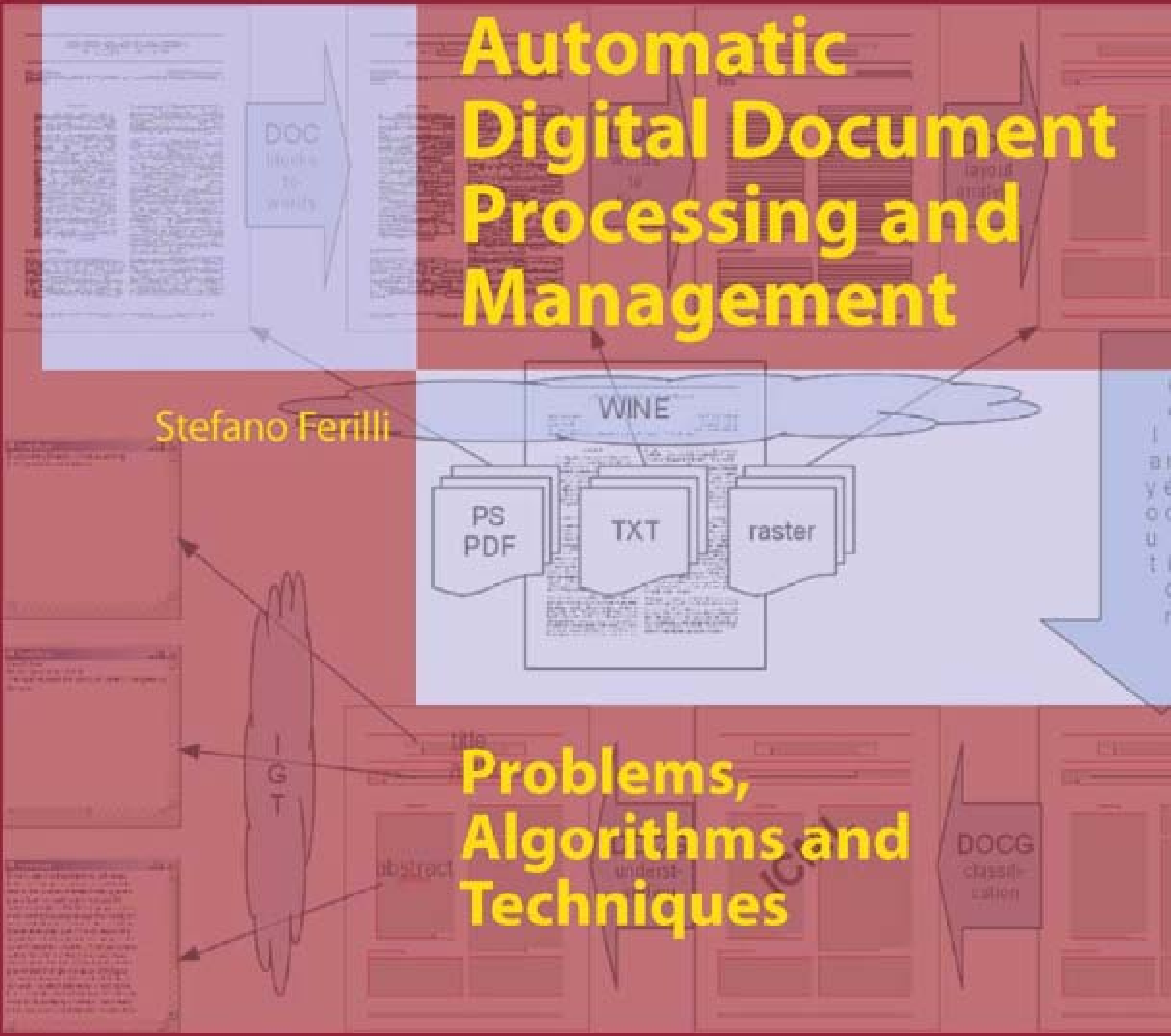
Advances in Pattern Recognition



# Automatic Digital Document Processing and Management

Stefano Ferilli

## Problems, Algorithms and Techniques

Springer

Advances in Pattern Recognition

Stefano Ferilli

# Automatic Digital Document Processing and Management

Problems, Algorithms and Techniques

Springer

Stefano Ferilli
Dipartimento di Informatica
Università di Bari
Via E. Orabona 4
70126 Bari
Italy
ferilli@di.uniba.it

*Series Editor*
Professor Sameer Singh, PhD
Research School of Informatics
Loughborough University
Loughborough
UK

ISSN 1617-7916
ISBN 978-0-85729-197-4 e-ISBN 978-0-85729-198-1
DOI 10.1007/978-0-85729-198-1
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

*Cover design*: VTEX, Vilnius

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

*Verba volant, scripta manent*

# Foreword

Imagine a world without documents! No books, magazines, email, laws, and recipes. For better or worse, in our world documents outnumber every other kind of artifact. The Library of Congress contains over 20 million books plus another 100 million items in the special collections. Google indexes about 20 billion web pages (July 2010). Both of these are growing like topsy. There are far more documents than houses, cars, electronic gadgets, socks, and even rubber bands.

Since the dawn of antiquity, documents have played an essential role in fostering civilizations. One could make a plausible argument that material and social progress are proportional to document density. It helps, of course, if the documents are widely available rather than kept under lock and key in a monastery or the royal library. The first chapter of this book highlights the rapid transition of juridical and commercial records from paper to electronic form. The rest of the book traces the contemporaneous evolution of digital document processing from esoteric research into a pervasive technology.

The variety of physical embodiments of paper documents (multivolume encyclopedias, pocket books, newspapers, magazines, passports, driver's licenses) is easily matched by the number of file types, generally known by acronyms like DOC, PDF, HTM, XML, TIF, GIF. The suffix indicates the type of processing appropriate for each file type. Differences between file types include how they are created, compressed, decompressed, and rendered in a human-readable format. Equally important is the balance between ease of modification of textual content (linguistic or logical components) and appearance (physical or layout components). Professor Ferilli offers an up-to-date tour of the architecture of common document file types, lists their areas of applicability, and provides detailed explanations of the notions underlying classical compression algorithms.

Important paper documents like deeds and stock certificates are written or printed in indelible ink on watermarked stock, and then locked into a vault. Digital files stored on personal computers or transmitted over the internet must be similarly protected against malware launched by curious or malicious hackers. Perhaps surprisingly, many of the elaborate and recondite measures used to ensure the authenticity, secrecy, and traceability of digital documents are rooted in Number Theory. This is one of the oldest branches of pure mathematics, with many counterintuitive theorems related to the factorization of integers into prime numbers. The difficulty of factorizing large numbers is of fundamental importance in modern cryptography. Nevertheless, some current systems also incorporate symbol substitutions and shuffles borrowed from ancient ciphers.

Because of its applications to computer security, cryptography has advanced more in the last 40 years than in the previous three thousand. Among widely used systems based on the secret (symmetric) key and public (asymmetric) key paradigms are the Data Encryption Standard (DES), Rivest Ciphers (RCn), the Rivest, Shamir, Adleman (RSA) algorithm, and the Digital Signature Algorithm (DSA). One-way encryption is often used to encrypt passwords and to generate a digital fingerprint that provides proof that a file has not been altered. Other methods can provide irrefutable evidence of the transmission of a document from one party to another. Professor Ferilli expertly guides the reader along the often tortuous paths from the basic mathematical theorems to the resulting security software.

There is a piquant contrast between the role of national governments in promoting the use of secure software to facilitate commerce and to grease the wheels of democracy, and its duty to restrict the propagation of secure software in order to protect its military value and to maintain the ability of law enforcement agents to access potentially criminal communications. This books reviews recent legislation with emphasis on relevant Italian, British and European Community laws.

While we are not yet ready to declare all written material that exists only on paper as legacy documents, that moment cannot be very far. Professor Ferilli is a key member of a research team that has made steady progress for upwards of 30 years on the conversion of scanned paper documents to digital formats. He presents a useful overview of the necessary techniques, ranging from the low-level preprocessing functions of binarization and skew correction to complex methods based on first-order logic for document classification and layout analysis. Many of the algorithms that he considers best-in-class have been incorporated, after further tuning, into his group's prototype document analysis systems.

For scanned documents, the transition from document image analysis to content analysis requires optical character recognition (OCR). Even born-digital documents may require OCR in the absence of software for reading intermediate file formats. OCR is discussed mainly from the perspective of open source developments because little public information is available on commercial products. Handwritten and hand-printed documents are outside the scope of this work.

Some documents such as musical scores, maps and engineering drawings are based primarily on long-established and specialized graphic conventions. They make use of application-specific file types and compression methods. Others, like postal envelopes, bank checks and invoice forms, are based on letters and digits but don't contain a succession of sentences. Except for some sections on general image processing techniques and color spaces, the focus of this book is on documents comprising mainly natural language.

The value of digital documents transcends ease of storage, transmission and reproduction. Digital representation also offers the potential of the use of computer programs to find documents relevant to a query from a corpus and to answer questions based on facts contained therein. The corpus may contain all the documents accessible on the World Wide Web, in a digital library, or in a domain-specific collection (e.g., of journals and conference reports related to digital document processing). For text-based documents, both information retrieval (IR) and query-answer (QA) systems require natural language processing (NLP). Procedures range from establishing the relative frequency, morphology and syntactic role of words to determining the sense, in a particular context, of words, phrases and sentences. Often simple relationships with arcane names, like synonymy, antinomy, hyperonymy, hyponymy, meronymy and holonymy, are sought between terms and concepts. Fortunately, the necessary linguistic resources like lexicons, dictionaries, thesauri, and grammars are readily available in digital form.

To avoid having to search through the entire collection for each query, documents in large collections—even the World Wide Web—are indexed according to their putative content. Metadata (data about the data, like catalog information) is extracted and stored separately. The relevant NLP, IR and IE techniques (based on both statistical methods and formal logic) and the management of large collection of documents are reviewed in the last two chapters.

The study of documents from the perspective of computer science is so enjoyable partly because it provides, as is evident from this volume, many opportunities to bridge culture and technology. The material presented in the following pages will be most valuable to the many researchers and students who already have a deep understanding of some aspect of document processing. It is such scholars who are likely to feel the need, and to harvest the benefits, of learning more about the growing gamut of techniques necessary to cope with the entire subject of digital document processing. Extensive references facilitate further exploration of this fascinating topic.

Rensselaer Polytechnic Institute Prof. George Nagy

# Preface

Automatic document processing plays a crucial role in the present society, due to the progressive spread of computer-readable documents in everyday life, from informal uses to more official exploitations. This holds not only for new documents, typically born digital, but also for legacy ones that undergo a digitization process in order to be exploited in computer-based environments. In turn, the increased availability of digital documents has caused a corresponding increase in users' needs and expectations. It is a very hot topic in these years, for both academy and industry, as witnessed by several flourishing research areas related to it and by the ever-increasing number and variety of applications available on the market. Indeed, the broad range of document kinds and formats existing today makes this subject a many-faceted and intrinsically multi-disciplinary one that joins the most diverse branches of knowledge, covering the whole spectrum of humanities, science and technology. It turns out to be a fairly complex domain even focusing on the Computer Science perspective alone, since almost all of its branches come into play in document processing, management, storage and retrieval, in order to support the several concerns involved in, and to solve the many problems raised from, application to real-world tasks. The resulting landscape calls for a reference text where all involved aspects are collected, described and related to each other.

This book concerns *Automatic Digital Document Processing and Management*, where the adjective 'digital' is interpreted as being associated to 'processing and management' rather than to 'document', thus including also digitized documents in the focus of interest, in addition to born-digital ones. It is conceived as a survey on the different issues involved in the principal stages of a digital document's life, aimed at providing a sufficiently complete and technically valid idea of the whole range of steps occurring in digital document handling and processing, instead of focusing particularly on any specific one of them. For many of such steps, fundamentals and established technology (or current proposals for questions still under investigation) are presented. Being the matter too wide and scattered, a complete coverage of the significant literature is infeasible. More important is making the reader acquainted of the main problems involved, of the Computer Science branches suitable for tackling them, and of some research milestones and interesting approaches available. Thus, after introducing each area of concern, a more detailed description is given of selected algorithms and techniques proposed in this field along the past decades. The choice was not made with the aim of indicating the best solutions available in the state-of-the-art (indeed, no experimental validation result is reported), but rather for the purpose of comparing different perspectives on how the various problems can be faced, and possibly complementary enough to give good chance of fruitful integration.

The organization of the book reflects the natural flow of phases in digital document processing: acquisition, representation, security, pre-processing, layout analysis, understanding, analysis of single components, information extraction, filing, indexing and retrieval. Specifically, three main parts are distinguished:

**Part I** deals with digital documents, their role and exploitation. Chapter 1 provides an introduction to documents, their history and their features, and to the specific digital perspective on them. Chapter 2 then overviews the current widespread formats for digital document representation, divided by category according to the degree of structure they express. Chapter 3 discusses technological solutions to ensure that digital documents can fulfill suitable security requirements allowing their exploitation in formal environments in which legal issues come into play.

**Part II** introduces important notions and tools concerning the geometrical and pictorial perspective on documents. Chapter 4 proposes a selection of the wide literature on image processing, with specific reference to techniques useful for handling images that represent a whole digitized document or just specific components thereof. Chapter 5 is devoted to the core of processing and representation issues related to the various steps a document goes through from its submission up to the identification of its class and relevant components.

**Part III** analyzes the ways in which useful information can be extracted from the documents in order to improve their subsequent exploitation, and is particularly focused on textual information (although a quick glance to the emerging field of image retrieval is also given). Chapter 6 surveys the landscape of Natural Language Processing resources and techniques developed to carry out linguistic analysis steps that are preliminary to further processing aimed at content handling. Chapter 7 closes the book dealing with the ultimate objective of document processing: being able to extract, retrieve and represent, possibly at a semantic level, the subject with which a document is concerned and the information it conveys.

Appendices A and B briefly recall fundamental Machine Learning notions, and describe as a case-study a prototypical framework for building an intelligent system aimed at merging in a tight cooperation and interaction most of the presented solutions, to provide a global approach to digital documents and libraries management.

The book aims at being self-contained as much as possible. Only basic computer science and high-school mathematical background is needed to be able to read and understand its content. General presentation of the various topics and more specific aspects thereof are neatly separated, in order to facilitate exploitation by readers interested in either of the two. The technical level is, when needed, sufficiently detailed to give a precise account of the matter presented, but not so deep and pervasive as to discourage non-professionals from usefully exploiting it. In particular, most very technical parts are limited to sub-subsections, so that they can be skipped without losing the general view and unity of the contents. To better support readers, particular care was put in the aids to consultation: the index reports both acronyms and their version in full, and in case of phrases includes entries for all component terms; the glossary collects notions that are referred to in different places of the book, so that a single reference is provided, avoiding redundancy; the acronym list is very detailed, including even items that are used only once in the text but can be needed in everyday practice on document processing; the final Reference section collects all the bibliography cited in the text.

The main novelty of this book lies in its bridging the gaps left by the current literature, where all works focus on specific sub-fields of digital document processing but do not frame them in a panoramic perspective of the whole subject nor provide links to related areas of interest. It is conceived as a monograph for practitioners that need a single and wide-spectrum *vade-mecum* to the many different aspects involved in digital document processing, along with the problems they pose, noteworthy solutions and practices proposed in the last decades, possible applications and open questions. It aims at acquainting the reader with the general field and at being complemented by other publications reported in the References for further in-depth and specific treatment of the various aspects it introduces. The possible uses, and connected benefits, are manifold. In an academic environment, it can be exploited as a textbook for undergraduate/graduate courses interested in a broad coverage of the topic.[1] Researchers may consider it as a bridge between their specific area of interest and the other disciplines, steps and issues involved in Digital Document Processing. Document-based organizations and final users can find it useful as well, as a repertoire of possible technological solutions to their needs.

Although care has been put on thorough proof-reading of the drafts, the size of this work makes it likely that some typos or other kinds of imprecisions are present in the final version. I apologize in advance for this, and will be grateful to anyone who will notify me about them.

Bari, Italy Stefano Ferilli

[1]The included material is too much for a semester, but the teacher can select which parts to stress more and which ones to just introduce.

# Acknowledgments

There are many persons that deserve my acknowledgments after finishing the writing of this book. First of all, I am grateful to Prof. Floriana Esposito for introducing me to the charming and challenging application field of Digital Document Processing as an interesting target of my research in Machine Learning. Also, I thank all the colleagues (and friends) in the Machine Learning research group at the University of Bari, with which I shared my adventure in Computer Science research along the last 15 years or so. Through stimulating discussions, useful suggestions and interesting ideas, they have been fundamental for deepening my understanding of document processing, and for defining the perspective, on it [i.e., on document processing], that [i.e. the perspective] underlies this work. Thanks to Professor George Nagy for the precious and insightful comments on parts of the book, to Wayne Wheeler at Springer, who believed in this project, and to Simon Rees for his continuous editorial support. Last but not least, a special mention goes to my family, and in particular to my wife and my parents, that I partly neglected in these years while attending to my professional interests, but that nevertheless always supported and encouraged me.

A credit is to be given to the whole community working on Open Source projects. Their products help me in everyday practice and have been fundamental for accomplishing this work. In particular, I would like to mention:

- the kubuntu Linux distribution (www.kubuntu.org)
- the OpenOffice.org suite (www.openoffice.org)
- LaTeX (http://www.latex-project.org/)
- the GIMP (www.gimp.org)
- Mozilla (www.mozilla.org)

The WIKIpedia project (www.wikipedia.org) has been often useful in providing introductory overviews of specific subjects and in suggesting interesting initial references for a more thorough treatment. Some figures in the text appear courtesy of the Institute for Electrical and Electronic Engineering (IEEE) and of Springer.

# Contents

## Part II Document Analysis

# Acronyms

*Many people are persuaded that the Computer Science community, in addition to the Military one, have a sort of maniacal inclination towards the use of acronyms. To enforce this, in the following a (short) list of most important acronyms of technologies and organizations referred to in this book is provided.*

| | |
|---|---|
| 2D | 2-Dimensional |
| 3D | 3-Dimensional |
| ANN | Artificial Neural Network, a Machine Learning technique |
| API | Application Programming Interface |
| ASCII | American Standard Code for Information Interchange |
| BMP | Basic Multilingual Plane, a portion of Unicode |
| BMP | BitMaP, an image representation format |
| BoW | Bag Of Words, a simplified representation of natural language texts |
| BoS | Bag Of Senses, a conceptual representation of natural language texts |
| CA | Certificate Authority |
| CBIR | Content-Based Image Retrieval |
| CIE | Commission Internationale d'Èclairage (www.cie.co.at) |
| CI | Concept Indexing |
| CMYK | Cyan Magenta Yellow Key, a color space |
| CPS | Certificate Practice Statement |
| CRC | Cyclic Redundancy Check, an error control system for data transmission |
| CRL | Certificate Revocation List |
| CRT | Cathode Ray Tube |
| CSL | Certificate Suspension List |
| CSS | Cascading Style Sheets, files containing formatting instructions for the presentation of a document written in a markup language, such as HTML or XML |
| CTM | Current Transformation Matrix, a PostScript component |
| DAG | Directed Acyclic Graph |
| DCT | Discrete Cosine Transform |

| | |
|---|---|
| DDC | Dewey Decimal Classification, an encoding scheme for document subjects |
| DES | Data Encryption Standard |
| DMCI | Dublin Core Metadata Initiative |
| DOI | Digital Object Identifier |
| DOM | Document Object Model, an XML document representation standard |
| DPCM | Differential Pulse Code Modulation |
| DPI | Dots Per Inch, a measure of printing/scanning quality |
| DPR | Decree of the President of the Republic (Decreto del Presidente della Repubblica), a kind of act in the Italian law system |
| DSA | Digital Signature Algorithm |
| DSS | Digital Signature Standard |
| DTD | Document Type Definition, a specification of biases on XML syntax |
| EBCDIC | Extended Binary Coded Decimal Interchange Code |
| FIPS | Federal Information Processing Standard |
| FN | False Negative |
| FOL | First-Order Logic |
| FP | False Positive |
| FTP | File Transfer Protocol, an Internet service |
| gcd | Greatest Common Divisor, the largest integer that divides given integers |
| GIF | Graphics Interchange Format, an image representation format |
| GPG | GNU Privacy Guard, a non-commercial implementation of PGP |
| GUI | Graphical User Interface |
| HMM | Hidden Markov Model, a Machine Learning technique |
| HSV | Hue Saturation Value, a color space |
| HSB | Hue Saturation Brightness, a color space |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| HTTPS | HyperText Transfer Protocol Secure |
| IANA | Internet Assigned Numbers Authority (http://www.iana.org) |
| ICC | International Color Consortium |
| ICT | Information and Communication Technologies |
| IDEA | International Data Encryption Algorithm |
| IE | Information Extraction |
| IEEE | Institute of Electrical and Electronics Engineers (http://www.ieee.org) |
| IEC | International Electrotechnical Commission (http://www.iec.ch) |
| IETF | Internet Engineering Task Force |
| IFD | Image File Directory, a data block in TIFF files |
| ILP | Inductive Logic Programming, a Machine Learning paradigm |
| IP | Internet Protocol |
| IR | Information Retrieval |
| ISBN | International Standard Book Number |
| ISO | International Organization for Standardization (http://www.iso.org) |
| JPEG | Joint Photographic Experts Group, an image representation format |
| $k$-NN | $k$-Nearest Neighbor, a Machine Learning technique |

| | |
|---|---|
| KE | Keyword Extraction, an Information Extraction task |
| LCD | Liquid Crystal Display |
| LED | Light-Emitting Diode |
| LIFO | Last In First Out, a stack-based data organization |
| LSA | Latent Semantic Analysis |
| LSI | Latent Semantic Indexing |
| LZW | Lempel–Ziv–Welch, a compression algorithm |
| MIME | Multipurpose Internet Mail Extensions, a standard to identify the representation format of various types of information |
| MIPS | Millions Instructions Per Second, a measure of computer processing speed |
| ML | Machine Learning |
| MNG | Multiple-image Network Graphics, an image representation format |
| NISO | National Information Standards Organization |
| NIST | National Institute of Standards and Technology |
| NLP | Natural Language Processing |
| NSA | National Security Agency |
| OASIS | Organization for the Advancement of Structured Information Standards |
| OCLC | On-line Computer Library Center |
| OCR | Optical Character Recognition |
| ODF | Open Document Format, a suite of formats for office documents |
| OOXML | Office Open XML, a suite of formats for office documents |
| PA | Public Administration |
| PCA | Principal Component Analysis |
| PCKS | Public Key Cryptography Standard |
| PCT | Processo Civile Telematico (Telematic Civil Proceedings) |
| PDF | Portable Document Format |
| PDL | Page Description Language |
| PES | Proposed Encryption Standard |
| PGP | Pretty Good Privacy, a digital signature standard (http://www.pgpi.com) |
| PKI | Public Key Infrastructure, a digital certificate scheme |
| PNG | Portable Network Graphics, an image representation format |
| PoS | Part of Speech |
| PS | PostScript, a digital document format |
| PUA | Private Use Area, a portion of Unicode |
| RA | Registration Authority |
| RFC | Request For Comment, a stage of the standardization process |
| RGB | Red Green Blue, a color space |
| RLE | Run Length Encoding, a compression algorithm |
| RLSA | Run Length Smoothing Algorithm, a document image segmentation algorithm |
| RSA | Rivest–Shamir–Adleman, a cryptographic system |
| RTF | Rich Text Format, a format for word processing documents |
| SAX | Simple API for XML |

| | |
|---|---|
| SGML | Structured Generalized Markup Language, standard ISO 8879:1986 |
| SHA | Secure Hash Algorithm |
| SIP | Supplementary Ideographic Plane, a portion of Unicode |
| SMP | Supplementary Multilingual Plane, a portion of Unicode |
| SSL | Secure Socket Layer, a secure protocol for TCP/IP Transport Level |
| SSP | Supplementary Special-purpose Plane, a portion of Unicode |
| SVD | Singular Value Decomposition |
| SVG | Scalable Vector Graphics, an image representation format |
| TC | Text Categorization |
| TCP | Transfer Control Protocol |
| TIFF | Tagged Image File Format |
| TLS | Transport Layer Security, a secure protocol for TCP/IP Transport Level |
| TN | True Negative |
| TP | True Positive |
| TTP | Trusted Third Party |
| UCS | Universal Character Set |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator, a standard address for objects on the Internet |
| UTF | Unicode/UCS Transformation Format |
| XML | eXtensible Markup Language, a format for Web document and information representation |
| XSL | eXtensible Stylesheet Language, a language to specify styles for XML documents |
| XSLT | XSL Transformations, a language to specify transformations for XML documents |
| YUV | A color space |
| W3C | (*World Wide Web Consortium*) A no-profit international organization that develops and fosters standards for the WWW (www.w3.org) |
| WSD | Word Sense Disambiguation |
| WWW | World Wide Web, the hypertextual section of the Internet |

# Part I
# Digital Documents

This part of the book introduces digital documents, providing a running definition for what is going to be intended as a *document* throughout the rest of the book, discussing a number of widespread formats for representing documents, each with its *pros and cons* with respect to the different needs for which they are produced and processed, and summarizing the most important problems and solutions involved in using digital documents in official environments.

Documents are fundamental in our lives because they are the means to convey information and/or the formal evidence of something. Although they date back to the first times of human civilization, the current technological development has turned upside-down many classical aspects of document creation, management, processing and exploitation. In particular, the advent of computer systems has made much easier document production and transmission, which is an opportunity but also a source of problems, due to the many different kinds of documents at hand, and to the different perspectives and needs of the institutions interested in document management and of their users. In Chap. 1, after reviewing various perspectives on documents and providing a historical survey of their evolution, various categories of documents are identified, and a particular focus on digital ones is given.

A problem to be faced very early when dealing with digital documents is how to represent them in a suitable machine-readable format. Thus, the current widespread formats for digital documents representation are presented in Chap. 2, divided by category according to the degree of structure they express. Formats that do not exhibit any high-level structure for their content are dealt with first: plain text and image formats. Then, the formats containing information on spatial placement of the document elements are introduced. Lastly, a selection of formats organized according to the content and function of the document components are presented, including Web formats and the official standard for text processing.

The characteristics of digital documents that are desirable for normal tasks pose severe security problems in official documents, which must be tackled and solved in order to foster digital document adoption in formal environments. Chapter 3 deals with security and legal aspects concerning digital documents and their production, exchange and exploitation. This is a hot topic as long as digital documents are progressively taking the place of classical paper ones also in the administration context, where issues such as property, non-withdrawal, certification, protection of content cannot be ignored. Hence, the current trends in cryptography and digital signature technologies to solve these issues are proposed, along with the legal initiatives in various countries to enforce digital document exploitation.

# Chapter 1
# Documents

The word *document* comes from the Latin word ‘documentum’, which has the same stem as the verb ‘doceo’ (meaning ‘to teach’), plus the suffix ‘-umentum’ (indicating a means for doing something). Hence, it is intended to denote ‘a means for teaching’ (in the same way as ‘instrument’ denotes a means to build, ‘monument’ denotes a means to warn, etc.). Dictionary definitions of a document are the following [2]:

1. Proof, Evidence
2. An original or official paper relied on as basis, proof or support of something
3. Something (as a photograph or a recording) that serves as evidence or proof
4. A writing conveying information
5. A material substance (as a coin or stone) having on it a representation of thoughts by means of some conventional mark or symbol.

The first definition is more general. The second one catches the most intuitive association of a document to a paper support, while the third one extends the definition to all other kinds of support that may have the same function. While all these definitions mainly focus on the bureaucratic, administrative or juridic aspects of documents, the fourth and fifth ones are more interested in its role of information bearer that can be exploited for study, research, information. Again, the former covers the classical meaning of documents as written papers, while the latter extends it to any kind of support and representation. Summing up, three aspects can be considered as relevant in identifying a document: its original meaning is captured by definitions 4 and 5, while definition 1 extends it to underline its importance as a proof of something, and definitions 2 and 3 in some way formally recognize this role in the social establishment.

## 1.1 A Juridic Perspective

A juridic definition of *document* is usually not provided by the regulations of the various countries, that assume it to be a quite natural and intuitive concept and hence take it for granted. Thus, there is no specific definition that identifies a particular object as a document. However, much literature has been produced about the subject in the past decades, and very interesting considerations can be found in the Italian landscape. Carnelutti[1] defines a document as “something that allows to know some fact” [8], in opposition to a witness, that is a person who reports a fact. On the other hand, Irti analyzes the problem putting more emphasis on the activity aimed at producing the document [9]. Under this perspective, the document becomes a *res signata* (a ‘marked object’), an *opus* (an ‘artifact’) resulting from human work. The marks are the outcome of an artificial process accomplished or regulated by the man itself, and intended to confer representativeness to the *res portata* (‘what is brought’, the content). As a consequence, the document is not the object in itself nor the mark in itself, but rather the object on which the man has acted.

Three main components must be taken into account when dealing with a document [3]:

- The *object*, or in any case “the physical or material element” to which such a meaning is given [7].
- The *fact* represented, that must be juridically relevant. From this perspective, documents can be distinguished between *direct* and *indirect* ones: the former place the interpreter before the actual fact (e.g., photographic or cinematographic documents), while the latter show a representation thereof derived from a mental processing [8].
- The *representation* provided. This perspective highlights the essentially intellectual aspect of the document domain: a *res* (an ‘object’) takes on the status of a document only because the person, who aims at exploiting it in that meaning, is provided with an intellectual code for understanding it [9].

As a consequence, it seems straightforward to conclude that the document, juridically intended, does not exist in nature, but exists only if the suitable circumstances hold to ascribe this particular meaning to an object [3].

## 1.2 History and Trends

A short historical survey may be useful to properly cast documents and their role. The most fundamental intellectual conquer of human beings of all times is probably the invention of language, which allowed them to ‘dump’ their thoughts on a physical support and to communicate to each other for sharing experience and coordinating their actions. On the one hand, improved communication allowed them to more effectively face the problems of everyday survival; on the other hand, the possibility of sharing experiences allowed them to boost their evolution, since every individual could get ‘for free’ previous knowledge without having to personally experience it (and everybody knows that *knowledge is power*, as the saying goes), and could just

[1] Francesco Carnelutti (1879–1965), a famous lawyer and jurist, main inspirer for the 1942 version of the Italian Civil Law.

focus on improving it. Originally, communication was just oral, which implied a strong bias due to the limited amount and effectiveness of human memory capability. Thus, the real turning point was reached when someone decided to concretely fix knowledge on a permanent physical means, which allowed its exploitation as a formal and objective proof independent on owner, place and time. This first happened in a pictorial fashion by explicitly representing the information as perceived by the senses, e.g., in the form of graffiti on cave walls. Then, by progressive levels of abstraction, symbols were introduced, first to represent concepts, as in the case of ideograms, and then by selecting and transforming some of them to represent basic vocal sounds to be variously combined in order to express all possible words and concepts without the need of continuously extending the set of symbols.

This new combination, made up of data/information/knowledge[2] plus a permanent physical means on which it is stored, is what we call a '*document*', and has characterized the last millennia of our evolution. In this general perspective, documents are multi-faceted objects, that can adopt and mix several different media (text, picture, sound, etc.), formalisms and supports (stone, clay, paper, etc.) to convey the desired information according to different needs and objectives. Hence, dealing with documents involves the ability to handle many kinds of different items, each with its peculiarities. From what is said above, it is clear (and everyone can realize it everyday) that documents are pervasive and essential elements in our existence. They are the foundations on which all social, administrative and scientific aspects of our civilization are built. As a consequence, the ability to produce, manage, handle, exchange and interpret them takes on a crucial importance in our society.

In the past, the accomplishment of all such tasks was affected by practical and logistic biases, because of the tangible (mainly paper) nature of the documents themselves, and thus typical problems were the cost for producing them, the consequent lack of copies thereof and the difficulty in transferring and obtaining them when needed. The introduction and spread of computer systems, of Computer Science and of telecommunications have led to overcoming large part of such barriers, and particularly of those related to the creation and distribution of documents.

## 1.3 Current Landscape

While up to recently documents were produced in paper format, and their digital counterpart was just a possible consequence carried out for specific purposes, nowadays we face the opposite situation: nearly all documents are produced and exchanged in digital format, and their transposition on a tangible, human-readable support has become a successive, in many cases optional, step. Additionally, significant efforts have been spent in the *digitization* of previously existing documents

[2]Usually, the three terms denote different but strictly related concepts: *data* are just values; they become *information* when an interpretation is given to them, while *knowledge* refers to pieces of information that are inter-related both among themselves and with other experience.

(i.e., their transposition into digital format), aimed at preservation and widespread fruition. Such a new approach, called *dematerialization*, is currently the general trend, even at governance levels, as proved by several laws issued in many countries to enforce the official exploitation of digital documents in the administration. A consequence has been the birth and spread of Digital Libraries, repositories that collect many different kinds of documents in digital format.

In this changed landscape, many document- and information-centered activities have found new opportunities to significantly improve both the quality and speed of work of their practitioners, and the effectiveness and accuracy of their outcomes, increasing as a consequence the productivity levels. An outstanding example is provided by the case of scientific research, for which the exchange, the continuous updating of, and the easiness of access to, data and theories are vital requirements that are indeed supported by the availability of documents in digital format and of efficient technologies to transmit them. However, as a tradeoff, this has also determined the obsolescence of classical work-flow practices and made more complex than ever the retrieval of useful information. Indeed, it gave rise to a dramatic proliferation of available documents, often of questionable quality and significance, which poses difficulties that are exactly opposite to those previously experienced for classical documents (a problem known as *information overloading*). In a nutshell, in the past the main difficulty consisted in gathering useful documents, but then it was easy to identify the desired information therein; now it is extremely easy to retrieve huge (and often overwhelming) quantities of documents, but then identifying useful information is in many cases almost impossible. In addition, documents are extremely varied (in format, type, language, etc.), with little or no structure, which gives little leverage for understanding the role and importance of their single components. Even more, potential users interested in accessing them belong to many different categories and are themselves extremely different as to aims and needs. All these issues make it significantly hard recognizing which information is to be considered as relevant, and extracting it in order to make it available to other applications or for specific tasks.

On the other hand, the information society in which we live bases almost all of its fundamental activities on documents content, so that easy and quick access to proper information, and its management and usability, become unavoidable factors for the economic, business-related, professional, social and cultural development and success. The World Bank, in the 1998 report [1], recognized that, for economically developed countries, knowledge, more than resources, has become the most important factor in defining the life standard, and that nowadays the majority of technologically advanced economies are in fact based on knowledge (again, *knowledge is power*).

Of course, putting human experts to analyze and annotate each and every single document in order to allow potential users to understand whether it is of interest to them or not is absolutely infeasible: the life-cycle of many documents is so short that they would become obsolete far sooner than they would get annotated. Hence, a strong motivation for the research in the field of automatic processing that is able to support or replace human intervention in all these knowledge-intensive activities by means of Artificial Intelligence techniques and systems. The automatic processing approach is obviously supported by the fact that the availability of digital documents makes it simple and allows to completely automatize, differently from their paper counterparts, the search for information in the document itself, at least at the syntactic level. Unfortunately, it is not immediate nor trivial bridging the gap from syntax to the semantic level, where content understanding comes into play. The increased computational power available nowadays, and the results of several years of research, have nevertheless allowed to start developing methodologies, techniques and systems that try to tackle automatically, at least partially, even these more complex aspects.

## 1.4 Types of Documents

The wide coverage of the definition of 'document' makes it impossible to provide an exhaustive systematization of the various kinds thereof. Nevertheless, some intuitive and useful categorizations of documents, based on different (although sometimes overlapping) perspectives, can be identified. In the following, a short list will be provided, according to which the matter of this book will be often organized:

**Support** A first distinction can be made between *tangible* or *intangible* supports.[3] The latter category includes digital documents, while the former can, in turn, be sensibly split into *written* documents (such as those on paper, papyrus or parchment) and documents whose content must be impressed on their supports (e.g., stone, clay, etc.) using other techniques.

**Time of production** A rough distinction can also be made between *legacy* documents (i.e., documents that were produced using classical, non-computerized techniques, for which an original digital source/counterpart is not available) and 'current' ones. The former typically correspond to tangible ones, and the latter to digital ones, but the overlapping is not so sharp. Indeed, tangible documents are still produced nowadays, although they have lost predominance, and the recent years have seen the widespread use of both kinds at the same time.

**Historical interest** Similar to the previous one, but in some sense more neat and born from a specific perspective, is the distinction between *historical* documents and 'normal' ones. Here, the focus is on the value of a document not just based on its content, but on its physical support as well. The document as an object, in this case, cannot be distinguished by its essence because even a scanned copy would not replace the original. Hence, this perspective is strictly related to the problems of preservation, restoration and wide access.

[3]Another term sometimes exploited to convey the same meaning is *physical*. However, it is a bit odd because even intangible supports require a physical implementation to be perceived, e.g., air is the physical support of the intangible 'sound' documents, and magnetic polarization or electrical signals are the physical support of the intangible 'digital' documents.

**Table 1.1** Some types of documents and their categorization

| | Support | Production | Interest | Medium | Structure | Formalism |
|---|---|---|---|---|---|---|
| Speech | intangible | any | normal | sound | no | natural language |
| Greek roll | tangible | legacy | historical | text, graphic | spatial | natural language |
| Book | tangible | legacy | any | text, graphic | spatial | natural language |
| Picture | tangible | legacy | any | graphic | no | light |
| Music | intangible | any | normal | sound | no | temperament |
| Program code | intangible | current | normal | text | content | programming language |
| Web page | intangible | current | normal | text, graphic, sound | content | tagged text |

**Medium** Another obvious distinction is based on the kind of medium that is used to convey the information. In the case of paper documents, the choice is limited between text and graphics, while in digital documents other kinds of media, such as sound, might be included.

**Structure** Moving towards the document content, different categories can be defined according to the degree of structure they exhibit. The concept of structure is typically referred to the meaningful arrangement of document components to form the document as a whole, but can sometimes be referred to the inner structure of the single components as well.

**Representation formalism** Also related to the content is the categorization by representation formalism. There are different formalisms for both the paper documents, whose interpretation is intended for humans, and for digital documents, whose interpretation is intended for computer and telecommunication systems. However, often the digital documents are intended for humans as well, in which cases both levels of representation are involved.

As can be easily noted, these kinds of categorizations show some degree of orthogonality, and thus allow several possible combinations thereof (Table 1.1 reports a few sample cases). Of course, not all combinations will bear the same importance or interest, but some are more typical or outstanding and deserve particular attention. Specifically, this book focuses on documents (and document components) from a visual and understanding perspective. Thus, a selection on the support and media types is immediately obtained: as to the former, we are interested in digital and paper documents only, while as to the latter only text and graphics will be considered. In other words, only 'printable' documents and components will be dealt with, not (e.g.) music.

An interesting essay on what is a document in general, and a *digital* document in particular, can be found in [5, 6], and will be summarized and discussed hereafter. A historical survey of the literature reveals that a 'document' is, traditionally, often intended as a ‘textual record’. Such a quite restrictive definition is extended by some authors to include any kind of artifact that fulfills the ‘object as a sign’ perspective. Underlying is the assumption that a document must be a physical item that is generally intended and perceived as a document, and, although less convincing for generic documents—but not, as we will see shortly, for digital documents—must also be the product of some (human) processing [4]. As a further requirement, it has to be cast into an organized system of knowledge. Indeed, the question concerning the distinction between medium, message and meaning is a very old one.

This question becomes again topical with digital technology that represents everything as a sequence of bits. In fact, once one accepts that a word processor output (probably the digital artifact most similar to what is classically intended as a document) is to be considered a document, it must also be accepted that everything having the same representation is a document as well. It is only a matter of interpretation, just like one alphabet (say, the Latin one) can express many different natural languages. This sets the question of ‘what a document is’ free from any dependence on specific supports because multi-media all reduce, at the very end, to a single medium that is a sequence of (possibly electronically stored) bits. Indeed, some define a digital document as anything that can be represented as a computer file. This is the very least definition one can think of, and at the same time the most general. At a deeper analysis, the point is that there is no medium at all: the bits exist by themselves, without the need for the support of physical means for making them concrete, although this is obviously needed to actually store and exploit them.

Moreover, accepting the thesis that what characterizes a document is its being a ‘proof’ of something, two consequences immediately spring out. First, it must carry information, and information is expressed and measured in bits [10]. Thus, anything that can (at least in principle, and even with some degree of approximation) be expressed in terms of sequences of bits is a document. Second, it must be fixed in some shape that can be distributed and permanently preserved. Hence, the document needs a support, and thus any support capable of, and actually carrying, significant bits in a stable way is a document from a physical perspective. Thus, digital documents can settle the controversy on whether the physical support is discriminant in determining if an object is a document or not.

The problem of digital documents, with respect to all other kinds of documents preceding the digital era, is that they are the first human artifact that is outside the human control. Indeed, all previous kinds of documents (principally text and picture ones, but others as well) could be, at least in principle, directly examined by humans without the need for external supporting devices, at least from the syntactic perspective. Although allowing an extremely more efficient processing, this imposes a noteworthy bias to their final users, and introduces the risk of not being able to access the document information content because of technology obsolescence factors, which after all spoils the object of the very feature that makes it a document.

## 1.5 Document-Based Environments

As already pointed out, almost all aspects of the social establishment in which we live, and for sure all formal aspects thereof, are based on documents, and represent sources of needs and problems for their management and processing. Again, providing a complete survey of the several kinds of document-based environments is not easy, due both to their variety and also to their many inter-relationships, but a rough discussion of this specific issue can be nevertheless useful to further increase awareness of the importance and complexity of the overall subject.

Two typical environments to which one thinks when considering documents are *archives* and *libraries*. Indeed, they are the traditional institutions in which humans have tried ever-since to store large amounts of documents for easy retrieval and consultation. Ancient civilizations used archives for official documents (just think of the repository of linear A and B clay tablets found in Knossos, Crete, and related to the Mycenaean age—*circa* 2000 BC), and had the dream of collecting as much as possible of the corpus of knowledge available at that time in designated places (famous projects of libraries date back to *circa* 1000 BC, and flourished throughout the ancient times; particularly noteworthy are the libraries of Pergamum, Greece—today Turkey—and Alexandria, Egypt). Archives are more oriented to private documents, while libraries are devoted to publications. In times when documents were tangible, and consulting a document involved its physical possession, the easiest solution to allow people have access to large amounts of information was to collect huge quantities of documents in a few places, and require the users to move to those places. Nowadays, on the other hand, the digital facility allows people to have access to the documents at any moment and in any place, and to be physically separated from them because they are easily duplicated and transmitted. Thus, although their aims are the same as those of classical libraries, digital libraries are very different as to organization and management.

Other places where documents play a central role are the offices of administrations, professionals and secretariats. The documents involved in such environments are usually bureaucratic ones, such as forms, letters and administration communications. Differently from archives, documents in this environment are of current use and interest, and hence require easy and immediate retrieval whenever needed. They must typically fulfill strict format specifications, as in the case of forms, bills and commercial letters, and their management involves taking into account aspects related to time and to compliance with laws and practices of other institutions.

The academic and research environments are another significant example of document-based environment. For some respects, they sum up the features of libraries, archives and offices: indeed, they have to store lots of documents for both consultation and administration purposes. However, some activities require further peculiar functionality that is not present in those settings. A typical example is conference management, where the basic functionality for document management and processing must be extended with support for publication of logistic and scientific information, submission and reviewing of papers, production of abstracts and proceedings, author and participant management, etc.

## 1.6 Document Processing Needs

Depending on various combinations of document types, environments and user objectives, lots of different document processing tasks can be identified, in order to satisfy the several needs of many document-based activities and institutions. In new documents, the needs are mainly related to semantics and interpretation of the content, and hence on information retrieval and extraction. Conversely, going back to legacy documents, additional problems, that tend towards syntax and representation subjects, arise and become stratified on top of those.

On legacy documents, the problem is how to turn them into a digital format that is suitable for their management using computer systems. The most straightforward way to obtain this is scanning, which usually introduces as a side-effect various kinds of noise, that must be removed. For instance, typical problems to be dealt with are bad illumination, page image distortions (such as skew angles and warped papers) and introduction of undesired layout elements, such as specks and border lines due to shadows. Additional problems are intrinsic to the original item:

- Presence of undesired layout elements, as in the case of bleedthrough;
- Overlapping components, such as stamps and background texture;
- Lack of layout standard, in the document organization or in its content, as for handwritten or non-standard alphabets and letters.

Moreover, often the digitization process must include extraction of the document content as well, in order to store each component thereof in a way that is suitable for its specific kind of content. In this case, simple scanning or photographing the document is not sufficient. Indeed, for being properly stored and processed, the textual part of the document should be identified and represented as text, while images should be cropped and stored separately. In turn, this might impose additional requirements on how scanning is performed.

A serious problem related to preservation of legacy documents is the durability of the copy and its actual exploitability in the future. Indeed, in spite of the appearance, paper is more durable than many other supports (e.g., magnetic disks and tapes are easily damaged, but also the actual endurance of current optical disks is not known). An example may clarify the matter: a few decades ago some Public Administrations decided to transpose many of their archived papers, often important official documents, onto microfilms. Microfilms have a limited quality, they are delicate and can be read only by specific instruments. Such instruments, and the making of microfilms itself, were very costly, and for this reason microfilm technology was soon abandoned in favor of scanning by means of computer systems. Those instruments are no longer produced, and the existing ones are progressively disappearing, so the preservation problem strikes back. Even worse, in some cases the administrations, after transposing on microfilms their legacy documents, have destroyed the originals to make room in their archives. Thus they now are not able to read the microfilms and do not have the original either: there is a real danger of having lost those documents. The lesson gained from this negative experience is that the adoption of a new technology must ensure that the documents will be accessible in the future, independently of the availability of the same instruments by which they were created.

For printable documents (probably the most widespread and important ones—like administrative records, deeds and certificates) the best solution might be to provide, in addition to the digital version (to be exploited in current uses) a few 'official' paper copies, to be stored and exploited in case, for some reason (e.g., technological obsolescence or failure, black-outs, etc.) the digital counterpart is not accessible.

On historical documents, the main interest relies in their restoration and preservation for cultural heritage purposes. Restricting to written documents, the source format is typically paper, papyrus or parchment, and hence digitization can be generally obtained by means of a scanning process in order to maintain the original aspect of the artifact, in addition to its content. All the problems of legacy documents are still present, but additional ones arise, due to the low quality and standardization of the original, such as in the case of missing pieces (particularly in ancient fragments).

On digital documents (both born-digital ones and legacy ones that have been digitized according to the above criteria), a first issue is to develop suitable representation formalisms and formats that can express different kinds of content in an effective, compact and machinable way (three requirements that are often opposite to each other, thus imposing some kind of trade-off). More generally, the interest lies in properly indexing them for improving retrieval performance, and in extracting from them relevant information that can be exploited for many purposes, such as indexing itself, categorization, understanding, etc.

Lastly, when switching from single documents to homogeneous collections, organizational aspects come also into play, and must be properly tackled. These aspects obviously include (or at least involve) the indexing and information extraction issues described in the previous paragraph, but also raise additional needs strictly related to the overall purposes of the collection, and not only to the document content itself. Categorization and clustering of documents is often a fundamental requirement in all contexts, but distribution and delivery of the proper documents to the various users of the system might play a crucial role as well. For instance, a digital library is interested in associating documents to categories of interest; an e-mail system can derive its success from the ability to precisely discriminate spam messages; an on-line bookshop aims at suggesting effective recommendations according to users' desires; in an office typical needs are interoperability, compliance with external specifications and security aspects; and so on.

## References

1. Knowledge for development: Tech. rep. The World Bank (1998/1999)
2. Merriam-Webster's Collegiate Dictionary, 10th edn. Merriam-Webster Inc. (1999)
3. Angelici, C.: Documentazione e documento (diritto civile). In: Enciclopedia Giuridica Treccani, vol. XI (1989) (in Italian)
4. Briet, S.: Qu'est-ce que la Documentation. EDIT, Paris (1951)
5. Buckland, M.K.: What is a 'document'? Journal of the American Society for Information Science **48**(9), 804–809 (1997)
6. Buckland, M.K.: What is a 'digital document'? Document Numérique **2**(2), 221–230 (1998)

7. Candian, A.: Documentazione e documento (teoria generale). In: Enciclopedia Giuridica Treccani (1964) (in Italian)
8. Carnelutti, F.: Documento—teoria moderna. In: Novissimo Digesto Italiano (1957) (in Italian)
9. Irti, N.: Sul concetto giuridico di documento. In: Riv. Trim. Dir. e Proc. Civ. (1969) (in Italian)
10. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press, Champaign (1949)

# Chapter 2
# Digital Formats

Recent developments in computer science disciplines have caused an ever-increasing spread of digital technologies, which has consequently turned upside down nearly all human activities in many of their aspects. At the same time, the fragmentation of the computer science landscape and the tough competition among software producers have led to a proliferation of different, and unfortunately often incompatible, ways of representing documents in digital format. Only very recently, initial efforts have been started aimed at rationalization and standardization of such formats, although it must be said that some (limited) range of variability is needed and desirable to be able to properly represent documents having very different characteristics (for instance, it is hardly conceivable that different kinds of content as, say, text, images and music will be ever represented using the same 'language').

In the following, some of the most widespread formats will be introduced, grouped according to the kind and degree of structure they allow expressing in a document content representation. Raster image formats will be dealt with more in-depth, for showing and comparing different perspectives on how visual information can be encoded. Also PS and PDF will be given somewhat larger room because they are famous from an end-usage perspective, but are not very well-known as to internal representation. Also for comparison purposes to the raster formats, a particular attention will be given to their image representation techniques. Markup languages will be discussed with reference to HTML and XML. However, the aim here will not be providing an HTML/XML programmer's manual (many valid books on this specific subject are available); rather, their representational rationale and approach will be stressed. HTML will be discussed with just the aim of considering a tag-based format and the possibilities it provides, while XML will be presented to give an idea of how flexible information representation can be obtained by a general tag definition mechanism supported by external processing techniques.

## 2.1 Compression Techniques

The availability of digital documents and the consequent need to store huge quantities thereof and to transmit them over computer networks have raised the need for reducing the amount of memory required to represent a document. In the former case, this would allow keeping together on one support large repositories of related documents; in the latter, this would enable quick delivery of documents to their final users. Such problems relate particularly, but not exclusively, to digital image representation, due to the large amount of data to be stored. Thus, a primary interest in this field has always been the development of effective and efficient techniques for *data compression*. *Lossless* compression techniques ensure that the original uncompressed data can be exactly restored, at the cost of a lower compression ratio. Conversely, *lossy* techniques allow exchanging a reasonable amount of quality in image reconstruction for significant improvement in compression performance. With this subject being pervasive in digital document representation, it will be discussed preliminarily in the presentation of the specific digital formats, by considering outstanding solutions that have been proposed and widely adopted so far.

**RLE (Run Length Encoding)** The *RLE* algorithm performs a lossless compression of input data based on sequences of identical values (called *runs*). It is a historical technique, originally exploited by fax machines and later adopted in image processing. Indeed, since in black&white images only two symbols are present, the chances to have long runs for a value, and hence the possibility of high compression rates, are significantly increased. The algorithm is quite easy: each run, instead of being represented explicitly, is translated by the encoding algorithm in a pair $(l, v)$ where $l$ is the length of the run and $v$ is the value of the run elements. Of course, the longer the runs in the sequence to be compressed, the better the compression ratio.

*Example 2.1* (RLE compression of a sample sequence) The sequence

**xyzzxyyzxywxy**

would be represented by RLE as

(1, x) (1, y) (2, z) (1, x) (2, y) (1, z) (1, x) (1, y) (1, w) (1, x) (1, y).

Assuming that an integer takes a byte just as a character, here the 'compressed' version of the original 13-byte sequence takes 22 bytes. This is due to the large cardinality of the alphabet with respect to the length of the sequence, and to the high variability of the symbols therein.

**Huffman Encoding** The compression strategy devised by *Huffman* [20] relies on a greedy algorithm to generate a dictionary, i.e., a symbol–code table, that allows obtaining a nearly *optimal* data compression from an information-theoretic viewpoint.[1] First of all, the alphabet is identified as the set of symbols appearing in the string to be encoded. Then, the frequency in such a string of each symbol of the alphabet is determined. Subsequently, a binary tree, whose nodes are associated to frequencies, is built as follows:

```
1. Set the symbols of the alphabet, with the corresponding frequencies, as leaf nodes
2. while the structure is not a tree (i.e., there is no single root yet)
   (a) Among the nodes that do not still have a parent, select two whose associated frequency is minimum
   (b) Insert a new node that becomes the parent of the two selected nodes and gets as associated frequency the sum of the frequencies of such nodes
3. In the resulting tree, mark each left branch as '1' and each right branch as '0'
```

Lastly, the binary code of each symbol is obtained as the sequence of branch labels in the path from the root to the corresponding leaf. Now, each symbol in the source sequence is replaced by the corresponding binary code. A consequence of this frequency-based procedure is that shorter codes are assigned to frequent symbols, and longer ones to rare symbols.

Decompression is performed using the symbol–code table, scanning the encoded binary string and emitting a symbol as soon as its code is recognized:

```
1. while the encoded sequence has not finished
   (a) Set the current code to be empty
   (b) while the current code does not correspond to any code in the table
       (i) Add the next binary digit in the encoded sequence to the current code
   (c) Output to the decompressed sequence the symbol corresponding to the code just recognized
```

Note that the code generation procedure is such that no code in the table is a prefix of another code, which ensures that as soon as a code is recognized left-to-right it is the correct one: neither a longer code can be recognized, nor recognition can stop before recognizing a code or having recognized a wrong code. Hence, the compressed sequence can be transmitted as a single and continuous binary sequence, without the need for explicit separators among codes.

*Example 2.2* (Huffman compression of a sample sequence) Assume the message to be encoded is **xyzzxyyzxywxy**. A node is created for each different symbol in the sequence: $x$, $y$, $z$ and $w$. The symbol frequencies are $\frac{4}{13}$ for $x$, $\frac{5}{13}$ for $y$, $\frac{3}{13}$ for $z$ and $\frac{1}{13}$ for $w$. The two nodes with minimum frequencies ($z$ and $w$) are connected to a new parent node (let us call it $I$), that gets the sum of their frequencies ($\frac{4}{13}$). Now the two nodes without a parent and with minimum frequencies are $x$ and $I$, that are connected to a new parent *II*, whose overall frequency is $\frac{8}{13}$. Lastly, the only two nodes without a parent, *II* and $y$, are connected to a new parent $R$ that completes the tree and becomes its root. From the resulting tree (depicted in Fig. 2.1) the

[1]As proved by Shannon [23], the number of bits required to specify sequences of length $N$, for large $N$, is equal to $N \cdot H$ (where $H$ is the source entropy).

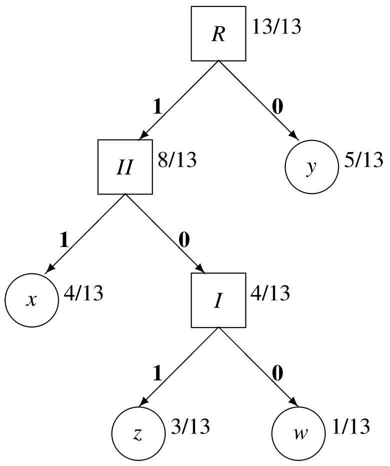


**Fig. 2.1** Binary tree that defines the Huffman codes for the symbols in string **xyzzxyyzxywxy**

following table is obtained:

| Symbol | x | y | z | w |
|---|---|---|---|---|
| Code | 11 | 0 | 101 | 100 |

that is used to encode the initial string:

| x | y | z | z | x | y | y | z | x | y | w | x | y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 0 | 101 | 101 | 11 | 0 | 0 | 101 | 11 | 0 | 100 | 11 | 0 |

Given the compressed string, and scanning it from left to right: 1 is not a code, 11 is recognized as the code of $x$; 0 is immediately recognized as the code of $y$; 1 is not a code, nor is 10, while 101 is the code of $z$; and so on.

**LZ77 and LZ78 (Lempel–Ziv)** A series of compression techniques was developed by Lempel and Ziv, and hence denoted by the acronym LZ followed by the year of release: *LZ77* [28] dates back to 1977, while *LZ78* [29] dates to 1978. Both are dictionary-based encoders, and represent a basis for several other techniques (among which LZW, to be presented next). The former looks backward to find duplicated data, and must start processing the input from its beginning. The latter scans forward the buffer, and allows random access to the input but requires the entire dictionary to be available. They have been proved to be equivalent to each other when the entire data is decompressed. In the following, we will focus on the former.

LZ77 adopts a representation based on the following elements:

**Literal** a single character in the source stream;

**Length–Distance** pair to be interpreted as "the next *length* characters are equal to the sequence of characters that precedes them of *distance* positions in the source stream".

The repetitions are searched in a limited buffer consisting of the last $N$ kB (where $N \in \{2, 4, 32\}$), whence it is called a *sliding window* technique. Implementations may adopt different strategies to distinguish *length–distance* pairs from *literals* and to output the encoded data. The original version exploits triples

$$(\mathit{length}, \mathit{distance}, \mathit{literal}),$$

where

- *length–distance* refers to the longest match found in the buffer, and
- *literal* is the character following the match

($\mathit{length} = 0$ if two consecutive characters can only be encoded as literals).

**LZW (Lempel–Ziv–Welch)** The information lossless compression technique by *Lempel–Ziv–Welch* (whence the acronym *LZW*) [26] leverages the redundancy of characters in a message and increases compression rate with the decreasing of the number of symbols used (e.g., colors in raster images). It uses fixed-length codes to encode symbol sequences of variable length that appear frequently, and inserts them in a symbol–code conversion dictionary $D$ (that is not stored, since it can be reconstructed during the decoding phase). If implemented as an array, the index of elements in $D$ corresponds to their code. In the following, $T$ will denote the source stream, composed on an alphabet $A$ made up of $n$ symbols, and $C$ the compressed one.

To start compression, one needs to know how many bits are available to represent each code ($m$ bits denote $2^m$ available codes, ranging in $[0, 2^m - 1]$). The symbols of $A$ are initially inserted in the first $n$ positions (from 0 to $n - 1$) of $D$. The compression algorithm continues exploiting an output string $C$ and two variables (a prefix string $P$, initially empty, and a symbol $s$):

```
1. $P$ is initially empty
2. **while** $T$ is not finished
   (a) $s \leftarrow$ next symbol in $T$
   (b) **if** string $Ps$ is already present in $D$
       (i) $P$ becomes $Ps$
   (c) **else**
       (i) Insert string $Ps$ in the first available position of $D$
       (ii) Append to $C$ the code of $P$ found in $D$
       (iii) $P$ becomes $s$
3. Append to $C$ the code of $P$ found in $D$
```

After compression, $D$ can be forgotten, since it is not needed for decompression. Due to the many accesses to $D$, it is usefully implemented using a hashing technique, based on the string as a key. Of course, there is the risk that all available codes in the table are exploited, and special steps have to be taken to handle the case in which an additional one is needed.

The only information needed during the decoding phase is the alphabet $A$, whose symbols are initially placed in the first $n$ positions of dictionary $D$. Encoding is such that all codes in the coded stream can be translated into a string. Decoding continues as follows, using two temporary variables $c$ and $o$ for the current and old codes, respectively:

```
1. $c \leftarrow$ first code from $C$
2. Append to $T$ the string for $c$ found in $D$
3. **while** $C$ is not finished
   (a) $o \leftarrow c$
   (b) $c \leftarrow$ next code in $C$
   (c) **if** $c$ is already present in $D$
        (i) Append to $T$ the string found in $D$ for $c$
       (ii) $P \leftarrow$ translation for $o$
      (iii) $s \leftarrow$ first symbol of translation for $c$
       (iv) Add $Ps$ to $D$
   (d) **else**
        (i) $P \leftarrow$ translation for $o$
       (ii) $s \leftarrow$ first character of $P$
      (iii) Append $Ps$ to $T$ and add it to $D$
```

Note that both coding and decoding generate the same character–code table. More space can be saved as follows. Notice that all insertions in $D$ are in the form $Ps$, where $P$ is already in $D$. Thus, instead of explicitly reporting $Ps$ in the new entry, it is possible to insert $(c, s)$, where $c$ is the code for $P$ (that usually has a shorter bit representation than $P$ itself).

*Example 2.3* (LZW compression of a sample sequence) Consider the following string to be compressed:

x y z z x y y z x y w x y

$\Downarrow$

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| String | w | x | y | z | xy | yz | zz | zx | xyy | yzx | xyw | wx |
| Compact | w | x | y | z | 1+y | 2+z | 3+z | 3+x | 4+y | 5+x | 4+w | 0+x |
| Code$_{10}$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Code$_2$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 |

$\Uparrow$

1 2 3 3 4 5 4 0 4

0001 0010 0011 0011 0100 0101 0100 0000 0100

In this case, the string to be compressed is 26 bits long (13 symbols $\times$ 2 bits needed to encode each symbol in an alphabet of 4) and the compressed string is 36 bits long (9 codes $\times$ 4 bits needed to represent each). This happens because the message is too short and varied, which generates many codes but does not provide the opportunity to reuse them.

For very long data streams and with a large alphabet (e.g., 256 symbols) the compression estimate is about 40% if applied to text.

**DEFLATE** *DEFLATE* [16] is a compression algorithm that joins LZ77 and Huffman. Compressed streams consist of sequences of blocks, each preceded by a 3-bit header, where the first bit is a flag saying whether that is the last block (**0** = false, **1** = true) and the other two express the encoding method for the block (the most used is **10**, corresponding to *dynamic Huffman encoding*, while **11** is reserved).

It consists of two steps:

1. Apply matching to find duplicate strings and replace them with backward references, obtaining the pairs
$$(\mathit{length}, \mathit{distance})$$
where $\mathit{length} \in [3, 258]$ and $\mathit{distance} \in [1, 32768]$.
2. Replace the original symbols with new symbols having length inversely proportional to their frequency of use, according to Huffman encoding. Specifically, literal and length alphabets are merged into a single alphabet 0–285, and encoded in a tree that provides room for 288 symbols, as follows:
    - 0–255 represent the possible literals;
    - 256 denotes the end of the block;
    - 257–285, combined with extra bits, express a match length of 3 to 258 bytes, where codes in each group from $256 + 4i + 1$ to $256 + 5i$, for $i = 0, \ldots, 5$, denote $2^i$ lengths using $i$ extra bits, and code 285 denotes length 258 (an extensional representation is provided in Table 2.1);
    - 286–287 are reserved (not used).

Distance values between 1 and 32768 are encoded using 32 symbols as follows:

- Symbols 0 to 3 directly represent distances 1 to 4.
- Symbols $(2 \cdot i + 4)$ and $(2 \cdot i + 5)$, for $i = 0, \ldots, 12$, denote distances from $(2^{i+2} + 1)$ to $(2^{i+3})$ as follows:

  $(2 \cdot i + 4)$ denotes the base value $2^{i+2} + 1$ (and distances up to $3 \cdot 2^{i+1}$),
  $(2 \cdot i + 5)$ denotes the base value $3 \cdot 2^{i+1} + 1$ (and distances up to $2^{i+3}$),

  and the actual distance is obtained by adding to the base value a displacement $d \in [0, 2^{i+1} - 1]$, expressed by $(i + 1)$ extra bits.
- Symbols 30 and 31 are reserved (not used).

These symbols are arranged in a distance tree, as well.

## 2.2 Non-structured Formats

By the attribute “non-structured”, in this section, we will refer to documents that do not explicitly organize the information they contain into structures that are significant from a geometrical and/or conceptual point of view.

**Table 2.1** Correspondence between codes and lengths in the DEFLATE compression technique

| Code | Extra bits | Length | Lengths per code |
|---|---|---|---|
| 257 | | 3 | |
| ⋮ | 0 | ⋮ | 1 |
| 264 | | 10 | |
| 265 | | 11–12 | |
| ⋮ | 1 | ⋮ | 2 |
| 268 | | 17–18 | |
| 269 | | 19–22 | |
| ⋮ | 2 | ⋮ | 4 |
| 272 | | 31–34 | |
| 273 | | 35–42 | |
| ⋮ | 3 | ⋮ | 8 |
| 276 | | 59–66 | |
| 277 | | 67–82 | |
| ⋮ | 4 | ⋮ | 16 |
| 280 | | 115–130 | |
| 281 | | 131–162 | |
| ⋮ | 5 | ⋮ | 32 |
| 284 | | 227–257 | |
| 285 | 0 | 258 | 1 |

### *2.2.1 Plain Text*

Plain text, usually denoted by the *TXT* extension, refers to a stream of characters represented according to some standard coding agreement. In particular, each character is represented as an integer (typically in binary or hexadecimal format[2]), to which the users assign a specific meaning (that can be identified with the symbol denoting that character). Usually, the characters to be represented are the letters of a particular writing system or *script* (the alphabet of a language, possibly extended with diacritic marks) plus punctuation marks and other useful symbols. In this format, the only semblance of a structure is given by the line separator, roughly indicating a kind of 'break' in the character flow.

This format is mainly intended for information exchange, not typography. Thus, each character represents the 'essence' of a symbol, abstracted from the many par-

[2]In the rest of this section, both notations will be used interchangeably, as needed.

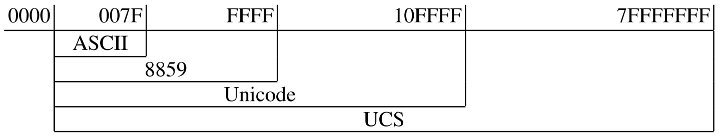


**Fig. 2.2** Ranges of hexadecimal codes exploited by various standards for character encoding

ticular shapes by which that symbol can be graphically denoted. Rather, it should ensure complete accessibility, independent of the particular platform (software, operating system and hardware architecture) in use, as long as there is an agreement on the coding standard to be used. As a consequence, there is a need to define shared conventions on which basis a given value is uniformly interpreted by all users as the same character. Figure 2.2 depicts the ranges of hexadecimal codes exploited by various standards that will be discussed in the following.

**ASCII** Historically, the coding agreement that has established in the computer community, becoming a *de facto* standard, is the *ASCII* (American Standard Code for Information Interchange).[3] It is based on the exploitation of 1 byte per character, leading to a total of 256 possible configurations (i.e., symbols that can be represented). In its original version, devised to support American texts (US-ASCII), it actually exploited only 7 bits, reserving the last bit for error checking purposes in data transmission, or leaving it unused. This allowed $2^7 = 128$ different configurations, to represent 95 printable characters plus some control codes. However, the worldwide spread of computer systems has subsequently called for an extension of this range to include further useful symbols as well. In particular, several kinds of codes were developed with the advent of the Internet, and for this reason their definition and assignment are ruled by the IANA (Internet Assigned Numbers Authority).

**ISO Latin** Many languages other than English often rely on scripts containing characters (accented letters, quotation marks, etc.) that are not provided for in ASCII. A first, straightforward solution was found in exploiting for coding purposes also the last bit of a character byte, which allowed 128 additional configurations/characters available. However, even the 256 total configurations obtained in this way were soon recognized to be insufficient to cover all Western writing systems. To accommodate this, several alternative exploitations of the new configurations coming from the use of the eighth bit were developed, leading to a family of standards known as ISO/IEC 8859. Specifically, 16 different extensions were defined, and labeled as the ISO/IEC 8859-$n$ standards (with $n = 1, \dots, 16$ denoting

[3]Another coding standard in use during the ancient times of computing machinery, abandoned later on, was the *EBCDIC* (Extended Binary Coded Decimal Interchange Code), which started from the *BCD* (Binary Coded Decimal) binary representation of decimal digits, from $0000_2 = 0$ to $1001_2 = 9$, used by early computers for performing arithmetical operations, and extended it by putting before four additional bits. The decimal digits are characterized by an initial $1111_2$ sequence, while the other configurations available allow defining various kinds of characters (alphabetic ones, punctuation marks, etc.).

the specific extension). Among them, the most used are the sets of Latin characters (*ISO Latin*), and specifically the ISO-8859-1 code. Obviously, the alternative sets of codes are incompatible with each other: the same (extended) configuration corresponds to different characters in different standards of the family. It should be noted that only printable characters are specified by such codes, leaving the remaining configurations unspecified and free for use as control characters.

**UNICODE** Although the one-byte-per-character setting is desirable for many reasons (efficiency in terms of memory, easy mapping, computer architecture compliance, etc.), it is not sufficient to effectively cover the whole set of languages and writing systems in the world. In order to collect in a unified set the different codes in the ISO/IEC 8859 family, and to allow the inclusion of still more scripts, avoiding incompatibility problems when switching from one to another, the *Unicode* [24] and *UCS* (Universal Character Set, also known as ISO/IEC 10646) [6] standards were developed, and later converged towards joint development. Here, each character is given a unique name and is represented as an abstract integer (called *code point*), usually referenced as 'U+' followed by its hexadecimal value.

Unicode defines a codespace of 1,114,112 potential code points in the range $000000_{16}$–$10FFFF_{16}$, logically divided into 17 *planes* ($00_{16}$–$10_{16}$), made up of $2^{16} = 65{,}536$ ($0000_{16}$–$FFFF_{16}$) code points each:

**Plane 0** (0000–FFFF) *Basic Multilingual Plane* (*BMP*)
**Plane 1** (10000–1FFFF) *Supplementary Multilingual Plane* (*SMP*)
**Plane 2** (20000–2FFFF) *Supplementary Ideographic Plane* (*SIP*)
**Planes 3 to 13** (30000–DFFFF) unassigned
**Plane 14** (E0000–EFFFF) *Supplementary Special-purpose Plane* (*SSP*)
**Planes 15** (F0000–FFFFF) **and 16** (100000–10FFFF) *Private Use Area* (*PUA*) – reserved

Only 100,713 code points are currently exploited (as of Version 5.1, April 2008, covering 75 scripts). Ranges of code points have been reserved for every current and ancient writing systems, already known or still to be discovered. The BMP, in particular, is devoted to support the unification of prior character sets as well as characters for writing systems in current use, and contains most of the character assignments so far. Code points in the BMP are denoted using just four digits ('U+nnnn'); for code points outside the BMP, five or six digits are used, as required. ISO/IEC 8859 characters all belong to the BMP, and hence are mapped on their Unicode/UCS counterparts via a U+nnnn notation.

*UTF* Code points are implemented using a technique called *UTF*. Actually, different kinds of UTF exist, some using configurations of bits of fixed length and others using variable-length encoding:

**UTF-8** sequences of one to six 8-bit code values
**UTF-16** sequences of one or two 16-bit code values
**UTF-32** or **UTF-4** fixed-length 4-byte code values
**UTF-2** fixed-length 2-byte code values (now replaced by UTF-8)

UTF-8 is the only platform-independent UTF. Conversely, UTF-16 and UTF-32 are platform-dependent concerning byte ordering, and are also incompatible with ASCII, which means that Unicode-aware programs are needed to handle them, even in cases when the file contains only ASCII characters.[4] For these reasons, 8-bit encodings (ASCII, ISO-8859-1, or UTF-8) are usually exploited for representing text even on platforms that are natively based on other formats.

Table 2.2 shows the 4-byte fixed-length and UTF-8 variable length representations of remarkable code points, while Table 2.3 reports a comparison of UTF-8 and UTF-16 for interesting ranges of code points.

UTF-8 adopts a segment-based management: a subset of frequent characters is represented using fewer bits, while special bit sequences in shorter configurations are used to indicate that the character representation takes more bits. Although this adds redundancy to the coded text, advantages outperform disadvantages (and, in any case, compression is not an aim of Unicode). For instance, such a variable-length solution allows saving memory in all the many cases in which the basic Latin script, without accented letters and typographic punctuation, is sufficient for the user's purposes (e.g., programming). It exploits up to four bytes to encode Unicode values, ranging 0–10FFFF; for the ISO/IEC 10646 standard, it can exploit even five or six bytes, to allow encoding values up to U+7FFFFFFF. The rules for obtaining UTF-8 codes are as follows:

1. *Single-byte codes start with 0 as the most significant bit*. This leaves 7 bits available for representing the actual character.
2. *In multi-byte codes, the number of consecutive 1's in the most significant bits of the first byte, before a 0 bit is met, denotes the number of bytes that make-up the multi-byte code; subsequent bytes of the multi-byte code start with 10 in the two most significant bits*. For a multi-byte code made up of $n$ bytes, this leaves $(7 - n) + 6 \cdot (n - 1)$ bits available for representing the actual character.

This ensures that no sequence of bytes corresponding to a character is ever contained in a longer sequence representing another character, and allows performing string matching in a text file using a byte-wise comparison (which is a significant help and allows for less complex algorithms). Additionally, if one or more bytes are lost because of transmission errors, decoding can still be synchronized again on the next character, this way limiting data loss.

UTF-8 is compliant to ISO/IEC 8859-1 and fully backward compatible to ASCII (and, additionally, non-ASCII UTF-8 characters are just ignored by legacy ASCII-based programs). Indeed, due to rule 1, UTF-8 represents values 0–127 ($00_{16}$–$7F_{16}$) using a single byte with the leftmost bit at 0, which is exactly the same representation as in ASCII (and hence an ASCII file and its UTF-8 counterpart are identical). As to ISO/IEC 8859-1, since it fully exploits 8 bits, it goes from 0 to 255 ($00_{16}$–$FF_{16}$). Its lower 128 characters are just as ASCII, and fall, as said, in the 1-byte

[4]As a trivial example of how tricky UTF-16 can be: usual C string handling cannot be applied because it would consider as string terminators the many 00000000 byte configurations in UTF-16 codes.

**Table 2.2** Comparison between a fixed-length (UTF-4) and a variable-length (UTF-8) encoding for noteworthy values of UCS

| Encoding | Value representation | | | | | | Standards boundaries | UTF-8 length |
|---|---|---|---|---|---|---|---|---|
| UCS | 0 0 | 0 0 | 0 0 | 0 1 | | | | 1-byte |
| UTF-4 | 00000000 | 00000000 | 00000000 | 00000001 | | | | |
| UTF-8 | 00000001 | | | | | | | |
| UCS | 0 0 | 0 0 | 0 0 | 7 F | | | | |
| UTF-4 | 00000000 | 00000000 | 00000000 | 01111111 | | | | |
| UTF-8 | 01111111 | | | | | | ASCII | |
| UCS | 0 0 | 0 0 | 0 0 | 8 0 | | | | 2-byte |
| UTF-4 | 00000000 | 00000000 | 00000000 | 10000000 | | | | |
| UTF-8 | 11000010 | 10000000 | | | | | | |
| UCS | 0 0 | 0 0 | 0 7 | F F | | | | |
| UTF-4 | 00000000 | 00000000 | 00000111 | 11111111 | | | | |
| UTF-8 | 11011111 | 10111111 | | | | | | |
| UCS | 0 0 | 0 0 | 0 8 | 0 0 | | | | 3-byte |
| UTF-4 | 00000000 | 00000000 | 00001000 | 00000000 | | | | |
| UTF-8 | 11100000 | 10100000 | 10000000 | | | | | |
| UCS | 0 0 | 0 0 | F F | F F | | | | |
| UTF-4 | 00000000 | 00000000 | 11111111 | 11111111 | | | | |
| UTF-8 | 11101111 | 10111111 | 10111111 | | | | BMP | |
| UCS | 0 0 | 0 1 | 0 0 | 0 0 | | | | 4-byte |
| UTF-4 | 00000000 | 00000001 | 00000000 | 00000000 | | | | |
| UTF-8 | 11110000 | 10010000 | 10000000 | 10000000 | | | | |
| UCS | 0 0 | 1 0 | F F | F F | | | Unicode | |
| UCS | 0 0 | 1 F | F F | F F | | | | |
| UTF-4 | 00000000 | 00011111 | 11111111 | 11111111 | | | | |
| UTF-8 | 11110111 | 10111111 | 10111111 | 10111111 | | | | |
| UCS | 0 0 | 2 0 | 0 0 | 0 0 | | | | 5-byte |
| UTF-4 | 00000000 | 00100000 | 00000000 | 00000000 | | | | |
| UTF-8 | 11111000 | 10001000 | 10000000 | 10000000 | 10000000 | | | |
| UCS | 0 3 | F F | F F | F F | | | | |
| UTF-4 | 00000011 | 11111111 | 11111111 | 11111111 | | | | |
| UTF-8 | 11111011 | 10111111 | 10111111 | 10111111 | 10111111 | | | |
| UCS | 0 4 | 0 0 | 0 0 | 0 0 | | | | 6-byte |
| UTF-4 | 00000100 | 00000000 | 00000000 | 00000000 | | | | |
| UTF-8 | 11111100 | 10000100 | 10000000 | 10000000 | 10000000 | 10000000 | | |
| UCS | 7 F | F F | F F | F F | | | | |
| UTF-4 | 01111111 | 11111111 | 11111111 | 11111111 | | | | |
| UTF-8 | 11111101 | 10111111 | 10111111 | 10111111 | 10111111 | 10111111 | UCS | |

**Table 2.3** UTF-8 and UTF-16 encodings for noteworthy ranges of code points. 0/1s denote fixed bit values, while x's denote bit positions available for the actual value representation. For short, in UTF-8, the notation $(B) \times N$ stands for $N$ bytes of the kind $B$

| Range (hex) | UTF-16 | UTF-8 |
|---|---|---|
| 000000–00007F | 00000000 0xxxxxxx | 0xxxxxxx |
| 000080–0007FF | 00000xxx xxxxxxxx | 110xxxxx + 10xxxxxx |
| 000800–00FFFF | xxxxxxxx xxxxxxxx | 1110xxxx + (10xxxxxx) × 2 |
| 010000–10FFFF | 110110xx xxxxxxxx 110111xx xxxxxxxx | 11110xxx + (10xxxxxx) × 3 |
| 00110000–001FFFFF | | 11110xxx + (10xxxxxx) × 3 |
| 00200000–003FFFFF | | 111110xx + (10xxxxxx) × 4 |
| 04000000–7FFFFFFF | | 1111110x + (10xxxxxx) × 5 |

representation, while its upper 128 characters (those extending ASCII, going from 128 to 255, i.e., $80_{16}$–$FF_{16}$) fall in the 2-byte representation ($080_{16}$–$7FF_{16}$, or 128–2047), thus they will be represented as 110000xx 10xxxxxx.

The rules for obtaining UTF-16 codes are as follows:

- For code points in the BMP ($0000_{16}$–$FFFF_{16}$), 16 bits (i.e., 2 bytes) are sufficient.
- To represent code-points over the BMP ($010000_{16}$–$10FFFF_{16}$), *surrogate pairs* (pairs of 16-bit words, each called a *surrogate*, to be considered as a single entity) are exploited. The first and second surrogate are denoted, respectively, by the bit sequences 110110 and 110111 in the first six positions, which avoids ambiguities between them and leaves $16 - 6 = 10$ available bits each for the actual value representation. First $10000_{16}$ is subtracted from the code point value, in order to obtain a value ranging in $00000_{16}$–$FFFFF_{16}$, that can be represented by 20 bits. Such resulting 20 bits are split into two subsequences made up of 10 bits each, assigned to the trailing 10 bits of the first and second surrogate, respectively.

This means that the first surrogate will take on values in the range $D800_{16}$–$DBFF_{16}$, and the second surrogate will take on values in the range $DC00_{16}$–$DFFF_{16}$.

*Example 2.4* (UTF-16 representation of a sample code point) The character at code point U+10FF3A is transformed by the subtraction into

$$\mathrm{FFFA}_{16} = 11111111111100111010_2$$

The first ten bits (1111111111) are placed in the first surrogate, that becomes $1101101111111111_2 = \mathrm{DBFF}_{16}$, and the second ten bits 1100111010 are placed in the second surrogate, that becomes $1101111100111010_2 = \mathrm{DF3A}_{16}$; overall, the original code point is represented as the sequence $(\mathrm{DBFF\ DF3A})_{16}$.

Of course, not to confuse a surrogate with a single 16-bit character, Unicode (and thus ISO/IEC 10646) are bound not to assign characters to any of the code points in the U+D800–U+DFFF range.

## *2.2.2 Images*

Image components play a very important role in many kinds of documents because they leverage on the human perception capabilities to compactly and immediately represent large amounts of information that even long textual descriptions could not satisfactorily express. While text is a type of linear and discrete information, visual information, however, is inherently multi-dimensional (2D still images, that will be the area of interest of this book, are characterized along three dimensions: two refer to space and one to color) and continuous. Thus, while the transposition of the former into a computer representation is straightforward, being both characterized by the same features, the translation of the latter posed severe problems, and raised the need for formats that could find a good trade-off among the amount of information to be preserved, the memory space demand and the manipulation processes to be supported. To make the whole thing even more difficult, all these requirements may have different relevance depending on the aims for which the image is produced.

### Color Spaces

A *color space* is a combination of a *color model*, i.e., a mathematical abstract model that allows representing colors in numeric form, and of a suitable mapping function of this model onto perceived colors. Such a model determines the colors that can be represented, as a combination of a set of basic parameters called *channels*.[5] Each channel takes on values in a range, that in the following we will assume to be [0, 1] unless otherwise stated. Of course, practical storing and transmission of these values in computer systems may suggest more comfortable byte-based representations, usually as discrete (integer) values in the $[0, N-1]$ interval (where $N = 2^k$ with $k$ the number of bits reserved for the representation of a channel). In such a case, a value $x \in [0, 1]$ can be suitably scaled to an $n \in [0, N-1]$ by the following formula:

$$n = \min\bigl(\mathrm{round}(N \cdot x), N-1\bigr)$$

and back from $n$ to $x$ with a trivial proportion. A typical value adopted for $N$, that is considered a good trade-off between space requirements, ease of manipulation and quality of the representation, is $2^8 = 256$, which yields a channel range $[0, 255]$. Indeed, $k = 8$ bits, or a byte, per channel ensures straightforward compliance to traditional computer science memory organization and measurement. It is also usual that a color is compactly identified as a single integer obtained by juxtaposing the corresponding channel values as consecutive bits, and hence adding or subtracting given amounts to such a compound value provides results that cannot be foreseen from a human perception viewpoint.

[5] An indication that one of the main interests towards images is their transmission.

**Table 2.4** The main colors as represented in coordinates of the different color spaces

| Color | R | G | B | C | M | Y | Y | U | V | H | S | V | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0.5 | 0.5 | – | 0 | 0 | 0 |
| R | 1 | 0 | 0 | 0 | 1 | 1 | 0.3 | 0.33 | 1 | 0 | 1 | 1 | 0.5 |
| G | 0 | 1 | 0 | 1 | 0 | 1 | 0.58 | 0.17 | 0.08 | 1/3 | 1 | 1 | 0.5 |
| B | 0 | 0 | 1 | 1 | 1 | 0 | 0.11 | 1 | 0.42 | 2/3 | 1 | 1 | 0.5 |
| C | 0 | 1 | 1 | 1 | 0 | 0 | 0.7 | 0.67 | 0 | 1/2 | 1 | 1 | 0.5 |
| M | 1 | 0 | 1 | 0 | 1 | 0 | 0.41 | 0.83 | 0.92 | 5/6 | 1 | 1 | 0.5 |
| Y | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0.58 | 1/6 | 1 | 1 | 0.5 |
| W | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0.5 | 0.5 | – | 0 | 1 | 1 |
| Gray | $x$ | $x$ | $x$ | $x$ | $x$ | $x$ | $x$ | 0.5 | 0.5 | – | 0 | $x$ | $x$ |

**RGB** The most widespread color space is *RGB* (or *three-color*), acronym of the colors on which it is based (Red, Green and Blue). It is an additive color system, meaning that any color is obtained by adding to black a mix of lights carrying different amounts of the base colors. The intensity of each color represents its shade and brightness. The three primary colors represent the dimensions along which any visible color is defined, and hence the whole space consists of a cube of unitary side. Each color is identified as a triple $(R, G, B)$ that locates a point in such a space (and *vice-versa*). In particular, the triples that represent the base colors correspond to the eight corners of the cube (see Table 2.4), while gray levels (corresponding to triples in which $R = G = B$) are in the main diagonal of such a cube, going from Black (usually denoted by $K$) to White ($W$). Note that this color space is *non-linear*, in the sense that its dimensions have no direct mapping to the typical dimensions underlying human perception. As a consequence, it is not foreseeable what will be the result of acting in a given way on each channel value because changing a single component changes at once many different parameters such as color tone, saturation and brightness.

The success of RGB is due to the fact that it underlies most analog and digital representation and transmission technologies, and is supported by many devices, from old CRT (Cathode Ray Tube) monitors to current LCD (Liquid Crystal Display) and LED (Light-Emitting Diode) screens, from scanners to cameras.

**YUV/YC$_b$C$_r$** The *YUV* color space has its main feature in the $Y$ component that denotes the *luminance* channel (i.e., the sum of primary colors), and corresponds to the overall intensity of light carried by a color. In practical terms, the luminance information alone yields grayscale colors, and hence is sufficient to display images on black&white screens. The values of the colors can be derived as the difference between the chrominance channels $U$ (difference from blue) and $V$ (difference from red). Green is obtained by subtracting from the luminance signal the signals transmitted for red and blue. $YC_bC_r$ is the international standardization of YUV.

YUV was originally developed to switch from a color RGB signal to a black and white one because the same signal could be straightforwardly exploited both by color receivers (using all three channels) and by black&white receivers (using only the $Y$ channel).

**CMY(K)** *CMY* is a color model named after the acronym of the basic colors that it exploits: Cyan, Magenta, and Yellow. It is a subtractive system, because starting from white (such as a sheet to be printed), light is progressively subtracted along the basic components to determine each color, up to the total subtraction that yields black.

Hence, in theory a separate black component is not necessary in CMY, because black can be obtained as the result of mixing the maximum level of the three basic colors. However, in practice the result obtained in this way is not exactly perceived as black by the human eye. For this reason, often an additional $K$ channel (where $K$ stands for blacK, also called *Key color*) is specifically exploited for the gray-level component of the colors, this way obtaining the *CMYK* (or *four-color*) color space.

CMY(K) is suitable for use in typography, and indeed its basic colors correspond to the colors of the inks used in color printers.

**HSV/HSB and HLS** The *HSV* (acronym of Hue Saturation Value) color space (sometimes referred to as *HSB*, where $B$ stands for Brightness) is a three-dimensional space where the vertical axis represents the brightness, the distance from such an axis denotes saturation, and the angle from the horizontal is the hue. This yields a cylinder having black on the whole lower base (but conventionally placed in its center, which is the origin of the axes), and a regular hexagon inscribed in the upper base circle, whose corners correspond to the $R$, $Y$, $G$, $C$, $B$, $M$ basic colors. White stands in the center of the upper base, and gray levels lie along the vertical. With black being in the whole base, the space is again non-linear.

*HLS* (short for Hue Luminance Saturation) is very similar to HSV: the vertical axis is still brightness (here called Luminance), the angle is still Hue and the distance (radius) is again Saturation, but the latter are defined here in a different way. In particular, black is again on the whole lower base (although conventionally placed in its center, which is the origin of the axes), but white now takes the whole upper base (although conventionally placed in its center), and the regular hexagon whose corners are the basic colors cuts the cylinder in the middle of its height. It is again non-linear, because black takes the whole lower base, and white takes the whole upper base.

**Comparison among Color Spaces** Table 2.4 compares the representations of main colors in different spaces. The color distribution is uniform in the RGB cube, while in HSV it is more dense towards the upper base and goes becoming more rare towards the lower base (for which reason it is often represented as an upside-down pyramid), and is more dense in the middle of the HLS cylinder, becoming more rare towards both its bases (hence the representation as a double pyramid, linked by their bases).

## Raster Graphics

The term *raster* refers to an image representation made up of dots, called *pixels* (a contraction of the words ‘picture elements’), stored in a matrix called *bitmap*. It is the result of a spatial discretization of the image, as if a regular grid, called *sampling grid*, were superimposed to an analog image: then, the portion of image that falls in each of the grid cells is mapped onto a pixel that approximately represents it as just one value. If also the values a pixel can take on are discrete, the original image is completely discretized. The number of possible values a pixel can take on is called the *density* (or *depth*) of the image. Each value represents a color, and can range from a binary distinction between black and white, to a scale of gray levels, to a given number of colors. The *resolution* corresponds to the number of (horizontal/vertical) grid meshes (pixels) that fit in a given linear distance (usually an *inch*, i.e., 2.54 cm); clearly, narrower meshes provide a closer approximation of the original image. In addition to using a given color space, some computer formats for image representation provide an additional (optional) *alpha channel* that expresses the degree of transparency/opacity of a pixel with respect to the background on which the image is to be displayed. Such a degree is denoted by means of a numeric value (whose range depends on the format in use, but is often the same as the other color space channels).

The amount of memory space needed to store the information concerning an image thus depends on both resolution and density, as reported in Table 2.5 along with other representation-related parameters. Indeed, a single bit per pixel is enough to distinguish among black and white; 4 bits (16 levels) can be used for low-quality gray-level or very-low-quality color images; 8 bits (and hence one byte) per pixel are fine for good-quality gray-level (using a single luminance channel) or for low-quality color images; three 8-bit channels (e.g., RGB), for a total of 3 bytes, allow expressing more than 16 million colors that are considered a sufficient approximation of what can be perceived by the eye (whence the term *true color* for this density).

There are many formats to save an image on file; the choice depends on several factors, among which the allowed density, the use of compression or not, and, if

**Table 2.5** Indicative space needed to store a non-compressed raster image having width $W$ and height $H$ for different density values, plus other parameters

| Density | 2 | 16 | 256 | 16 mil. |
|---|---|---|---|---|
| Usual exploitation | (black&white) | gray-level or color | | true color |
| Bits/pixel | 1 | 4 | 8 | 24 |
| Image size (in bytes) | $W \cdot H/8$ | $W \cdot H/2$ | $W \cdot H$ | $W \cdot H \cdot 3$ |
| Usual number of channels | 1 | 1 | 1 | 3 |

**Table 2.6** Bitmap file header

| Byte | Name | stdval | Description |
|---|---|---|---|
| 1–2 | bfType | 19778 | The characters ‘BM’ to indicate it is a BMP file |
| 3–6 | bfSize | | File size in bytes |
| 7–8 | bfReserved1 | 0 | Always 0 |
| 9–10 | bfReserved2 | 0 | Always 0 |
| 11–14 | bfOffBits | 1078 | Offset from the file start to the first data byte (the beginning of the pixel map) |

used, its being information lossy or lossless,[6] the algorithm and the compression ratio.

**BMP (BitMaP)** Introduced in 1990 by Microsoft for Windows 3.0, not patented, the BMP format soon gained wide acceptance by graphic programs. It allows depths of 1, 4, 8, 16, 24 or 32 bits/pixel. Although provided for the cases of 16 and 256 colors, compression is usually not exploited due to the inefficiency of the RLE (lossless) algorithm. Thus, the most used (uncompressed) version has a representation on disk similar to its RAM counterpart. Even if this improves read and write speed, because the processor is not in charge of thoroughly processing the data contained in the file, the drawback is that the space needed to represent an image is quite large, which prevents the use of this format on the Web. Another shortcoming is the fact that, in version 3 (the most commonly used), differently from versions 4 and 5, the alpha channel is not provided and personalized color spaces cannot be defined. The latest version allows exploiting a color profile taken from an external file, and embedding JPEG and PNG images (to be introduced later in this section).

An image in BMP format is made up as follows:

**Bitmap file header** Identifies the file as a BMP-encoded one (see Table 2.6).

**Bitmap information header** Reports the size in pixels of the image, the number of colors used (referred to the device on which the bitmap was created), and the horizontal and vertical resolution of the output device that, together with the width and height in pixels, determine the print size of the image in true size; see Table 2.7.

**Color palette** An array (used only for depth 1, 4 or 8) with as many elements as the number of colors used in the image, each represented by a RGBQUAD, a 4-byte structure organized as follows:

| Byte | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Name | rgbBlue | rgbGreen | rgbRed | rgbReserved |
| Description | Amount of blue | Amount of green | Amount of red | 0 (unused) |

[6]An image compressed lossily, if repeatedly saved, will tend to lose quality, up to not being able to recognize its content anymore.

**Table 2.7** Bitmap information header

| Byte | Name | stdval | Description |
|---|---|---|---|
| 15–18 | biSize | 40 | Size of the Bitmap information header (in bytes) |
| 19–22 | biWidth | 100 | Image width (in pixels) |
| 23–26 | biHeight | 100 | Absolute value = image height (in pixels); sign = scan direction of the lines in the pixel map:<br>+ bottom-up (most common variant)<br>− top-down |
| 27–28 | biPlanes | 1 | Number of planes of the device |
| 29–30 | biBitCount | 8 | Number of bits per pixel |
| 31–34 | biCompression | 0 | Pixel map compression:<br>0 (BI_RGB) not compressed<br>1 (BI_RLE8) compressed using RLE. Valid only for biBitCount = 8 and biHeight > 0<br>2 (BI_RLE4) compressed using RLE. Valid only for biBitCount = 4 and biHeight > 0<br>3 (BI_BITFIELDS) not compressed and encoded according to personalized color masks. Valid only for biBitCount ∈ {16, 32}; unusual<br>4 (BI_JPEG) the bitmap embeds a JPEG image (in version 5)<br>5 (BI_PNG) the bitmap embeds a PNG image (in version 5) |
| 35–38 | biSizeImage | 0 | Size of the pixel map buffer (in bytes). Can be 0 when biCompression = BI_RGB |
| 39–42 | biXPelsPerMeter | 0 | Horizontal resolution of the output device (in pixels/meter); 0 if unspecified. |
| 43–46 | biYPelsPerMeter | 0 | Vertical resolution of the output device (in pixels/meter); 0 if unspecified |
| 47–50 | biClrUsed | 0 | Number of colors used in the bitmap<br>if biBitCount = 1: 0<br>if biBitCount ∈ {4, 8}: number of entries actually used in the color palette; 0 indicates the maximum (16 or 256)<br>else: number of entries in the color palette (0 meaning no palette). For depths greater than 8 bits/pixel, the palette is not needed, but it can optimize the image representation |
| 51–54 | biClrImportant | 0 | if biBitCount ∈ {1, 4, 8}: number of colors used in the image; 0 indicates all colors in the palette<br>else if a palette exists and contains all colors used in the image: number of colors<br>else: 0 |

**Image pixels** An array of bytes containing the data that make up the actual image. It is organized by image rows (called *scanlines*), usually stored in the file bottom-up, each with the corresponding pixels from left to right (thus the first pixel is the bottom-left one in the image, and the last is the top-right one). Each pixel consists of a color, expressed as an index in the palette (for depth 1, 4 or 8) or directly as its RGB chromatic components, one after the other (for larger depth values). For instance, in two-color (usually, but not necessarily, black&white) images the palette contains two RGBQUADs, and each bit in the array represents a pixel: 0 indicates the former color in the palette, while 1 indicates the latter. The length in bytes of each scanline thus depends on the number of colors, format and size of the bitmap; in any case, it must be a multiple of 4 (groups of 32 bits), otherwise NUL bytes are added until such a requirement is fulfilled. In version 5 this structure can also embed JPG or PNG images.

The overall size in bytes of an image will be:

$$54 + 4 \cdot \left( \mathrm{u}(15 - b) \cdot 2^b + h \cdot \left\lceil \frac{w \cdot b}{32} \right\rceil \right),$$

where 54 is the size of the first two structures, $b$ is the depth in bits/pixel, and $h$ and $w$ denote, respectively, the height and width of the image in pixels. $2^b$ yields the size in bytes of the palette, and $\mathrm{u}(x)$ denotes the Heaviside unit function: for depth values between 1 and 8 it yields 1; for depth values greater or equal to 16 it yields 0, and hence no space is allocated for the palette. Value 32 as a denominator of the ceiling function is exploited to obtain multiples of 4 bytes, as required by the specifications.

**GIF (Graphics Interchange Format)** Released in 1987 by Compuserve for image transmission on the Internet, *GIF* [1] is today supported by all browsers, and widely exploited in Web pages thanks to its useful features, among which fast displaying, efficiency, the availability of transparency, the possibility to create short animations that include several images and to have a progressive rendering. It allows using at most 256 colors in each image, chosen from a palette called *Color Table*, which makes it not suitable for photographic (halftone) images. A binary alpha channel is supported. It exploits the LZW compression algorithm, that until 2003 in America (and until 2004 in the rest of the world) was patented, for which reason whoever wrote software that generated GIF images had to pay a fee to CompuServe and Unisys.

GIF is organized into blocks and extensions, possibly made up of sub-blocks and belonging to three categories:

**Control** blocks Image-processing information and hardware-setting parameters.

*Header* Takes the first 6 bytes: the first three are characters that denote the format ('GIF'), the other three specify the version ('87a' or '89a').

*Logical Screen Descriptor* Always present next to the header, contains global information on image rendering according to the structure reported in Table 2.8. The *logical screen* denotes the area (the monitor, a window, etc.) where the image is to be displayed.

**Table 2.8** Organization of a GIF logical screen descriptor

| Byte | Name | Type | Description |
|---|---|---|---|
| 0–1 | LogicalScreenWidth | Unsigned | Horizontal coordinate in the logical screen where the top-left corner of the image is to be displayed |
| 2–3 | LogicalScreenHeight | Unsigned | Vertical coordinate in the logical screen where the top-left corner of the image is to be displayed |
| 4 | Packed field | | Specifies: $g$ a flag (usually true) that indicates the presence of a global palette (*Global Color Table*), used by default for all images that do not have a local one; $k$ the color resolution, a 3-bit integer meaning that the palette colors are chosen from an RGB space defined on $k+1$ bits per channel; $s$ a flag that indicates whether colors in the palette are ordered by decreasing importance; $i$ the palette size, a 3-bit integer meaning that $i+1$ bits are used for the index (and hence at most $2^{i+1}$ entries are allowed) |
| 5 | BackgroundColorIndex | Byte | The alpha channel, denoted as the index of the palette color that, during visualization, is replaced by the background |
| 6 | PixelAspectRatio | Byte | The width/height ratio for pixels, ranging from 4:1 to 1:4, approximated in $1/64$th increments; 0 no aspect ratio information; [1..255] $(\mathit{PixelAspectRatio}+15)/64$ |

*Color Table* A color palette that can be local (valid only for the image immediately following it) or global (valid for all images that do not have a local one). Both kinds have the same structure that consists of a sequence of $3 \cdot 2^{i+1}$ bytes representing RGB color triplets.

*Graphic Control Extension* Controls the visualization of the immediately subsequent image, such as what to do after the image has been displayed (e.g., freezing the image, or going back to the background or to what was in place before), whether user input is needed to continue processing, the visualization delay (in $1/100$th of second) before continuing processing (important for animations), local color table and transparency settings (as in the Logical Screen Descriptor). Animations include one such block for each frame that makes up the animation.

*Trailer* Indicates the end of the GIF file.

**Graphic-Rendering** blocks Information needed to render an image.

*Image Descriptor* Contains the actual compressed image (in the *Table Based Image Data* sub-block), plus information concerning its size and position in the logical screen (in pixels), the local color table (as in the Logical Screen Descrip-

**Table 2.9** Header block of a TIFF file

| Byte | Description |
|---|---|
| 0–1 | Two characters denoting the byte ordering used in the file:<br>II *little-endian* (from the least to the most significant bit), used by Microsoft<br>MM *big-endian* (from the most to the least significant bit), used by the Macintosh |
| 2–3 | The number 42 (denoting the format TIFF) |
| 4–7 | The offset in byte of the first IFD |

tor) and the presence of *interlacing* (i.e., the possibility to display it in stages by progressively refining the representation).

*Plain Text Extension* Allows rendering text, encoded as 7-bit ASCII, as images on the screen. Defines the features of grids in which each character is rendered as a raster image. Not all interpreters support this extension.

**Special Purpose** blocks Information not affecting image processing.

*Comment Extension* Contains comments that are not going to be displayed.

*Application Extension* Contains data that can be exploited by specific programs to add special effects or particular image manipulations.

The image is compressed using the LZW algorithm, where the alphabet is known: it is the number of possible pixel values, reported in the header. Actually, GIF applies a modification to the LZW algorithm to handle cases in which the initially defined size of the compression codes ($N$ bits per pixel) turns out to be insufficient and must be extended dynamically. This is obtained by adding to $A$ two additional meta-symbols: $C$ (clear, used to tell to the encoder to re-initialize the string table and to reset the compression size to $N + 1$ for having more codes available) and $E$ (end of data).

**TIFF (Tagged Image File Format)** Developed by Microsoft and Aldus (that subsequently joined Adobe, which to date holds the patent), *TIFF* [2] is currently the most used, flexible and reliable technique for storing bitmap images in black&white, gray-scale, color scale, in RGB, CMYK, $\text{YC}_b\text{C}_r$ representation. As a drawback, the file size is quite large. There are no limits to the size in pixel that an image can reach, nor to the depth in bits. A TIFF file can be saved with or without compression, using the LZW or the Huffman method. It can embed meta-information (the most common concerns resolution, compression, contour trace, color model, ICC profile) in memory locations called *tags*. Extensions of the format can be created by registering new tags with Adobe, but the possibility of adding new functionality causes incompatibility between graphic programs and lack of support on the browsers' side. It is supported by the most common operating systems, and two versions of it exist, one for Windows and one for Macintosh, that differ because of the byte ordering.

The file structure includes a data block (*header*) organized as in Table 2.9, plus one or more blocks called *Image File Directories* (IFD), each of which (supposing it starts at byte $I$) is structured as follows:

**Table 2.10** Structure of a TIFF Image File Directory entry. Note that each field is a one-dimensional array, containing the specified number of values

| Byte | Description |
|---|---|
| 0–1 | Tag that identifies the field |
| 2–3 | Type:<br>1. BYTE, 8-bit unsigned integer<br>2. ASCII, 7-bit value<br>3. SHORT, 2-byte unsigned integer<br>4. LONG, 4-byte unsigned integer<br>5. RATIONAL, made up of 2 LONGs (numerator and denominator)<br>6. SBYTE, 8-bit integer in two's complement<br>7. UNDEFINED, byte that may contain everything<br>8. SSHORT, 2-byte integer in two's complement<br>9. SLONG, 4-byte integer in two's complement<br>10. SRATIONAL, made up of 2 SLONGs (numerator and denominator)<br>11. FLOAT, in single precision (4 bytes)<br>12. DOUBLE, in double precision (8 bytes) |
| 4–7 | Number of values of the indicated type |
| 8–11 | Offset to which the value is placed (or the value itself if it can be represented in 4 bytes) |

- Bytes $I$, $I+1$: the number of directory entries in the IFD (say $N$)
- Bytes from $I+2+12\cdot(k-1)$ to $I+2+12\cdot k-1$: the $k$-th directory entry (made up of 12 bytes and organized as in Table 2.10), for $1 \le k \le N$
- Bytes from $I+2+12\cdot N$ to $I+2+12\cdot N+3$: the address of the next IFD (0 if it does not exist).

Each TIFF file must have at least one IFD that, in turn, must contain at least one entry. The *Tag* field in the entry expresses as a numeric value which parameter is being defined (e.g., 256 = ImageWidth, 257 = ImageLength, 258 = BitsPerSample, 259 = Compression, etc.). Different kinds of images require different parameters, and a parameter may take on different values in different kinds of images.

Images can also be stored in frames, which allows for a quick access to images having a large size, and can be split into several 'pages' (e.g., all pages that make up a single document can be collected into a single file).

**JPEG (Joint Photographic Experts Group)** *JPEG* [15, 19], named after the acronym of the group that in 1992 defined its standard, aims at significantly reducing the size of raster images, mainly of halftone pictures such as photographs, at the cost of a lower quality in enlargements. Indeed, it provides lossy compression and true color. It is ideal for efficient transmission and exploitation of images on the Web, even in the case of photographs including many details, which makes it the most widely known and spread raster format. However, it is not suitable for images that are to be modified because each time the image is saved an additional compression is applied, and thus increasingly more information will be lost. In such a case, it is wise to save intermediate stages of the artifact in a lossless format. Meta-information cannot be embedded in the file.

The dramatic file size reduction with reasonable loss in terms of quality is obtained at the expenses of information that, in any case, the eye cannot perceive, or perceives negligibly, according to studies in human physiology. JPEG encoding of an image is very complex, and its thorough discussion is out of the scope of this book. However, a high-level, rough survey of its most important phases can be useful to give the reader an idea of the involved techniques.

1. Switch from the RGB color space channels to the YUV ones. This singles out luminance, to which the eye is more sensible than it is to colors. Thus, luminance ($Y$) can be preserved in the next step, while losing in chrominance.
2. Reduce chrominance ($U$, $V$) components by subsampling[7] (available factors are 4:4:4, 4:2:2, 4:2:0). This replaces $2 \times 1$, $2 \times 2$ or larger pixel blocks by a single value equal to the average of their components, and already reduces by 50–60% the image size.
3. Split each channel in blocks of $8 \times 8$ pixels, and transpose each value to an interval centered around the zero.
4. Apply to each such block the *Discrete Cosine Transform* (*DCT*) that transposes the image into a *frequency-domain* representation. The forward formula is:

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)u\pi}{16}\right]$$

   while the inverse formula, to go back to the original image, is:

$$f(x,y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} F(u,v)C(u)C(v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)u\pi}{16}\right],$$

   where $f(x,y)$ is the original pixel, $F(u,v)$ is the DCT coefficient and $C(u)$, $V(u)$ are normalization factors. The resulting $8 \times 8$ matrix, whose values are rounded to the closest integer, aggregates most of the signal in the top-left corner, called DC coefficient (that is the average of the image luminance), while the other cells are called AC coefficients, as represented in Fig. 2.3 on the left.
5. Divide each DCT coefficient by the corresponding value (between 0 and 1) in an $8 \times 8$ *quantization table*. This amplifies the effect of the previous step. Since the eye can note small differences in brightness over a broad area, but cannot distinguish in detail high frequency brightness variations, high frequencies can be cut off (using lower values in the quantization table, while higher values are

[7] The *subsampling* scheme is commonly expressed as an $R:f:s$ code that refers to a conceptual region having height of 2 rows (pixels), where:

| | |
|---|---|
| $R$ | width of the conceptual region (*horizontal sampling reference*), usually 4; |
| $f$ | number of chrominance samples in the first row of $R$ pixels; |
| $s$ | number of (additional) chrominance samples in the second row of $R$ pixels. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $DC$ | $AC_{01}$ | $AC_{02}$ | $AC_{03}$ | $AC_{04}$ | $AC_{05}$ | $AC_{06}$ | $AC_{07}$ |
| $AC_{10}$ | $AC_{11}$ | $AC_{12}$ | $AC_{13}$ | $AC_{14}$ | $AC_{15}$ | $AC_{16}$ | $AC_{17}$ |
| $AC_{20}$ | $AC_{21}$ | $AC_{22}$ | $AC_{23}$ | $AC_{24}$ | $AC_{25}$ | $AC_{26}$ | $AC_{27}$ |
| $AC_{30}$ | $AC_{31}$ | $AC_{32}$ | $AC_{33}$ | $AC_{34}$ | $AC_{35}$ | $AC_{36}$ | $AC_{37}$ |
| $AC_{40}$ | $AC_{41}$ | $AC_{42}$ | $AC_{43}$ | $AC_{44}$ | $AC_{45}$ | $AC_{46}$ | $AC_{47}$ |
| $AC_{50}$ | $AC_{51}$ | $AC_{52}$ | $AC_{53}$ | $AC_{54}$ | $AC_{55}$ | $AC_{56}$ | $AC_{57}$ |
| $AC_{60}$ | $AC_{61}$ | $AC_{62}$ | $AC_{63}$ | $AC_{64}$ | $AC_{65}$ | $AC_{66}$ | $AC_{67}$ |
| $AC_{70}$ | $AC_{71}$ | $AC_{72}$ | $AC_{73}$ | $AC_{74}$ | $AC_{75}$ | $AC_{76}$ | $AC_{77}$ |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 6 | 7 | 15 | 16 | 28 | 29 |
| 3 | 5 | 8 | 14 | 17 | 27 | 30 | 43 |
| 4 | 9 | 13 | 18 | 26 | 31 | 42 | 44 |
| 10 | 12 | 19 | 25 | 32 | 41 | 45 | 54 |
| 11 | 20 | 24 | 33 | 40 | 46 | 53 | 55 |
| 21 | 23 | 34 | 39 | 47 | 52 | 56 | 61 |
| 22 | 35 | 38 | 48 | 51 | 57 | 60 | 62 |
| 36 | 37 | 49 | 50 | 58 | 59 | 63 | 64 |

**Fig. 2.3** Schema of a JPEG DCT coefficients (on the *left*), and sequence of the corresponding zigzag traversal (on the *right*)

used for exalting low frequencies). This is the most information-lossy step. Each transformed block represents a frequency spectrum.

6. Rearrange AC cells in a 64-elements vector following a zigzag route (as in the schema on the right in Fig. 2.3), which increases the chance that similar cells become adjacent.
7. Perform final lossless *entropy encoding*, exploiting several algorithms:

   - RLE compression on the AC components, creating pairs (*skip*, *value*) where *skip* is the number of values equal to 0 and *value* is the next value different than zero. Since the array resulting from the zigzag reading contains many consecutive 0 values, this method saves significant space.
   - *Differential Pulse Code Modulation* (*DPCM*) compression on the DC component: the DC component of the $i$th block is encoded as the difference with respect to the preceding block ($DC_i - DC_{i-1}$); indeed, it turned out that there exists a statistical relationship between DC components of consecutive blocks.

     *Example 2.5* (Compression in JPEG) The sequence 150, 147, 153, 145, 152, 160 is stored by DPCM as the value 150 followed by the differences with the remaining values: −3, 6, −8, 7, 8.

   - Huffman encoding for the final data. In this way, a further compression of the initial data is obtained, and, as a consequence, a further reduction of the JPEG image size.

A JPEG file is made up of *segments*, each started by a 2-byte *marker* where the first byte is always $\mathrm{FF}_{16}$ and the second denotes the type of segment. If any, the third and fourth bytes indicate the length of the data. In entropy-coded data (only), immediately following any $\mathrm{FF}_{16}$ byte, a $00_{16}$ byte is inserted by the encoder, to distinguish it from a marker. Decoders just skip this $00_{16}$ byte (a technique called *byte stuffing*).

**PNG (Portable Network Graphics)** *PNG* is an open and free format [7], created by some independent developers as an alternative to the GIF for compressed images, and approved by W3C in 1996. It has been accepted as ISO/IEC 15948:2003 standard. A motivation for it came from GIF being patented, and from the purpose (announced in 1995) of its owners (CompuServe and Unisys) to impose a fee to third parties that included GIF encoders in their software. Thus, many of its features are similar to GIF, as reported in the following list:

**Compression** Mandatory in PNG, exploits the information lossless *Zlib* algorithm[8] [17], that yields results in general 20% better than GIF, and can be significantly improved by using filters that suitably rearrange the data that make up the image.

**Error check** The *CRC*-32 (32-bits *Cyclic Redundancy Check*) system associates check values to each data block, and is able to immediately identify any corruption in the information saved or transmitted through the Internet.

**Color** Full 24-bit RGB (true color) mode is supported as a range of colors for images.

**Alpha channel** A transparency degree ranging over 254 levels of opacity is allowed.

**Interlacing** 1/64th of the data is sufficient to obtain the first, rough visualization of the image.

**Gamma correction** Allows, although approximately, to balance the differences in visualization of images on different devices.

Thus, compared to GIF, PNG improves performance (interlacing is much faster, compression rates are much higher) and effectiveness (the number of colors that can be represented is not limited to a maximum of 256, the alpha channel is not limited to a binary choice between fully transparent and completely opaque), but does not provide support for animated images (although a new format, *Multiple-image Network Graphics* or *MNG*, has been defined to overcome this lack).

Encoding an image in PNG format is obtained through the following steps:

1. **Pass extraction**: in order to obtain a progressive visualization, the pixels in a PNG image can be grouped in a series of small images, called *reduced images* or *passes*.
2. **Serialization** by scanline, top-down among scanlines and left-to-right within scanlines.
3. **Filtering** of each scanline, using one of the available filters.
4. **Compression** of each filtered scanline.
5. **Chunking**: the compressed image is split into packets of conventional size called *chunks*, to which an error checking code is attached.
6. Creation of the **Datastream** in which chunks are inserted.

Textual descriptions and other auxiliary information, that is not to be exploited during decompression, can be embedded in the file: a short description of the image, the background color, the chromatic range, the ICC profile of the color space, the image histograms, the date of last modification and the transparency (if not found in the file). The file size is larger than JPEG, but compression is lossless.

[8] A variant of the *LZ77*, developed by J.-L. Gailly for the compression part (used in zip and gzip) and by M. Adler for the decompression part (used in gzip and unzip), and almost always exploited nowadays in ZIP compression. It can be optimized for specific types of data.

**DjVu (DejaVu)** *DjVu* ("déjà vu") [8] was being developed since 1996 at AT&T Labs, and first released in 1999. It is intended to tackle the main problems related to document digitization, preservation and transmission. Its main motivation is that the huge number of legacy paper documents to be preserved cannot be cheaply and quickly re-written in a natively digital format, and sometimes it is not desirable either because (part of) their intrinsic value comes from their original visual aspect. Thus, the only technology that can be sensibly applied is scanning, whose drawback is that the image representation of color documents requires significant amounts of space. Usual ways to deal with this problem are lossy compression, as in JPEG, or color depth reduction to gray-level or black&white. However, sometimes the lower definition due to compression is unacceptable, and color cannot be stripped off without making the document meaningless. In these cases, the file size seriously affects its transmittability which, in turn, prevents its embedding into Web pages and hence severely limits access to it. DjVu solves all these problems at once by preserving the original aspect of the document, without losing in quality on sensible components (such as text), and keeping the size of the outcome within limits that significantly outperform JPEG, GIF and PDF. To give an idea, a 300 dpi scanned full color A4 page would take 25 MB that can be represented in less than 100 kB.

The key for this optimal solution lies in the selective application of different types and rates of compression to different kinds of components: text and drawings, usually being more important to the human reader, are saved in high quality (to appear sharp and well-defined), while stronger compression can be applied to halftone pictures in order to preserve their overall appearance at a sufficient tradeoff between quality and space. It is also a progressive format: most important components (text ones) are displayed first and quickly, and the others (pictures and then background) are added afterwards as long as the corresponding data is gained. A DjVu document may contain multiple (images of) pages, placed in a single file or in multiple sources: the former solution is more comfortable for handling, but involves a serialization for their exploitation (a page cannot be displayed until all the previous ones have been loaded), while the latter allows for quicker response and selective transmission. Additional useful features that a DjVu file can contain is meta-data (e.g., hyperlinks) in the form of annotations, a hidden text layer that allows reading plain text corresponding to what is displayed, and pre-computed thumbnails of the pages that can be immediately available to the interpreter.

Several types of compression are provided by DjVu. The *JB2* data compression model is exploited for binary (black&white) images, where white can be considered as being the background and black as the significant content (usually text and drawings). It is a bitonal technique that leverages the repetition of nearly identical shapes to obtain high compression rates, and essentially stores a dictionary of prototypical shapes and represents each actual shape as its difference from one of the prototypes. *IW44* wavelet representation is exploited for pictures. Other compression methods are available as well.

According to its content type, a DjVu page consists of an image in one of the following formats:

**Table 2.11** Structure of a DjVu chunk

| Field | Type | Description |
|---|---|---|
| ID | Byte [4] | An ID describing the use of the chunk as a string, e.g.,<br><br>FORM container chunk<br>FORM:DJVM multi-page document<br>FORM:DJVU single-page document<br>FORM:DJVI shared data (e.g., the shape dictionary)<br>FORM:THUM embedded thumbnails chunks<br>Djbz shared shape table<br>Sjbz mask data (BZZ-compressed JB2 bi-tonal)<br>FG44 foreground data (IW44-encoded)<br>BG44 background data (IW44-encoded)<br>TH44 thumbnails data (IW44-encoded)<br>FGbz color data for the shapes (JB2)<br>BGjp background image (JPEG-encoded)<br>FGjp foreground image (JPEG-encoded) |
| Length | 32-bit integer | The length of the chunk data (in Big Endian byte ordering) |
| Data | Byte [Length] | The chunk data |

**Photo** used for color or gray-level photographic images, compressed using IW44.

**Bi-level** used for black&white images, handled using JB2.

**Compound** used for pages that mix text and pictures. The background and foreground are identified, separated and represented in two different layers. A third layer (called *mask layer*) is used to distinguish between background and foreground. The background contains paper texture (to preserve the original look-and-feel of the document) and pictures, while the foreground contains text and drawings. The foreground is represented as a bi-level image, called *foreground mask*, encoded in JB2 where black denotes the foreground pixels and white the background ones. Foreground colors can be represented by specifying the color of each foreground shape separately, or as a small image, sub-sampled with a factor ranging from 1 to 12 (usually 12: smaller values do not significantly increase quality), that is scaled up to the original size of the document and drawn on the background image. The background image is obtained through progressive IW44 refinements, scaled down with a sub-sampling factor between 1 and 12 (usually 3, smaller if high-quality pictures are to be represented, higher if no pictures are present).

DjVu files are represented in UTF-8 encoded Unicode, and identified by a header having hexadecimal values 41 54 26 54 in the first four bytes. They are structured in a variable number of *chunks*, that contain different types of information according to the field structure described in Table 2.11. An external FORM chunk contains all the others, with no further nesting. Each chunk must begin on an even byte boundary, including an initial 00 padding byte to ensure this in case it does not hold (as in the beginning of the file). Chunks of unknown type are simply ignored.

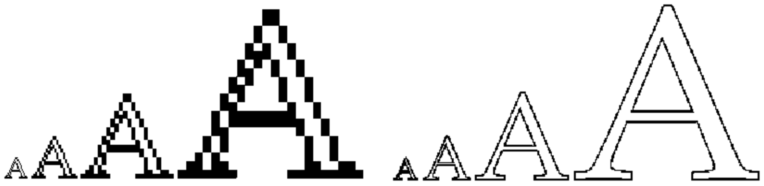


**Fig. 2.4** Comparison between enlargements in a raster format (on the *left*) and in a vector one (on the *right*)

### Vector Graphic

*Vector Graphic* is a technique to describe images exploiting mathematical/geometrical primitives, such as points, lines, curves and polygons. Further information can be attached to each element as well (e.g., color). Compared to raster formats, in which images are described as a grid of syntactically unrelated colored points, the advantages of vectorial images are: better quality (in particular, when zooming or up-scaling them—see Fig. 2.4), limited memory requirements, easy modification (each component can be managed separately and independently of the others by acting on its descriptive primitives), progressive rendering facility, and easier analysis (the description language might allow applying logical deduction and other kinds of inference to identify desired shapes). On the other hand, it is very hard to describe photographs (because the details to be translated into mathematical primitives would be a huge number). Moreover, very detailed images, such as architecture and engineering projects, might require the processor to perform significantly heavier calculations to display them.

**SVG (Scalable Vector Graphic)** Produced by W3C, SVG [25] dates back to 1999. In addition to vectorial shapes, it allows embedding raster graphics and text, and attaching information to any component (e.g., name, available colors, relationships to other objects). It is suitable for high-complexity images, such as architecture and engineering projects (it allows scaling up or down an image at any ratio), or 3D animations. The main disadvantage that can, however, be tackled by means of suitable tools is given by the grammar complexity. All Internet browsers, except Internet Explorer, support SVG. Although conceived as a vector graphics markup language, it may also act as a page description language (PDL), like PDF, since all the functionality required to place each element in a precise location on the page is provided.

Everything in SVG is a graphic element (a list is provided in Table 2.12) that can be a generic shape, a text or a reference to another graphic element. The features include nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility. Among the most relevant are:

**Shapes** Basic Shapes (straight-lines, polylines, closed polygons, circles and ellipses, rectangles possibly with rounded corners) and Paths (simple or compound shape outlines drawn with curved or straight lines) can be drawn. Ends of lines or

**Table 2.12** Graphical elements available in SVG

| Element | Attributes | Description |
|---|---|---|
| SVG | x, y, width, height, allowZoomAndPan | The most external graphical element that contains all other elements and has the specified *width*, *height* and $(x, y)$ position (useful for displaying the image inside a Web page). The *allowZoomAndPan* flag concerns the possibility to zoom and pan the image |
| Rect | x, y, width, height | Rectangle having the specified *width* and *height*. Optionally, the horizontal and vertical radii $(rx, ry)$ of ellipses used to smooth the angles can be specified |
| Circle | cx, cy, r | Circle centered at $(cx, cy)$ and having radius $r$ |
| Ellipse | cx, cy, rx, ry | Ellipse centered at $(cx, cy)$ and having horizontal and vertical radii $rx$ and $ry$, respectively |
| Line | x1, y1, x2, y2 | Straight line having extremes at $(x1, y1)$ and $(x2, y2)$ |
| Polyline | List of points | The sequence of points—pairs of $(x, y)$ coordinates—that make up the curve, separated by commas |
| Polygon | List of points | A polyline automatically closed, whose points are separated by commas or blanks |
| Text | string, x, y | A string to be displayed |
| Image | x, y, width, height, xlink:href | Allows embedding a PNG or JPEG raster image, and also textual descriptions (useful to classify a document containing also images) |

vertices of polygons can be represented by symbols called *markers* (e.g., arrowheads).

**Text** Characters are represented in Unicode expressed as in XML. Text can flow bidirectionally (left-to-right and right-to-left), vertically or along curved paths. Fonts can reference either external font files, such as system fonts, or *SVG fonts*, whose glyphs are defined in SVG. The latter avoid problems in case of missing font files on the machine where the image is displayed.

**Colors** Painting allows filling and/or outlining shapes using a color, a gradient or a pattern. Fills can have various degrees of transparency. Colors are specified using symbolic names, hexadecimal values preceded by #, decimal or percentage RGB triples like $rgb(\cdot, \cdot, \cdot)$. (Color or transparency) gradients can be linear or radial, and may involve any number of colors as well as repeats. Patterns are based on predefined raster or vector graphic objects, possibly repeated in any direction. Gradients and patterns can be animated. Clipping, Masking and Composition allow using graphic elements for defining inside/outside regions that can be painted independently. Different levels of opacity in clipping paths and masks are blended to obtain the color and opacity of every image pixel.

**Effects** Animations can be continuous, loop and repeat. Interactivity is ensured through hyperlinks or association of image elements (including animations) to (mouse-, keyboard- or image-related) events that, if caught and handled by scripting languages, may trigger various kinds of actions.

**Metadata** According to the W3C's Semantic Web initiative, the Dublin Core (e.g., title, creator/author, subject, description, etc.—see Sect. 5.4.2) or other metadata schemes can be used, plus elements where authors can provide further plain-text descriptions to help indexing, search and retrieval.

SVG sources are pure XML and support DOM [27] (see Sect. 5.1.2). Thus, their content is suitable for other kinds of processing than just visualization: using XSL transformations, uninteresting components can be filtered out, and embedded metadata can be displayed or textually described or reproduced by means of a speech synthesizer. The use of CSSs makes graphical formatting and page layout simple and efficient: their modification is obtained by simply changing the style sheet, without accessing the source code of the document. The files contain many repeated text strings, which makes the use of compression techniques particularly effective. Specifically, using the *gzip* technique on SVG images yields the *SVGZ* format, whose space reduction reaches up to 20% of the original size.

## 2.3 Layout-Based Formats

This section deals with those formats, often referred to as *Page Description Languages* (*PDL*s) that are structured in the perspective of document displaying. They focus on the visual/geometrical aspect of documents, by specifying the position in the page of each component thereof. The most widespread formats in this category, thanks to their portability and universality (independence on platform and on the software exploited to create and handle the documents), are the PostScript and the PDF, which will be introduced in the next sections.

**PS (PostScript)** PostScript (*PS*) [22] is a real programming language (it even allows writing structured programs) for the description and interpretation of pages, developed in 1982 by Adobe Systems. Originally intended for controlling printer devices, and indeed widely exploited in typography, it has been subsequently used for the description of pages and images. It allows providing a detailed description of the printing process of a document, and is characterized by a representation that is independent from the devices on which the pages are to be displayed. Also text characters are considered as graphic elements in PS, which prevents text in a document from being read as such, and hence copied and pasted. Documents generated in this format can be virtually 'printed' on file using suitable drivers; when such files are later interpreted, the original document is perfectly reproduced on different kinds of devices. It has the advantage of not bearing viruses. One of the most famous PS interpreters is *GhostScript*[9] that provides a prompt to enter commands and a 'device' on which graphic elements are drawn.

PS exploits postfix notation and is based on an operator stack, which makes easier command interpretation although seriously affecting code readability. Many kinds of objects can be represented: characters, geometrical shapes, and (color, grayscale or black&white) raster images. They are built using 2D graphic operators, and placed in a specified position in the page. Objects handled by PS are divided into two categories:

**simple** Boolean, fontID, Integer, Mark, Name, Null, Operator, Real, Save
**compound** Array, Condition, Dictionary, File, Gstate, Lock, packedarray, String, Procedure.

Any object in PS can be 'executed': execution of some types of objects (e.g., Integers) pushes them on the stack, while execution of other types of objects (e.g., Operators) triggers actions that, usually, consume objects in the stack.

*Example 2.6* Sample computation of $5 + 3 = 8$ in GhostScript:

| | |
|---|---|
| `GS>` | GhostScript interpreter ready |
| `GS>5 3` | Integer objects `5` and `3` are 'executed', i.e., pushed on the stack |
| `GS<2>` | The prompt indicates that two objects are present in the stack |
| `GS<2>add` | The Operator object `add` extracts two elements from the operators stack (in case of a stack containing fewer than two objects, the execution would terminate issuing an error), sums them and inserts the result on the stack |
| `GS<1>` | The prompt says that only one object is present in the stack |
| `GS<1>pstack` | The `pstack` Operator displays the stack content |
| `8` | (in this case, 8) |
| `GS<1>` | Still one object in the stack, interpreter ready |

PS allows defining procedures, to be called/run subsequently, by enclosing a sequence of commands in curly brackets and assigning them an identifier (of type Name) through the `def` operator.[10] When it is called, the interpreter runs in sequence the objects in curly brackets.

*Example 2.7* Sample definition of a single `printSum` procedure in GhostScript, that includes the sum and print operations:

[9]An open-source project (http://pages.cs.wisc.edu/~ghost/) that does not directly handle PS and PDF formats, this way being able to handle some differences in the various versions or slangs of such formats. An associated viewer for PS files, called *GSview*, is also maintained in the project.

[10]In this section, PostScript code and operators will be denoted using a `teletype` font. Operators that can be used with several numbers of parameters are disambiguated by appending the number *n* of parameters in the form `operator/n`.

| | |
|---|---|
| `GS>/printSum {add pstack} def` | procedure definition |
| `GS>5 3` | push of two Integers on the stack |
| `GS<2>printSum` | two objects in the stack; procedure call |
| `8` | result |
| `GS<1>` | one object (the result) in the stack |

The interpreter reads commands that define objects (such as characters, lines and images). Each command runs an associated parametric graphic procedure that, based on its parameters, suitably places *marks* on a *virtual page*. The virtual page is distinct from (it is just a representation in temporary storage of) the physical device (printer or display). The interpretation of a PS document is based on a so-called *painting model* that exploits a 'current page' in which it progressively adds the marks (a mark could overshadow the previous ones). When all information for the current page has been read (and hence the page is complete), `showpage` causes the interpreter to call the page printing procedure: all marks in the virtual page are rendered on the output, i.e., transformed into actual drawings on the device, and the current page becomes again empty, ready for another description.

*Dictionaries* are objects in which a key-value list can be defined. They can be used as a kind of variables or to store procedures. All pre-defined PS operators correspond to procedures defined in the read-only dictionary **systemdict**. The interpreter maintains a separate stack of dictionaries, whose top element is called *current dictionary*. At runtime, whenever a name that does not correspond to a simple type is referenced, a look-up for it is started from the current dictionary down through the dictionary stack. For instance, groups of at most 256 text characters (*character set*s) are represented by programs that define a dictionary (*Font Dictionary*). Such a dictionary contains an **Encoding** vector whose 256 elements are names, each corresponding to a drawing procedure for a character. Each character is denoted by an integer between 0 and 255, used as an index to access the Encoding vector of the character set currently in use. When referenced, the name is extracted and the corresponding drawing procedure is run. Each character set defines different names for its characters. ASCII characters are defined in the **StandardEncoding** vector.

A *path* is a sequence of points, lines and curves, possibly connected to each other, that describe geometrical shapes, trajectories or areas of any kind. A path is made up of one or more sub-paths (straight and curved segments connected to one another). A new path is started by calling `newpath`, always followed by `moveto`, that adds a new non-consecutive sub-path to a previous sub-path (if any). The path can be open or closed (the latter obtained using `closepath`).

The reference system to locate any point in the page consists by default of an ideally infinite Cartesian plane whose origin is placed in the bottom-left corner of the page, and whose horizontal ($x$) and vertical ($y$) axes grow, respectively, towards the right and up-wise. Points are denoted by real-valued coordinates measured in *dots* (corresponding to $1/72$th of inch, as defined in the print industry). Independence on the particular device in use is obtained by considering two coordinate systems: one referring to the user (*user space*) and one referring to the output device (*device space*) on which the page will be subsequently displayed. The former is independent of the latter, whose origin can be placed at any point in the page to fit different printing modes and resolutions and several kinds of supports that can be exploited for visualization. To define the device space, it suffices to indicate the axes origin (by default in the bottom-left with respect to the support on which it will be displayed), their orientation and the unit of measure. The interpreter automatically, and often implicitly, converts the user space coordinates into device space ones. The default settings of the user space can be changed by applying operators that can, e.g., rotate the page or translate the coordinates. Switching from a pair of coordinates $(x, y)$ in the current user space to a new pair $(x', y')$ in the device space is obtained through the following linear equations:

$$x' = ax + cy + t_x, \qquad y' = bx + dy + t_y.$$

The coefficients for the transformations are defined by a $3 \times 3$ matrix,[11] called *Current Transformation Matrix* (*CTM*), always present and equal to

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix},$$

and represented as an array of 6 elements $[a\ b\ c\ d\ t_x\ t_y]$ (where the last column, being fixed, is omitted). The possibility of modifying the matrix to distort and move the user system is often exploited also to efficiently manage recurring objects in the document: each object is defined just once using a separate reference system (whose origin is usually placed in the bottom-left corner of the element to be drawn), and then it is drawn multiple times in different places of the page by simply moving the reference system each time it appears. Printing drivers that thoroughly use this technique will produce more compact documents.

The *Graphic State* is the framework in which the operators (implicitly or explicitly) act. It consists of a structure that contains various graphic parameter settings (see Table 2.13), including color, font type, line thickness, the *current path* and the CTM. It is organized LIFO, which complies with the structure in which objects are typically stored (generally independent on each other and nested at various level of depth). It contains objects, but it is not an object itself, and hence it cannot be directly accessed by programs. It can be handled only using two operators that allow changing the internal graphic state without modifying those around it:

- `gstate` pushes in a stack the whole graphic state
- `grestore` pops from the stack the values of a graphic state

Operators are grouped into 7 main categories:

**Graphic state** operators handle the graphic state.

**Coordinate system and matrix** operators handle the CTM by combining translations, rotations, reflections, inclinations and scale reductions/enlargements.

[11] The most common operations that modify the matrix are *translation* of the axes origin, *rotation* of the system of Cartesian axes by a given angle, *scaling* that independently changes the unit of measure of the axes, and *concatenation* that applies a linear transformation to the coordinate system.

**Table 2.13** PostScript graphic state parameters

| Parameter | Type | Description |
|---|---|---|
| Device-independent | | |
| CTM | array | Current transformation matrix |
| Position | 2 numbers | Coordinates of the current point in the user space (initially undefined) |
| Path | (internal) | Current path, that is, the implicit parameter of some path operators (initially empty) |
| Clipping path | (internal) | Defines the borders of the area in which the output can be cut (initially the entire page) |
| Clipping path stack | (internal) | Stores the clipping paths saved through `clipsave` and not yet returned by `cliprestore` (Type 3) |
| Color space | array | Color space in which the values are to be interpreted (initially *DeviceGray*) |
| Color | (several) | Varies according to the specified color space (initially black) |
| Font | dictionary | Contains the set of graphical shapes to represent the characters in a font style |
| Line width | number | Thickness exploited by lines (initially 1.0) |
| Line cap | integer | Shape of lines end (initially square) |
| Line join | integer | Shape of conjunctions of segments |
| Miter limit | number | Maximum length of a miter line for `stroke` |
| Dash pattern | array and numbers | Style for drawing lines by `stroke` |
| Stroke adjustment | boolean | Defines whether resolution is to be compensated in case of too thin a thickness (Type 2) |
| Device-dependent | | |
| Color rendering | dictionary | Collection of parameters to transform CIE-based colors into values suitable for the device color (Type 2) |
| Overprint | boolean | Specifies whether the underlying area is to be overwritten during printing (Type 2) |
| Black generation | procedure | Computes the amount of black to be used when converting from RGB to CMYK (Type 2) |
| Undercolor removal | procedure | Based on the quantity of black used by the `black generation` procedure computes the amount of the other colors to be used (Type 2) |
| Transfer | procedure | Correction in case of transfer of pages on particular devices |
| halftone | (several) | Defines a screen for rendering gray levels and colors |
| Flatness | numbers | Precision for curve rendering on output devices |
| Smoothness | numbers | Precision for gradient rendering on output devices (Type 3) |
| Device | (internal) | Internal structure that represents the current state of the output device |

**Table 2.14** PostScript graphic procedures

| | |
|---|---|
| `erasepage` | paints the page in white |
| `showpage` | prints the page on the physical device |
| `fill` | fills the current path with the current color (if it denotes a closed line) |
| `eofill` | fills with the current color the internal part (as defined by the 'even–odd rule') of the 'current path' |
| `stroke` | draws a line along the points in the current path |
| `ufill` | fills with the current color a path given in input (called *userpath*) |
| `ueofill` | fills with the current color the internal part (as defined by the 'even–odd rule') of a 'userpath' given in input |
| `ustroke` | draws a line along the points in a *userpath* given in input |
| `rectfill` | fills with the current color a rectangle defined in input |
| `rectstroke` | draws the contour of a rectangle defined in input |
| `image` | draws a raster image |
| `colorimage` | draws a color raster image |
| `imagemask` | uses a raster image as a mask to locate zones to be filled with the current color |
| `show` | prints on the page the characters in a string |
| `ashow` | prints on the page the characters in a string by spacing them of the number of points given in input |
| `kshow` | runs the procedure defined in input while printing on the page the characters of a string |
| `widthshow` | as `show`, but modifies character width and height (and spacing accordingly) |
| `awidthshow` | combines the effects of `ashow` and `widthshow` |
| `xshow` | prints the characters in a string on the page using as width of each the values defined in a vector in input |
| `yshow` | prints the characters in a string on the page using as height of each the values defined in a vector in input |
| `xyshow` | combines the effects of `xshow` and `yshow` |
| `glyphshow` | prints a character identified by a name (associated to a draw procedure) |
| `cshow` | prints the characters in a string using a drawing procedure defined in input |

**Path construction** operators are the only way to modify the current path to be added to the graphic state that describes the shapes using the parameters reported in the CTM. They do not place marks on the page (a job of the painting operators).

**Painting** operators draw on output, by referring to the information contained in the graphic state, graphic elements (shapes, lines and raster images) that have been placed in the current path and handle the sampled images. The clipping path contained in the graphic state limits the region of the page that is affected by the painting operators, the only that will be displayed in output. Everything that can be drawn in PS on a device goes through a graphic procedure (a list is provided in Table 2.14).

**Glyph and Font** operators draw *glyph* characters, also using path and painting operators.

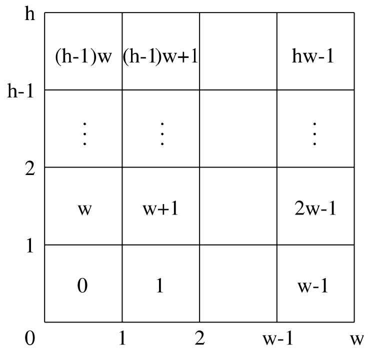


**Fig. 2.5** Ordering of samples' scanning in PS images

**Device setup** operators define an association between raster memory and physical output on which the image is to be displayed.

**Output** operators, having completed the image description, transmit the page to the output.

Operators exploit both implicit (among which the current **path**, **color**, **line width** and **font**) and explicit parameters. Parameters are always read-only to avoid errors of inconsistency due to modifications not updated. Some have biases on the type or range their values must belong to. Numeric values are always stored as real numbers, and are forced to fall in the required ranges, independently of the initial specifications. Lastly, the current path, clipping path and the device parameters are objects internal to the graphic state, and for this reason cannot be accessed by programs.

Raster images are regarded as rectangles of $h \times w$ units in a Cartesian reference system having its origin in the bottom-left vertex. They are represented as *sampled images*, i.e., as sequences of sampled arrays, where the samples contain color-related information and the arrays are obtained by linewise scanning the image from the origin until the top-right point, as represented in Fig. 2.5.

To represent an image, some (inter-related) parameters must be specified, that will be implicitly or explicitly used by the operators:

- The format of the source image: width (number of columns) and height (number of rows), number of components per sample and number of bits per component.
- The capacity in bits of the source image, given by the product

$$height \times width \times components \times bits/component.$$

- The correspondence between user space coordinates and external coordinate system, to define the region in which the image will be inserted.
- A mapping between the values of the source image components and the respective values in the current color space.
- The stream of data that make up the image samples.

Thanks to the independence from the output devices on which the documents will be displayed, properties such as resolution, scan ordering of samples, orientation of image and others are to be intended as referred to the images and not to the devices, although the actual resolution that can be obtained obviously depends on the properties of the latter.

Three *levels* of PS exist, having increasing expressive power. As to image handling, the difference can be summarized as follows:

1. Supports almost only images defined in a DeviceGray color space (graylevel). It is rendered on output using `image/5` that processes the image data coming only from specific procedures and not directly from files or strings. The number of bits per component ranges only between 1 and 8. Some level 1 interpreters handle images with three or four components per value, by using `colorimage`.
2. Adds to the features of level 1 `image/1` (whose parameter is an image dictionary in which much more information can be inserted to define more precisely the image features) the possibility of using 12 bits per component and using files and strings as image data sources.
3. Adds to `imagemask` of previous levels two more operators for color masking.

The number of components per color varies according to the Device Space in use (1 component for DeviceGray, 3 for DeviceRGB, etc.) that, in turn, depends on the operator used:

- `image/5` exploits only DeviceGray;
- `colorimage` refers to the value taken by the **ncomp** parameter: DeviceGray if it is 1, DeviceRGB if it is 3, and DeviceCMYK if it is 4;
- `image/1` refers the current color space.

In turn, each component is made up of $n \in \{1, 2, 4, 8, 12\}$ bits that can represent $2^n$ (i.e., from 2 to 4096) values interpreted as integers in the range $[0, 2^n - 1]$. Image data (*samples*) are stored as byte streams that are split in units, starting from the most significant bit, based on the number of bits/component, so that each unit encodes a value for a color component. The encoded values are interpreted in two ways to obtain the corresponding color:

- `colorimage` and `image/5` map the integer range $[0, 2^n - 1]$ onto $[0.0, 1.0]$;
- `image/1` uses the **Decode** parameter of the corresponding dictionary.

Halftoning techniques are used to approximate color values of components. The stream length must be a multiple of 8 bits (if it is not, padding bits are introduced, that will be ignored by the interpreter during the decoding phase). The bytes can be interpreted as 8-bit integers, starting from the high order bit:

$$\dots\ \underbrace{\overbrace{0010}^{\text{high}}\ \overbrace{1001}^{\text{low}}}_{\text{byte}}\ 0001\ \ 0010\ \ 01\ \dots$$

The byte stream is passed to the interpreter by means of files, strings or procedures (depending on the level). Its organization may vary when many components per sample are provided: for a single data source, the component values (e.g., red/green/blue for the DeviceRGB color space) are adjacent for each sample; in a multiple data source, conversely, there are first all values for the red component, followed by all values of the green one and lastly by the blue ones, and components can be distributed over different sources.

The CTM to modify the coordinate system is provided as a separate operator in the case of `image/5`, or, in the case of `image/1`, as a dictionary that contains the parameters needed to render the image on the page. It allows using any color space, defining an encoding for the sample values different from the standard one, inserting an interpolation among samples and making explicit particular values to mask a color.

Different kinds of dictionaries can be chosen by specifying the **ImageType** parameter inside the dictionary itself. If the value is 1, an image will be represented in an opaque rectangle; if it is 3 or 4 (valid in level 3 language), different levels of color masking can be used, so that new marks added to the page do not to completely overwrite those previously placed in the same area of the page. The **Decode** parameter defines a linear mapping of the integer values of color components in order to exploit them as parameters of `setcolor` in the color space. It is an array of pairs that represent the minimum and maximum value for such a mapping, as shown in Table 2.15. The output value is

$$c = D_{\min} + \left( i \cdot \frac{D_{\max} - D_{\min}}{2^n - 1} \right),$$

where $n =$ BitsPerComponent, $i$ is the input value and $D_{\min}$, $D_{\max}$ are the values in the array. It yields $c = D_{\min}$ for $i = 0$, $c = D_{\max}$ for $i = 2^n - 1$ and intermediate values among these extremes for the other values of $i$.

*Example 2.8* An excerpt of code referred to an image using type 1 dictionary:

| | |
|---|---|
| `/DeviceRGB setcolorspace` | color space identification |
| `45 140 translate` | place the image in the bottom-left corner |
| `132 132 scale` | scale the image to a 132 × 132 unit square |
| `<<` | beginning of the dictionary |
| `  /ImageType 1` | dictionary identification |
| `  /Width 256` | width |
| `  /Height 256` | height |
| `  /BitsPerComponent 8` | bits per component |
| `  /Decode [0 1 0 1 0 1]` | array for color coding |
| `  /ImageMatrix [256 0 0 -256 0 256]` | CTM array |
| `  /DataSource /ASCIIHexDecode filter` | source data obtained by a filter in hexadecimal values |
| `>>` | end of dictionary |
| `image` | |
| ... | stream of hexadecimal values representing the 256 × 256 = 65536 image samples |
| `>` | mark for the end of source data |

**Table 2.15** Configuration of the decode array according to the most frequent color spaces to be mapped

| Color space | Decode array |
|---|---|
| DeviceGray | [0 1] |
| DeviceRGB | [0 1 0 1 0 1] |
| DeviceCMYK | [0 1 0 1 0 1 0 1] |
| CIEBasedABC | [0 1 0 1 0 1] |
| DeviceN | [0 1 0 1 ... 0 1] with *N* pairs of [0 1] |

**Table 2.16** Type 1 dictionary parameters for PostScript

| Parameter | Type | Optional | Value |
|---|---|---|---|
| ImageType | integer | No | 1 (the dictionary, and hence image, type) |
| Width | integer | No | Width in samples |
| Height | integer | No | Height in samples |
| ImageMatrix | array | No | Six numbers that define the transformation from user space to image space |
| MultipleDataSources | boolean | Yes | If true the samples are provided through different sources, otherwise (default) are packed in the same data stream |
| DataSource | (various) | No | The source from which the data stream is to be drawn. If MultipleDataSources is true, it is an array of as many components as in the color space, otherwise it is a single file, procedure or string |
| BitsPerComponent | integer | No | Number of bits used to represent each color component (1, 2, 4, 8 or 12) |
| Decode | array | No | Values describe how the image sample is to be mapped in the range of values of the proper color space |
| Interpolate | boolean | Yes | Denotes the presence of interpolation in the image visualization |

Type 3 dictionaries allow exploiting an *explicit mask*, i.e., specifying an area of the image to be masked (in such a case, it is not required that the image and the mask have the same resolution, but they must have the same position in the page). It uses two more sub-dictionaries, the *image data dictionary* (**DataDict**) and the *mask dictionary* (**MaskDict**), that are similar to a Type 1 dictionary, except for some restrictions applied to the parameters. The parameters for Type 1 dictionaries are reported in Table 2.16, while those for Type 3 dictionaries can be found in Table 2.17.

A Type 4 dictionary, instead of an area to be masked, identifies a range of colors to be used as a mask (a technique called *color key masking*) that are not displayed. It does not use sub-dictionaries, and differs from Type 1 because of the presence of pa-

**Table 2.17** Type 3 dictionary parameters for PostScript

| Parameter | Type | Optional | Value |
|---|---|---|---|
| ImageType | integer | No | 3 (code that identifies the dictionary type) |
| DataDict | dictionary | No | Reference to a type 1 dictionary modified to contain information on the image |
| MaskDict | dictionary | No | Reference to a type 1 dictionary modified to contain masking information |
| InterleaveType | integer | No | Code that identifies the kind of organization of the image and of the mask. Possible values: 1. Image and mask samples interleaved by sample and contained in the same data source 2. Image and mask samples interleaved by row and contained in the same data source 3. Image and mask contained in different data sources |

**Table 2.18** Type 4 dictionary parameters for PostScript

| Parameter | Type | Optional | Value |
|---|---|---|---|
| ImageType | Integer | No | 4 (code identifying the dictionary type) |
| MaskColor | array | No | Integers that specify the colors to be masked; its size varies from $n$ to $2n$ values, where $n$ is the number of components per color |
| Width | integer | No | Width of the image in samples |
| Height | integer | No | Height of the image in samples |
| ImageMatrix | array | No | 6 numbers that define the transformation from user space to image space |
| MultipleDataSources | boolean | Yes | Specifies whether the source is single or multiple |
| DataSource | (various) | No | The source from which the data stream is to be drawn. If MultipleDataSources is true, it is an array with as many components as in the color space, otherwise it is a single file, procedure or string |
| BitsPerComponent | integer | No | Number of bits used to represent each color component. Allowed values are 1, 2, 4, 8 and 12 |
| Decode | array | No | Values that describe how the sample image is to be mapped in the range of values of the proper color space |
| Interpolate | boolean | Yes | Presence of interpolation in the image visualization |

rameter **MaskColor** that denotes the range of colors to be masked. The parameters for Type 4 dictionaries are reported in Table 2.18.

**PDF (Portable Document Format)** *PDF* [14] is an evolution of PS developed by Adobe, and in some respects represents a slang thereof. However, differently from PS, it is a document presentation format rather than a programming language. This means that there is no need for an interpreter to be able to write and display documents, but it is sufficient to read the descriptions included in the PDF file itself. Its specifications are public domain; it is compatible to any printer, flexible (it allows character replacement and the insertion of links, bookmarks and notes) and readable in third-party applications through suitable plug-ins. Differently from PS, text displayed on screen is internally represented using character codes, and hence can be copied and pasted in other applications. Thanks also to this, the size of a PDF document is much smaller than the corresponding PS counterpart (often about 1/10th of it).

PDF provides for color space components that use 1, 2, 4, 8 or 16 bits (i.e., 2, 4, 16, 256 or 65536 values, respectively), which is another difference with respect to PS. To represent a sampled image, it is necessary to provide in the dictionary of operator `Xobject` the parameters height, width, number of bits per component and color space (from which the number of components needed is inferred). Color components are interleaved by sample (e.g., in the case of the DeviceRGB color space, the three components—red, green and blue—of a sample are followed by the three components of the next sample in the same order, and so on).

Each image has its own coordinate system, called *image space*, similar to that of PS (the image is split in a grid of $h \times w$ samples, whose Cartesian system has the $x$ axis oriented towards the right and the $y$ axis up), but with the origin in the top-left corner. The scan of samples in the grid starts from the origin and goes on horizontally, linewise. The correspondence between user space and image space is fixed and can be described by the matrix

$$\begin{bmatrix} \frac{1}{w} & 0 & 0 & -\frac{1}{h} & 0 & 1 \end{bmatrix},$$

where the user space coordinate $(0, 0)$ corresponds to the image space coordinate $(0, h)$. The code to modify the transformation matrix is enclosed between the `q` and `Q` commands, that save and restore the image graphic state, respectively; command `cm` modifies the matrix; command `Do` draws the image on the device.

*Example 2.9* Sample code to draw an image ‘sampleimage’:

| | |
|---|---|
| `q` | saves the graphic state |
| `  1 0 0 1 100 200 cm` | translation |
| `  0.707 0.707 -0.707 0.707 0 0 cm` | rotation |
| `  150 0 0 80 0 0 cm` | scaling |
| `  /sampleimage Do` | image drawing |
| `Q` | graphic state restoration |

PDF provides two methods of different expressive power to describe and handle images:

**XObject** is used for any kind of image. In `Xobject` images, the data stream is made up of two parts: a dictionary (*Image Dictionary*) containing image-related information and a data container that encloses the image source data. The dictionary contains approximately 20 parameters, by which all the features needed to display the image on a device can be handled. Many parameters are inter-related, and their values must be consistent not to raise an error. Different operators could handle it in different ways: Table 2.19 reports some of the main parameters, in the form suitable for use by the `Do` operator.

*Example 2.10* Sample code for an `Xobject` image:

```
...
22 0 obj
<< /Type /XObject
  /SubType /Image
  /Width 256
  /Height 256
  /ColorSpace /DeviceGray
  /BitPerComponent 8
  /Length 83183
>>
stream
 aclkjlkj5kjlcilÃ ixinxosaxjhaf ...   | source data stream (65536 samples)
endstream
endobj
...
```

**Inline** allows handling only small-sized images, and applies restrictions to their properties. In an `Inline` object, all information needed to describe the image is embedded in a single data stream. It allows including, before the image source data stream, a short definition of the way in which the stream is to be interpreted (rather than creating an object, such as a dictionary, that embeds it). The object structure is defined by delimitation operators in the following form:

| | |
|---|---|
| `BI` | beginning of the Inline object |
| `...` | parameters definition |
| `ID` | beginning of the source data stream |
| `...` | image source data stream |
| `EI` | end of the Inline object |

Nested `BI`s and `EI`s are allowed, while `ID` must appear only between `BI` and `EI`, with at most a blank after it so that the first following character is the first character of the stream. The parameters that can be used between `BI` and `EI` are almost all the same as for XObject, and show different abbreviations and syntax, as reported in Table 2.20.

**Table 2.19** Some PDF image dictionary parameters, in the form required by `Do`

| Parameter | Type | Optional | Value |
|---|---|---|---|
| Type | name | Yes | Type of object this dictionary refers to (in this case, XObject) |
| Subtype | name | No | Type of XObject described by this dictionary (in this case, Image) |
| Width | integer | No | Width in samples |
| Height | integer | No | Height in samples |
| ColorSpace | name or array | No | Type of color space used |
| BitsPerComponent | integer | No | Number of bits used per color component |
| Intent | name | Yes | A color name to be used for the image rendering intent |
| ImageMask | boolean | Yes | Indicates whether the image is a mask. If it is true, BitsPerComponent must be equal to 1 and Mask and Colorspace must not appear |
| Mask | stream or array | Yes | If it is a stream, the mask to be applied to the image; if it is an array, the range of colors to be masked |
| Decode | array | Yes | Numbers that describe (using the same formula as in PS) how to map the image samples in the range of values suitable for the defined color space |
| Interpolate | boolean | Yes | Presence of interpolation |
| Alternates | array | Yes | Alternate dictionaries that can be used for the image |
| Name | name | No | Name by which the image to which the dictionary refers is referenced in the subdictionary XObject |
| Metadata | stream | Yes | A data stream that can be used to include further textual information |
| OC | dictionary | Yes | A further group of dictionaries in case additional information is needed to describe the image |

*Example 2.11* Sample code for an `Inline` image:

```
q                                  | save graphic state
BI                                 | beginning of Inline object
  /W 256                           | width in samples
  /H 256                           | height in samples
  /CS /RGB                         | color space
  /BPC 8                           | bits per component
  /F [/A85 /LZW]                   | filters
ID                                 | beginning of data stream
... slckiu7jncso8nlssjo98ciks ...  | data stream
EI                                 | end of Inline object
Q                                  | restore graphic state
```

**Table 2.20** PDF abbreviations synopsis

| BI–EI operators parameters | | Color spaces | |
|---|---|---|---|
| Internal name | Abbreviation | Internal name | Abbreviation |
| BitPerComponent | BPC | DeviceGray | G |
| ColorSpace | CS | DeviceRGB | RGB |
| Decode | D | DeviceCMYK | CMYK |
| DecodeParms | DP | Indexed | I |
| Filter | F | ASCIIHexDecode | AHx |
| Height | H | ASCII85Decode | A85 |
| ImageMask | IM | LZWDecode | LZW |
| Intent | none | FlateDecode | Fl |
| Interpolate | I | RunLengthDecode | RL |
| Width | W | CCITTFaxDecode | CCF |
| | | DCTDecode | DCT |

A further way for describing an image in PDF is the **XObject form**. It is a stream that contains a sequence of graphic objects (such as paths, text and images) and can be considered as a model to be exploited several times in different pages of the same document. In addition to document description optimization, it helps in:

- Modeling pages (e.g., forms for bureaucratic documents);
- Describing logos to be displayed on each page background by using a mask to make them semi-transparent;
- Creating a kind of form called **group XObject**, useful to group different graphic elements and handle them as a single object;
- Creating a kind of form called **reference XObject** that can be used to transfer content from a PDF document to another.

## 2.4 Content-Oriented Formats

By the attribute 'content-oriented' we refer to formats that, when defining the document components, specify a set of conceptual properties thereof, instead of determining their position in the document. Thus, the visual aspect that the document gets after displaying is, in some sense, a consequence of its components' properties. Such formats are usually described using declarative languages that allow specifying 'what' is to be obtained, rather than 'how' to obtain it (differently from procedural languages).

Several well-known formats belong to this category. Some that will not be thoroughly discussed in this book are:

**LaTeX** is used to specify the properties of the elements that make up a document from a typographic perspective [21], and is based on the TeX language by

D. Knuth. This format requires a compiler to produce the actual document (typically, final output formats of the compiled version are PS and PDF), but ensures that the rendering is perfectly the same on any hardware/software platform. Different typographic styles can be defined, and the compiler acts as a typographer that, given a plain text manuscript annotated with indications on the role and desired appearance of the various passages, typesets it in order to obtain the most balanced layout possible according to the indicated style.

**DOC** is a binary, proprietary format developed by Microsoft for files produced by the Word word processing program of the Office suite [11]. Latest versions of such a suite exploit a new XML-based, open but patented format called **OOXML** (*Office Open XML*) that since August 15, 2008 has become an ISO standard (ISO/IEC DIS 29500), notwithstanding an open and free ISO standard for the same purposes (the OpenDocument Format, discussed below) already existed since May 1, 2006. In particular, for what concerns word processing, DOC has been replaced by **DOCX**.

**RTF** (*Rich Text Format*) was born for allowing document interchange between different word processors. It was introduced by Microsoft with the aim of creating a standard for formatted text. It provides the same potential as the DOC format, but its specification is—at least in its original version—public domain. Most word processing programs can read and write in this format, but many of them add proprietary extensions, which results in limited portability.

### *2.4.1 Tag-Based Formats*

A comfortable way to specify the properties of a document's components is using *tags* spread along the document itself, that express information concerning the document structure and content within the document itself by means of particular attributes of the components to which they are associated. In general, a very important distinction is made between two kinds of tags:

**Logical** tags that express the role played by the indicated component in the document;

**Physical** tags that directly specify the way in which the component is to be displayed.

Tags of the former kind do not specify any particular visual attribute for the component they denote: it is in charge of the interpreter properly and consistently displaying it, and different interpreters, or even the same interpreter at different times, could adopt different display attributes for a given logical tag. On the other hand, physical tags do not carry any information about the component role. Usually, the former are to be preferred to the latter, at least for a systematic identification of components because this would allow transposing the same document through different styles by just changing the interpreter and/or the attributes underlying the tag. Conversely, the latter can be used occasionally for describing how specific components are to be rendered, independently of the particular style in use.

The category of *tag-based* formats essentially includes two well-known Web-oriented languages: HTML and XML. Thousands of pages and many professional works have been written about these formats, and hence it would be pretentious (and out of our scope) to give here a complete presentation thereof. Nevertheless, a quick overview is useful to introduce their approach and compare it to those of the other formats presented in this book.

**HTML (HyperText Markup Language)** A *hypertext* is a set of texts interconnected by links that allow jumping from a given place in a document to (a specific place of) another document. The *World Wide Web* (*WWW*) is the most widespread and famous example of hypertext nowadays. *HTML* [3] is a tag-based language developed to support the creation and presentation of hypertextual information (such as Web documents) in a universal and hardware-independent way, and to allow the definition of a simple and straightforward interface for 'browsing' among hypermedia documents, with the only help of a pointing device (e.g., a mouse). It is an application of the *SGML* (*Structured Generalized Markup Language*, used in the editorial field), and represents the current standard for Web page description, whose definition is fixed by the W3C.[12]

HTML allows defining the format of a document (size, type and style of the characters, kind of text justification, etc.) and including hyperlinks to other documents or to multimedia objects (such as images, icons, videos, sounds) or even to other Internet services (FTP, Gopher, etc.), still keeping the (sometimes complex) commands underlying those links transparent to the user. Its main strengths lie in the possibility of including images in the documents and in its universality, flexibility, richness and compactness.

An HTML file is encoded in standard ASCII, that can be easily handled by any text editor (although dedicated editors improve readability). Special characters, reserved to HTML (<, >, `&`) or not provided by the ASCII, are expressed as *escape sequences* (sequences of characters that are displayed in the document as a single symbol), starting with an ampersand `&` and ending with an (optional) semi-colon `;`. Their conversion is the task of the interpreter. A sample of escape sequences is reported in Table 2.21.

The visual presentation of the document is driven by structural and stylistic indications provided by specifying content- and formatting-related *directives* to an interpreter. Directives are interleaved with the text that represents the actual content, delimited by brackets (< and >) to be distinguished from it, and are made up of a code (called *tag*, a term sometimes improperly used to denote the whole directive), possibly followed by the specification of *attributes* (some are mandatory, some optional, depending on the particular tag) that modify their effect:

```
<TAG attribute="value" ... attribute="value">
```

[12]Some software applications that produce HTML documents exploit extensions not included in the official format definition, and hence part of their output represents semi-proprietary code that might not be properly displayed on some platforms.

**Table 2.21** Some special characters and their escape sequences in HTML

| Symbol | Escape | Note | Symbol | Escape | Symbol | Escape |
|---|---|---|---|---|---|---|
| < | &lt; | less than | À | &Agrave; | à | &agrave; |
| > | &gt; | greater than | É | &Eacute; | é | &eacute; |
| & | &amp; | ampersand | È | &Egrave; | è | &egrave; |
| © | &copy; | copyright | Ì | &Igrave; | ì | &igrave; |
| ... | | | Ò | &Ograve; | ò | &ograve; |
| | | | Ù | &Ugrave; | ù | &ugrave; |

Not all tags are supported by all browsers, but unknown ones are simply ignored instead of raising an error. Commands are not case-sensitive (although, to improve human readability, a widespread and useful agreement is to write tags in upper-case and attributes in lower-case). Some tags denote the presence of a single element (e.g., a line break or an image); more often they apply to a piece of content, and hence require a corresponding directive that indicates the end of their scope. In the following, we will call the former *single* tags, and the latter *paired* tags. The closing directive of a paired tag (that can be sometimes omitted, being implicitly assumed by the interpreter when certain other directives are found) is denoted by a slash `/` before the tag:

`... <TAG attribute="value" ... >` content affected by the tag `</TAG> ...`

Attributes, if any, must be specified in the opening directive only. Comments, ignored by the interpreter, can be included as follows:

<!– text of the comment ->

The high-level structure of an HTML document is as follows:

```
<HTML>          beginning of the document
  <HEAD>        beginning of the heading
  ...           meta-information
  </HEAD>       end of the heading
  <BODY>        beginning of the actual content
  ...           actual content
  </BODY>       end of the actual content
</HTML>         end of the document
```

where the tag `HTML` delimits the document specification, the tag `HEAD` delimits a heading (that expresses general information on the document that is not to be displayed), and the tag `BODY` delimits the actual hypertextual content of the document (that is to be displayed). Some possible attributes for the `BODY` tag are: `background` (to specify a background image for the page), `bgcolor` (to specify a background color for the page), etc.

The heading typically contains a document title (paired tag `TITLE`, in which spacings are significant), useful for identifying it in other contexts, that is usually displayed in the window title bar. Additional meta-information on the document can be specified using the single tag `META`, in the form:

```
<META name="..." content="...">
```

where the `name` attribute denotes the kind of meta-information (typical values are: **generator**, the software used to produce the HTML code; **author**, the author of the document; **keywords**, a list of keywords related to the document content; **description**, a textual description of the content) and `content` specifies the actual meta-information.

The main elements that can be included in the body are: titles, lists (possibly nested at will), images, sounds, videos and Java programs (*applets*), hyperlinks, character-formatting styles. It should be noted that the interpreter generally ignores carriage returns, and considers tabbings or multiple spaces as a single blank. Thus, new lines and spacings in the source file are exploited just for improving human readability of the code. To prevent the document from appearing as a single paragraph, suitable tags that cause layout formatting must be used, such as text splitting ones:

- Line breaks (*Break Rule*), `BR`;
- Horizontal line separators (*Hard Rule*), `HR`;
- Paragraph boundaries (using the paired tag `P` or also the single directive `<P>` that displays a vertical space between the content before and after it).

The only exception is the *preformatted* text, delimited by the paired tag `PRE`, that is displayed with fixed size characters, in which spaces, new lines an tabbings are significant (it is useful for displaying program listings). However, HTML tags, hypertextual references and links to other documents can still be used in its scope.

Images are enclosed using the directive

```
<IMG src="filename">
```

where the attribute `src` specifies the name and path of the image file. Supported formats are *XBM* (*X BitMap*), GIF, JPEG. Vertical alignment of the image with respect to the surrounding text can be specified using the `valign` attribute (allowed values are **top** = upper border, **middle** = centered, **bottom** = lower border, that is the default). Using the optional `height` and `width` attributes, one or both display dimensions of the image can be independently resized (if just one is specified, the other is consequently set so to preserve the original ratio), in pixels or as a percentage of the browser window (by specifying the `%` symbol after the value). The (optional) attribute `alt` allows specifying an *alternate* textual description of the image, to be displayed when the mouse passes over it or by textual browsers that cannot display images.

HTML allows applying special character-formatting styles to pieces of text. The ‘physical style’ is defined by the user and shown by the browser, while ‘logical styles’ are configured by the interpreter. Both exploit paired tags. A scheme of the available tags for physical styles, along with the typical corresponding logical styles (and tags), available in HTML is provided in Table 2.22.

**Table 2.22** Logical and physical character-formatting style tags in HTML

| Physical style | **Italics** | **Bold** | **Underlined** | **TeleType**-like |
|---|---|---|---|---|
| Tag | `I` | `B` | `U` | `TT` (fixed-size font characters) |
| Corresponding Logical styles | `EM` *emphasize* | `STRONG` *note* | | `CODE` for program *code* |
| | `CITE` *citation* | | | `SAMP` *sample* (for examples) |
| | `VAR` *variable* | | | `KBD` *keyboard* (for names of keys) |
| | `DFN` *definition* | | | |

Section headings are identified by the family of tags `H`$n$, where $n$ specifies the depth level (and hence the character size) in a 1 to 6 scale. Level 1 headings are the most important, usually displayed with larger characters in bold face. In complete documents, they usually express the document title, while in documents that represent sections of wider works (e.g., chapters of a book), they should be referred to the content of that particular section, using the `TITLE` tag in the heading to provide the structural reference (e.g., the title of the whole book plus that of the chapter).

As to lists, the following kinds are supported (each followed by the associated paired tag):

**Unordered List** `UL`
**Ordered List** `OL`
**Description List** `DL`

Elements of ordered or unordered lists are identified by the paired `LI` tag (*list item*), while in description lists the title of the item is identified by the paired tag `DT` (*Description Title*) and its description by the paired tag `DD` (*Description Description*). All item tags are allowed also as single tags. A paragraph separator between list elements is not needed, but each item may contain multiple paragraphs, lists or other descriptive information. Hence, an arbitrary nesting can be specified. Unordered lists are displayed differently by different browsers, as regards indentation and symbols for each level (typically a dot for the first level and a square for the second).

Tables are specified row-wise, according to the following structure:

```
<TABLE>
  <TR> ... </TR>
  ...
  <TR> ... </TR>
</TABLE>
```

where possible attributes of the `TABLE` tag are `border` (thickness of the line to draw the table grid), `cellspacing` (internal margin for the cell content), `cellpadding` (spacing among cells), `width`. Each *table row* tag `TR` delimits the single cells in a row, specified using the paired tags `TH` (for *heading* ones, to be highlighted) or `TD` (for content, or *description*, ones). `TR`, `TH` and `TD` can specify the horizontal or vertical alignment of the cell content using the attributes `align` (values **left**, **right**, **center**) and `valign` (**top**, **middle**, **bottom**), respectively. `TH` and `TD` additionally provide attributes `nowrap` (automatic wrapping of the contained text), `rowspan` and `colspan` (to span a cell over several rows or columns, respectively). The paired tag `CAPTION` allows specifing an explanation, with attribute `align` (possible values are **top**, **bottom**, **left**, **right**).

Hypertextual links are regions of a document that, when selected, take the user to another document that can be on the same computer or on a different one. They typically appear highlighted (usually colored and underlined, but this setting can be changed), and are specified by delimiting the region with an *anchor* paired tag

```
<A href="..."> ... </A>
```

where the `href` attribute represents the (relative or absolute) URL of the linked document. It is also possible to specify anchors to specific document sections, useful for moving directly to intermediate passages. The target place (*named anchor*) must be labeled using the (paired or single) directive

```
<A name="reference">
```

that defines an identifier for it, to be exploited as a suffix for the URL in the links, as follows:

```
<A href="filename.html#reference"> ... </A>
```

(the file name being optional if the target document is the same as the starting one).

The paired tag `ADDRESS` specifies the author of a document and a way to contact him (usually an e-mail address). It is typically the last directive in the file, whose content is placed on a new line, left-justified.

Colors are expressed in RGB format, as a sequence #RRGGBB of pairs of hexadecimal digits for each basic color (RR = red, GG = green, BB = blue), or using symbolic names for some predefined colors: **black** (#000000), **silver** (#C0C0C0), **gray** (#808080), **white** (#FFFFFF), **maroon** (#800000), **green** (#008000), **lime** (#00FF00), **olive** (#808000), **yellow** (#FFFF00), **navy** (#000080).

An interesting feature is the possibility of splitting a single window into *frames*, a sort of tables each of whose cells can contain a separate document. This is obtained by the paired tag `FRAMESET`, with attributes `rows` and `cols` to specify the window splitting as a comma-separated list of values (in pixels or, if followed by `%`, in percentage of the window size; a `*` denotes all the available space). Other allowed attributes are: `src` (the document to be displayed in the cell), `name` (an identifier for referencing the frame), `scrolling` (with values **yes**, **no** and **auto**, to allow scrolling of the content of a cell), `noresize` (to prevent window resizing from the user), `marginwidth` and `marginheight` (to define the spacing among cells). A `NOFRAME` tag can be used to specify what to display in case the browser does not support frames.

*Example 2.12* A sample specification of frames in an HTML document.

```
<FRAMESET rows="25%, 75%">
  <FRAME noresize name="top" src="t.html" scrolling="no">
  <FRAMESET cols="150,*,150">
    <FRAME name="left" src="l.html">
```

```
      <FRAME name="center" src="c.html" scrolling="yes">
      <FRAME name="right" src="r.html" scrolling="auto">
    </FRAMESET>
  </FRAMESET>
```

The window is vertically divided in two rows, taking 1/4 (25%) and 3/4 (75%) of the whole allowed height, respectively. The first row contains a frame named ‘top’, in which the source HTML is loaded from file ‘t.html’, and displayed so that scrolling is not permitted (scrolling widgets are not shown), even in the case the source is too large to fit the available frame space. The second row, in turn, is horizontally split into three frames (columns), named ‘left’, ‘center’ and ‘right’, respectively, of which the first and the last one take exactly 150 pixels, while the middle one takes all the remaining space (depending on the overall window width). These columns contain the hypertexts whose sources are to be loaded from files ‘l.html’. ‘c.html’ and ‘r.html’, respectively, of which the second one always shows scrolling widgets, while the last one shows them only if the source content exceeds the available frame space.

**XML (eXtensible Markup Language)** Developed by W3C for use in the Internet, in order to make more flexible and personalizable HTML, *XML* subsequently established also as the main language for information representation. It was designed for ease of implementation and for interoperability with both SGML and HTML [9, 13]. Indeed, it provides a shared syntax that can be understood by machines, allows homogeneous processing and enforces information storing, handling and interchange. It is acceptable by human users as well, since it provides the generic blocks for building a language, according to a comfortable tag-based syntax (just like HTML). It is a subset of SGML,[13] focused on documents and data, in which no keyword is pre-defined (as in HTML), but a set of rules is provided according to which other languages (e.g., HTML) can be written. Of course, this leaves the user with the problem of choosing the tags and, if needed, with defining their intended meaning and behavior.

The structure of an XML document requires, in the first row of the file, a declaration concerning the language version (mandatory) and the character encoding used (UTF-8 by default):

```
<?xml version="..." encoding="..."?>
```

followed by the actual content.

The content is made up of *elements*, each delimited by directives in the same formalism as in HTML, but denoted using user-defined tags. Elements (and hence directives) can be nested, but a *root* element that contains all the others and defines the type of XML document is mandatory. The nesting order is relevant, and represents a good indicator for the choice of elements or attributes. Directives can even be empty (`<tag></tag>`, often abbreviated in `<tag />`).

[13]Note that HTML is an *application*, not a *subset*, of SGML.

As in HTML, tag effects can be modified by using *attributes* whose values are delimited by single or double quotes. Attribute values cannot be empty, and only text is allowed therein. They are often exploited for specifying metadata that express properties of the elements, and their ordering is not significant to XML (although it can be relevant to the applications that will process the file).

An *entity* is any unit of content, started by `&` and ended by `;` (like escape sequences in HTML). Some entities are pre-defined (*escape sequences*):

- `lt` (<),
- `gt` (>),
- `apos` (′),
- `quot` (″),
- `amp` (&);

others are used as shortcuts for human convenience. References to characters are represented by UNICODE code points *n*, expressed in decimal (`&#`*n*`;`) or hexadecimal (`&#x`*n*`;`) notation.

Comments are as in HTML:

`<!--` text of the comment `-->`

and cannot contain the `--` sequence of characters. They are exclusively for human use (differently from HTML, where instructions for applications can be enclosed), because other specific tools are provided to contain information intended for machines. For this reason, the use of comments must be avoided in machine-intended documents.

As to syntax, tag and attribute names are case sensitive, must start with a letter or an underscore and may continue with any mix of letters, digits, underscores, dots (not colons, that are reserved for namespaces, to be discussed below). Some names (those starting with `xml`) are reserved. Well-formed documents require that elements are correctly nested and that all open tags are closed. Checking the validity of an XML document consists in a check for admissibility of well-formed expressions, and is useful for machine-readability purposes.

*Namespaces* are used for disambiguation of identical names used in different contexts, according to the concept of *URI* (*Uniform Resource Identifier*). A namespace is denoted by a name, and defines a *vocabulary* whose elements are referred to a specific context and are independent from those in other namespaces. An XML document can specify which namespaces are going to be exploited: then, elements and attributes having the same name but coming from different contexts can be used together in that document by prefixing each of them with the intended namespace, separated by a colon: `namespace:element` and `namespace:attribute`, respectively. Namespaces are themselves URIs of the vocabulary `xmlns:namespace`.

*Example 2.13* A short XML document:

```
<?xml version="1.0" encoding="us-ascii"?>
<countries>
```

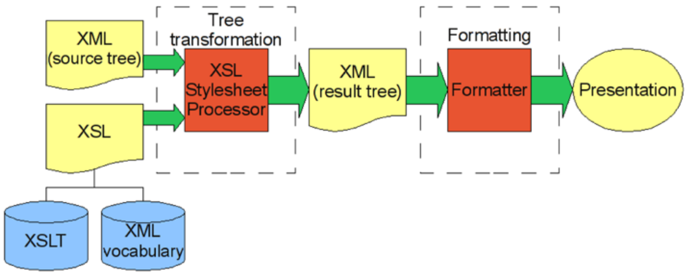


**Fig. 2.6** XSL transformation of an XML document

```
    <country id="c01">
      <name>Italy</name>
      <abbr>ITA</abbr>
    </country>
    <country id="c02">
      <name>United States of America</name>
      <abbr>USA</abbr>
    </country>
    <country id="boh"/>
  </countries>
```

In addition to the abstract syntactic correctness, some kind of semantic correctness may be required to the used tags and attributes and to their nesting. The types of elements and attributes allowed in an XML document can be expressed by means of *DTD* (*Document Type Definition*) files. An XML document that must fulfill a DTD can reference it by means of a `<!DOCTYPE ...>` declaration placed in the second line of the file. It indicates which is the root element of the XML document (that can even be different from the root of the DTD). If a DTD is specified, the elements in the XML document must be instances of the types expressed in the DTD. Specific procedures are in charge of processing the document with respect to its declared DTD, and of rejecting invalid documents.

*XSL* (*eXtensible Stylesheet Language*) is a language that allows specifying styles for displaying XML documents [10]. In this way, the same XML file can be presented in different ways: as a Web page, a printable page, speech, etc. The transformation takes place in two steps, as depicted in Fig. 2.6:

1. **Tree transformation**, carried out on an XML document (that represents the *source tree*) and an XSL document by an *XSL Stylesheet processor* that produces a *result tree*;
2. **Formatting**, carried out on the result tree by a *formatter* that produces the final presentation.

The XSL exploits an *XSLT* (*XSL Transformations*, defined by the W3C [5]) language for transforming XML documents and an XML vocabulary for specifying formatting semantics. The former describes how the document is transformed into another XML document that uses the latter. The result tree is a new representation of the XML source that can be used by the formatter for producing the final presentation.

XML processing tools are parsers that take an XML file as input and return information related to them. They fall in two categories: *event-based* and the *DOM* (*Document Object Model*—see Sect. 5.1.2). The former act by means of calls to APIs (program functions) any time a parsing is involved, or by means of a standard API library called SAX (available in Java, Python, Perl). The latter, whose standard is defined by the W3C, load the XML document content into main memory in the form of a tree. Three levels of DOM exist, each including several specifications for different aspects thereof. XSLT processors define transformations of an XML document into another XML or HTML document. For instance, *Xpath* [4] is a very flexible query language for addressing parts of an XML document, designed to be used by XSLT but used also to run processing functions.

### *2.4.2 Office Formats*

The most straightforward and classical way for producing a natively digital document is using a Word Processor, usually included in a suite of office-related programs. Several software packages of this kind have been developed over the years by various software houses. Up to recently, each of them used to save the documents in a different proprietary format, often binary. This might not be a problem while the very same version of the application is used, but becomes a strong limitation as soon as one needs to manipulate the document content, directly or using another program, in order to carry out operations that are not provided by the original application used for producing them. The only way out in these cases is to pass through intermediate, neutral or 'universal' formats, such as RTF or *CSV* (*Comma-Separated Values*). However, just because of them being neutral, they are often not able to preserve all the original settings of the document. For this reason, many Public Administrations have recently required their applications to adopt open formats and save the documents as pure text (typically XML), so that the content can be accessed independently of the visualization application.

Today only two office application suites are widely exploited and compete on the market: the Microsoft Office and OpenOffice.org suites. While the former has a long tradition, the latter is much more recent, but has gained large consensus in the community thanks to its being free, open-source, portable, rapidly improving its features and performance, and based on an open, free and ISO-standardized format. Conversely, the former is commercial and based on proprietary formats that only recently have been accepted as an ISO standard. For these reasons, here we will focus on the latter.

**ODF (OpenDocument Format)** The *OpenDocument* format (*ODF*) [12, 18] was developed for the office applications suite OpenOffice.org whose aim is to provide users with the same applications on any hardware and software platform, and to make them able to access any data and functionality by means of open formats and components that are based on the XML language and on API technology, respectively. The OASIS (Organization for the Advancement of Structured Information Standards) decided to adopt this format and to further develop and standardize it, resulting in the definition as an ISO standard (ISO 26300) since 2006.

The OpenDocument format is an idealized representation of a document structure, which makes it easily exploitable, adaptable and extensible. An OpenDocument consists of a set of files and directories that contain information on the settings and content of the document itself, saved altogether in a compressed file in ZIP format.[14] A document generated by the OpenOffice.org word processor (*Writer*), identified by the *ODT* acronym (for OpenDocument Text) is made up of the following elements:

- `mimetype` a file containing a single row that reports the MIME content type of the document;
- `content.xml` the actual document content;
- `styles.xml` information on the styles used by the document content (that have been separated for enhanced flexibility);
- `meta.xml` meta-information on the content (such as author, modification date, etc.);
- `settings.html` setting information that is specific to the application (window size, zoom rate, etc.);
- `META-INF/manifest.xml` list of all other files contained in the ZIP archive, needed by the application to be able to identify (and read) the document;
- `Configurations2` a folder;
- `Pictures` a folder that contains all the images that are present in the document (it can be empty or even missing).

It should be noted that the XML code is often placed on a single row to save space by avoiding line break characters. The only mandatory files are 'content.xml' and 'manifest.xml'. In principle, it is possible to create them manually and to properly insert them in a ZIP file, and the resulting document could be read as an OpenDocument (although, of course, it would be seen as pure text and would not report any further style, setting or other kind of information).

## References

1. Graphics Interchange Format (sm) specification—version 89a. Tech. rep., Compuserve Inc. (1990)
2. TIFF specification—revision 6.0. Tech. rep., Adobe Systems Incorporated (1992)
3. HTML 4.01 specification—W3C recommendation. Tech. rep., W3C (1999)
4. XML Path Language (XPath) 1.0—W3C recommendation. Tech. rep., W3C (1999)

[14]This representation is inspired by JAR files (*Java ARchives*), used by the Java programming language to save applications.

5. Transformations, X.S.L.: (XSLT) 1.0—W3C recommendation. Tech. rep., W3C (1999)
6. International standard ISO/IEC 10646: Information technology—Universal Multiple-octet coded Character Set (UCS). Tech. rep., ISO/IEC (2003)
7. Portable Network Graphics (PNG) specification, 2nd edn.—W3C recommendation. Tech. rep., W3C (2003)
8. Lizardtech djvu reference—version 3. Tech. rep., Lizardtech, A Celartem Company (2005)
9. Extensible Markup Language (XML) 1.1, 2nd edn.—W3C recommendation. Tech. rep., W3C (2006)
10. Extensible Stylesheet Language (XSL) 1.1—W3C recommendation. Tech. rep., W3C (2006)
11. Microsoft Office Word 97–2007 binary file format specification [*.doc]. Tech. rep., Microsoft Corporation (2007)
12. Open Document Format for office applications (OpenDocument) v1.1—OASIS standard. Tech. rep., OASIS (2007)
13. Extensible Markup Language (XML) 1.0, 5th edn.—W3C recommendation. Tech. rep., W3C (2008)
14. Adobe Systems Incorporated: PDF Reference—Adobe Portable Document Format Version 1.3, 2nd edn. Addison-Wesley, Reading (2000)
15. International Telegraph and Telephone Consultative Committee (CCITT): Recommendation t.81. Tech. rep., International Telecommunication Union (ITU) (92)
16. Deutsch, P.: Deflate compressed data format specification 1.3. Tech. rep. RFC1951 (1996)
17. Deutsch, P., Gailly, J.L.: Zlib compressed data format specification 3.3. Tech. rep. RFC1950 (1996)
18. Eisenberg, J.: OASIS OpenDocument Essentials—Using OASIS OpenDocument XML. Friends of OpenDocument (2005)
19. Hamilton, E.: JPEG file interchange format—version 1.2. Tech. rep. (1992)
20. Huffman, D.: A method for the construction of minimum-redundancy codes. In: Proceedings of the I.R.E, pp. 1098–1102 (1952)
21. Lamport, L.: $\LaTeX$, A Document Preparation System—User's Guide and Reference Manual, 2nd edn. Addison-Wesley, Reading (1994)
22. Reid, G.: Thinking in PostScript. Addison-Wesley, Reading (1990)
23. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press, Champaign (1949)
24. The Unicode Consortium: The Unicode Standard, Version 5.0, 5th edn. Addison-Wesley, Reading (2006)
25. W3C SVG Working Group: Scalable Vector Graphics (SVG) 1.1 specification. Tech. rep., W3C (2003)
26. Welch, T.: A technique for high-performance data compression. IEEE Computer **17**(6), 8–19 (1984)
27. Wood, L.: Programming the Web: The W3C DOM specification. IEEE Internet Computing **3**(1), 48–54 (1999)
28. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23**(3), 337–343 (1977)
29. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory **24**(5), 530–536 (1978)

# Chapter 3
# Legal and Security Aspects

A *digital document* is a kind of electronic document in which information is described as a sequence of binary values obtained by exploiting electronic devices and processing techniques of various kind. Examples are digitization by means of an acquisition device (scanner) or direct creation by means of an office-automation application program. The binary values can express just the appearance of the document (as in the case of scanned documents), or its content (as in the case of a word-processor output). In any case, the document can be seen as a whole sequence of bits, independently of their specific meaning in the document representation (that depends on the format and standard used), and hence, in some sense, as a (very large) integer value.

Digital documents, by their nature, have different characteristics than classical paper ones, among which the most relevant and well-known are flexibility, easy cloning and easy transmission (intended as ease of sending a document from a source to a target, through some kind of channel). These features, although in general can be deemed as desirable, pose severe security problems as soon as personal privacy or public validity are taken into account. Indeed, a document might be reserved to selected and authorized readers only, or its content could be required to be sufficiently guaranteed from manipulations in order to act as a proof from a juridic viewpoint or in the Public Administration. In these cases, the possibility of easily creating, modifying and cloning digital documents has ever since seriously affected their validity and relevance for exploitation in formal environments. Even the easy transmission raises the problem of possible interception of confidential documents and their fraudulent modification by malicious persons. Actually, both issues are not specific to digital documents, but have been ever-since a problem (just think of military communications during a war) and, because of this, several methods for secure data transmission (and corresponding techniques to break them) have been developed since ancient times. Thus, a choice must be made between abandoning digital documents (and their advantages) in formal environments or undertaking an effort to find a solution for their weaknesses. Since the presence and spread of digital documents nowadays cannot be ignored or limited, the latter option is nearly mandatory, and hence the study of how to secure digital content and its transmission has become a hot topic in the last years [27, 32].

## 3.1 Cryptography

The need for hiding the content of a message to other persons has ever since been felt by people, for military reasons but also just for privacy. In a nutshell, the problem involves three individuals **A**, **B** and **E** (often represented by the names Alice, Bob and Eve by practitioners), and can be expressed as follows: Alice wants to send a secret message $M$ to Bob, with Eve being an enemy that wants to read the message. Thus, a study of methodologies for obtaining such a result has been carried out through the centuries [30]. Due to their straightforward exploitation, and to the undeniable advantage they can bring, in war environments (an event considered by many historians as determinant for the Allies to win World War II was the break of the German secret communication system called Enigma), in the US these methodologies are considered in all respects as war weapons, and hence their exploitation, circulation and export are strictly ruled by law and strongly limited.

### *3.1.1 Basics*

Before proceeding any further and going into technical details, a vocabulary of fundamental terms must be provided. The usual term to denote the science of inventing systems that can make a message understandable to its legitimate addressee only, and incomprehensible to whoever else comes into possession of it, is *cryptography* (from the Greek 'kryptós' = to hide and 'gráphein' = to write), defined as "the enciphering and deciphering of messages in secret code or cipher" (1658). Conversely, *cryptanalysis* denotes "the theory of solving cryptograms or cryptographic systems; the art of devising methods for this" (1923), i.e., the art of breaking such systems. Lastly, *cryptology* is "the scientific study of cryptography and cryptanalysis" (1935), and hence encompasses both [5].

The original message, i.e., the sequence of letters (or, more generally, *symbols*) that are to be securely transmitted, is called *plaintext*. The operation of 'obscuring' the plaintext, i.e., transforming it in such a way that only selected persons can read and understand its content, is named *encoding* or *encryption*. It is based on some kind of *code*, and is performed using an *algorithm* (or *method*), that is the set of operations that allow performing encryption, and back. The *key* is the specific word, or sentence, "that allows us, through the application of the algorithm, to perform encryption", i.e., the code that allows activating the algorithm. Thus, if the algorithm is the computation procedure, the key is the way the algorithm operates in a particular situation. The 'obscured' message, i.e., the sequence of symbols/letters that are output by the algorithm and will be actually transmitted, is called *ciphertext* or *cryptogram*. The algorithm also allows performing the opposite transformation, from the ciphertext to the plaintext; such an inverse process is called *decoding* if it is carried out by the legitimate addressee, or *decryption* if it is carried out by another person (the *enemy*) that "tries to violate the secrecy of someone else's message". More precisely, this term indicates "someone who does not own the key of a cipher, but fraudulently tries to know the content of a ciphertext using indirect systems". A *cipher* (or *cipher system* or *cryptographic system*) is a system made up of a set of plaintexts, the corresponding ciphertexts, the algorithm and the keys.

Two perspectives can be taken on cryptography. The former (often adopted in the military environment) requires, for enhanced security, to keep secret the algorithm. The latter is based on the assumption that the security of a cipher must not depend on the secrecy of the algorithm used, but just on the secrecy of the key (also known as the *Kerkhoffs principle* [24, 25]), which allows the algorithm to be known even to the enemy, and requires just the key to be kept secret. Modern cryptographic systems are based on the latter assumption. In this setting, the system is said to be *strong* or *robust* if it "is based on a large size of the key and on a significant difficulty to invert the encoding key", where the size of the key is given by the number of symbols (i.e., in the case of binary-encoded information, by the number of bits) by which it is made up.

*Transposition ciphers* are cipher systems in which the symbols in the plaintext are the same as in the ciphertext, but re-arranged according to some strategy. Conversely, *substitution ciphers* are those in which each symbol of the plaintext alphabet is replaced by a corresponding symbol of the ciphertext alphabet, according to a regular rule (referred to as the *substitution*). They can be further divided into two categories:

**mono-alphabetic** use a single secret alphabet to encrypt a plaintext, so that each symbol in the plaintext is always transformed into the same symbol of the secret alphabet;

**multi-alphabetic** exploit several secret alphabets to encrypt a single plaintext, so that different occurrences of the same letter in the plaintext are transformed into possibly different symbols.

A fundamental distinction of cryptographic systems is based on the way they operate on the key for performing encryption and decoding. Classical algorithms are *symmetric*, meaning that they carry out both processes using the very same key, which raises the problem of the distribution of keys among the intended users. Indeed, needless to say, whoever comes into possession of the key would be able to decrypt the messages, and hence the key itself turns out to be a secret message to be transmitted. This obviously represents a flaw of the approach, that significantly tampers its reliability, for a two-fold reason: first, spreading several copies of the key implies the risk that it can be stolen from one of the owners; second, personal delivery of the key to those who are to use it could be infeasible (or in any case represent an undesirable overload), and sending the key by means of third-party vectors involves the chance that it is intercepted while being transmitted. Delegation of manual delivery to selected and reliable persons would imply huge costs for personnel (as experienced in past decades by the Departments of Defense of the various countries), even in the case that the meetings for key delivery are made periodically, giving the receiver each time a list of keys, to be used one for each of the messages he will receive in the following period.

Conversely, in *asymmetric* cryptography the encryption and decoding steps exploit different keys when applying the cryptographic algorithm: a *public* key, by which the plaintext is encrypted, and a *private* key, used to go back to the plaintext starting from the ciphertext (for this reason, it is also called 'public key cryptography'). Stealing, intercepting or even legally owning the former is useless to decrypt the message. An idea to obtain such an effect could be the following:

- **A** sends to **B** the message $M$ encrypted using a key $k_A$ (let us call it $M_A$);
- **B** cannot read the message, but encrypts $M_A$ using another key $k_B$ and sends the outcome $M_{AB}$ back to **A**;
- Now, **A** decodes $M_{AB}$ using again $k_A$, and sends the outcome $M_B$ (the original message $M$ now encrypted only by $k_B$) to **B**;
- Finally, **B** can decode $M_B$ using $k_B$ and obtain $M$.

But the cheat would not work since encryption algorithms are typically composed serially, and not in parallel, i.e., applying one on top of the other requires appling the opposite sequence in decoding (*Last In First Out* technique, or *LIFO*). However, the fundamental idea of using two keys, one for the sender and the other for the receiver, is not infeasible. In particular, a feature of bi-directional functions, which makes them useless in solving the above problem, is that it is easy to apply their inverse and obtain the input from the output. Conversely, in uni-directional functions, given the output, it is still very hard to go back to the original input that generated it. Thus, the solution to the problem is to be searched in the domain of the latter.

### 3.1.2 Short History

The Greek historian Polybius (ca. 203–120 B.C.) designed a $5 \times 5$ checkboard (called *Polybius square*) in which the alphabet is arranged so that each symbol corresponds to (and is encrypted as) a pair of integers between 1 and 5, corresponding to the row and column, respectively, of the cell in which the symbol is found. For instance, $\theta$ is encoded as $(2, 3)$:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | $\alpha$ | $\beta$ | $\gamma$ | $\delta$ | $\epsilon$ |
| 2 | $\zeta$ | $\eta$ | $\theta$ | $\iota$ | $\kappa$ |
| 3 | $\lambda$ | $\mu$ | $\nu$ | $\xi$ | o |
| 4 | $\pi$ | $\rho$ | $\sigma$ | $\tau$ | $\upsilon$ |
| 5 | $\phi$ | $\chi$ | $\psi$ | $\omega$ | |

The Roman leader Julius Caesar (100–44 B.C.) developed the so-called *Caesar's cipher* where the algorithm consists in a linear shift of letters, while the key corresponds to the amount of translation. For instance, a key 6 means a translation of 6 positions to the right: A encoded becomes G, and so on:

| Plain | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cipher | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | X | Y | Z | A | B | C | D | E | F |

One of the main techniques for breaking a cipher consists in reasoning about the frequency of the symbols in the ciphertext, and mapping them onto symbols having a similar frequency in the language to obtain the plaintext (some letters, such as vowels, usually have significantly higher frequency than the others; some others, such as the 'q', have significantly lower frequency; etc.). After identifying a few prominent symbols, the others can be obtained by sensibly completing the partial words discovered thus far (e.g., a 'q' is almost invariably followed by a 'u'; 'aqua_iu_' might sensibly be 'aquarium' or 'Aquarius'; etc.). Since the beginning of the fourteenth century, in order to mislead the attempts to apply a statistical analysis of symbol frequencies, some tricks were introduced, such as the use of several symbols in the ciphertext to encrypt the same plaintext symbol (particularly the vowels) and the so-called *nulls* (symbols spread through the ciphertext that do not correspond to any symbol in the plaintext).

In 1883, Kerkhoffs stated six requirements for encryption systems [24, 25]:

1. Being practically, if not mathematically, indecipherable;
2. Being not secret and allowed to even fall in the hands of the enemy without harm;
3. Using keys that do not require written notes to be communicated and retained, and that can be changed and modified at will by the users;
4. Being applicable to telegraphic communications;
5. Being portable, and needing just one person for maintenance or functioning;
6. Being easy to use and not straining the user (e.g., with a long series of rules to be remembered and observed).

Even the Enigma, the famous encoding machine exploited by the Germans during World War II to encrypt and decode secret messages, whose cipher was broken thanks to the contributions of Alan Turing among others, was 'just' an (extremely complicated) version of the approach that consists in associating each symbol in the plaintext to one of the cipher alphabet according to a key.

Computer cryptography gained attention in the 1960s when computers were sufficiently spread in the commercial environment to be exploited for encryption and decoding of reserved business messages and for data transmission. If information was to be exchanged among different organizations, a common agreement was required, whence the motivation for the definition of encryption standards. As to the Kerkhoffs' requirements: 1, 2 and 4 still hold; 3 holds only in part (the key is not written but actually cannot be communicated verbally either); 5 and 6 become meaningless, due to the algorithm being executed by a computer rather than a human.

### *3.1.3 Digital Cryptography*

In digital documents, two levels of representation are present: the original (human-understandable) message and its (computer-readable) encoding as a sequence of binary digits. Because encryption and decoding are to be carried out by computer systems, the latter is more suitable, and hence is considered for automatic application of cryptographic algorithms. Thus, the source and target alphabets are made up of just two symbols (**0** and **1**), which leaves no room for complex encryption (the only possible transformation would be turning each binary digit in its complement). The solution consists in applying the algorithm on sequences of binary symbols, rather than single ones. Since the whole binary representation of the plaintext would be too long to be processed at once, it is usually split into pieces (*blocks*) having a fixed bit-length, each of which can be considered as a number and is encrypted into a corresponding piece of the ciphertext (that can be considered again as a number), and *vice versa*. Except for this, the principles and methods are the same as for normal symbols. The ciphertext consists in the concatenation of the encrypted counterpart of the single blocks (and *vice versa*), that can be seen as a very large binary number. In digital cryptography, keys are numbers as well. The larger the space in which a key can be chosen (i.e., the number of bits that can be used to express it), the harder breaking the system.

In other words, encryption and decoding can be seen as integer functions that transform a number (the plaintext) into another (the ciphertext, or *vice versa*) according to a third number (the key). Specifically, two mathematical concepts turned out to be fundamental for building these techniques:

**modulus** function consisting of the remainder of a division: $x \equiv y \bmod b$ (to be read as "$x$ is *congruent* $y$ *modulus* $b$") means that the remainder of the integer division between $x$ and $b$ is the same as for $y$ and $b$.

**prime numbers** integers that can be divided only by themselves and by 1.
In particular, two numbers are said to be *relatively prime*, or *coprime*, if their greatest common divisor[1] (*gcd*) is equal to 1.

The *modulus* is an example of uni-directional function: given the remainder of an integer division, one cannot know (univocally) the original number that generated it. Thus, an opportunity can be working in *modular arithmetic*, where the results of operations are reduced to their remainder with respect to a given integer, and hence the enemy has no hint from the remainder to guess which number gave it as a result. For instance, exponential functions in modular arithmetics are unidirectional.

The principles underlying asymmetric cryptography have evolved from an attempt to attack two of the most prominent weaknesses in symmetric cryptography: the distribution of the key and the suitability for creating digital signatures (see Sect. 3.3). In this respect, a turning-point in the history of cryptography was marked by the findings of the research team made up by W. Diffie, M. Hellman and

[1]The *greatest common divisor* (*gcd*) of a set of integers is the largest positive integer that divides all of them without yielding a remainder. In the case of two integers $a$ and $b$, denoted $\gcd(a,b)$, a famous algorithm for its computation was invented by Euclid [28]. Since the greatest common divisor of two numbers does not change if the smaller is subtracted from the larger one, the Euclidean algorithm carries out such a subtraction a first time, and then repeats the process on the result and the subtrahend of the last previous operation, until the result is zero. When that occurs, the *gcd* is the last minuend/subtrahend. By reversing the steps, a property known as Bézout's identity states that the *gcd* of two numbers can be expressed as a linear combination thereof with two other integers $x$ and $y$ (where typically either of the two is negative): $\gcd(a,b) = ax + by$.

R. Merkle. Specifically, Diffie was the inventor of the *asymmetric key* cipher, published in 1975 and based on two keys instead of just one [15, 16, 18]. The former, used to decode messages, is called *private* key, and is known only to the receiver; the latter, used for encryption, is called a *public* key, and can be known to anyone. Thus, anybody can encrypt messages and send them to the receiver who is the only one able to decode and read them. Differently from the symmetric approach, no preliminary explicit information exchange is needed. The sender himself is not able to decode his own encrypted message. The problem of finding such a special function that is unidirectional, but can be reversed only using another key, was solved thanks to the research carried out by R. Rivest, A. Shamir and L. Adleman.[2] For ensuring interoperability, a series of *Public Key Cryptography Standards* has been developed and published under the acronym *PKCS #n* (where $n = 1, \dots, 15$). A very famous one, used for digital signatures, is PKCS #7, defined by RFC 2315 [23].

**DES (Data Encryption Standard)** The *DES* [1] is a composite cipher directly derived from a previous very powerful algorithm developed by H. Feistel, called *Lucifer* [31], available at those times. It is a symmetric system (both the sender of the message and its receiver exploit the same secret key). Presented in 1975 at IBM, this technique was adopted as a standard, in its form with 64-bit blocks and a 56-bit key[3] (that allowed expressing numbers up to 100 million billions) with the name of DES on 23 November 1976, and has become the official cipher system of the US Government yet since 1977. It has been certified for reliability by the NIST (National Institute of Standards and Technology) every five years until 1993, and it is still currently the American official standard for information encryption and the secret key cipher most used in computer science environments. If exploited by a single user, the DES can be a very good system for saving files on a hard disk in an encrypted form.

It requires splitting the binary plaintext into $(8n)$-bit blocks (corresponding to $n$ bytes) and applying to each successive encryptions (consisting of transpositions and substitutions of bits) based on an $(8m)$-bit key $k$, according to the following steps:

1. Split the block into two halves of $4n$ bits each, called $\text{Left}^0$ and $\text{Right}^0$.
2. For $i = 1 \dots 16$, carry out a *round* as follows:
   (a) $\text{Right}^{i+1} \leftarrow F(\text{Right}^i, k_i) \oplus \text{Left}^i$
   (b) $\text{Left}^{i+1} \leftarrow \text{Right}^i$.
3. Return the concatenation of $\text{Left}^{16}$ and $\text{Right}^{16}$.

[2] It should be mentioned, however, that, as later ascertained, the most important results in this field were independently invented also by J. Ellis, C. Cocks and M. Williamson during their work at the GCHQ (Government Communications HeadQuarters) of the British Government [12, 19], but they had to keep their findings secret because of military reasons.

[3] The American National Security Agency (*NSA*) thought that such length was sufficiently easy to break for their powerful computers, but not for those of other organizations. Indeed, for security reasons they wanted to keep the chance of intercepting and decrypting messages exchanged in the US, and wanted to be the only one to have such a privilege.

While decoding is carried out by just inverting the steps:

1. Split the block into two halves of $4n$ bits each, Left$^{16}$ and Right$^{16}$.
2. For $i = 16 \dots 1$, carry out a *round* as follows:
   (a) $\text{Left}^{i-1} \leftarrow F(\text{Left}^i, k_i) \oplus \text{Right}^i$
   (b) $\text{Right}^{i-1} \leftarrow \text{Left}^i$.
3. Return the concatenation of Left$^0$ and Right$^0$.

Here

- $\oplus$ is the XOR bit operation;
- $k_i$ is a subkey for the $i$th round computed starting from the key $k$;
- $F$ is a complex deformation function (whose definition is outside the scope of this presentation) that scrambles the bits.

The problem of the DES is, as for all symmetric systems, the distribution of keys.

**IDEA (International Data Encryption Algorithm)** *IDEA* [26] was developed between 1990 and 1992 and first published in 1991 as a potential replacement for the DES (whence its former name *PES*, or *Proposed Encryption Standard*). It is patented in several countries until 2010–2011, and it has never been broken yet, for which reason various standard-enforcing organizations recommend it, and the PGP standard (see Sect. 3.1.3) adopted it for message encryption. The IDEA is a symmetric block cipher that applies on 64-bit data blocks eight *rounds* of a complex permutation–substitution network scheme in which 52 16-bit *subkeys* obtained from a 128-bit key $K$ are exploited, as follows:

For each 64-bit plaintext block $B$,

1. Split $B$ into four 16-bit blocks $B_{0,1}, \dots, B_{0,4}$
2. For $i = 0, \dots, 7$,
   (a) Split $K$ into eight 16-bit blocks and consider only the first six thereof: $K_{(i \cdot 6)+1}, \dots, K_{(i \cdot 6)+6}$
   (b) $P_{i,1} = B_{i,1} \times_{16} K_{(i \cdot 6)+1}$
   (c) $P_{i,2} = B_{i,2} +_{16} K_{(i \cdot 6)+2}$
   (d) $P_{i,3} = B_{i,3} +_{16} K_{(i \cdot 6)+3}$
   (e) $P_{i,4} = B_{i,4} \times_{16} K_{(i \cdot 6)+4}$
   (f) $P_{i,13} = P_{i,1} \oplus P_{i,3}$
   (g) $P_{i,24} = P_{i,2} \oplus P_{i,4}$
   (h) $P_{i,5} = P_{i,13} \times_{16} K_{(i \cdot 6)+5}$
   (i) $P_6 = (P_{i,24} +_{16} P_{i,5}) \times_{16} K_{(i \cdot 6)+6}$
   (j) $B_{i+1,1} = P_{i,1} \oplus P_{i,6}$
   (k) $B_{i+1,2} = P_{i,3} \oplus P_{i,6}$
   (l) $P_{i,56} = P_{i,5} +_{16} P_{i,6}$
   (m) $B_{i+1,3} = P_{i,2} \oplus P_{i,56}$
   (n) $B_{i+1,4} = P_{i,4} \oplus P_{i,56}$
   (o) $\mathcal{R}^L_{25}(K)$

**Table 3.1** Subkeys for decoding in the IDEA cipher

| | Operation | $1/K_n$ | $-K_n$ | $-K_n$ | $1/K_n$ | $K_n$ | $K_n$ |
|---|---|---|---|---|---|---|---|
| Round | Decoding subkey $K'_{(i\cdot 6)+j}$, $j =$ | 1 | 2 | 3 | 4 | 5 | 6 |
| $i = 0, \dots, 7$ | Corresponding $K_n = K_{6\cdot(7-i)+k}$, $k =$ | 7 | 9 | 8 | 10 | 5 | 6 |
| Final | Decoding subkey | $K'_{49}$ | $K'_{50}$ | $K'_{51}$ | $K'_{52}$ | | |
| transformation | Corresponding $K_n$ | $K_1$ | $K_2$ | $K_3$ | $K_4$ | | |

3. Perform a *final transformation* by combining the resulting sub-blocks with the first four 16-bit subkeys $K_{49}$, $K_{50}$, $K_{51}$, $K_{52}$ of the current $K$ as follows:
   (a) $B_{9,1} = B_{8,1} \times_{16} K_{49}$
   (b) $B_{9,2} = B_{8,2} +_{16} K_{50}$
   (c) $B_{9,3} = B_{8,3} +_{16} K_{51}$
   (d) $B_{9,4} = B_{8,4} \times_{16} K_{52}$.

Here

- $\oplus$ is the logical bit-wise XOR operation;
- $+_{16}$ denotes summation modulus $2^{16} = 65536$;
- $\times_{16}$ stands for multiplication modulus $2^{16} + 1 = 65537$;
- $\mathcal{R}^L_n(x)$ is the left-rotation of $x$ by $n$ bits.

Decoding takes place applying the very same steps as above, but using 52 different subkeys $\{K'_j\}_{j=1,\dots,52}$, each obtained as the reciprocal (modulus $2^{16} + 1$) or opposite (modulus $2^{16}$) of one of the encryption keys according to the schema reported in Table 3.1.

The algorithm was designed to resist attacks based on differential cryptanalysis, and at the time of writing only 6 steps of it have been broken using linear cryptanalysis. For a brute force attack to be successful against a 128-bit key, it is estimated that a million billions years are needed in the best case.

**Key Exchange Method** As already pointed out, in symmetric cryptography, before sending a secret message to **B**, **A** must first provide him with the key that is another secret message itself and must be preserved from the risk of stealing or interception. The solution to this problem was found by Hellman [17], and presented at the National Computer Conference in June 1976. It is called a *key exchange* method, and takes place as follows:

1. **A** and **B** agree about two integers $Y$ and $P$, where $P$ must be prime and $Y < P$.
2. **A** and **B** choose a number each (say $a$ and $b$, respectively), and keep it secret, but transmit to the other the value resulting from applying to this number the function $Y^X \bmod P$:
   - **A** transmits $\alpha = Y^a \bmod P$ to **B**
   - **B** transmits $\beta = Y^b \bmod P$ to **A**.

3. Now, each of them obtains exactly the same key as follows:
   - **A** computes the key as $\beta^a \bmod P$
   - **B** computes the key as $\alpha^b \bmod P$.

It is important to point out that the values exchanged by the two actors ($Y$, $P$, $\alpha$ and $\beta$) can even be publicly known, without resulting in the risk of an unsafe communication. Indeed, even intercepting these values, **E** would not be able in any case to get the key because he misses both $a$ and $b$ to apply the last functions.

*Example 3.1* (Use of the Key Exchange Method) Suppose that **A** and **B** agree on $Y = 4$ and $P = 5$. Then **A** chooses $a = 3$ and transmits to **B** the value $\alpha = 4^3 \bmod 5 = 4$, while **B** chooses $b = 2$ and transmits to **A** the value $\beta = 4^2 \bmod 5 = 1$. Lastly, **A** computes the key as $1^3 \bmod 5 = 1$ and **B** obtains just the same value as $4^2 \bmod 5 = 1$.

**RSA (Rivest, Shamir, Adleman)** Developed in 1977, the *RSA* cipher (named after the acronym of its creators: Rivest, Shamir, Adleman) represented a milestone in the history of cryptology. It was the first public key cryptographic system. Nowadays, almost all secure transactions on the Web (i.e., those performed using the HTTPS—HyperText Transfer Protocol Secure) are based on the RSA.

The system relies on a few mathematical results:

**Euler's *totient* function** $\varphi(n)$ of a positive integer $n$, defined to be the number of positive integers less than or equal to $n$ that are coprime to $n$.
If $n$ is a prime number, it is not divided by any of its preceding integers, except 1, and hence $\varphi(n) = n - 1$.
If $n = pq$ with $p$ and $q$ prime numbers, $n$ is factorized only by $p$ and $q$, and hence also all combinations of their preceding integers will be coprime to their product: $\varphi(n) = (p - 1)(q - 1)$.

**Fermat's little theorem** Given a prime number $p$, for any integer $a$ it holds that $a^p \equiv a \bmod p$, i.e., $a^p - a \equiv 0 \bmod p$.
Given a prime number $p$ and an integer $a$ coprime to $p$ (i.e., not evenly divisible by $p$), it holds that $a^{p-1} \equiv 1 \bmod p$, i.e., $a^{p-1} - 1 \equiv 0 \bmod p$.

**Euler–Fermat theorem** Given a positive integer $n$ and an integer $a$ coprime to $n$, it holds $a^{\varphi(n)} \equiv 1 \bmod n$.

The former is used in different ways by the RSA algorithm to generate the public and private keys, through the following steps:

1. Choose two prime numbers $p$ and $q$, with $p \neq q$ (such numbers must be kept secret and, for better security, should be chosen uniformly at random, and have similar bit-length)
2. Take $n = p \cdot q$
3. Compute $b = \varphi(n) = \varphi(pq) = (p - 1) \cdot (q - 1)$
4. Choose an integer $e$ between 1 and $b$ that is relatively prime to $b$ (it should preferably have a short addition chain for improving efficiency, and not be small for improving security)

5. Compute $d$ such that $e \cdot d \equiv 1 \bmod b$ (an extension of Euclid's algorithm for the gcd can be useful to speed up computation)
6. Thus, one gets:

   **Public key** used for encryption: $(e, n)$
   **Private key** used for decoding: $(d, n)$

   where $e$ can be used by different persons, but each must have a different $n$.

Given $M$, the binary-encoded message, the encryption function is:[4]

$$C = M^e \bmod n.$$

Then, to decode the message the formula is

$$M = C^d \bmod n.$$

Due to the modulus in these formulæ, the output is always an integer between 0 and $n-1$, which thus represents the maximum allowed value for a plaintext (respectively, ciphertext) block to be encrypted (respectively, decoded). In other words, a block must have a bit-length $k \leq \log_2(n)$, where $2^k < n \leq 2^{k+1}$.

*Example 3.2* (Application of the RSA algorithm)
**B** chooses $p = 5$ and $q = 11$. This yields $n = 55$, and $b = 40$.
The first integer relatively prime to 40 is $e = 3$, from which $d \cdot 3 = 1 \bmod 40 \Rightarrow d = 27$.
Hence, the public key is $(3, 55)$ and the private key is $(27, 55)$.
Since $n = 55$, the maximum bit-length of each block is $k = 5$ (because $2^5 = 32 < 55 \leq 2^6 = 64$).
Suppose that **A** wants to send a message consisting of a single character $M =$"Y" (say, for 'yes') to **B**. She first transforms it into binary form (e.g., using the ASCII code the 'Y' becomes 10001001) and then splits it into two 4-bits blocks (to have equal-length blocks, indeed splitting into 5-bit blocks would have required the addition of padding bits): $M_1 = 1000$ and $M_2 = 1001$, i.e., 8 and 9 in decimal. Then, she applies to each block the encoding formula:
$C_1 = 8^3 \bmod 55 = 512 \bmod 55 = 17$ and $C_2 = 9^3 \bmod 55 = 729 \bmod 55 = 14$,
and sends $C_1$ and $C_2$ to **B**. On the other side, **B** recovers the original message as $M_1 = 17^{27} \bmod 55 = 17^{8 \cdot 3+2+1} \bmod 55 = ((17^8 \bmod 55)^3 \cdot (17^2 \bmod 55) \cdot (17^1 \bmod 55)) \bmod 55 = (26^3 \cdot 14 \cdot 17) \bmod 55 = 4183088 \bmod 55 = 8$, and
$M_2 = 14^27 \bmod 55 = 14^{8 \cdot 3+2+1} \bmod 55 = ((14^8 \bmod 55)^3 \cdot (14^2 \bmod 55) \cdot (14^1 \bmod 55)) \bmod 55 = (16^3 \cdot 31 \cdot 14) \bmod 55 = 1777664 \bmod 55 = 9$.

Just to give an idea of how secure the RSA cipher is, let us sketch a brief discussion on whether it is possible, and by which effort, to go back to the private key, knowing the public one. The following approaches are available:

[4]Any integer $y$ can be decomposed as a sum of different powers of 2: $y = p_1 + \cdots + p_n$. Thus, the following simplification can be applied when computing the modulus of high powers:

$$x^y \bmod b = x^{p_1 + \cdots + p_n} \bmod b = \left(x^{p_1} \cdots x^{p_n}\right) \bmod b = \left(\left(x^{p_1} \bmod b\right) \cdots \left(x^{p_n} \bmod b\right)\right) \bmod b.$$

**Brute force** involves the tentative application of all possible private keys.

**Mathematical attacks** aim at discovering the private key by means of several mathematical techniques, all of which can be reduced, in terms of complexity, to the factorization of the product of two prime numbers:

- Factorize $n$ into its two prime factors $p$ and $q$. This is equivalent to determining $\varphi(n)$ that, in turn, allows finding out the private key $d$.
- Directly determine $\varphi(n)$, without first determining $p$ and $q$.
- Directly determine the private key $d$ without first determining $\varphi(n)$.

**Time attacks** are based on the runtime of the decryption algorithm.

**Selected ciphertext attacks** leverage the features of the RSA algorithm.

The system security is founded on the difficulty in splitting its core element $n$, that represents the flexible component of the unidirectional function, into its two prime factors. Most techniques for cryptographic analysis of RSA try this way but, using the algorithms currently known, it seems to be too costly in terms of time. It is possible that in the future a quick factorization algorithm will be invented, thus making the system useless, but unproductive research in the last two millennia suggests this event to be very unlikely, and that probably this cannot be done in principle.

The defense techniques against a brute force approach are the same as those used for other encryption systems: exploiting a key chosen in a large space of keys. The larger the number of bits for the private key, the more resistant the algorithm. Unfortunately, given the computations needed to generate the key and for the encryption/decoding, the larger the size of the key, the slower the system. Thus, the security of the RSA algorithm improves as long as the length of the key increases (although its performance decays). For large values of $n$, with sufficiently large prime factors $p$ and $q$ (several hundreds of decimal digits, to stand the current computational power of computers), finding them from $n$ is quite difficult. Theoretically, it is always possible to use such a long key that all the computers in the world joined together could not find the solution within millions of years.

However, factorization is nowadays not as difficult as in the past. In 1977, an article about the RSA, written by M. Gardner and published in the Mathematical Games section of the Scientific American journal, challenged the readers to decrypt a message given $n$ (consisting of 129 decimal digits, i.e., approximately 428 bits), offering a reward to whoever would provide the plaintext sequence. Indeed, it was expected not to happen before 40 quadrillions years, but in April 1994 a group claimed the prize after just eight months working on the Internet. In the meantime, new challenges have been published, the last of which involved a key 200 decimal digits long (i.e., 663 bits). The effort to factorize a key made up of 155 decimal digits, using a quite fast algorithm, is of 8000 MIPS per year (Millions Instructions Per Second per year). A personal computer endowed with a single-core processor running at 2 GHz speed provides nearly 500 MIPS. One must consider, however, that in the last years very efficient factorization algorithms have been developed, which requires the length of the key to be increased. It seems reasonable to foresee that, in the near future, it will be necessary to exploit keys having length equal to 1024–2048 bits.

**DSA (Digital Signature Algorithm)** *DSA* is an asymmetric encryption algorithm, dating back to 1991 and later adopted for the DSS standard (see Sect. 3.1.3) that bases its security on the difficulty of computing discrete logarithms. It applies to a group of users whose members adopt three shared public parameters that underlie the cipher and based on them compute their public and private keys. Preliminarily, a pair of integers $(L, N)$ must be chosen, that defines the security level of the final keys: the larger the values of $L$ and $N$, the more security is provided. Then, the shared parameters are defined as follows:

1. Choose a prime number $p$ having bit-length $L$ (or, in other words, such that $2^{L-1} < p < 2^L$);
2. Choose a prime number $q$ having bit-length $N$ (i.e., such that $2^{N-1} < q < 2^N$) that is a divisor of $(p - 1)$;
3. Choose an integer $h$ such that $1 < h < (p - 1)$, and compute $g = h^{(p-1)/q}$ until it holds that $h^{(p-1)/q} \bmod p > 1$ (usually $h = 2$ fulfills the requirement).

Given the shared (public) parameters $p$, $q$ and $g$, each user in the group selects a private key and generates a corresponding public key as follows:

1. The private key $x$ must be an integer chosen (preferably at random) between 1 and $(q - 1)$ (i.e., $0 < x < q$).
2. Once the private key $x$ has been generated, the public key $y$ is obtained by the formula

$$y = g^x \bmod p.$$

Computation of the public key $y$ based on the private key $x$ is relatively easy. Nevertheless, given $y$, it is computationally impossible (for large prime numbers), even for the most efficient algorithm currently known, to identify $x$, that is the discrete logarithm of $y$ in base $g$ modulus $p$. Hence, the DSA encryption algorithm can be considered as reliable.

## 3.2 Message Fingerprint

The *digital fingerprint* of a message consists of a code, computed on the basis of the message content, that can be exploited for integrity check and assurance. Just as a physical fingerprint ensures that its owner is univocally identified, in the same way a digital fingerprint can guarantee that the message from which it was computed has not changed. Indeed, two pieces of digital content that differ for even one symbol or bit would yield different codes, and hence changes to the document after the code has been generated would be easily discovered by computing the new code and comparing it to the original one. Fingerprints are often exploited to point out transmission errors when sending or downloading files through the Internet, but their use for proving that an official document has not been maliciously modified is straightforwardly apparent.

The digital fingerprint of a document is generally obtained exploiting a *hashing* algorithm that (based on the document content) generates a secure and fixed-length numeric code, called *digest*, that in turn allows univocally identifying the digital document (where, obviously, univocity must be intended in a probabilistic way). The larger the bit-length allowed for the digest, the less the chance of collisions (i.e., different documents having the same hash value). Just to give an idea, a security threshold for an algorithm using a 160-bit digest is a collision every $2^{80}$ different messages. A desirable property of such hash functions is the *avalanche effect*, according to which even slight changes in the input yield large changes in the output [20]. A famous algorithm is MD5 [29], but it was recently proved to be quite unreliable [33] (collisions can be found at an unacceptable rate), so that many organizations and frameworks based on it are compelled to switch to other techniques.

**SHA (Secure Hash Algorithm)** The *SHA*, accepted as FIPS 180 standard, is actually a family of methods for computing the digital fingerprint of a document, denoted as SHA-0, SHA-1 and SHA-2 and, recently, SHA-3 [2, 3, 6, 7]. Specific algorithms in the family differ as to level of security and digest length. The current version is SHA-2, further split into several subversions SHA-$n$ where $n$ represents the bit-length of the digest they produce (SHA-224, SHA-256, SHA-384, and SHA-512). Older versions (SHA-0 and SHA-1) work on 32-bit words, apply to messages at most $(2^{64} - 1)$-bits long that are split into 512-bit chunks, and produce 160-bit digests using a technique based on an 80-step loop. They became obsolete because of security flaws (hash collisions were actually found for SHA-0, while for SHA-1, although no collision has been found yet, algorithms have been designed that should produce collisions every $2^{52}$ cases). The SHA-2 series added bit-shift to the operations exploited for computing the digest. Moreover, SHA-224 and SHA-256 reduced the number of loop steps to 64, while SHA-384 and SHA-512 restored the number of loop steps to 80 and extended the block size to 1024 bits, the maximum length of messages to $2^{128} - 1$ bits, and the word size to 64 bits. In all cases, the digest is the result of progressively updating an initial value according to partial hash values computed for each chunk of the message and stored in a temporary buffer. Both the (partial) digest and the buffer are seen as the concatenation of several word-sized *registers* that are handled separately during the various steps of the algorithm. All numbers are handled as $b$-bit unsigned integers in big-endian byte ordering, and all operations are performed modulus $2^b$, where $b$ is the register bit-length.

Here we will focus on the SHA-512 standard that splits both the digest and the buffer (both 512-bit long) into eight registers ($H_i$ and $B_i$, for $i = 0, \dots, 7$, respectively). The computation on a binary message $M$ proceeds as follows:

1. **For** $i = 1, \dots, 8$,
   $H_{i-1} \leftarrow$ first 64 bits of the fractional part of the $i$th prime number's square root
2. **For** $i = 1, \dots, 80$,
   $K_{i-1} \leftarrow$ first 64 bits of the fractional part of the $i$th prime number's cube root
3. Append to $M$ a **1** bit followed by 1 to 1024 **0** bits, as needed to make the final bit-length $L$ congruent 896 modulus 1024 ($L \equiv 896 \bmod 1024$)
4. Append to the previous result the original message length as a 128-bit unsigned integer
5. Split the message into 1024-bit chunks

6. **For** each chunk,
   (a) Compute 80 64-bit words $W_i$
   $W_0 \dots W_{15}$ are obtained breaking the chunks in 64-bit words
   **For** $i = 16 \dots 79$,
$$\begin{aligned} W_i = & W_{i-16} +_{64} \mathcal{R}_1^R(W_{i-15}) \oplus \mathcal{R}_8^R(W_{i-15}) \oplus \mathcal{S}_7^R(W_{i-15}) \\ & +_{64} W_{i-7} +_{64} \mathcal{R}_{19}^R(W_{i-2}) \oplus \mathcal{R}_{61}^R(W_{i-2}) \oplus \mathcal{S}_6^R(W_{i-2}) \end{aligned}$$
   (b) **For** $i = 0, \dots, 7$, $B_i \leftarrow H_i$ (initializes buffer for this chunk)
   (c) **For** $t = 0, \dots, 79$,
      (i) $T = B_7 +_{64} (\mathcal{R}_{14}^R(B_4) \oplus \mathcal{R}_{18}^R(B_4) \oplus \mathcal{R}_{41}^R(B_4))$
      $+_{64} \text{ch}(B_4, B_5, B_6) +_{64} K_t +_{64} W_t$
      (ii) **For** $i = 7, \dots, 1$, $B_i \leftarrow B_{i-1}$
      (iii) $B_4 \leftarrow B_4 +_{64} T$
      (iv) $B_0 \leftarrow T +_{64} ((\mathcal{R}_{28}^R(B_0) \oplus \mathcal{R}_{34}^R(B_0) \oplus \mathcal{R}_{39}^R(B_0)) +_{64} \text{maj}(B_0, B_1, B_2))$
   (d) **For** $i = 0, \dots, 7$, $H_i \leftarrow H_i + B_i$ (adds this chunk's hash to result so far).
   Here the following notation is exploited:

   $\mathcal{R}_n^R(x)$ right round rotation of $n$ bits of the 64-bit argument $x$;
   $\mathcal{S}_n^R(x)$ right shift of $n$ bits of the 64-bit argument $x$, filling by **0**'s on the left;
   $\oplus$ XOR bit-wise operation;
   $\otimes$ AND bit-wise operation;
   $\ominus$ NOT bit-wise operation;
   $\text{ch}(X, Y, Z) = (X \otimes Y) \oplus ((\ominus X) \otimes Z)$ *conditional function*
   equivalent to IF $X$ THEN $Y$ ELSE $Z$;
   $\text{maj}(X, Y, Z) = (X \otimes Y) \oplus (X \otimes Z) \oplus (Y \otimes Z)$ *majority function*
   true if and only if the majority of its arguments is true;
   $+_{64}$ sum modulus $2^{64}$.

7. Output as digest the final 512-bit hash value (concatenation of the registers):
$$H_0 H_1 H_2 H_3 H_4 H_5 H_6 H_7.$$

The first two steps, respectively, initialize the hash value and produce 80 additive constants to be exploited during the 80 rounds. These values can be considered as random 64-bit configurations that should scramble any regularity in the input data.[5] The next three steps yield a message of length equal to $N \cdot 1024$ bits. Hence, it can be processed in 1024-bit blocks $M_i$ $(i = 1, \dots, N)$. The initial hash value is updated after processing each block by adding to each of its components a new value derived from that block. In short,

1. $Hash_0 = IV$
2. $Hash_i^j = Hash_{i-1}^j +_{64} Buffer_i^j$ (for $j = 0, \dots, 7$)
3. $MD = Hash_N$

where

[5]The first eight prime numbers range between 2 and 19, while the first 80 prime numbers range between 2 and 311.

- *IV* is the initial value of the hash code;[6]
- $Hash_i$ is the partial hash value computed on the $i$th block of the message output after its last (i.e., the 80th) processing step;
- $Hash_i^j$ and $Buffer_i^j$ denote the $j$th word/register of the hash code and of the buffer, respectively;
- *MD* is the final value of the hash code (Message Digest).

The core of a block processing consists of an 80-step loop that yields a 512-bit value to be added to the overall hash digest. Such a value is stored in a buffer that initially gets the same value as the current partial digest after the last processed block ($Hash_{i-1}$), and is updated in each step $t$ of the loop according to its previous content and a 64-bit value $W_t$ derived from the 1024-bit block under processing ($M_i$). In the first 16 processing steps, the value $W_t$ is equal to the corresponding word in the message block. For the remaining 64 steps, it is obtained by a mix of rotation, shift, XOR, AND, NOT and sum operations carried out on the values of some of the previous words. The output of the 80th step is added to the partial digest up to the previous block ($Hash_{i-1}$) to yield $Hash_i$. The sum (modulus $2^{64}$) is computed independently for each of the eight word/register pairs of the buffer with each of the corresponding words in $Hash_{i-1}$.

## 3.3 Digital Signature

The authors of legacy documents confirm the content and undertake all consequences thereof by means of autographic signatures that consist of the "modification of a paper document by affixing a mark that can be traced back to its author, forgery actions excepted". According to jurists [11], a signature must be:

**named** it must make explicit the name of its producer;
**readable** its handwriting must be comprehensible to whoever reads it;
**recognizable** it must allow identifying its author;
**non-reusable** it must establish a unique relationship to the signed document;

and perform the following functions:

**Indication** of who has produced the document;
**Declaration** the signer undertakes paternity of the document;
**Proof** it must be possible to produce it as a juridically valid proof.

[6]In hexadecimal, the initial hash value (split by register) is:

- $H_0 = 6A09E667F3BCC908$
- $H_1 = BB67AE8584CAA73B$
- $H_2 = 3C6EF372FE94F82B$
- $H_3 = A54FF53A5F1D36F1$
- $H_4 = 510E527FADE682D1$
- $H_5 = 9B05688C2B3E6C1F$
- $H_6 = 1F83D9ABFB41BD6B$
- $H_7 = 5BE0CD19137E2179$.

The ever-increasing use of electronic messages and documents, and their adoption in formal environments such as the Public Administration or the business, raises the need for the development of an equivalent counterpart, for digital information, of what classical autographic signatures are for paper documents. Indeed, simple message encryption protects the subjects involved in a communication from malicious third parties, but does not protect them from each other in cases in which there is no complete trust between them. For instance, the receiver could produce a message and pretend it comes from the sender, or the sender could deny having sent a message: digital information, differently from handwritten text, cannot be analyzed from a graphological viewpoint in order to assess its paternity. *Digital signature* techniques were developed purposely to play such a role. Obviously, a digital signature cannot consist of a mark that modifies a document (e.g., a scanned image of an autographic signature), but rather it must be an authentication mechanism that allows univocally identifying the author of a digital message or document, based on a code that he adds to the message itself and that acts as a signature for it.

In order to be considered reliable and to avoid possible controversies (or to allow the resolution thereof, in case they arise), a digital signature must guarantee security from the following perspectives:

**Integrity** the document cannot be modified after the signature was affixed (a guarantee for the signee);
**Authentication** the author of the document (and possibly the date and time of issue) must be univocally determined (a guarantee for third parties);
**Non-repudiation** the author must not be allowed to deny having signed the document (a guarantee for the receiver).

Operationally, compliance to the aforementioned properties and additional desirable ones can be carried into effect by enforcing the following main requirements [32]: it must

- Consist of a bit configuration that depends on the signed message, to prevent anybody from modifying the message;
- Exploit sender-specific information, to prevent him from denying to be the author of the message;
- Be easily produced, verified and acknowledged;
- Be computationally impossible to be forged (by making a counterfeited message for an existing digital signature, or making a fake digital signature starting from a given message);
- Allow retaining a copy of it.

Several techniques must concur to make up a digital signature procedure that fulfills the above requirements. Integrity can be ensured with the adoption of a method for producing the digital fingerprint of the document to be signed, in order to prevent it from being changed. As regards authentication, a straightforward solution consists in applying an encryption technique to secure the document. More specifically, an asymmetric technique is needed, but inverting the normal procedure, i.e., using the private key for encryption and the public one for the corresponding decoding. Indeed, the use of a private key that is known only to the author and nobody else implies that, if his public key works in decoding the message, he is the only person that can have produced the encryption, and hence as a side-effect ensures non-repudiation as well. Summing up, the digital signature of a digital document consists of an additional file to be provided attached to that document, obtained by computing a digital fingerprint of the message (that links the signature to the single, specific document) signed with a private key (that links the signature to a unique author, whose identity can be checked using the corresponding public key). Actually, the private key could be applied for encrypting directly the whole document, but it is usually avoided, and the document fingerprint is preferred instead, both because it would be a computationally heavy task and require a long time, and because in some cases it is not desirable that the digital signature check unveils the document content to the verifier. Both objects (the document and the signature) must be enclosed in a *digital envelope* and sent to the receiver.

### 3.3.1 Management

In practice, different approaches have been proposed for creating and managing digital signatures. They can be partitioned into *direct* and *arbitrated* ones, according to whether the sender and receiver deal directly with each other, or pass through a mediator called the *arbiter*. In the former approach, only the parties that carry out the communication, i.e., the sender and the receiver, are involved, and hence they must fully trust each other. In the latter approach, the sender transmits the message and signature to the arbiter, that checks their origin and integrity and forwards them to the receiver, along with a timestamp witnessing the date and time of issue. To show the procedure in the two cases, let us assume that a sender $S$ wants to transmit a message $M$ to a receiver $R$, possibly through an arbiter $A$, and exploit the following notation:

- $[O_1, \ldots, O_n]$ denotes a bundle made up of the items $O_1, \ldots, O_n$.
- $O_K$ means encrypting the item $O$ with key $K$ that can be one of the following:
  - $Pr(X)$ private key of $X$
  - $Pu(X)$ public key of $X$
  - $K(X, Y)$ encryption key between $X$ and $Y$ (it can be a shared secret key if a symmetric encryption is used, or a public/private key, as required, in case of asymmetric cipher).
- $H(\cdot)$ hash (fingerprint) function.
- $I$ identifier of the sender.
- $T$ timestamp.

The direct scheme works as follows:

1. $S$ computes $\overline{H} = H(M)$.
2. $S$ sends $[M, \overline{H}_{Pr(S)}]_{K(S,R)}$ to $R$.
3. $R$ decodes the bundle using $K(S, R)$, obtaining $M$ and $\overline{H}_{Pr(S)}$.
4. $R$ decodes $\overline{H}_{Pr(S)}$ using $Pu(S)$ and compares $\overline{H}$ to the value $H(M)$ he directly computes on $M$.

The sender creates a digital signature by using his private key to encrypt the whole message, or just a hash code thereof. Hence, to decode it, the receiver is assumed to know the public key of the sender. The receiver needs just to store the plaintext with its associated signature as a proof to be used subsequently for settling possible controversies. Secrecy from third parties can be guaranteed by further encrypting the whole message plus the signature.[7] The shortcoming of this approach is that its validity clearly depends on the security of the sender's private key. If a sender wants subsequently to deny having sent a message, he could maintain that the private key got lost or was stolen, and that someone else forged the signature. To prevent this, one might require that each message includes also a timestamp $T$, and that in case the private key is stolen its owner immediately declares such an event.

The arbitrated scheme proceeds as follows:

1. $S$ computes $\overline{H} = H(M)$.
2. $S$ sends $[M, [I, \overline{H}]_{K(S,A)}]$ to $A$.
3. $A$ decodes the inner bundle using $K(S, A)$ to check $I$ and compare $\overline{H}$ to the hash $H(M)$ that he computes directly on $M$.
4. If the tests are passed, $A$ sends to $R$ the bundle $[M, I, [I, \overline{H}]_{K(S,A)}, T]_{K(A,R)}$.

Here, the keys used for communication between the sender and the arbiter are different than those used between the arbiter and the receiver. The arbiter checks the message and its signature using suitable tests to verify their origin and content. If the tests are passed, he adds a timestamp and an acknowledgment of validity, and forwards to the receiver an encrypted bundle containing the message plus all check information. Thus, the sender cannot deny having sent the message. If confidentiality is an issue the original message may be encrypted, so that the arbiter can only certify the signature and hash value, but cannot actually read the message. In such a case, the message $M$ is to be replaced by its encrypted version $M_{K(S,R)}$ throughout all the steps of the above procedure. The arbiter plays a delicate and crucial role in this kind of schema and both parties must significantly trust the fact that the arbitration mechanism works correctly and fairly. Moreover, the sender must trust that the arbiter will not reveal their key nor forge a signature, and the receiver must trust that the arbiter will not certify messages not coming from the sender or having wrong hashcodes.

[7] In such a case, it is important that the signature is inside the latter cryptogram. Indeed, in case of quarrel, a third party summoned to observe the message and its signature could just exploit the sender's public key on the inner cryptogram. Conversely, if the signature were computed on the encrypted message, yielding $[[M, \overline{H}]_{K(S,R)}]_{Pr(S)}$, he should know the secret decryption key $K(S, R)$ to access the message and corresponding hash value to be checked.

To be widely accepted, the management of a digital signature must be carried out based on standards that ensure a high level of security by specifying suitable methods for the generation of private and public keys and algorithms for encoding and decoding messages.

**DSS (Digital Signature Standard)** One of the most important digital signature techniques available nowadays has been established in the FIPS 186 standard, also known as *DSS*, proposed and published by the NIST (National Institute of Standards and Technology) in 1993 and subsequently updated as an answer to inquiries on the security of the standard or for minor revisions [4, 8]. In its original version, it exploits the SHA for generating the digital fingerprint of a document and the DSA for generating the digital signature, but an extended version, known as FIPS 186-2, has been defined that introduces the support for digital signature algorithms exploiting RSA encryption.

The way the DSA-based DSS works is quite simple. Informally, the hash code of the document to be signed is generated, and input to a signature function along with a randomly generated number for that particular signature. The signature function depends also on the private key of the sender and on the set of public parameters ($p$, $q$ and $g$) shared by the group of persons that are responsible for the transmission (see Sect. 3.1.3). This set globally represents a kind of public key for the group. The resulting signature is made up of two components: $s$ and $r$. The receiver, having obtained the document, generates its hash code and inputs it to a checking function, along with the sender's public key. If the signature is valid, the checking function outputs a value corresponding to the $r$ component of the signature.

The original DSS specification requires that the pair $(L, N)$ is set to $(l \cdot 64, 160)$, where $8 \leq l \leq 16$ (put another way, the length of $p$ must be between 512 and 1024 bits, inclusive, with increments of 64 bits), but more recent updates of the standard suggest that acceptable security levels today require values of $(2048, 224)$ or $(2048, 256)$ and that soon values of $(3072, 256)$ will be needed [8]. In any case, an important requirement is that $N$ must be less than or equal to the hash output length (determined by the $d$ parameter in the case of SHA-$d$).

Given a message $M$, the procedure to obtain its digital signature consists in repeating the following steps:

1. Take a random integer $k$ such that $0 < k < q$.
2. Compute
$$r = \big(g^k \bmod p\big) \bmod q.$$
3. Compute
$$s = \big[k^{-1}\big(H(M) + x \cdot r\big)\big] \bmod q,$$
   where $H(M)$ is the digital fingerprint of the message $M$ computed as a hash function according to the SHA-$d$ algorithm

until $s \neq 0$ and $r \neq 0$.

The message signature is represented by the pair $(r, s)$.

Verification of the signature takes place as follows:

1. If the conditions $0 < r < q$ and $0 < s < q$ are not satisfied, the signature is rejected
2. Otherwise, the signature is considered valid if $v = r$, where[8]

- $v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$
- $u_1 = [H(M)w] \bmod q$
- $u_2 = (rw) \bmod q$
- $w = r^{-1} \bmod q$.

In the more recent RSA-based approach, the message to be signed is sent to a hash function that produces a secure hash code of fixed length. Such a code is then encrypted using the sender's private key, in order to make up the signature. Then, the message and its associated signature are sent. The receiver, upon reception of the signed message, autonomously computes the hash code of the document and, using the public key, decodes the digital signature. If the two hash codes (the one generated by the receiver and the one sent by the message author) agree, the document can be deemed as valid.

**OpenPGP Standard** OpenPGP is an Internet standard for secure message transmission, born as an evolution of a previous human rights project started by P. Zimmermann for ensuring privacy and secure communication to everybody. Indeed, in 1991 Zimmermann published for free on the Internet *PGP* (*Pretty Good Privacy*) [21, 35], an email asymmetric encryption software tool aimed at providing encryption facilities that are not too much computationally demanding, so that they can be run on widespread computer systems, and are user-friendly, so as to hide all technical and complex aspects of encryption and decoding (e.g., the private and public key are automatically generated at random based on the movement of the mouse). After coming out from a criminal investigation, promoted by the US Government for violation of export restrictions on cryptographic software (considered in the US as a war weapon), but later abandoned because of the success obtained by PGP, Zimmermann founded a company whose subsequent purchaser made PGP a commercial product whose development is still ongoing.

The basic mechanism underlying PGP consists in encrypting the message through a common, quick symmetric algorithm run on a key randomly generated for that session, but then encrypting the symmetric key itself (which is much shorter than the message) through asymmetric encryption, and sending both to the receiver. More precisely, the IDEA method performs the former encryption, while the RSA algorithm is exploited for the latter. Both are patented technologies which caused a number of troubles to the straightforward spread of PGP throughout the world. Another facility provided by PGP is digital signature. Summing up, IDEA is used to

[8]Correctness of the procedure can be proved as follows. First, by definition of $g$ and due to Fermat's little theorem, $g^q \equiv h^{p-1} \equiv 1 (\bmod\ p)$. Since $g > 1$ and $q$ is prime, $g$ must have order $q$.
Due to the definition of $s$, $k \equiv H(m)s^{-1} + xrs^{-1} \equiv H(m)w + xrw \bmod q$.
Since $g$ has order $q$ it follows that $g^k \equiv g^{H(m)w} g^{xrw} \equiv g^{H(m)w} y^{rw} \equiv g^{u1} y^{u2} \bmod p$.
Finally, the DSA is correct because $r = (g^k \bmod p) \bmod q = (g^{u1} y^{u2} \bmod p) \bmod q = v$.

encrypt the plaintext, the RSA encrypts the IDEA key, and an additional step applies the signature.

OpenPGP was developed as an evolution of the original PGP, with the aim of providing a non-proprietary protocol to encode and sign messages, certify and exchange keys. It is currently established by the RFC 4880 standard [10], fulfilled by PGP and *GPG* (*GNU Privacy Guard*, a non-commercial implementation), that describes a number of methods for the management of digital signature systems, for the computation of a document's digital fingerprint, and for symmetric and asymmetric key encoding. Some of the algorithms supported by OpenPGP for the various steps are:

**asymmetric encoding** DSA, RSA, RSA-E, RSA-S, ELG-E (Elgamal);
**symmetric encoding** IDEA, TripleDES, CAST5, BlowFish, AES at 128/192/256 bit, TwoFish;
**digital fingerprint** MD5, SHA-1, RIPE-MD/160, SHA-224/256/384/512.

For the management of the digital signature, the focus is on encryption algorithms based on asymmetric key and on algorithms for digital fingerprint computation. The most widespread algorithms for asymmetric encoding are the DSA and the RSA, used also in other digital signature standards, such as the DSS. For computing the digital fingerprint the most widespread algorithms are the MD5 and the SHA.

In the OpenPGP standard, a direct signature scheme is exploited, i.e., the sender generates the pair of keys and provides his public key for the verification. To provide some trust for the actual users' identity, OpenPGP relies on a Web of Trust scheme [9] (see Sect. 3.3.2). Thus, the GPG users themselves are in charge of assessing their confidence in a certificate being genuine, according to their trust in those who supported that certificate by signing it.

### *3.3.2 Trusting and Certificates*

A shortcoming of the direct signature scheme lies in the impossibility to guarantee that the association key-user is genuine, i.e., that a public key actually comes from the person or organization to which it is claimed to belong, so that malicious users could exploit false identities or pretend to be someone else. To make up for this lack, a widespread solution is represented by the introduction of *digital certificates*, i.e., electronic documents (files) that unambiguously identify signees. They are obtained by means of someone who guarantees the real association of a public key to a specific user, unambiguously identified. In such a case, the digital envelope to be sent to the receiver must include three items (files): the document/message, its digital signature and the associated certificate.

A digital certificate typically reports the following information:

**ID** A serial number that uniquely identifies the certificate.
**Authority** The entity that verified the information and issued the certificate.
**Validity** The starting and expiration dates of the certification.

**User** The data identifying the entity to be certified (e.g., name, address, etc.).
**Key** The public key to be certified and its intended use (e.g., encryption, signature, certificate signing, etc.).
**Signature Algorithm** The algorithm used to create the signature.
**Fingerprint** The hash of the certificate, to ensure that it has not been forged, and the algorithm used to produce such a fingerprint.

Translated into natural language, a certificate means approximately "I hereby certify, under my responsibility, that its owner is actually who he claims to be", and hence that the information it contains (and specifically the association between the reported identity data and public key) is valid. This claim is technically implemented in the issuer digitally signing the certificate, and consequently undertaking the responsibility of the certificate itself. A certificate is *self-signed* when it is signed/issued by the same entity that it certifies. Of course, the trust a certificate deserves is strictly related to the reliability of its issuer, which can be assessed according to two different strategies, that give rise to the following schemes:

**Public Key Infrastructure** (*PKI*) in which the signature belongs to a *Trusted Third Party* (*TTP*), often called a *Certificate Authority* (*CA*);
**Web of Trust** in which the certificate is either self-signed by the user or it is signed by other users that *endorse* its validity.

A PKI must declare the security policy it conforms to, the procedure it follows for emission, registration, suspension and revocation of certificates (*Certificate Practice Statement*, or *CPS*), its system of CAs, its system of *Registration Authorities* (*RA*s) for user registration and authentication, and its certificate server. A CA, in particular, is required to bring up all proper actions aimed at verifying the certified entities' identity, in order to justify the trust that the users of their certificates place on them. In case of problems, a CA can suspend or even completely withdraw a certificate, inserting it into a *Certificate Suspension List* (*CSL*) or into a *Certificate Revocation List* (*CRL*) accordingly. Then, the question of who guarantees for the issuer arises. A possible solution is requiring the CA to exhibit a certificate issued by a higher-level CA in a hierarchy of increasingly important CAs. Users, typically exploiting suitable software, check that the private key used to sign a certificate matches the public key in the CA's certificate. This does not completely solve the problem, but limits it to the highest CA only (*root*), which provides the ultimate assurance in each particular PKI organization. Obviously, there being nobody that guarantees for the root, its certificate (also called *root certificate*) can only be self-signed. It goes without saying that, if the root certificate is fake, the whole tree of CAs and certificates in the PKI becomes untrustable, and hence the procedure for issuing root certificates must be accomplished with extreme care in order to avoid any kind of bug or flaw.

In a Web of Trust scheme, the responsibility of guaranteeing a certificate is not charged on a single, formally trusted, entity, but it is shared among several entities that claim that certificate to be reliable. More specifically, each user produces a self-signed certificate that is subsequently additionally signed by other users that confirm its being reliable (i.e., that the identity and public key contained therein are genuine and actually correspond). In this case, other users base their trust about that certificate being correct on these additional signatures, qualitatively (they might trust—or untrust—specific users that signed that certificate, and as a consequence trust—or untrust—that certificate), or quantitatively (their confidence in the reliability of the certificate might increase with the number of additional signatures supporting it). Also in this case the problem arises of who guarantees for the actual identity of the additional supporters. Some of them could be personally known to the user, and hence their public keys might have been passed directly to him. Otherwise, the user must take into account some degree of hazard in 'propagating' the trust from known persons to the persons they guarantee for. Some CAs also maintain a Web of Trust, counting and managing the endorsements to the subscribing certificates.

To ensure automatic processing by security-aware software applications, and interoperability between different CAs, certificates must be produced according to pre-defined standards. In particular, the current official standard, called X.509, is established by RFC 5280 [13], that also specifies regulations for *CRL*s, a useful means for identifying certificates that for some reason must not be trusted anymore.

Certificates are thoroughly exploited in the *HTTPS* (HTTP *Secure*) Internet environment for Web sites, that enforces secure communication and data integrity on TCP/IP networks by underlying HTTP with encrypted source-destination communication at the transport level using the *Secure Socket Layer* (*SSL*) protocol or, more recently, the *Transport Layer Security* (*TLS*) protocol. In this setting, each Web operator must obtain a certificate by applying to a CA, indicating the site name, contact email address, and company information. The usual check performed by the CA consists in verifying that the contact email address for the Web site specified in the certificate request corresponds to that supplied in the public domain name registrar. If the application is successful, the public certificate is issued by the CA. It will be provided by the Web server hosting the site to the Web browsers of the users connecting to it, in order to prove that the site identification is reliable. The root certificates of some CAs that fulfill given management standards can be natively embedded in the Web browsers: in case they are, the browser will consider as trusted all the sites having certificates depending on those CAs; in case they are not, the root certificates will be considered as self-signed certificates. If the browser successfully checks the certificate, the connection can be established and the transaction performed, while in the opposite case a warning message is displayed to the user, asking him to abort or confirm the connection under his own responsibility. The system might not be completely safe due to the fact that three strictly related roles come into play, whose underlying actors could be different, and hence the relationship among them could be deceived: the purchaser of the certificate, the Web site operator, and the author of the Web site content. Thus, strictly speaking, an HTTPS Web site is not 'secure' at all: the end user can only be sure that the registered email address corresponds to the certified one and that the site address itself is the original one, but it could be operated by someone different than the person that registered the domain name, or its content could have been hacked.

## 3.4 Legal Aspects

As long as computers progressively become the main means for the organization and management of economic and administrative activities, the classical document management settings turn out to be a more and more dramatic bottleneck. Indeed, the emphasis on paper documents endowed with autographic signatures and various kinds of seals and stamps as a guarantee of originality and validity, in addition to the problems already discussed, also imposes a (useless) double transformation from digital to printed format and back: first, paper documents must be produced from the digital source data that are available to one party, and then the information must be extracted from paper, transformed and stored again in digital format by the other party. For this reason, all governments are progressively switching towards acceptance of natively digital documents as legally valid proofs. Of course, the peculiarities inborn in digital information make security a fundamental issue, and as a consequence the digital signature system is gaining wider and wider application in the legal and institutional domain. For this reason, it may be interesting to draw a picture of the past and current situation in the law in force about this subject.

Let us recall a few prominent events:

- On 3 December 1998, 33 countries from all-over the world signed in Austria the *Wassenaar Arrangement* on 'Export Controls for Conventional Arms and Dual-Use Goods and Technologies', in which the importance of encryption products is recognized, and limits are imposed on their export.
- In Great Britain, the historical promoter of computer technology in Europe, a 'UK Draft Legislation on Electronic Commerce', that formally acknowledges encryption and digital signatures, was presented to Parliament in July 1999. It preceded the 'UK Electronic Communication Bill', later 'UK Electronic Communication Act', presented to the House of Commons on 18 November 1999, that acknowledges legal validity of electronic signatures. Commenting upon such an event, an article titled "D-day for e-commerce—now it's time to tackle phone charges" published in 'The Guardian' on 20 November 1999 notices that the UK is 2 years behind the US in that subject. Further related documents are the Electronic Commerce Bill/Act issued in 2006 and the Digital Britain White Paper [14]. More recently, a Queen's Speech held on 18 November 2009 announces: "My Government will introduce a Bill to ensure communications infrastructure that is fit for the digital age, supports future economic growth, delivers competitive communications and enhances public service broadcasting".
- The first regulation of this subject in Europe dates back to the 1999/93/CE Directive (13 Dec.), envisaging a shared framework for digital signatures in the Community. Then, in 2006, as a result of an overview of the status of accomplishment of the regulation, the European Union expressed the intention to update and adapt the 1999 directive, in order to further boost this technological tool.
- However, Italy has since 1997 adopted a positive attitude towards full reception of digital technologies as a legally valid means for the support of formal transactions. Indeed, at that time, in Italy there were already regulations concerning the introduction and acknowledgment of digital documents and digital signatures.

Given the above timeline, it turns out that a crucial period has been the biennium 1998–1999, but that Italy has played a pioneering role, so that the late start of a shared European model has required some of the Italian regulations to be changed. For this reason, and given the impossibility to deal with all the different situations that are present on this subject in the various countries, the rest of this section will describe in more depth the Italian experience as a prototypical case study [34] that can be considered representative of the evolution of regulations on this subject and of the perspectives adopted by law on these technical matters. Of course, a complete treatment of the whole corpus of acts on the subject would be out of the scope of this section, and only a few most significant items will be presented.

In the following, the various acts will be identified by their category (using the Italian acronym where applicable[9]) plus their progressive number and/or date of issue, further specifying the relevant (sub)sections only when needed. They will be cited directly within the text, instead of providing separate pointers to them in the Reference section.

### *3.4.1 A Law Approach*

Although, as already noticed, no juridic definition of ‘document’ is available, the Italian law does provide a definition of what a *digital* document is: according to L 59/1997 (15 Mar.), that represents a milestone for the official adoption of digital technologies in formal transactions, a *digital document* is to be intended as the “digital representation of juridically relevant deeds, facts or data” (art. 1, par. 1). Hence, it can be identified in any sequence of bits or, more practically, in any computer file. Moreover, “The deeds, data and documents produced by the Public Administration or by privates using computer or telematic systems, the agreements contracted using the same means, as well as their filing and transmission using computer systems, are valid and relevant to all intents and purposes of law” (art. 15, par. 2). As a side-effect, this introduces the need for a secure way of signing them. Criteria and modalities for applying such a principle are stated in the DPR 513/1997 (10 Nov.), where several fundamental concepts are introduced and defined (art. 1), among which Certification and CAs:

[9]Talking of legal matters in different languages is not easy, due to the extremely different concepts, practices and procedures that have been developed and are currently in use in each country, and to the lack of a correspondence among them. A list and a short explanation of relevant law act categories in Italy is:

**L** (Legge) Act approved by the Parliament.

**DL** (Decreto Legge) Law by decree: an act provisionally issued by the Government, that needs formal approval by the Parliament.

**DPR** (Decreto del Presidente della Repubblica) Decree of the President of the Republic.

**DPCM** (Decreto del Presidente del Consiglio dei Ministri) Decree of the President of the Council of Ministers.

**DM** (Decreto Ministeriale) Decree of a Minister.

**Validation system** The computer and cryptography system that is able to generate and affix a digital signature or to check its validity.

**Asymmetric keys** The pair of (private and public) encryption keys, mutually related, to be exploited in the context of systems for validation or encryption of digital documents.

**Private key** The element of the pair of asymmetric keys, intended to be known only to its regular holder, by which the digital signature is affixed on a digital document or a digital document previously encrypted using the corresponding public key can be decoded.

**Public key** The element of the pair of asymmetric keys, intended to be made public, by which the digital signature affixed on the document by the regular holder of the asymmetric keys can be checked or digital documents to be transmitted to the regular holder of such keys can be encrypted.

**Certification** The outcome of the computer process that is applied to the public key and that can be checked by validation systems by which the one-to-one correspondence between a public key and the regular holder to which it belongs is guaranteed; the regular holder is identified and the period of validity of the key itself and the deadline of expiration of its related certificate are stated (in any case not more than three years).

**Temporal validation** The outcome of the computer process by which are assigned, to one or more digital documents, a date and a time that can be objected to third parties.

**Electronic address** The identifier of a physical or logical resource that is able to receive and record digital documents.

**Certifier** Public or private subject that carries out the certification, issues the public key certificate, publishes it along with the public key itself, publishes and updates the lists of suspended and revoked certificates.

The same act assigns to a digital document endowed with a digital signature the same legal validity according to the Civil Law (art. 2702) as a traditional paper document endowed with an autographic signature. These notions pave the way for the adoption of the digital signature as "an authentication mechanism that allows the author of a message to add a code that acts as a signature. The signature is created using the hash code of the message (that can be considered as its digital fingerprint) and encrypting it by means of the author's private key. The signature guarantees the origin and integrity of the message". In particular, mutual authentication protocols "allow the communicating entities to guarantee to each other their own identity and to exchange session keys"; one-way authentication, in contrast, just "guarantees to the receiver that the message actually comes from the indicated sender".

"Technical rules for the generation, transmission, preservation, duplication, reproduction and validation, also temporal, of digital documents" according to art. 3, par. 1 of this first regulation on digital signature were provided in the DPCM issued 8 Feb. 1999, where a *signature device* is defined as "an electronic device programmable only in its origin, capable of at least storing in a protected way the private key and generating inside of it digital signatures". DPR 445/2000 (28 Dec.) collected in a single text all regulations concerning administrative documentation, and to the juridic acknowledgment of the relevance and validity of digital documents and signatures added the requirements a digital document must fulfill.

After all these regulations, the European directive 1999/93/CE came with 15 articles and 4 attachments in which the requirements for qualified certificates, for the CAs that issue such certificates and for the secure signature devices are defined, and recommendations are given for the secure signature. Such a new framework required some changes to the Italian law, that were carried out by DL 10/2002 (23 Jan.), and implemented by DPR 137/2003 (7 Apr.) that assigns to digital documents the same validity as a legal proof as all other means recognized by the Civil Law (art. 2712). The DPCM issued 8 Feb. 1999 was updated by the DPCM issued 13 Jan. 2004, specifying the regulations for setting up public lists of certificates (art. 29, par. 1), while the DPCM issued 2 July 2004 defines the "Competence concerning electronic signature certifiers". Also related, although not specifically concerning digital signature, is the DPR 68/2004 (11 Feb.), concerning "Regulations including requirements for the exploitation of certified electronic mail", according to L 3/2003 (16 Jan.), art. 27.

The most recent significant contribution is represented by the DL 82/2005 (7 Mar.), titled "Digital Administration Act" (*Codice dell'Amministrazione Digitale*), as modified by the DL 159/2006 (4 Apr.), where the various concepts, already introduced, of digital signature, validation system, certification and asymmetric keys are explained, and the following kinds of signatures are defined, in order of increasing reliability (art. 1, par. 1):

**Electronic signature** The set of data in electronic form, attached or connected by means of a logic association to other electronic data, used as a means for digital identification according to the CE regulations.

**Qualified electronic signature** The electronic signature obtained by means of a computer procedure that guarantees its unambiguous association to the signee, created using means on which the signee can have an exclusive control and associated to the data to which it refers in such a way to allow pointing out whether the data themselves have been subsequently modified, that is based on a qualified certificate and carried out exploiting a secure device for the creation of the signature.

**Digital signature** A particular kind of qualified electronic signature based on a system of encryption keys, one public and one private, related to each other, that allows the legal owner using the private key and the addressee using the public key, respectively, to reveal and to check the provenance and the integrity of a digital document or of a set of digital documents.

The European regulation (art. 2) defines only the electronic signature and another kind, called *Advanced electronic signature*, that can be ranked between the electronic signature and the qualified one in the above list. The latter adds to the features of the former the integrity of the document and the exclusiveness of the signee. The qualified signature, although not formally defined in the CE Directive, is suggested by art. 3 (par. 4) and art. 5 thereof, and adds to the advanced one the use of a secure device and a certificate released by a qualified certifier. Finally, the digital signature adds to the qualified one the fact that the certifier is accredited: thus, the kind of signature required in Italy in 1997 (by DPR 513/1997, art. 10) was already stronger than what required by the European regulations issued two years later. Indeed, the digital signature derives from a digital document perspective on the subject, and hence has the strictest requirements and the largest validity; conversely, the other three kinds of signature derive from an electronic commerce perspective: they can be issued by any certifier (even not accredited, in which case their validity is limited to the organization in which they are exploited, but does not extend to the legal level). According to DL 82/2005, the former two kinds of signature can be only discretionally accepted as a proof, while the latter two kinds have full proving power: "A digital document, signed using a digital signature or other kind of qualified electronic signature, has the effect provided for by art. 2702 of the Civil Law. The use of the signature device is assumed to be traceable back to its owner, except if he gives proof of the contrary" (art. 21, par. 2).

According to the European regulations (art. 3) the certification services must not be authorized, but for application in the PA the single governments may add further requirements. In any case, the procedure of qualification is provided for, and the certifiers that are qualified in a country are automatically qualified in all other UE countries. A committee must be set up to check the fulfillment of security requirements by the signature devices and software (art. 9, 10). Qualified certificates issued by non-EU countries are valid if they fulfill the Directive requirements and there are agreements with the EU (art. 7), while DPR 513/1997 in Italy acknowledges them if they are operated under license or authorization of an EU country (art. 8, par. 4).

A very recent update in the field is represented by DPCM issued 30 Mar. 2009, reporting "Technical rules concerning digital signatures".

### *3.4.2 Public Administration Initiatives*

The responsibility for enforcing and supporting the widespread use of technologies in the Italian Public Administration and professional environment is in charge of the *DigitPA* Authority, recently established as a replacement of the former authority *CNIPA* (and still formerly *AIPA*) by DL 177/2009 (1 Dec.), concerning "Reorganization of the National Center for Computer Science in the Public Administration", in fulfillment of L 69/2009 (18 Jun.), art. 24. Its activities include Accessibility, Continuing Operation, Dematerialization, e-Government, internal efficiency of the PA, Digital Signature, Open Standards (encompassing Open Formats and Open Source software), Certified Electronic Mail, Digital Registry, Security, Territorial Information Systems, and others. Many important and successful activities have been carried out by the AIPA/CNIPA/DigitPA during the years.

**Digital Signature** First and most important, the Digital Signature initiative is one of the foundations for an e-Government framework. In the PA, its objectives proceed along three lines of intervention:

- Spreading of the digital signature in the Administration, by assigning one to all executives and officials and training them for its usage;

- Making secure remote applications and services;
- Stimulating the use of digital signature by groups of external users such as professionals, enterprises, etc.

DigitPA has issued various regulations on how to obtain and use digital signature, among which:[10]

- Circular 22/1999 (22 Jul.): enrollment modalities for the certifiers.
- Circular 24/2000 (19 Jun.): guidelines for certifiers interoperability (specifying regulations for digital certificates).
- Resolution 51/2000 (23 Nov.): preservation of digital documents.
- Circular 27/2001 (16 Feb.): enrollment procedures for the PA roll of CAs.
- Resolution 42/2001: preservation of digital documents, also on optical support.
- Circular 48/2005 (6 Sep.): procedure to submit an application for the public list of certifiers (envisaged in the DPR 445/2000 art. 28, par. 1).
- Resolution 4/2005 (17 Feb.): rules for the acknowledgment and verification of digital documents.
- Resolution 34/2006 (18 May): technical rules for the definition of the profile of encryption envelope for digital signatures in XML language.
- Resolution 45/2009 (21 May): rules for acknowledgment and verification of digital documents.

It acts as a root in a PKI, where private CAs can be accepted, provided they give assurance about their verifying the applicants' identity, their fulfilling the technical regulations of law, their not storing the users' private keys, and their being ISO-9002 certified. Thus, it signs the accepted CAs' certificates, while its certificate is self-signed. As of 2009, it has accredited 17 private CAs, the first of which were Infocamere (registered on 6 April 2000) and Postecom (registered on 20 April 2000). The number of authorized certifiers, and of existing projects and products suitable for document certification by means of digital signature, has requested a significant effort to reach a complete interoperability in the exploitation and the applications of digital signature. Nevertheless, the success obtained in Italy (nearly 4 millions of digital signatures assigned, the largest amount all over Europe) proves that it is possible to obtain a free and secure exchange of digital documents even in official environments.

Specifically, three different kinds of keys for digital signatures are provided for normal subscription of documents, certification (subscription of certificates) and timestamping. As to the last one, the DPCM 52/1999 (8 Feb.) defines the following procedure: first, the digital document fingerprint is generated and sent to the CA, that generates the timestamp according to an international standard (art. 53) and subsequently signs the resulting file. The signed timestamp must be produced within one minute from the request, and stays valid for one month. Different keys must be exploited to produce different timestamps, even for the same fingerprint.

[10] A *circular* is a document reporting clarifications and directions on how a new act is to be interpreted and applied.

Computers or other removable storage supports provide too little security to be of use for storing digital signature data. For this reason, the traditional means used for this purpose is a *Smart Card*, a plastic card similar to a Credit Card, that can be easily carried in a pocket, and that can be activated only using a secret Personal Identity Number (PIN) known exclusively to its owner. It contains a microchip capable of generating the keys and applying the digital signature, by performing a cryptographic algorithm that, starting from the (currently 160-bit long) digest of a document, produces a (currently 160-bit long) string signed with the private key. Smart cards are built in such a way that export or copy of the private key from their storage to the outside is precluded. The smart card is provided with a "signature kit" that includes all the needed hardware and software, i.e.,

- A *smart card* (or other secure device, such as a USB key) containing the signature certificate and the secret key, to be activated using a PIN;
- A *smart card reader*, generally acting on a USB, COM or even WiFi port, that allows a computer to exchange information with the smart-card chip;
- The *digital signature management software* that supports the user in digitally signing documents and in verifying digitally signed ones;
- The *software that allows importing the certificates* of digital signature and authentication (i.e., the non-secret part of the digital signature system) in the browser's cache to be used during the Internet browsing or by the e-mail client.

The user must preliminarily install the smart-card reader (by loading on the PC the needed drivers), connect the reader to a computer port, install the signature and the import software. Then, he must actually read from the smart-card, import and store the certificates in the Internet or e-mail client. Recently *cryptographic tokens* became available, USB keys that work without the need for all such preparation steps, by directly providing the needed software (instead of installing it on the PC).

The formal procedure users must carry out for obtaining a digital signature and related certificate is as follows: after sending to a CA proper documentation aimed at assessing his identity, the applicant obtains a signature kit by which he generates the keys using the included devices, according to a procedure defined by the CA. Then, he requests a certificate for the public key generated this way; the CA checks that such a public key is not already certified by other CAs and asks the user to prove possession of the corresponding private key, by subscribing some test documents; if the test succeeds, the certificate is generated (according to DPCM 8 Feb. 1999, art. 42), published in the register of certificates, sent to the applicant along with a timestamp and a reserved code for revoking the certificate, and finally recorded in the control journal.

**Certified e-mail** A complementary initiative is the *Certified Electronic Mail* (Posta Elettronica Certificata, *PEC*), an e-mail system in which the sender obtains electronic documents that prove, with full legal validity, the outcome of two fundamental moments in the transmission of digital documents, i.e., their posting and delivery. As to the former, the provider of the certified e-mail service will provide the sender with a receipt that confirms (and is a valid legal proof of) the successful mailing of a message, also specifying possible related attachments. In the same way, when the message reaches the addressee, the provider will give the sender a receipt confirming the success (or failure) in its delivery, with a precise temporal information. In case the sender loses the receipts, a digital track of the operations performed, stored by law for a period of 30 months, allows the reproduction, with the same juridic value as the originals, of the receipts themselves. Related regulations are:

- DPR 68/2005 (11 Feb.): regulations for the exploitation of certified electronic mail (according to L 3/2003 16 Jan., art. 27);
- DM 2 Nov. 2005: technical rules for the production, transmission and validation, also temporal, of certified electronic mail;
- DPCM 6 May 2009: provision concerning release and exploitation of certified electronic mailbox assigned to citizens.

Starting 2010, all professionals registered in National rolls are obliged by law to own a certified electronic mailbox.

**Electronic Identity Card & National Services Card** The *Electronic Identity Card* (Carta d'Identità Elettronica, CIE) is an initiative that aims at providing all Italian citizens with a plastic card endowed with a magnetic stripe and a microprocessor. The former is devoted to store the identity data (also graphically reported on the card), such as personal data and Fiscal Code,[11] blood group, health options, but also predisposed to store information needed to generate biometric keys, digital signature, electoral certification, etc. The microchip is aimed at acting as a Service Card for the identification of the owner on the Internet and the fruition of telematic services, such as electronic payments between privates and the PA, etc. Relevant regulations are:

- L 127/1997 (15 May) art. 2, par. 10;
- L 191/1998 (16 Jun.) art. 2, par. 4;
- DPCM 437/1999 (22 Oct.);
- DM (Interior) 19 Jul. 2000 (technical and security rules).

The *National Services Card* (Carta Nazionale dei Servizi, CNS) initiative also consists of a plastic card released to citizens in order to electronically identify them on the Internet by means of data stored on an embedded microprocessor. Differently from the CIE, however, it has no writings on it, so it cannot be used for direct personal identification. Since the format in which information is stored on CNSs is compatible to that of CIEs, CIEs are going to completely replace them.

**Telematic Civil Proceedings** An example of demanding on-field application of the above regulations, and of the strong commitment of the Italian PA towards the pervasive exploitation of digital technologies, is the *Telematic Civil Proceedings* (Processo Civile Telematico, PCT) initiative. It represents one of the most relevant

[11] An alphanumeric code assigned from the Italian PA to all resident people, initially for taxpaying purposes, but then for unique identification, in general.

e-Government projects started in Italy, aimed at improving communication between the people involved in the Civil Proceedings by introducing the Information and Communication Technologies (ICT). Two categories of persons involved in the process can be distinguished: the *public* parties (e.g., chancellors, judges, bailiffs, State and Public Administration advocacies), usually acting in the Judicial Office, and the *private* ones (lawyers, Court and Party Technical Advisors, notaries, etc.) that need to interact with the Judicial Office from the outside to exploit its services or to accomplish their tasks. The PCT establishes, defines, regulates and organizes the modality by which judicial documents in electronic format are produced, recorded, notified, consulted and exploited by both the public and the private parties using telematic and computer systems. Advantages of the new setting include delocalization, extension of offices availability up to 24 hours a day, complete and timely checks, automatic export of the data for the management software and electronic agenda of the lawyers' offices, no need for intermediate personnel, cost reduction.

The PCT[12] [22] is based on the following regulations:

- DPR 123/2001 (13 Feb.) concerning "Regulation of the use of computer and telematic instruments in the civil and administrative proceedings, and in the proceedings before the jurisdictional sections of the Accounting Law-court".
- DM (Justice) 14 Oct. 2004 and DM 17 Jul. 2008 reporting "Technical and operating rules for the exploitation of computer and telematic instruments in the civil proceedings".
- DM 27 Apr. 2009 specifying "New procedural rules concerning the keeping of digital registers in the Justice Administration".
- DM 10 Jul. 2009 defining the "Adoption of specifications for structuring digital models" as required by art. 62, par. 2, of the DM 17 Jul. 2008.

According to the DPR 123/2001, it is "permitted the making, communication and notification of deeds in the Civil Proceedings by means of digital documents, in the ways provided by the present regulation. The activity of transmission, communication or notification of the digital documents is carried out through telematic technology using the Civil Information System". The whole project structure of the PCT is centered on the transformation of the traditional paper deed in its equivalent (as regards juridical reliability and validity) electronic counterpart. The latter is not just a word processor-made file, but a file digitally signed using an asymmetric key cipher according to the rules provided by the DPR 445/2000. Indeed, a digitally signed document is in every respect considered juridically equivalent to an autographically signed paper document, since the counterpart can be sure of the authorship and integrity of the file by using the public key of its creator. For this reason, involved people must be acknowledged according to an authentication procedure that once again exploits an asymmetric key cryptographic system.

The communication between a private party and the involved Judicial Office takes place as depicted in Fig. 3.1. The private party connects to the public Internet, and hence is totally free of any temporal and spatial bias for carrying out the desired

[12]Website http://www.processotelematico.giustizia.it/pdapublic/.

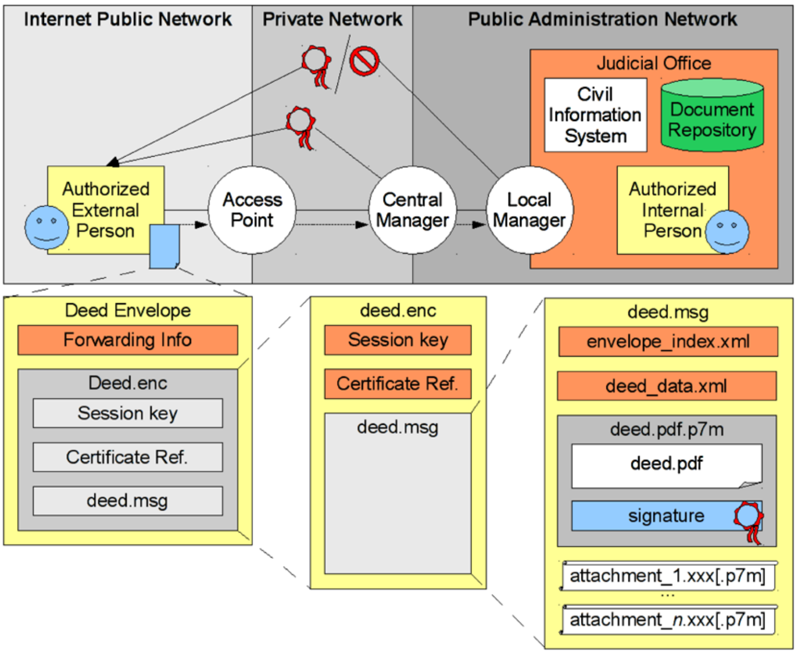


**Fig. 3.1** Organization of the data-transmission-based civil proceedings and structure of a deed

procedure. The gate towards the protected area is represented by an authorized *Access Point* that is acknowledged by a *Central Manager* and deals with it through a Private Network. The data generated by an Authorized External Person (the private party) are passed by the Access Point to the Central Manager that, in turn, forwards them through the Public Administration (private) Network to the *Local Manager* of the Judicial Office in question. Such a Local Manager deals with the Document Repository and the Civil Information System of the office, and can deliver the information to the proper Authorized Internal Person of the office (the public party), if needed.

A fundamental step in the implementation of the PCT is the certification process that underlies the delivery of the telematic services such as the Access Points. Also private organizations can be authorized by the DGSIA (Direzione Generale SIA Ministero Giustizia) to act as Access Points. Indeed, each private party must be registered to one (and only one) Access Point, either that instituted by its Order Council or by delegating to the National Forensic Council (CNF), if certified as Access Point, or by delegating a private organization certified by the Department of Justice for fulfilling the needed ‘moral’ and technical requirements, or choosing directly a certificated Access Point. The Access Points, whose homepages are available on the Internet, provide any indication to properly configure the Certified Electronic Mail of the PCT (PECPCT), and the other software to perform the telematic connections of the PCT. They also provide the software released by the Department of Justice, and publish all the lists, data and information needed or useful for the proceedings.

All connections (both for consultation and for transmission and recording of deeds) pass through the Access Point.

Before being able to exploit the services of the PCT through the Internet, a private party must be registered to the system, own a Certified e-mail Box and a digital signature issued by a CA acknowledged by DigitPA. The digital signature smart-card contains both a *signature certificate* and an *authentication certificate*. The private party uses the former to digitally sign the proceedings (digital) deeds, and the latter when, connecting to the telematic proceedings services to consult the record-office records and the proceedings dossiers or to deposit acts, needs to be identified in a secure way. The latter is similar to the former, but its role is different: it allows the identification of who is connected to the Internet. Indeed, because of the way the Internet was designed, one cannot be sure about the physical person that in a given moment is connected from a computer to another to exchange information or use services. Using the authentication certificate and the corresponding smart-card it is possible to identify the subjects that are carrying out a telematic transaction. On its side, the CA publishes the certificate on the Internet, guarantees that it actually corresponds to the identifying data of its owner, and ensures its validity in time (indeed, it could be suspended or revoked before its normal expiration).

From its homepage, the private party logs in and allows the system to recognize him by activating the authentication certificate contained in his smart-card. He puts his smart-card in the reader connected to the computer and unlocks it by entering his PIN. The system identifies the private party, checks that he is entitled to proceed (e.g., if he is a lawyer he must not be suspended, canceled or struck off from the Law List roll), and allows him to accomplish his task. For simple consultation, he can access all the data (records, documents and deeds), deadlines, events happened and everything else concerning the proceedings he is involved in (a service called *PolisWeb*). He can also get a copy of the documents wherever he is.

Asynchronous services, such as recording of a deed, have a more complex management. The process from the creation of the document to its transmission involves several steps, as depicted in the lower part of Fig. 3.1:

1. **Creation** The private party produces a document (say `deed.xxx`) using a *writer* software that either exploits a third-party Word Processor (such as Microsoft Word or OpenOffice Writer) or provides one by itself (in both cases predefined deed templates are available). It can also connect to the database of a software for the management of the lawyer's office.
2. **Export in the required format** If the deed was not produced in one of the required formats (XML or, more recently, PDF) the document is exported into one of them (e.g., `deed.pdf`).
3. **Digital signature** The deed is digitally signed through a procedure activated by the writer program (obtaining a file `deed.pdf.p7m`): from that moment on, it cannot be modified any further (in case, modifications are to be made on the original document and a new version must be produced and signed).
4. **Attachment of documents and Folding** The deed is inserted into an 'envelope' along with the attachments, if any (`attachment_`*n*`.xxx[.p7m]`). The envelope must be encrypted using the public key of the Judicial Office to which the

deed is sent for recording (`deed.enc`). Such a key must be manually downloaded from the Website of the Department of Justice; it is foreseen that forthcoming versions of the writer program will take care of getting it from lists published by the Access Points or other official sites, and of the download and updating (they usually expire after 2–3 years) of the certificates.
5. **Fold encoding** The bundle of the deed and its attachments, after being encrypted, becomes a single attachment to the e-mail message that contains also any useful routing information (in XML format).
6. **Fold sending** The e-mail message prepared this way by the software can be simply sent through the Access Point.

Some of these steps are carried out, automatically and transparently to the lawyer, by the *writer* software.

The Access Point, after identifying the lawyer (using the identification certificate), verifies that no obstacles exist for his action and forwards the message, through the Central Manager, to the Local Manager of the Judicial Office where it is to be recorded. As soon as the Central Manager server undertakes the message, an e-mail message of confirmation is automatically generated and sent to the sender through the Access Point. This will be a proof of deposit during the proceedings. In the meantime, the message containing the deed is forwarded to the Local Manager that, after decoding it, checks that it is free of viruses or malware, verifies its completeness, its formal correctness, the availability of suitable references for associating it to the proper proceeding (if already existing) and, in the end, in case of positive outcome sends a message that definitively validates the automatic procedure, or otherwise notifies the failure along with possible causes. Clearly, the successful completion of the procedure is proved by the two messages sent, respectively, by the Central and Local Manager.

## References

1. Data Encryption Standard. Tech. rep. FIPS Pub. 46-1, National Bureau of Standards, Washington, DC, USA (1987)
2. Secure Hash Standard (SHS). Tech. rep. FIPS PUB 180, National Institute of Standards and Technology (1993)
3. Secure Hash Standard (SHS). Tech. rep. FIPS PUB 180-1, National Institute of Standards and Technology (1995)
4. Digital Signature Standard (DSS). Tech. rep. FIPS PUB 186-1, National Institute of Standards and Technology (1998)
5. Merriam-Webster's Collegiate Dictionary, 10th edn. Merriam-Webster Inc. (1999)
6. Secure Hash Standard (SHS)—amended 25 February 2004. Tech. rep. FIPS PUB 180-2, National Institute of Standards and Technology (2002)
7. Secure Hash Standard (SHS). Tech. rep. FIPS PUB 180-3, National Institute of Standards and Technology (2008)
8. Digital Signature Standard (DSS). Tech. rep. FIPS PUB 186-3, National Institute of Standards and Technology (2009)
9. Abdul-Rahman, A.: The PGP trust model. EDI-Forum (1997)

10. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP message format. Tech. rep. RFC 4880, IETF (2007)
11. Carnelutti, F.: Studi sulla sottoscrizione. Rivista di Diritto Commerciale, p. 509 ss. (1929) (in Italian)
12. Cocks, C.C.: A note on 'non-secret encryption'. Tech. rep., GCHQ (1973)
13. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, T.: Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) profile. Tech. rep. RFC 5280, Internet Engineering Task Force (IETF) (2008)
14. Department for Culture, Media and Sport, Department for Business, Innovation and Skills: Digital Britain—final report. Tech. rep., UK Government (2009)
15. Diffie, W.: An overview of public key cryptography. IEEE Communications Society Magazine **16**, 24–32 (1978)
16. Diffie, W.: The first ten years of public-key cryptography. Proceedings of the IEEE **76**(5), 560–577 (1988)
17. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**, 644–654 (1976)
18. Diffie, W., Hellman, M.: Privacy and authentication: an introduction to cryptography. Proceedings of the IEEE **67**, 397–427 (1979)
19. Ellis, J.H.: The possibility of secure non-secret digital encryption. Tech. rep., GCHQ (1970)
20. Feistel, H.: Cryptography and computer privacy. Scientific American **128**(5) (1973)
21. Garfinkel, S.: PGP: Pretty Good Privacy. O'Reilly (1994)
22. Gattuso, A.: Processo telematico. Mondo Professionisti **I**(13), III–VI (2007) (in Italian)
23. Kaliski, B.: Pkcs #7: Cryptographic message syntax. Tech. rep. RFC 2315, IETF (1998)
24. Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires **IX**, 5–38 (1883)
25. Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires **IX**, 161–191 (1883)
26. Lai, X.: On the Design and Security of Block Ciphers. ETH Series in Information Processing, vol. 1. Hartung-Gorre (1992)
27. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
28. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, Cambridge (2007)
29. Rivest, R.: The MD5 message-digest algorithm. Tech. rep. RFC 1321, Network Working Group (1992)
30. Singh, S.: The Code Book. Doubleday, New York (1999)
31. Sorkin, A.: Lucifer a cryptographic algorithm. Cryptologia **8**(1), 22–24 (1984)
32. Stallings, W.: Cryptography and Network Security. Principles and Practice, 3rd edn. Prentice Hall, New York (2002)
33. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., Weger, B.D.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology. Lecture Notes in Computer Science, vol. 5677, pp. 55–69. Springer, Berlin (2009)
34. Tosi, E.: Il codice del Diritto dell'Informatica e di Internet, VI edn. (2007) I codici vigenti. La Tribuna (in Italian)
35. Zimmermann, P.R.: The Official PGP User's Guide. MIT Press, New York (1995)

# Part II
# Document Analysis

A critical phase towards the extraction and fruitful exploitation of the information contained in digital documents is the analysis of their structure. According to the intrinsic properties and spatial arrangement of the various layout components, it aims at determining the type of document as a whole and, then, which parts of it deserve further specific attention. This activity involves both low-level tasks, dealing with perceptual aspects, and higher-level ones, closely related to the semantics and roles played by the objects of interest.

While the semantics of text is explicit, in images concepts are implicit and the perception aspect is totally predominant. The image processing issue, that represents a whole branch of Computer Science in its own, is of great interest in document processing both because digitized documents are images by themselves, in which relevant components are to be identified, and because some document components are in fact images, from which interesting information is to be extracted. Chapter 4 introduces a set of techniques that can be profitably exploited in the specific area of interest of document processing. First color-related tools are presented, that allow to switch to a more comfortable representation depending on the task at hand or to progressively decrease the color and luminance information up to black&white. Then, general methods for manipulating the image content are proposed, ranging from geometrical transformation, to edge enhancement, to connected components processing. Finally, some techniques to find refined edges in an image, as a step towards the identification of the represented object, are dealt with.

One of the main distinguishing features of a document is its *layout*, as determined by the organization of, and reciprocal relationships among, the single components that make it up. For many tasks, one can afford to work at the level of single pages, since the various pages in multi-page documents are usually sufficiently unrelated to be processed separately. Chapter 5 discusses the processing steps that lead from the original document to the identification of its class and of the role played by its single components according to their geometrical aspect: digitization (if any), low-level pre-processing for documents in the form of images or expressed in term of very elementary layout components, optical character recognition, layout analysis and document image understanding. This results in two distinct but related structures for a document (the layout and the logical one), for which suitable representation techniques are introduced as well.

# Chapter 4
# Image Processing

Text has an important advantage from an information communication perspective: if it represents a significant expression in a known language, its semantics is (at least in most cases) explicit. On the other hand, in images concepts are implicit and the perception aspect is totally predominant. Just for this reason, from a computer processing perspective objects of the latter kind are very complex to be dealt with. Indeed, focusing on their basic components (points or geometrical shapes), compared to those of text (symbols or words), the gap separating syntax from the corresponding semantic interpretation is very wide (as supported by considerations in [5, pp. 25 ff.]) because words have been purposely defined by humans to carry a high-level meaning, whereas shapes are by themselves unlabeled and much more variable. As an additional problem, the visual perception and understanding procedures that take place in humans are not fully understood, yet.

The image processing issue is of great interest in the wider landscape of document processing, from a twofold perspective. On the one hand, it is needed as a preliminary phase that supports the layout analysis of digitized documents, in order to step from the raw pixel-level representation to the level of more significant aggregates as basic components. On the other hand, after document image analysis and understanding have been performed, it is to be applied to interesting components that are images in order to index them or extract from them useful information related to the document content. Although the specific image processing techniques typically exploited in these two steps are in large part different, nevertheless some significant overlap exists (e.g., color handling and edge finding algorithms can find thorough application in both). This suggested providing an introduction to this subject separate from its particular applications, resulting in the present chapter that can be useful as a single common reference for concepts that come into play in either aspect of document processing. However, the subject of *Image Processing* represents a whole branch of Computer Science in its own, and hence might deserve a much deeper treatment, as can be provided by specific literature references such as [1, 3, 11].

## 4.1 Basics

An *image* can be formally represented as a mathematical function $I$ that maps each pair of coordinates $(x, y)$ in a (finite) 2D space of size $m \times n$ to a unique value $I(x, y)$. Informally, the coordinates represent a spot in the image, while the function value represents the associated color. In the discrete perspective of raster images, the coordinates are integers ranging in $[0, n-1] \times [0, m-1]$. Working on images (e.g., for extracting information from them) often requires that they are properly treated in order to turn them into a representation that is more useful for the intended processing aims. In such a context, vector graphics are based on operators that describe lines, curves, characters or other geometrically expressible shapes, and hence explicitly denote noteworthy subsets of related points in the image. This higher level of abstraction, in turn, surely makes easier the interpretation of the image content. Conversely, in the case of raster graphics, each single point that makes up the image is syntactically unrelated to all the others, although semantically it clearly is not. This adds significant complexity to image interpretation, and requires additional efforts to identify the implicit relationships among points in an image, but represents the most frequent situation to be handled in document processing.

Image representations, and particularly those coming from real-world production procedures and devices, can be affected by unwanted *noise* that overlaps the original signal and distorts it, changing as a consequence the correct function values. In this event, the image must be cleaned up of such a noise. In other cases, some features of the image are to be enhanced. This can be obtained by exploiting proper operators. *Image transformation operators* are functions that, applied to an image $I_{[m \times n]}$, transform it into another one $I'$, usually having the same size. They can be grouped into three categories, according to whether the output value at the point $(x, y)$ (called in the following the 'reference') depends on:

- The input value at the $(x, y)$ point alone (*point operators*);
- The input values in a neighborhood of the $(x, y)$ point alone (*local operators*);
- All the values in the input image (*global operators*).

Any task that is carried out on an image can be expressed in terms of such operators.

**Convolution and Correlation** Given two functions $I$ and $M$ having the same dimensionality (the two-dimensional, discrete case is of interest for image processing), the mathematical operation of *linear convolution* (often denoted by $I \star M$) is defined as:

$$I'(u, v) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} I(u - j, v - j) \cdot M(i, j).$$

Working on raster images, a particular case of it is exploited, in which $I$ is associated to an $m \times n$ image matrix (containing the values of the pixels that make up the image), and $M$ is associated to another (usually smaller) $m' \times n'$ matrix (called the *mask*, or *filter*, or *kernel*). $I$ and $M$ take on the values of the corresponding matrix for all coordinate pairs inside its boundaries, and zero for all the others. Usually, the filter sides have odd length ($m' = 2m'' + 1$, $n' = 2n'' + 1$) and the origin of its coordinates is assumed to be in the centroid position ($R_M = [-m'', m''] \times [-n'', n'']$), called the *hot spot*. This yields the *linear correlation* operation (technically, a convolution with a reflected filter matrix [1]):

$$I'(u, v) = \sum_{(i,j)\in R_M} I(u + j, v + j) \cdot M(i, j).$$

Intuitively, $M$ is overlap to $I$ in all possible ways[1] and, for each overlapping position, a value is generated as a linear combination of the pairs of values in cells that are associated by the current overlapping. Each such value is assigned to the element in the resulting matrix/image $I'$ corresponding to the hot spot position.

The actual values in the kernel to be exploited depend on the effect one aims at obtaining. A typical shape of the kernel is a square of size $3 \times 3$ ($M_{3\times 3}$), as follows:

| | | |
|---|---|---|
| $w_1$ | $w_2$ | $w_3$ |
| $w_4$ | $w_5$ | $w_6$ |
| $w_7$ | $w_8$ | $w_9$ |

Given a generic portion of the image $I$ that happens to overlap the kernel for one of its possible displacements, consisting of the neighborhood of the reference $I(x, y)$:

| | | |
|---|---|---|
| $I(x-1, y-1)$ | $I(x, y-1)$ | $I(x+1, y+1)$ |
| $I(x-1, y)$ | $I(x, y)$ | $I(x+1, y)$ |
| $I(x-1, y+1)$ | $I(x, y+1)$ | $I(x+1, y+1)$ |

application of the operator returns the following value that is assigned, in the transformed image $I'$, to the position of the reference:

$$\begin{aligned} I'(x, y) = {} & w_1 \cdot I(x-1, y-1) + w_2 \cdot I(x-1, y) + w_3 \cdot I(x-1, y+1) \\ & + w_4 \cdot I(x, y-1) + w_5 \cdot I(x, y) + w_6 \cdot I(x, y+1) \\ & + w_7 \cdot I(x+1, y-1) + w_8 \cdot I(x+1, y) + w_9 \cdot I(x+1, y+1). \end{aligned}$$

Note that in practice the kernel output must be saved on a separate image, not to affect the next applications of the kernel after having changed a pixel value. Note also that when the pixels on the border of $I$ play the role of references, the border of the mask would fall out of the image, and hence the mask would not be fully exploited on them; for this reason, often they are reported in $I'$ as they are in $I$, without any transformation.

[1]Operationally, this can be obtained, e.g., by starting from the top-left position of the latter and sequentially moving it to all next positions top-down, left-to-right.

*Example 4.1* (Linear correlation of an image) Consider a $5 \times 5$ matrix $I$ representing a gray-level image, whose elements correspond to the pixel values, a $3 \times 3$ filter $M$ and the resulting matrix/image $I'$, as follows:

$$I = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \mathit{1} & \mathit{3} & \mathit{0} & 0 \\ \hline 0 & \mathit{3} & \mathit{4} & \mathit{0} & 0 \\ \hline 0 & \mathit{0} & \mathit{0} & \mathit{0} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \qquad M = \begin{array}{|c|c|c|} \hline 2 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \qquad I' = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & \mathbf{2} & 6 & 0 \\ \hline 0 & 0 & 6 & 8 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

The effect of the filter consists in moving each pixel one position below and to the right from the original image, and doubling its intensity. Indeed, the value taken by the reference pixel when applying the kernel corresponds to twice the value of the pixel in the image associated to the top-left corner of the mask, while all other pixels are ignored being multiplied by 0. For instance, applying the mask over the emphasized positions in $I$ returns the value to be set in the position of $I'$ reported in bold, as follows:

$$(1 \cdot 2) + (3 \cdot 0) + (0 \cdot 0) + (3 \cdot 0) + (4 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) = 2.$$

The mask is applied 9 times, to the emphasized positions in $I$.

## 4.2 Color Representation

As already pointed out, there are several agreements and standards, called *color spaces*, according to which colors can be represented. Each one emphasizes different perspectives on the colors, and hence turns out to be more suitable to support some kinds of processing and less suitable to other applications. Throughout this section, the color components will be assumed (unless otherwise stated) to range in $[0, 1]$, but a transposition to other intervals $[0 \dots M_c]$ (e.g., the more standard 8-bit representation $[0 \dots 255]$ traditionally adopted in Computer Science) is clearly straightforward:

$$c \in [0, 1] \to c' = c \cdot M_c \in [0, M_c]$$

(possibly rounded if integer values are to be considered).

An example of a point operator is the *negative* of a pixel value expressed through $n_c$ color components $c_i \in [0, 1]$ that yields a new color whose components are $c'_i = 1 - c_i$ (where $i = 1, \dots, n_c$).

### *4.2.1 Color Space Conversions*

Being able to switch from a representation to another is often a key factor towards effective solutions of image-related problems. In the following, the most important conversions between relevant color spaces will be presented.

**RGB–YUV** The amount of luminance in an RGB color corresponds to the average of the three channel values:

$$Y = \frac{R+G+B}{3}.$$

However, since the human eye perceives red and green as being brighter than blue, a properly weighted average turns out to be more effective. The usual weights are as follows:

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \in [0, 1].$$

To convert from RGB to YUV, two more values must be computed:

$$U = \frac{B - Y + 0.886}{1.772} \in [0, 1],$$
$$V = \frac{R - Y + 0.771}{1.542} \in [0, 1],$$

or, as reported in [1],

$$U = -0.147 \cdot R - 0.289 \cdot G + 0.436 \cdot B = 0.492 \cdot (B - Y),$$
$$V = 0.615 \cdot R - 0.515 \cdot G - 0.1 \cdot B = 0.877 \cdot (R - Y).$$

The inverse conversion is obtained as:

$$R = Y + 1.14 \cdot V,$$
$$G = Y - 0.395 \cdot U - 0.581 \cdot V,$$
$$B = Y + 2.032 \cdot U.$$

**RGB–YC$_b$C$_r$** $YC_bC_r$ is obtained from an $(R, G, B)$ color triple by computing $Y$ as above and the other components as follows:

$$C_b = -0.169 \cdot R - 0.331 \cdot G + 0.5 \cdot B,$$
$$C_r = 0.5 \cdot R - 0.419 \cdot G - 0.081 \cdot B,$$

while the inverse is:

$$R = Y + 1.403 \cdot C_r,$$
$$G = Y - 0.344 \cdot C_b - 0.714 \cdot C_r,$$
$$B = Y + 1.773 \cdot C_b.$$

**RGB–CMY(K)** In CMY, since the three subtractive basic colors are the complements of the additive ones, one gets:

$$\left.\begin{array}{l} C = 1 - R \\ M = 1 - G \\ Y = 1 - B \end{array}\right\} \quad \Rightarrow \quad \left\{\begin{array}{l} R = 1 - C \\ G = 1 - M \\ B = 1 - Y. \end{array}\right.$$

In the *CMYK* color space, the additional $K$ black component is defined as:

$$K = \min(C, M, Y).$$

Once black is available, it may contribute in defining not only gray level colors, but other colors as well, this way allowing to reduce the intensity of the other $C$, $M$, $Y$ components accordingly:

$$C_K = C - K,$$
$$M_K = M - K,$$
$$Y_K = Y - K.$$

**RGB–HSV** To switch from an RGB triple $(R, G, B)$ to the corresponding HSV representation, some calculations are needed that depend on its maximum $M = \max(R, G, B)$ and minimum $m = \min(R, G, B)$ values, and on their difference $r = M - m$ representing the range of variability for the component values of that color. Particular cases are those of gray levels, where $R = G = B \Leftrightarrow r = 0$, and specifically that of black, where $R = G = B = 0 \Leftrightarrow M = m = r = 0$.

Concerning Saturation $S$ and Value (i.e., luminance) $V$, the computation is straightforward:

$$S = \begin{cases} r/M & \text{if } M > 0, \\ 0 & \text{otherwise,} \end{cases}$$
$$V = M.$$

As to Hue, the conversion is more complicated, depending on the value of $r$. First of all, the hue is undefined for gray levels, where $r = 0 \Rightarrow S = 0$. In all other cases, when $r > 0$, a *normalization* step with respect to such a distance is performed:

$$R' = \frac{M - R}{r}, \qquad G' = \frac{M - G}{r}, \qquad B' = \frac{M - B}{r}$$

from which a provisional hue value is obtained:

$$H' = \left\{\begin{array}{ll} B' - G' & \text{if } R = M \\ R' - B' + 2 & \text{if } G = M \\ G' - R' + 4 & \text{if } B = M \end{array}\right\} \in [-1, 5]$$

to be normalized as follows:

$$H = 1/6 \cdot \begin{cases} H' + 6 & \text{if } H' < 0, \\ H' & \text{otherwise.} \end{cases}$$

Expressed as an angle, measured in degrees, it simply amounts to $H \cdot 360°$.

Back from HSV to RGB, first the appropriate color sector is to be determined as

$$H' = (6 \cdot H) \bmod 6 \in [0, 6[$$

according to which some intermediate values must be computed:

$$c_1 = \lfloor H' \rfloor, \qquad c_2 = H' - c_1,$$
$$x = (1 - S) \cdot V, \qquad y = \big(1 - (S \cdot c_2)\big) \cdot V, \qquad z = \big(1 - \big(S \cdot (1 - c_2)\big)\big) \cdot V,$$

to finally obtain:

$$(R, G, B) = \begin{cases} (V, z, x) & \text{if } c_1 = 0, \\ (y, V, x) & \text{if } c_1 = 1, \\ (x, V, z) & \text{if } c_1 = 2, \\ (x, y, V) & \text{if } c_1 = 3, \\ (z, x, V) & \text{if } c_1 = 4, \\ (V, x, y) & \text{if } c_1 = 5. \end{cases}$$

**RGB–HLS** From RGB to HLS, considering $M$, $m$ and $r$ as for the RGB-to-HSV conversion:

$$H \text{ is as for the RGB-to-HSV conversion,}$$
$$L = \frac{M + m}{2},$$
$$S = \begin{cases} 0 & \text{if } L = 0 \vee L = 1, \\ 0.5 \cdot \frac{r}{L} & \text{if } L \in ]0, 0.5], \\ 0.5 \cdot \frac{r}{1-L} & \text{if } L \in ]0.5, 1[. \end{cases}$$

Back from HLS to RGB, the computation is straightforward for the two extremes

- black ($L = 0$): $(R, G, B) = (0, 0, 0)$,
- white ($L = 1$): $(R, G, B) = (1, 1, 1)$,

while in all other cases the values $H'$, $c_1$ and $c_2$ are to be first determined as for the RGB-to-HSV conversion, and additionally:

$$d = \begin{cases} S \cdot L & \text{if } L \leq 0.5, \\ S \cdot (L - 1) & \text{if } L > 0.5, \end{cases}$$

in order to compute:

$$(R, G, B) = \begin{cases} (w, z, x) & \text{if } c_1 = 0, \\ (y, w, x) & \text{if } c_1 = 1, \\ (x, w, z) & \text{if } c_1 = 2, \\ (x, y, w) & \text{if } c_1 = 3, \\ (z, x, w) & \text{if } c_1 = 4, \\ (w, x, y) & \text{if } c_1 = 5, \end{cases}$$

where $w = L + d$, $x = L - d$, $y = w - (w - x) \cdot c_2$, $z = x + (w - x) \cdot c_2$.

### 4.2.2 Colorimetric Color Spaces

In addition to the classical spaces purposely developed for compliance with existing image acquisition or rendering hardware (such as scanners, displays or printers), the pure processing perspective on images has called for the definition of abstract color spaces whose features are device-independent and, possibly, more closely resemble human perception. Definition of a standard for such a theoretical representation, independent of physical devices, is in charge of the CIE.

**XYZ** The *XYZ* color space is named after the three abstract parameters $X$, $Y$ and $Z$ (where $Y$ is the luminance, as usual) that it uses as a base such that any color can be obtained as a summation of their components. It turns out to be a cone-shaped space that has the black $B$ in its origin and that does not include the primary colors. It is non-linear with respect to human visual perception.

Color hues in this space are defined as follows:

$$x = \frac{X}{X + Y + Z},$$
$$y = \frac{Y}{X + Y + Z},$$
$$z = \frac{Z}{X + Y + Z}.$$

Since $x + y + z = 1$, only two coordinates (usually $x$ and $y$) are sufficient to denote a color, while the third one ($z$) can be just derived by differencing.

Usually, a special focus is given on the plane $X + Y + Z = 1$ that corresponds to a triangle in the XYZ space. Consider any conceivable color point $C = (X_C, Y_C, Z_C)$, and the $\overline{BC}$ segment joining it with the origin (representing black). Then, the point of intersection between the aforementioned segment and plane/triangle determines the chromatic coordinates $c = (x_c, y_c, z_c)$ associated to $C$ that overall represent its hue and saturation. Now, the $Z$ coordinate $z_c$ can be just dropped, thus obtaining the projection of such a point on the $X$–$Y$ plane.

Since infinitely many colors correspond to such coordinates, depending on all possible luminance values, in order to go back to the original color, additional information about its luminance $Y$ must be provided. Starting from such a $(Y, x, y)$ triple, for $y > 0$, the remaining values are obtained as:

$$X = x \cdot Y/y, \qquad Z = z \cdot Y/y = (1 - x - y) \cdot Y/y.$$

The gray levels lie at the *neutral point* $X = Y = Z = 1$, i.e., $x = y = 1/3$, and zero saturation. XYZ can be turned to RGB as a simple linear transformation.

**L*a*b*** Another abstract color space for image processing is $L^*a^*b^*$, where $L^*$ denotes the luminance, while the color components $a^*$ and $b^*$ specify the hue and saturation features referred to the green–red axis and to the blue–yellow axis, respectively. It is particularly interesting because, differently from all spaces seen so far, it is linear and hence its behavior is closer to human intuition (changing two colors by the same amount will produce a similar amount of perceived changes). It is based on the choice of a white reference point $W = (X_W, Y_W, Z_W)$, and is obtained by the XYZ format as follows:

$$\begin{aligned} L^* &= 116 \cdot Y' - 16 \in [0, 100], \\ a^* &= 500 \cdot (X' - Y') \in [-127, +127], \\ b^* &= 200 \cdot (Y' - Z') \in [-127, +127], \end{aligned}$$

where $X' = f(\frac{X}{X_W})$, $Y' = f(\frac{Y}{Y_W})$, $Z' = f(\frac{Z}{Z_W})$, and

$$f(c) = \begin{cases} c^{\frac{1}{3}} & \text{if } c > 0.008856, \\ 7.787 \cdot c + \frac{16}{116} & \text{if } c \le 0.008856. \end{cases}$$

Since such a space is linear, the distance between two colors $C_1 = (L_1^*, a_1^*, b_1^*)$ and $C_2 = (L_2^*, a_2^*, b_2^*)$ is simply the Euclidean distance (i.e., the norm) between the corresponding vectors:

$$d(C_1, C_2) = \|C_1 - C_2\| = \sqrt{\left(L_1^* - L_2^*\right)^2 + \left(a_1^* - a_2^*\right)^2 + \left(b_1^* - b_2^*\right)^2}.$$

The conversions from XYZ to RGB and back are obtained as:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = M_{RGB} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

and

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M_{RGB}^{-1} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix},$$

where

$$M_{RGB} = \begin{pmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix}$$

and

$$M_{RGB}^{-1} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix}.$$

The three column vectors in $M_{RGB}^{-1}$ are, respectively, the coordinates of R, G and B in the XYZ space, and hence can be straightforwardly exploited for conversion purposes.

## 4.3 Color Depth Reduction

*Color depth reduction* refers to the possibility of reducing the amount of chrominance information in a color image, up to complete removal of colors or, yet more radically, to a neat distinction between black&white.

### *4.3.1 Desaturation*

*Desaturating* a color means reducing the amount of color in a continuous manner, according to a parameter $s \in [0, 1]$ that represents the amount of saturation to be left in the color (i.e., 0 removes all saturation while 1 leaves the color unchanged). Using the RGB space, a color is represented as a triple of components $(R, G, B)$, all of which must be reduced equally to preserve the relative proportions. Thus, the formula for obtaining the desaturated color $(R_d, G_d, B_d)$ applies a linear interpolation between the original color and the corresponding $Y$ gray level in the RGB space:

$$R_d = Y + s \cdot (R - Y),$$

$$G_d = Y + s \cdot (G - Y),$$

$$B_d = Y + s \cdot (B - Y),$$

where $Y$ is computed as in the transformation from RGB to YUV space (see Sect. 4.2.1).

### 4.3.2 Grayscale (Luminance)

The conversion of a color image into a gray-level one is obtained by computing for each pixel, starting from its original $(R, G, B)$ values, the corresponding gray value according to the luminance formula in Sect. 4.2.1 (rounded to the closest integer in case the value has to be discrete):

$$l = \text{round}(Y).$$

If the gray-level image is still to be represented in the RGB color space, the computed value must be set for all RGB components: $(l, l, l)$.

### 4.3.3 Black&White (Binarization)

A gray-level image $I(x, y)$ can be *binarized*—i.e., transformed into a black&white one $I'(x, y)$—by applying to each of its pixels a *thresholding* that, based on its gray level being smaller or larger than a luminance threshold $\bar{t}$, changes the pixel to full black or to full white, respectively. Thus, the choice of $\bar{t}$ induces a partition of the set of gray levels into two groups: the one below the threshold (denoted in the following by $L$ for 'left') corresponds to the abstraction of the foreground (i.e., the objects in the scene), while the one above the threshold (denoted by $R$ for 'right') abstracts the background. More formally, the black&white image resulting from binarization can be considered as a matrix $I'$, having the same size as the original image, each of whose elements $I'(x, y)$ contains a boolean (**True/False** or 1/0) value expressing the outcome of the comparison between the original luminance value of the corresponding pixel $I(x, y)$ and the threshold, as follows:

$$I'(x, y) = \begin{cases} 0 & \text{if } I(x, y) < \bar{t}, \\ 1 & \text{otherwise,} \end{cases}$$

where 0/**False** denotes black (i.e., the pixel belonging to $L$) and 1/**True** denotes white (i.e., the pixel belonging to $R$). The higher the threshold, the fewer levels of gray are filtered out to $R$, and hence the darker the image. Thus, identifying a proper threshold that preserves the interesting details without resulting in meaningless black blobs is an issue that deserves attention.

**Otsu Thresholding** A method to automatically identify an optimal threshold to be used for binarizing an image, based on the information contained in its gray-levels histogram, was proposed by *Otsu* [9]. For it to be effective, the intended background must be made up of gray levels sufficiently different than those of the foreground objects. The threshold is computed as the luminance level $\bar{t}$ that minimizes the weighted intra-group (or *within*) variance between $L$ and $R$:

$$\sigma_w^2(t) = \omega_L(t)\sigma_L^2(t) + \omega_R(t)\sigma_R^2(t)$$

or, equivalently, maximizes the corresponding inter-group (or *between*) variance:

$$\begin{aligned}
\sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) \\
&\quad \text{that is, the weighted variance of the cluster means around the overall mean} \\
&= \omega_L(t)\big[\mu_L(t) - \mu\big]^2 + \omega_R(t)\big[\mu_R(t) - \mu\big]^2 \\
&\quad \text{since } \mu = \omega_L(t)\mu_L(t) + \omega_R(t)\mu_R(t), \text{ by substitution} \\
&= \omega_L(t)\omega_R(t)\big[\mu_L(t) - \mu_R(t)\big]^2,
\end{aligned}$$

where $\sigma^2$ is the combined variance, $\mu_L(t)$ and $\mu_R(t)$ are the means of groups $L$ and $R$ determined by $t$, and $\mu$ is the combined mean.

The weights are specific for each threshold, and are taken to be the class probability of $L$ and $R$ over the whole set of pixels. Consider, for each gray level $t$, the ratio $p(t)$ of pixels in the image having that gray level over the total number of pixels in the image (i.e., the probability of a pixel in the image having that gray level).[2] Then, the probability distributions are:

$$\omega_L(t) = \sum_{i \in L} p(i), \qquad \omega_R(t) = \sum_{i \in R} p(i) = 1 - \omega_L(t).$$

In practice, the statistics are not computed from scratch for each candidate threshold, but an iterative procedure can be exploited to quickly derive the statistics for a level by just updating those of the immediately previous level: After computing the initial values $\omega_L(0)$, $\omega_R(0)$, $\mu_L(0)$ and $\mu_R(0)$, the next values can be obtained as:

$$\omega_L(t+1) = \omega_L(t) + p(t), \qquad \omega_R(t+1) = \omega_R(t+1) - p(t) = 1 - \omega_L(t+1),$$

$$\mu_L(t+1) = \frac{\mu_L(t)\omega_L(t) + p(t) \cdot t}{\omega_L(t+1)}, \qquad \mu_R(t+1) = \frac{\mu_R(t)\omega_R(t) + p(t) \cdot t}{\omega_R(t+1)}.$$

Summing up, the variance is computed in turn for each candidate split, remembering the maximum computed so far and the threshold by which it was reached.

## 4.4 Content Processing

Although an image as a whole conveys an overall meaning, a prominent role in its proper interpretation is often played by specific subparts thereof. This section proposes some techniques for identifying, describing and processing such noteworthy subparts. A *component* of an image $I$ is a subset of its pixels, usually determined according to some underlying property. Consider a boolean property $\mathscr{P}$ that applies

[2]Given the histogram $H(t)$ reporting the number of pixels in the image for each gray level $t$, and the total number $n$ of pixels in the image, $p(t) = H(t)/n$.

to each pixel $p \in I$, so that $p$ can be labeled as 'true' or as 'false' according to its fulfilling $\mathscr{P}$ or not. Then, the component induced by $\mathscr{P}$ is

$$C_{\mathscr{P}}(I) = \{p \in I \mid \mathscr{P}(p) = \text{true}\}.$$

This general definition allows identifying components of any complexity, which is useful because the image subpart of interest does not necessarily correspond to simple geometrical shapes, and often represents an irregular set of pixels.

## *4.4.1 Geometrical Transformations*

A basic set of tools to act on images consists of operations that allow moving their pixels from some place in the image to some other. Although they are often applied to the whole image, it is not infrequent that just a part of it is to be moved. A comfortable way to extract the interesting part, starting from the set $C_{\mathscr{P}}(I)$ of pixels of interest in $I$, is to consider it as a sub-image on its own, and hence to represent it as a new rectangle which is the smallest rectangle in the original image that includes all interesting pixels. This is called the *bounding box* of $C_{\mathscr{P}}(I)$, defined as:

$$\begin{aligned} I'_{\mathscr{P}} = \Big\{ (x', y') \in I \mid & \arg\min_x\big((x,y) \in C_{\mathscr{P}}(I)\big) \le x' \le \arg\max_x\big((x,y) \in C_{\mathscr{P}}(I)\big) \\ & \wedge \arg\min_y\big((x,y) \in C_{\mathscr{P}}(I)\big) \le y' \le \arg\max_y\big((x,y) \in C_{\mathscr{P}}(I)\big) \Big\}. \end{aligned}$$

The transformations of a pair of coordinates $(x, y)$ onto a new pair $(x', y')$, we are going to take into account, are the following:

**Translation** by a displacement $(d_x, d_y)$, where any value of the displacement can be positive or negative, yields

$$(x', y') = (x + d_x, y + d_y).$$

**Mirroring** of an image sized $n \times m$ can take place horizontally, yielding

$$(x', y') = (m - x - 1, y),$$

or vertically, yielding

$$(x', y') = (x, n - y - 1).$$

**Rotation** by an angle $\theta$ with respect to a point $(x_0, y_0)$ yields

$$(x', y') = \big((x - x_0)\cos\theta + x'_0, (y - y_0)\sin\theta + y'_0\big)$$

where $(x'_0, y'_0)$ is the point in $I'$ corresponding to $(x_0, y_0)$ in $I$.

After computing the transformed coordinates, the resulting values can be copied into the same image or, more frequently, into a new image $I'$. While for mirroring the image size stays invariant, and every pixel in the result has a corresponding value in the original image $I$, for translation and rotation some transformed coordinates might fall outside the original image boundaries, and new pixels might come into play for areas not considered in $I$. Thus, for the sake of generality, it is wiser to assume that the new image has size $I'_{[n' \times m']}$, and that a default value $D$ (e.g., $D = 0$) is considered for all positions in $I'$ that have no counterpart in $I_{[n \times m]}$. Then, the transformation may look, for each pixel in the target image, to the corresponding pixel into the original one, as follows:

$$I'(x', y') = \begin{cases} I(x, y) & \text{if } 0 \le x \le m - 1 \wedge 0 \le y \le n - 1, \\ D & \text{otherwise}, \end{cases}$$

where the pixel $I(x, y)$ to be copied is found, for translation, at

$$x = x' - d_x,$$
$$y = y' - d_y,$$

and, for rotation, at

$$x = (x' - x'_0) \cdot \cos\theta - (y' - y'_0) \cdot \sin\theta + x_0,$$
$$y = (x' - x'_0) \cdot \sin\theta + (y' - y'_0) \cdot \cos\theta + y_0.$$

As for rotation, the latter approach is mandatory because, since the transformation formulæ include trigonometric (real-valued) functions, the resulting coordinates are not integer, and must be consequently rounded. Unfortunately, this may result in different original coordinates collapsing onto the same transformed coordinate, and hence in some target coordinates being not considered in the transformation (the larger the distance from a multiple of the right angle, the more missing pixels, as shown in Fig. 4.1).

### 4.4.2 Edge Enhancement

Much of the perceptual capability of humans (and other animals as well) is heavily based on the recognition of edges that usually denote the limit between two objects in the scene or between objects and the background [8]. Since contours and vertices in images are typically perceived by the eye as sudden changes in luminance, due to the shadow projected by the objects when hit by a directional light, in the following only the graylevel component of the image will be taken into account. For the very same reason, many of the techniques for edge detection based on variations in luminance exploit linear correlation with derivative filters. Such filters are to be considered as local operators because they provide the output value $I'(x, y)$ for

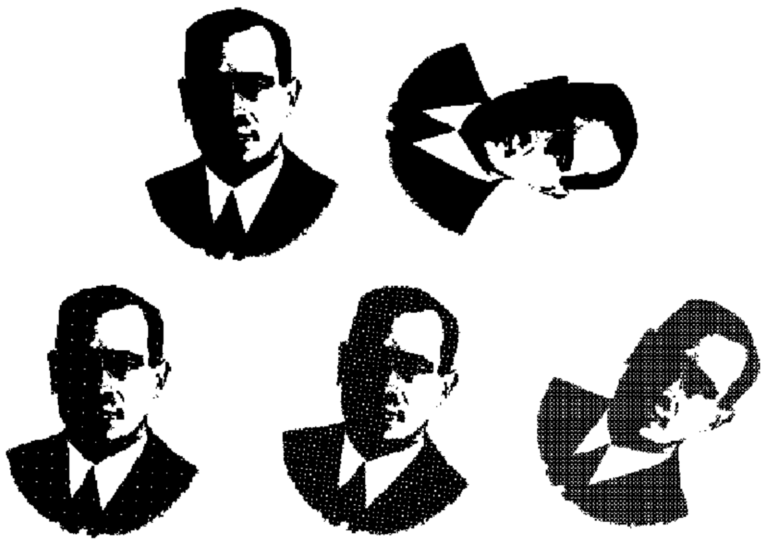


**Fig. 4.1** Noise due to collapsing coordinates in forward rotation of the *top-left image* (from *left* to *right*: *top row* 0°, 90°; *bottom row* 5°, 15°, 45° clockwise rotation)

each pixel in the transformed image as a function of the values of a suitable neighborhood of the corresponding pixel $I(x, y)$ in the original image. This ensures that the outcome is less affected by noise in the image around the pixel under processing.

**Derivative Filters** It has been just pointed out that object edges are characterized, from a perceptual viewpoint, by more or less sudden changes in luminance. If images are seen as mathematical functions, the suitable tool in charge of determining the amount of change is represented by derivative functions. Indeed, being able to locate the image regions where variations in luminance take place most prominently allows emphasizing them while smoothing the remaining regions, obtaining in this way a new version of the image where the edges are more evident. Such an enhanced image can then be exploited as a more comfortable basis than the original one for applying further techniques aimed at the final edge detection. More precisely, due to images being two-dimensional functions, *gradients* come into play, i.e., the extension to $n$-dimensional spaces of the concept of the derivative. Application of the gradient to a continuous function yields at each point a vector characterized by:

**Direction and Sense** the direction along which the maximum possible function variation for that point is obtained;
**Magnitude** the maximum value associated to the function variation.

A measure/estimation of the amount of change in a two-dimensional space (such as an image) can be obtained by computing the modulus of the gradient at every point, and can be represented by producing a map (another image) in which the regions having large gradient values are emphasized. In particular, the partial derivatives of the image with respect to the horizontal and vertical directions must be computed, by means of the corresponding gradient components:

$$\nabla I = \frac{\partial I}{\partial x}\bar{i} + \frac{\partial I}{\partial y}\bar{j},$$

where $\bar{i}$ and $\bar{j}$ are the unit vectors in directions $x$ and $y$, respectively. It is a derivative filter because the gradient is a first derivative of the image. If the gradient is computed on the luminance feature of an image, the outcome corresponds to the object edges. An approximation of the strength of the retrieved edge can be computed as the sum of the magnitude of the horizontal and vertical gradient components:

$$|G| = |G_x| + |G_y|,$$

where $G_x$ is the filter for direction $x$, and $G_y$ is the filter for direction $y$.

A first, simple computation of the gradient in a point $I(x, y)$ takes into account only the strictly horizontal and vertical directions:

$$G_x = \frac{I(x+1, y) - I(x-1, y)}{2},$$
$$G_y = \frac{I(x, y+1) - I(x, y-1)}{2}.$$

However, also the diagonal directions usually yield a significant contribution to the variation, in which case a correlation exploiting a filter based on a $3 \times 3$ mask that covers the whole neighborhood of a pixel is more indicated. These ideas underlie the technique proposed by *Prewitt*, that equally weights all directions:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}.$$

A similar filter that takes into account both the horizontal/vertical directions and the diagonal ones, but stresses the former components more than the latter, was proposed by *Sobel* [2], and is currently the most widely known and exploited:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}.$$

The operations to be carried out are:

1. Correlation of the original image with the filter for direction $x$ that yields the values $I_x(x, y)$;
2. Correlation of the original image with the filter for direction $y$ that yields the values $I_y(x, y)$;
3. Creation of an image $I'(x, y)$, each of whose pixels represents the absolute magnitude of the gradient in that point, obtained as a modulus (Euclidean distance)

of the values of components $I_x(x, y)$ and $I_y(x, y)$ produced by the directional filters in the corresponding position of the image:

$$I'(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2}.$$

Another famous filter is the *Laplacian*, based on second derivatives and defined as [10]:

$$\begin{aligned}\nabla^2 I(x, y) &= \frac{\partial^2 I}{\partial x^2}(x, y) + \frac{\partial^2 I}{\partial y^2}(x, y)\\ &= \big[\big(I(x, y) - I(x-1, y)\big) - \big(I(x+1, y) - I(x, y)\big)\big]\\ &\quad + \big[\big(I(x, y) - I(x, y-1)\big) - \big(I(x, y+1) - I(x, y)\big)\big]\\ &= 4 \cdot I(x, y) - \big(I(x-1, y) + I(x+1, y) + I(x, y-1) + I(x, y+1)\big)\end{aligned}$$

that can be represented by the following mask:

$$L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix},$$

further variants of which are [1]:

$$L_8 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \text{and} \quad L_{12} = \begin{bmatrix} -1 & -2 & -1 \\ -2 & 12 & -2 \\ -1 & -2 & -1 \end{bmatrix}.$$

### 4.4.3 Connectivity

Adjacency between pixels of an image $I$ can be defined according to two different *neighborhood* strategies, one stricter and one looser, as follows:

**4-neighborhood** referred to the horizontal or vertical directions only;
**8-neighborhood** including the diagonal directions as well.

Thus, a pixel can have at most 4 possible neighbors in the former case, and at most 8 possible neighbors in the latter, as graphically depicted in Fig. 4.2. In the following, $\mathcal{N}_k(p)$ will denote the set of $k$-neighbors ($k \in \{4, 8\}$) of a pixel $p$.

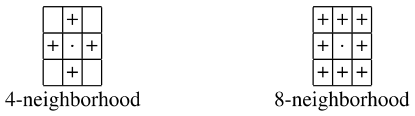


**Fig. 4.2** Neighborhood definitions for a reference pixel (denoted by ·). +'s denote the positions adjacent to · that are taken into account by each connection strategy

Having fixed a $k$-neighborhood approach and a boolean property $\mathscr{P}$, a *connected component* of $I$ with respect to $\mathscr{P}$ is a subset $C$ of its pixels such that

- $C \subseteq C_{\mathscr{P}}(I)$ ($\mathscr{P}$ is true for all pixels in $C$), AND
- For any two pixels in $C$ there exists a chain of pairwise $k$-neighbor pixels in $C$ that starts from the former and ends in the latter, AND
- There is no pixel $p \in I$ such that $\mathscr{P}(p) = \text{true}$, $p$ is $k$-neighbor to a pixel in $C$ and $p \notin C$.

In particular, a pixel $p \in C$ is said to be [10]:

- *Isolated* if $\mathscr{N}_k(p) \subseteq \overline{C}$;
- *Interior* if $\mathscr{N}_k(p) \subseteq C$;
- A *border* pixel otherwise ($\mathscr{N}_k(p) \cap C \neq \emptyset \wedge \mathscr{N}_k(p) \cap \overline{C} \neq \emptyset$); in particular, it is
  - An *arc* pixel if it has exactly two opposite neighbors in $C$ (while all the other neighbors are in $\overline{C}$);
  - An *arc end* if it has exactly one neighbor in $C$ (while all the others are in $\overline{C}$),

where $\overline{C} = I \setminus C$ is the complement of $C$.

Of course, several connected components can be found in a given image. However, the overall set of connected components in an image is uniquely identified, and represents a partition of the pixels in the image for which $\mathscr{P}$ is true. A subset of $I$ that has only one connected component is called *connected.*

The very same considerations that are made on the subset $S \subseteq I$ of pixels for which $\mathscr{P}$ is true, and in particular the analysis of connected components, can be made on its complement $\overline{S}$ using 'false' instead of 'true'. When working on $k$-connectivity in $S$, the opposite connectivity (let us denote this by $\overline{k}$ in the following: $\overline{k} = 8$ if $k = 4$, and *vice versa*) is exploited for $\overline{S}$. Consider $I$ as surrounded by a border of pixels labeled 'false'. The component of $\overline{S}$ consisting of the elements connected to any of these pixels is called the *outside* of $S$. Every other component of $\overline{S}$, if any, is called a *hole* in $S$; pixels in a hole are said to be *inside* $S$. If $S$ is connected and has no holes, it is called *simply* connected; if it has holes, it is called *multiply* connected.

Connected components are often useful because each of them can be extracted from the image and processed separately from the others. Connection is typically exploited in binary (i.e., black&white) images, where $\mathscr{P}(p) =$ 'pixel $p$ is black': in such a case, each connected component is usually interpreted as an object represented in the image.

**Flood Filling** A way for finding connected components of pixels sharing a given boolean property $\mathscr{P}$ is inspired by similarity to pouring a liquid into a container, so that it spreads along the bottom of the container until all of it has been covered up to the borders. Starting from a pixel fulfilling $\mathscr{P}$, the set of pixels that belong to the associated connected component is found by progressive propagation of one of the notions of connection between adjacent pixels. Both recursive and iterative versions of such an algorithm can be defined, but here the latter option is proposed for the sake of efficiency. As for all iterative implementations of recursive algorithms, it is necessary to explicitly handle a data structure that contains the information implicitly handled by the system stack in the recursive version: in this case, a set $B$ of pixels.

```
1. $P \leftarrow \{$set of pixels in $I$ that fulfill property $\mathscr{P}\}$
2. Select $\overline{s} \in P$ as a starting pixel
3. $B \leftarrow \{\overline{s}\}$
4. $C \leftarrow \emptyset$
5. **while** $B \neq \emptyset$
   (a) Select a $b \in B$
   (b) $B \leftarrow B \setminus \{b\}$
   (c) $C \leftarrow C \cup \{b\}$
   (d) **for each** $p \in \mathscr{P}_k(b)$:
       (i) **if** $p \notin C$: $B \leftarrow B \cup \{p\}$
```

$B$ represents the current 'boundary' of the connected component under construction, from which the 'flood' can be expanded; implementing it as a queue (which determines a FIFO behavior for the choice in step 5(a)) has the effect of progressively spreading the flood in all directions. $C$ is the connected component under construction, represented as a set of pixels as well. Each connected pixel in the $k$-neighbors of the boundary, considered in turn, is a candidate for flood expansion unless it has already been considered (i.e., it is already in $C$).

If the image includes several connected components, the pixels in $C$ can be removed from $P$ and the algorithm can be restarted, in order to obtain the next component. It might be desirable to produce a map of the connected components in the image $I$, where each pixel is labeled with a number expressing the connected component it belongs to. To this purpose, a matrix $L$ having the same size as $I$ is defined, whose elements are initially set at 0. Then, a top-down, left-to-right scanning of $I$ is started, until a pixel for which $\mathscr{P}$ is 'true' is found. The above algorithm is started on that pixel, setting at 1 the elements in $L$ corresponding to pixels that are included in $C$. After termination of the algorithm, the scanning of $I$ is carried on until a pixel is found for which $\mathscr{P}$ is 'true' and the corresponding element of $L$ is still at 0. The above algorithm is restarted on such a pixel, using the next available integer as a label, and this procedure is repeated until the bottom-right corner of $I$ is reached, at which moment the current integer represents the number of connected components found, and $L$ is the map.

**Border Following** Given a connected component $C$, it might be interesting to extract its border $B$, which completely identifies it but is lighter to store and represent. Let $B$ be the set of border elements of $C$ having neighbors in some particular component of $\overline{C}$. $B$ is obviously connected. Moreover, starting at any element of $B$ and successively moving to neighboring elements according to some strategy, it is possible to 'follow around' $B$ and return to the starting point. The algorithm is as follows:

```
1. Arbitrarily choose $p_0 \in B$ as the initial element
2. **if** $\mathscr{N}_4(p_0) \cap C = \emptyset$ then
   (a) $C = \{p_0\} = B$, and hence $p_0$ represents the whole border
```

```
3. else
   (a) Take $y_1^{(0)} \in \mathcal{N}_4(p_0) \cap \overline{C}$ (at least one such element exists, since $B$ is the border)
   (b) $i \leftarrow 0$
   (c) repeat
         (i) Let $\mathcal{N}_8(p_i) = \{y_1^{(i)}, \dots, y_8^{(i)}\}$
        (ii) Let $y_{j+1}^{(i)} \in \mathcal{N}_8(p_i) \cap \mathcal{N}_4(p_i) \cap C$ be the first in counterclockwise order
             starting from $y_1^{(i)}$
       (iii) if $y_j^{(i)} \in \overline{C}$ then
               (A) $y_1^{(i+1)} \leftarrow y_j^{(i)}$
               (B) $p_{i+1} \leftarrow y_{j+1}^{(i)} \in B$
             else ($y_j^{(i)} \in C$)
               (A) $y_1^{(i+1)} \leftarrow y_{j-1}^{(i)} \in \overline{C}$
               (B) $p_{i+1} \leftarrow y_j^{(i)} \in B$ (because $y_{j-1}^{(i)} \in \overline{C}$)
        (iv) $i \leftarrow i + 1$
       until $p_i = p_0$ (back to the starting point)
```

The sequence $\langle p_0, p_1, \dots \rangle$ includes all the pixels in the border of $C$, ordered counterclockwise. In addition to the explicit representation of each pixel as a pair of coordinates, the border can be compactly represented as a *chaincode*, i.e., as the (coordinates of the) starting point followed by the sequence of directions that are to be taken to locate each next pixel in the border with respect to the immediately previous one, with reference to a pre-defined schema, e.g.,

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 4 | · | 0 |
| 5 | 6 | 7 |

Some applications require a map of the borders that delimit the connected components in an image $I$, where each border is labeled with a different integer. This can be obtained by defining a matrix $L$ having the same size as $I$, whose elements are all initially set at 0. Then, $I$ is scanned top-down, left-to-right until a transition from pixels for which $\mathcal{P}$ is false to pixels for which it is true or *vice-versa* is found (representing the beginning of a connected component or of a hole in the image, respectively). On the 'true' pixel of such a transition, the above algorithm is started, setting at 1 the elements of $L$ that correspond to pixels included in the border sequence. Once the algorithm terminates, the image scanning is continued, and the algorithm is restarted on the next transition (unless the corresponding 'true' pixel is already labeled in $L$) using the next available integer as a label. The same procedure is repeated until the bottom-right corner of $I$ is reached, at which moment the current integer represents the number of borders found, and $L$ is the map.

**Dilation and Erosion** Some image processing tasks require to shrink and/or grow connected components in a given image, which can be thought of as removing or adding, respectively, border layers from/to the connected component. Considering an image $I$ as a set of pixels, both tasks can be resolved by means of set operations involving elements of the connected component $C \subseteq I$ and a *structuring element* $S$ centered on an element (called the *hot spot*) that specifies how its shape must grow or, conversely, which shapes should be shrunk to single points. The image operators in charge of obtaining such effects, each of which is the converse of the other, are, respectively:

**dilation** that extends a connected component $C$ by adding a layer along each border of $C$ (both those on the outside and those of the holes), producing a new component:

$$C \oplus S = C \cup \{p \in I \cap S(c) \mid c \in C\};$$

**erosion** that reduces a connected component $C$ by removing a layer from each border of $C$ (both those on the outside and those of the holes), producing a new component that preserves all the inner pixels of $C$ (those for which all pixels in the structuring element are still in $C$):

$$C \ominus S = \{p \in C \mid S(p) \subseteq C\} \subseteq C.$$

Here, $S(h)$ denotes the set of pixels identified by centering a structuring element $S$ on pixel $h$. Interestingly enough, eroding a component is the same as dilating the corresponding complement. In the classical definitions, $S = \mathcal{N}_k(p)$ is the set of $k$-neighbors of the hot spot. As usual, both $k = 4$ and $k = 8$ can be exploited: they result in addition or removal of a layer one pixel thick along each border of $C$ (both those on the outside and those of the holes). However, any other kind of structuring element can be exploited, to obtain fancier dilation and erosion effects.

One exploitation of dilation and erosion is for removing from an image noise in the form of small specks: by a few iterated erosions, such specks are completely deleted, after which applying an equivalent number of dilations the survivor components are restored in their original shape. Other applications are for reducing a connected components to a single point that represents it, or to a set of lines that represent its skeleton (by progressive erosion), or for making thicker or thinner a shape (e.g., to compensate for digitization defects).

**Opening and Closing** Based on dilation and erosion, two extremely useful operations can be defined, *open*:

$$C \circ S = (C \ominus S) \oplus S$$

and *close*:

$$C \bullet S = (C \oplus S) \ominus S.$$

The former is useful to eliminate from an image specks whose shape is smaller than $S$; the latter can be exploited to fill holes and fissures in the image whose shape is smaller than $S$. They are idempotent (i.e., after the first application, further application using the same structuring element does not change the result) and dual of each other (i.e., applying one on $C$ or the other on $\overline{C}$ yields the same result):

$$C \circ S = \overline{\overline{C} \bullet S}, \qquad C \bullet S = \overline{\overline{C} \circ S}.$$

## 4.5 Edge Detection

*Edge detection* is an image processing activity aimed at finding the boundaries of the elements represented in images. In fact, this is often a preliminary step to the identification and recognition of interesting image details, such as objects or peculiar patterns. The possible approaches can be divided into two categories, each of which in turn includes several techniques:

**Fixed** methods detect shapes in the image by comparison to a set of stored templates, which significantly limits the number of shapes that can be actually recognized. Indeed, being infeasible to store all possible shape templates, it may happen that the shape of interest is not present in the database. These include, among others, Pixel luminance (e.g., Image thresholding); Template matching; the Hough Transform.

**Flexible** methods repeatedly modify a tentative shape until it overlaps the target shape that is present in the image. They can be distinguished according to the function exploited to determine the degree of correspondence between the data in the image and the tentative shape, and also according to the kind of tentative shape exploited. Examples of this category are Deformable Templates and Active Contour Models.

A naive *Image Thresholding* technique consists in identifying the set of pixels that delimit an object by enhancing the changes in luminance through proper filters (such as those introduced in previous sections), then binarizing the image (the previous step should have improved the starting situation for the threshold determination). Each connected component obtained in this way can be considered an object shape, possibly to be refined (e.g., using erosion or contour following) and matched against available templates. Figure 4.3 shows an application of this procedure.

The *Deformable Template* technique consists in modifying a reference model (the *template*) made up of one or more shapes, in order to match them as much as possible with the shape to be recognized in the image at hand. One of the earlier approaches aimed at extracting facial features for identification purposes (by matching eyes to circles enclosed in opposite parabolæ) [13]. Each point $(x, y)$ of the template shapes 'has' (i.e., is associated to) an energy $E(x, y)$, and the energy of a shape is given by the sum of the energies of all its points. Modifications to the template are made by changing the shapes' position and orientation, but keeping unvaried the spatial relationship among them. Such changes take place by searching for the shape-defining parameter values that maximize their energy. Energy is normalized for each shape (e.g., based on its perimeter) to avoid maximization tending towards the shape having the largest size. If available, more information about the shape to

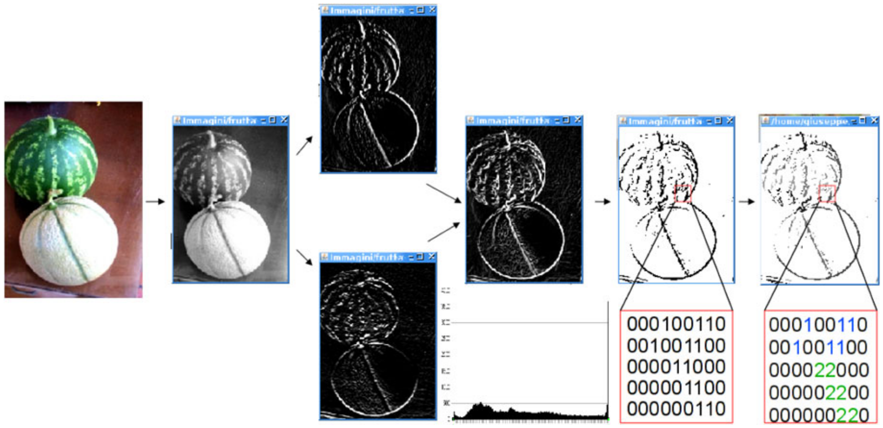


**Fig. 4.3** Naive technique for identifying objects in a raster image. From *left* to *right*: original image, gray level representation, Sobel horizontal and vertical components, merge of such components with associated, Otsu thresholding, connected component separation

be extracted can be exploited by the procedure. In order to maximize $E$, however, many parameter values come into play. Since iterating over all combinations of possible values for each parameter is infeasible, optimization techniques are exploited, or fewer parameters are taken into account,[3] as in the snakes approach presented afterward.

**Canny** A state-of-the-art algorithm for edge detection, although improvements thereof are available, is considered the one proposed by Canny [2]. It can find step- and ramp-shaped edges (even in images affected by white noise), and turns out to be optimal with respect to several parameters: precision and recall of the retrieved edges (*detection*), tolerance in the retrieved edge compared to the actual one (*localization*), and avoidance of multiple outcomes for a single edge (*one response*: a single edge should be returned for each point of the actual edge). The first and last criteria are strictly related, in that worst edges are eliminated from competing candidates.

The algorithm applies repeatedly the following basic steps:

1. Convolve $I$ with a Gaussian of standard deviation $\sigma$, $G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$;
2. Estimate for each pixel in the image the local direction $d$ normal to the edge;
3. Find the location of the edges using non-maximal suppression;
4. Compute the magnitude of the edge;
5. Threshold edges in the image to eliminate spurious responses,

[3] A way to obtain a 'useful' smaller set of parameters is using *Principal Component Analysis* (*PCA*), a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables (the *principal components*), each of which accounts for a portion of variability in the data (and can be ranked according to how much variability it catches).

for ascending values of the *scale* $\sigma$, and the final information about edges at multiple scale is aggregated using the 'feature synthesis' approach. However, since it is quite time-consuming, often these additional steps are omitted, if thorough robustness of the outcome is not strictly needed.

Step 1 blurs the image, using for convolution a Gaussian mask of size $n \times n$, where the value of $n$ must be defined. Some authors suggest taking it as the smallest odd integer $n \geq 6\sigma$ to provide 'full' power to the Gaussian filter (smaller filters would result in less smoothing).

Then, as to step 2, for each point $(x, y)$ in the image, the magnitude and angle/direction of the gradient are, respectively,

$$M(x,y) = \sqrt{\left(\frac{\partial f_s}{\partial x}\right)^2 + \left(\frac{\partial f_s}{\partial y}\right)^2} \quad \text{and} \quad \alpha(x,y) = \tan^{-1}\left[\frac{\partial f_s}{\partial y} \Big/ \frac{\partial f_s}{\partial x}\right].$$

Since $M$ is obtained using the gradient, it contains crests around the local maxima, and thus such values must be thinned using *non-maxima suppression*. For this, a finite number of directions to be considered must be specified. In an 8-neighborhood approach, since pairs of opposite directions both denote the same edge, just four (the horizontal, the vertical and the two diagonals) can be considered, placed at 45° from each other. The edge direction is orthogonal to that of the gradient expressed by $\alpha(x, y)$. Thus, the non-maxima suppression schema might be as follows:

```
1. Find the direction $d$ (among the above four) closest to $\alpha(x, y)$
2. if the value $M(x, y)$ is smaller than at least two of its neighbors along $d$
   (a) then set $g_N(x, y) = 0$ (suppression)
   (b) else set $g_N(x, y) = M(x, y)$,
```

where $g_N(x, y)$ is the final image.

Lastly, $g_N(x, y)$ must be thresholded to reduce false edge points. Too low thresholds yield false positives, while too high ones yield false negatives (i.e., eliminate correct edges). Canny uses thresholding based on hysteresis, and hence two thresholds: a lower one $T_L$ and a higher one $T_H$, suggesting the latter to be twice or three times as large as the former. Thus, two additional images are obtained, respectively:

$$g_{NH}(x,y) = \begin{cases} g_N(x,y) & \text{if } g_N(x,y) \geq T_H, \\ 0 & \text{otherwise,} \end{cases}$$

$$g_{NL}(x,y) = \begin{cases} g_N(x,y) & \text{if } T_L \leq g_N(x,y) < T_H, \\ 0 & \text{otherwise.} \end{cases}$$

The $g_N(x, y) < T_H$ condition in the definition of $g_{NL}(x, y)$ ensures that the two images have disjoint non-null values. Indeed, otherwise if $T_L < T_H$, the set of non-null pixels in the former (*strong* edges) would be a subset of those in the latter (*weak* edges).

All strong edges are directly considered as edges. Obviously, they are typically affected by significant fragmentation (the higher the $T_H$, the more empty regions). Longer edges are obtained as follows:

```
1. for all edge (i.e., non-zero) pixels $p$ in $g_{NH}(x, y)$
   (a) Label as valid edge pixels all weak pixels in $g_{NL}(x, y)$ that are connected to
       $p$ (e.g., by 8-connectivity)
2. Set to zero all pixels in $g_{NL}(x, y)$ that have not been labeled as valid edge pixels
3. Add to $g_{NH}(x, y)$ all non-zero pixels in $g_{NL}(x, y)$
```

In practice, there is no real need for two additional images, since everything can be done directly during non-maxima suppression.

**Hough Transform** The *Hough Transform* [6] is a parametric method to identify a shape within a distribution of points. Of course, drawing on the image all possible instances of the shape and counting the number of overlapping points for each is infeasible (although finite, due to the image being finite). Hough proposed switching on the opposite perspective, and counting for each point in the original image the possible shapes that pass through that point. The shape must be parametrically defined, which makes such a technique very suitable for regular geometrical shapes. This is less limiting than it might seem because regular geometrical shapes indeed pervasively occur as sub-components of real-world images. The basic idea consists in searching for the shapes in the space defined by the shape parameters (*parameter space*).

The algorithm is based on an *accumulator*, a multi-dimensional array that counts how many foreground points in the image are intersected by each possible shape, and in short works as follows:

```
1. for all non-background points $(x, y)$ in the image $I$
   (a) for all values $(v_1, \dots, v_n)$ of the set of $n$ parameters, each sampled in the
       suitable range with a discrete step $\Delta_i$
       (i) Increase the accumulator cell addressed by the coordinates $(v_1, \dots, v_n)$
   (b) Select as significant instances of the shape in the image the points in the
       parameter space having larger accumulator values associated.
```

A number of issues must be dealt with in order to practically apply this algorithm. First of all, the selection of 'high' accumulator points is not trivial: taking the absolute maximum would return only a few prominent instances of the shape (and in general smaller instances are underestimated because they are made up of fewer points so that some kind of normalization would be desirable). To tackle this, a threshold is often used, or all local maxima (points in the accumulator whose neighbors are smaller) are selected. In both cases, however, the approximations applied during the algorithm execution cause a shape to be actually represented by a set of neighbor points rather than by a single point in the parameter space: hence, proper techniques are needed to identify such regions, and their centroid is usually taken as the single shape representative. Another issue concerns efficiency: smaller $\Delta$ increments cause more refined shape identification, but increase runtime and memory requirements for storing and filling the accumulator. In the following, the case of straight lines will be discussed in more depth, to provide an understanding of the method. Application to other shapes will be just sketched.

It is well-known that the equation of a line in the $X$–$Y$ plane is

$$y = ax + b,$$

where $a$ denotes the slope (0 for horizontal, negative for descending towards the right-hand-side, positive for ascending towards the right-hand-side, infinite for the vertical) and $b$ is the height on the $Y$ axis for $x = 0$. Having fixed an image point $p = (\overline{x}, \overline{y})$, all possible lines passing through $p$ satisfy the equation

$$b = -a\overline{x} + \overline{y},$$

where now the constants are $\overline{x}$ and $\overline{y}$ and the variables $a$ and $b$ (or, equivalently, $b$ is a function of $a$). Thus, in the new plane $A$–$B$ that represents the parameter space for lines, the set of all lines passing through $p$ is itself a line (a point in the image is represented by a line in the parameter space, and *vice versa*). Now, finding the segment that connects two points means finding the $a$ and $b$ parameters such that both instances of the above equation applied to the coordinates of the two points are satisfied. Put it another way, the intersection of two lines in the parameter space denotes (the parameters of) a line in the image that passes through the two points to which the lines in the parameter space are associated. More generally, the set of lines in the parameter space that meet each other in a given point represents all points in the image that belong to the same line: the larger the cardinality of such a set, the more significant such a line in the image. For identifying segments in the image, the accumulator can be extended to store, in addition to the number of points intersected by the lines, the segment extremes as well, that are checked and updated each time a new point is added.

Actually, the linear equation for lines is a bit odd because vertical lines, corresponding to infinite slope, cannot be represented. A more suitable representation is the polar one that expresses them in terms of angle $\theta$ over the $X$ axis and radius $r$:

$$r = \overline{x}\cos\theta + \overline{y}\sin\theta,$$

where $r$ is a function of $\theta$. The angle can range in $[0, \pi[$ only (the other angles will denote just the same lines, considered in the opposite direction). The maximum segment length to be considered corresponds to the image diagonal, and hence, if the representation places the origin in the center of the image, the maximum radius is just one half of it:

$$\overline{r} = \frac{1}{2}\sqrt{\text{width}(I)^2 + \text{height}(I)^2} = \frac{1}{2}\sqrt{n^2 + m^2}.$$

The same principle can be applied to more complex shapes, but the number of parameters, and hence the problems of efficiency, may grow significantly. As to circles, the equation of a circle having center $(x_c, y_c)$ and radius $r$ is

$$r^2 = (x - x_c)^2 + (y - y_c)^2,$$

where, having fixed a point $p = (\overline{x}, \overline{y})$ in the image, there are three degrees of freedom (and hence parameters, and hence dimensions of the accumulator): $x_c$, $y_c$ and $r$.

For each given radius $r$, the set of all circles passing through point $p$ in the image corresponds to a circle of radius $r$ in the parameter space (a symmetry with the case of lines), and hence all circles of any radius for $p$ correspond to cones in the whole space. For ellipses, the matter is yet more complex, since there are five parameters ($x_c$ and $y_c$, coordinates of the center, $r_M$ and $r_m$, the radii, and $\theta$, the slope of the major diameter over the $X$ axis), which makes memory requirements quite stringent even for large $\Delta$'s.

**Polygonal Approximation** *Polygonal approximation* of contours is concerned with describing complex shapes as polygons each of whose sides represents a piece of contour which is actually more complex than a straight segment, while preserving the overall 'flavor' of the original shape. This allows a more compact representation, and can be deemed as a feature extraction technique (e.g., for supporting comparison between shapes). Two strategies can be applied: a *sequential* one that follows the shape contour, deciding at each point if a vertex of the polygon should be placed there or not, and an *iterative* one where starting from a triangle the side that less fits the curve is repeatedly identified and properly split, until a termination condition is met.

A fast sequential algorithm (at partial expenses of precision) for performing such a task was proposed in [12], where each point in the shape is examined only once to decide whether it may represent a vertex of the target polygon. Differently from other techniques, each polygon side starts and ends on points actually belonging to the shape. The original shape, made up of $n$ points, can be described explicitly as a sequence of points $\langle p_0, \dots, p_n \rangle$ (which allows expressing non-uniformly spaced points) or by delineating its contour (e.g., using a border following technique) and representing it, for the sake of compactness, as a chaincode $\langle p_0, d_1, \dots, d_n \rangle$ where $p_0$ is the starting point and $d_1, \dots, d_n$ are the directions that identify each of the next points $p_1, \dots, p_n$ in the shape, with $p_n = p_0$.

The algorithm exploits a threshold $T$ that represents the maximum allowed area deviation *per length unit* of the polygon side (and hence, the larger the $T$, the longer the segments and the looser the approximation of the original shape), expressed in pixels:

```
1. Initialize $v_1 \leftarrow p_0 = (x_0, y_0)$, $i \leftarrow 0$, $j \leftarrow 1$
2. **repeat**
   (a) Assume $v_j = (0, 0)$, by translating the coordinate system if necessary
   (b) Initialize $f_i = 0$
   (c) **repeat**
         (i) $i \leftarrow i + 1$
        (ii) $L_i = \sqrt{x_i^2 + y_i^2}$, the length of the current line segment from $v_j$ to $p_i = (x_i, y_i)$
       (iii) Update the deviation accumulator $f_i = f_{i-1} + \Delta f_i$, where $\Delta f_i = x_i \cdot \Delta y_i - y_i \cdot \Delta x_i$ with $\Delta x_i, \Delta y_i$ the increments[4]
       **until** $(|f_i| > T \cdot L_i) \vee (p_i = p_0)$
```

[4]If each point $p_i$ is a neighbor of $p_{i-1}$ (i.e., points are equally spaced), $\Delta x_i, \Delta y_i \in \{0, 1\}$.

```
    (d) $j \leftarrow j + 1$ (found a new vertex in the polygon)
    (e) $i \leftarrow i - 1$ (the previous point was the last acceptable one)
    (f) $v_j = p_i$ is the next vertex in the polygon, because the longest acceptable
        segment $\overline{v_{j-1}v_j}$ has been found
until $p_i = p_0$ (back to the starting point)
```

In the end, the sequence of $\{v_j\}_{j=1,\dots,m}$ denotes the set of vertices of the polygon approximating the original shape, and $m$ is the number of sides of such a polygon.

The authors report that the increment caused by extending the end of a segment starting in $v^j$ from a pixel $p_{i-1}$ to the next pixel $p_i$ can be expressed as

$$\Delta f_i = L_i \cdot D_i,$$

where $D_i$ is the (signed) distance between $p_{i-1}$ and the new segment $\overline{v_j p_i}$, which (by straightforward geometrical considerations) turns out to be the doubled area of the triangle $p_{i-1}v_j p_i$. This yields a total area deviation for a segment reaching the $k$th point after $v_j$ equal to $f_k/2 = \sum_{i=1}^{k} \Delta f_i/2 = \sum_{i=1}^{k} L_i D_i/2$.

A weakness of the method is that very narrow peaks in the shape to be approximated could be lost in the polygon, since the area deviation accumulated by skipping them in the current segment is very low. In case they are to be preserved as significant, a slight modification of the algorithm is needed. While building a segment starting in $v_j$, the farthest point $\overline{p}$ from $v_j$ is stored (candidates can be noticed because the length $L$ of the current segment decreases in the next point); if $L$ keeps decreasing for several points, the point $p_i$ whose distance $D$ from $\overline{p}$ reaches the value of $T$ is considered as well. Then, the next vertex $v_{j+1}$ is taken as $\overline{p}$ for having a better approximation (with the disadvantage of additional calculations to restart again from $\overline{p}$ and follow again the partial contour from $\overline{p}$ to $p_i$), or as $p_i$.

A lossless way to improve speed is to avoid computing the square root in the inner loop by testing the squared condition $f_i^2 > T^2 \cdot L_i^2$, where $L_i^2$ can be obtained by just updating the previous value:[5]

$$L_i^2 = L_{i-1} + 2 \cdot (x_i \Delta x_i + y_i \Delta y_i) + (\Delta x_i)^2 + (\Delta y_i)^2.$$

Further, but lossy, speed-up can be obtained by approximating $L_i$ as $|x_i| + |y_i| \geq L_i$ (which overestimates the value and hence yields rougher shape approximations) or as $\max(|x_i|, |y_i|) \leq L_i$ (which underestimates the value and hence yields finer shape approximations), with the advice that the orientation of the line segments affects the closeness of such an approximation to the actual value.

The authors also suggest that, having fixed an even number $k$ of consecutive pixels in the shape, the likelihood that a point $p_{i+k/2}$ is a corner is proportional to

[5]Note once again that, for uniformly spaced points, $\Delta x_i, \Delta y_i \in \{0, 1\} \Rightarrow (\Delta x_i)^2, (\Delta y_i)^2 \in \{0, 1\}$ as well, which clearly makes this computation trivial.

the area deviation for the line segment $\overline{p_i p_{i+j}}$ divided by the length of the current segment.

**Snakes** *Snakes* [7] are an active contour technique aimed at identifying the outline of the main object (the salient contour) represented in an image. Instead of finding separate pieces (e.g., segments) of the contour, this strategy works on a closed shape having vertices $\mathbf{v} = \{\mathbf{v}_0, \dots, \mathbf{v}_n\}$ (where $\mathbf{v}_n = \mathbf{v}_0$), and progressively moves and deforms its sides and corners according to several energy factors. Since the approach performs an energy minimization based on local optima rather than global ones, different alternative solutions are possible, and the desired one can be returned if the snake is initialized close to it by someone or by some preceding technique. Then, it carries out the rest of the task autonomously. The model is active in that it dynamically minimizes its energy functional. It is named after the behavior of the contours while being deformed because of energy minimization, resembling a snake.

A snake can be represented parametrically as $\mathbf{v}(s) = (x(s), y(s))$, where $s \in [0, 1]$ are the points of the snake normalized into $[0, 1]$. In a discrete reference system with step $h$, such as an image, $\mathbf{v}_i = (x_i, y_i) = (x(ih), y(ih))$. The basic model is a *controlled continuity* spline influenced by image forces and external constraints forces generated to minimize the overall energy functional of the snake, defined as

$$E_{\text{snake}} = \int_0^1 E_{\text{int}}\mathbf{v}(s) + E_{\text{img}}\mathbf{v}(s) + E_{\text{con}}\mathbf{v}(s)ds$$

or, in its discrete form,

$$E_{\text{snake}} = \sum_{i=1}^{n-1} E_{\text{int}}(i) + E_{\text{img}}(i) + E_{\text{con}}(i),$$

where $E_{\text{int}}$ is the internal energy of the spline due to bending, used to impose a piecewise smoothness constraint; $E_{\text{img}}$ represents the energy related to some image feature (typically, the image brightness) for attracting the snake towards edges, lines and salient contours; and $E_{\text{con}}$ gives rise to the external constraint forces in charge of placing the snake near the desired local minimum.

The internal energy

$$E_{\text{int}} = \frac{1}{2}\left(\alpha(s)\left\|\frac{\partial \mathbf{v}}{\partial s}(s)\right\|^2 + \beta(s)\left\|\frac{\partial^2 \mathbf{v}}{\partial s^2}(s)\right\|^2\right)$$

with discrete formulation

$$E_{\text{int}}(i) = \frac{1}{2}\left(\alpha_i \frac{|\mathbf{v}_i - \mathbf{v}_{i-1}|^2}{h^2} + \beta_i \frac{|\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}|^2}{h^4}\right)$$

controls the behavior of the snake both as a membrane and as a thin plate, weighting their relative importance by means of parameters $\alpha(s)$ and $\beta(s)$, respectively (setting $\beta(\overline{s}) = 0$ causes a second-order discontinuity, i.e., generates an angle, at $\overline{s}$). The use of an iterative technique based on sparse matrix methods allows processing the whole set of points in the snake in a linear number of steps.

The total image energy is a combination of three elements (energy functionals) that attract the snake towards lines, edges and terminations in the image, respectively, weighted differently to obtain snakes with different behavior:

$$E_{\text{img}} = w_{\text{line}} \cdot E_{\text{line}} + w_{\text{edge}} \cdot E_{\text{edge}} + w_{\text{term}} \cdot E_{\text{term}}.$$

The line component can just work on image intensity: $E_{\text{line}} = I(x, y)$ (the sign of $w_{\text{line}}$ determines whether the snake is attracted by light or dark contours). Another option consists in following the *scale space* approach, taking the snake to equilibrium on a blurry image and then fine-tuning the details by progressively reducing the level of blur. The component in this case becomes $E_{\text{line}} = -(G_\sigma \star \nabla^2 I)^2$, where $G_\sigma$ is a Gaussian having standard deviation $\sigma$. Minima of this functional lie on zero-crossings of $G_\sigma \star \nabla^2 I$, that attract the snake but do not affect its smoothness. The edge component pulls towards large image gradients: $E_{\text{edge}} = -|\nabla I(x, y)|^2$. Finally, terminations of line segments and corners are identified by the curvature of level lines in a slightly smoothed image $C = G_\sigma \star I$ as follows: given the gradient angle $\theta = \tan^{-1}(\frac{\partial C}{\partial y} / \frac{\partial C}{\partial x})$, $\mathbf{n} = (\cos\theta, \sin\theta)$ the unit vector along the gradient direction and $\mathbf{n}_\perp = (-\sin\theta, \cos\theta)$ the unit vector perpendicular thereof,

$$E_{\text{term}} = \frac{\partial\theta}{\partial\mathbf{n}_\perp} = \frac{\partial^2 C/\partial\mathbf{n}_\perp^2}{\partial C/\partial\mathbf{n}} = \frac{\frac{\partial^2 C}{\partial y^2}\left(\frac{\partial C}{\partial x}\right)^2 - 2\frac{\partial^2 C}{\partial x \partial y}\frac{\partial C}{\partial x}\frac{\partial C}{\partial y} + \frac{\partial^2 C}{\partial x^2}\left(\frac{\partial C}{\partial y}\right)^2}{\left(\left(\frac{\partial C}{\partial x}\right)^2 + \left(\frac{\partial C}{\partial y}\right)^2\right)^{3/2}}.$$

External constraints can be represented as ‘springs’ anchored to pairs of points $(x_1, x_2)$, which results in adding $-k(x_1 - x_2)^2$ to $E_{\text{con}}$, or repulsion forces $1/r^2$ (the $1/r$ energy functional is clipped near $r = 0$ to prevent numerical instability).

To improve the final outcome of the snake, preventing it from being misled by local energy optima, it has been proposed to work using two interrelated snakes (each point in either has a correspondent in the other), one initialized outside the shape to be found, and the other initialized inside of it [4]. The former progressively shrinks, while the latter progressively expands, towards the target. When both reach their equilibrium but still do not match each other, an additional *driving force* $g(t)$ is activated on the snake with higher energy to make it evolve by moving towards the other, until its energy starts decreasing. Incidentally, the energies of the two snakes must be rotation, translation and scale invariant to be comparable to each other.

The complete contour energy considered in [4] is

$$E_{\text{snake}} = \frac{1}{N}\sum_{i=0}^{N-1} \lambda E_{\text{int}}(v_i) + (1-\lambda)E_{\text{ext}}(v_i),$$

where $E_{\text{ext}} = E_{\text{img}} + E_{\text{con}} = E_{\text{edge}}$ and $\lambda$ weights the relative importance of the internal and external energy in determining the snake evolution ($\lambda = 1$ ignores image forces, while $\lambda = 0$ causes the snake shape to depend on image forces only).

The evolution of each vertex $\mathbf{v}_i^t$ at time $t$ to $\mathbf{v}_i^{t+1}$ at time $t+1$, tailored for not exceeding one pixel, is

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \frac{1}{2}\left(\lambda \frac{\mathbf{e}_i}{h} + (1-\lambda)F_i\right) + g(t)\frac{\mathbf{u}_i - \mathbf{v}_i^t}{|\mathbf{u}_i - \mathbf{v}_i^t|},$$

carried out until $\max_i |\mathbf{v}_i^{t+1} - \mathbf{v}_i^t| < \delta$ (for a fixed $\delta$), where

- $\mathbf{u} = \{\mathbf{u}_1, \dots, \mathbf{u}_n\}$ is the other snake,
- $F_i = -k\left(\begin{smallmatrix} \frac{\partial E_{\text{ext}}}{\partial x}|_{\mathbf{v}_i} \\ \frac{\partial E_{\text{ext}}}{\partial y}|_{\mathbf{v}_i} \end{smallmatrix}\right)$ are the external forces, normalized by a constant $k$ to the interval $[-1, 1]$,
- $\mathbf{e}_i = \frac{1}{2}(\mathbf{v}_{i-1} - \mathbf{v}_{i+1}) - \mathbf{v}_i + \theta_i \frac{1}{2}\mathbf{R}(\mathbf{v}_{i-1} - \mathbf{v}_{i+1})$ are the internal forces, that pull $\mathbf{v}_i$ towards its estimated target position according to the desired shape (R is a $+90°$ rotation matrix and $\theta$ is related to the local tangent angle).

The points can be forced to be evenly spaced ($|\mathbf{v}_{i-1} - \mathbf{v}_i| = |\mathbf{v}_{i+1} - \mathbf{v}_i|$). If the internal forces aim at promoting a circular shape, the internal angle at $\mathbf{v}_i$ must be $\varphi_i = \frac{(N-2)\pi}{N}$, and $\theta = \cot(\frac{\varphi}{2}) = \cot(\frac{(N-2)\pi}{2N}) = \tan(\frac{\pi}{N})$. A negative $\theta$ may cause the contour to become concave. To ensure that the contour does not expand or contract indefinitely, it must hold that $\sum_{i=0}^{N-1} \varphi_i = (N-2)\pi$ and $0 < \varphi_i < 2\pi$.

## References

1. Burger, W., Burge, M.J.: Digital Image Processing. Texts in Computer Science. Springer, Berlin (2008)
2. Canny, J.F.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **8**(6), 679–698 (1986)
3. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Prentice Hall, New York (2008)
4. Gunn, S.R., Nixon, M.S.: A robust snake implementation; a dual active contour. IEEE Transactions on Pattern Analysis and Machine Intelligence **19**(1), 63–68 (1997)
5. Hanson, N.R.: Patterns of Discovery—An Inquiry into the Conceptual Foundations of Science. Cambridge University Press, Cambridge (1958)
6. Hough, P.V.: Method and means for recognizing complex patterns. Tech. rep. 3069654, US Patent (1962)
7. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. International Journal of Computer Vision **1**(4), 321–331 (1988)
8. Lindsay, P.H., Norman, D.A.: Human Information Processing: Introduction to Psychology. Academic Press, San Deigo (1977)
9. Otsu, N.: A threshold selection method from gray-level histogram. IEEE Transactions on Systems, Man, and Cybernetics **9**(1), 62–66 (1979)
10. Rosenfeld, A.: Picture Processing by Computer. Academic Press, San Diego (1969)
11. Sonka, M., Hlavac, V., Boyle, R.: Image Processing, Analysis, and Machine Vision, 3rd (international) edn. Thomson, Washington (2008)
12. Wall, K., Danielsson, P.E.: A fast sequential method for polygonal approximation of digital curves. Computer Vision, Graphics, and Image Processing **28**, 220–227 (1984)
13. Yuille, A.L.: Deformable templates for face recognition. Journal of Cognitive Neuroscience **3**(1), 59–70 (1991)

# Chapter 5
# Document Image Analysis

One of the main distinguishing features of a document is its outward appearance (its *layout*), as determined by the organization of, and reciprocal relationships among, the single components that make it up (e.g., title, authors and abstract of a scientific paper; addressee, sender logo, subject, date and signature of a commercial letter; etc.). It is so important that very different component organizations have been designed and established for different kinds of documents (e.g., scientific papers, newspapers, commercial letters, bills, etc.), while documents of the same kind typically show very similar appearance (e.g., all scientific papers have a prominent title followed by authors, affiliations and abstract; all commercial letters have the sender logo and addressee on the top and the signature at the bottom), although subtle differences in organization are often exploited for characterizing their specific sources (e.g., scientific papers by different publishers). The importance of such a feature is due to its immediately hitting the visual perception level, so that a quick glance is sufficient to humans for identifying the document kind and its relevant components, even without actually reading their content. As a consequence, great effort has been spent in the computer science research to work on this level and reproduce the human abilities in understanding a document layout. *Document Image Analysis* is concerned with the automatic interpretation of images of documents [39].

## 5.1 Document Structures

Any document is characterized by a geometrical/perceptual structure in which its components are visually organized, referred to as its *layout structure*. It is a fundamental element for human recognition and understanding of the document, and defines a hierarchy of abstract representations of the document spatial organization. If such a hierarchy is graphically represented as a tree, its root corresponds to the entire document (usually a sequence of pages), while the leaves (corresponding to the bottom of the abstraction hierarchy) are the *basic blocks* that can be found in the document, and internal nodes represent different levels of compound components (typically words, lines, groups of lines, frames, pages). Since multi-page documents can be processed at the level of separate pages, in the following sections each single page will be considered as a *document* by itself. While not representing a limitation, this is important because in this way, whenever only some pages in a document are significant to the aims for which it is processed, irrelevant ones can be simply ignored. At the page level, several groups of strictly related visual components can be distinguished, called *frames*, that are conventionally identified as rectangular areas of interest each of which should ideally play a specific logical role in the document itself (e.g., the title or the author in an article). The process of identifying the basic components in a document and then reconstructing their hierarchical organization is called *layout analysis*.

After detecting the geometrical structure of a document, its components can be associated to a logical function expressing the role they play in the document (e.g., title, authors and sections in a scientific paper; sender, addressee, subject and date in a commercial letter; etc.). Indeed, the role played by a layout component represents meta-information that could be exploited to label the document in order to help its filing, handling and retrieval in a collection or library. The logical components can, in turn, be organized in a hierarchical structure, called *logical structure*, resulting from the progressive split of the document content in smaller and smaller parts on the grounds of the human-perceptible meaning of their content. The leaves of such a structure are the basic logical components, such as authors and title of a journal paper or the date and place of a commercial letter. Internal nodes of the structure represent compound elements, such as the heading of a paper, that encompasses title and authors. The root of the logical structure is the document class, such as 'scientific paper of the Springer-Verlag Lecture Notes series' or 'commercial letter'.

Figure 5.1 shows a document along with its layout and logical structures. An ideal layout analysis process should result in the identification of a set of frames, such that a non-ambiguous correspondence could be established between each of them and a possible logical component. In practice, however, an approximation in which, although the retrieved frames do not exactly match the theoretical ones, it is nevertheless possible to distinguish their logical meaning, can be considered a satisfactory output of a layout analyzer. Moreover, often only some of the document components are useful for accomplishing a task and need to be correctly identified. In any case, layout analysis is a fundamental step towards the interpretation of a document. Once the logical components of interest have been found, they must be processed according to the type of their content in order to extract the information they carry. In some sense, each component undergoes a sequence of steps that (although specific for each component) is analogous to that formerly applied to the whole document: pre-processing, classification, understanding.

The recognition of these structures in a digital document starts from their description available in the source file. The information aggregates expressed by the source file formats are typically of very low level: pixels in the case of images, (fragments of) words and graphical items in semi-structured formats, or organizational blocks in the case of structured formats. The first two provide spatial information about the components, but lack in organizational information, while the converse holds for the last kind. Thus, independently of the source representation, the processing

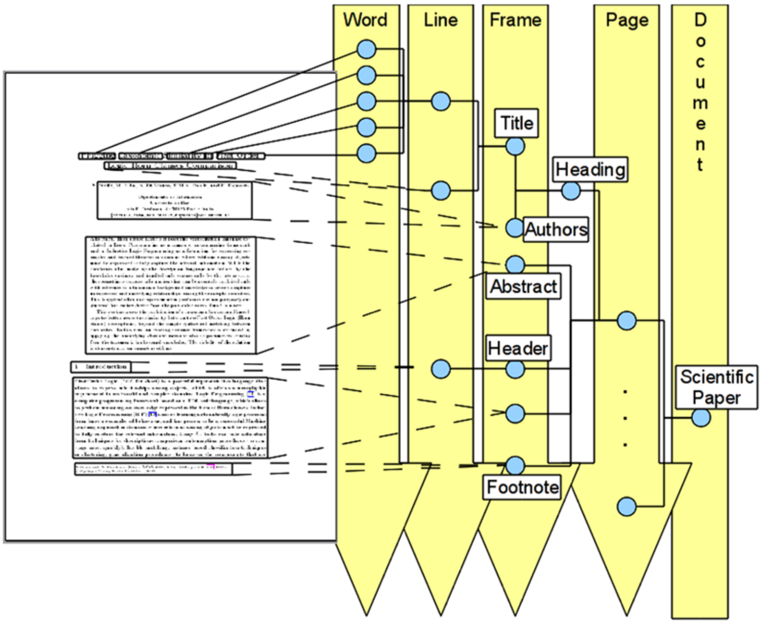


**Fig. 5.1** Overlapped layout and logical structures in the first page of a scientific paper. The former includes the nodes in the tree surrounded by vertical stripes and arrows, the latter includes the labeled nodes

steps aimed at extracting high-level structures from its layout need suitable representation formalisms to describe the document component organization they take as input or produce as output. There are several perspectives for these higher level representations. The layout structure description requires the ability of representing the geometrical organization of components, while the logical structure poses to the need for expressing the role of components.

### *5.1.1 Spatial Description*

Dealing with a document layout, the spatial representation of components becomes of crucial importance, and hence deserves a deeper treatment. Here we will focus on the perceptual aspects, although much research has been also devoted to the development of computer structures for storing spatial data [40]. In this perspective, given a set of regions in the space, several kinds of *spatial relations* come into play, among which [17]:

**Topological** (corresponding to the mathematical concept of *topology*) deal with neighborhood and overlap. **Ordinal** relations are an (often sufficient) simplification thereof that considers only containment (which is indeed an ordering relationship[1]) [27].

**Direction** (corresponding to the mathematical concept of *order*) consider order in space.

**Metric** (corresponding to the mathematical concept of *algebra*) refer to the possibility of making measurements, such as distances.

Here we are interested in the first two kinds.

**4-Intersection Model** According to the *4-Intersection* model [17], so called because all possible cases can be represented in a $2 \times 2$ matrix,[2] topological relations can be expressed mathematically by considering each region $R$ as a set of points in the plane, and distinguishing its interior $R_i$ and its boundary $R_b$. Given two regions $R'$ and $R''$, each of the following situations or its negation must occur:

1. $\exists x : x \in R'_b \wedge x \in R''_b$ (boundaries touch each other)
2. $\exists y : y \in R'_i \wedge y \in R''_b$ (a boundary crosses an interior)
3. $\exists w : w \in R'_b \wedge w \in R''_i$ (a boundary crosses an interior)
4. $\exists z : z \in R'_i \wedge z \in R''_i$ (common interiors)

for a total of $2^4 = 16$ combinations, of which only eight are valid:

$$\begin{array}{lcll}
\text{disjoint}(R', R'') & \Leftrightarrow & \neg(1) \wedge \neg(2) \wedge \neg(3) \wedge \neg(4); & \\
\text{meet}(R', R'') & \Leftrightarrow & (1) \wedge \neg(2) \wedge \neg(3) \wedge \neg(4); & \\
\text{overlap}(R', R'') & \Leftrightarrow & (1) \wedge (2) \wedge (3) \wedge (4); & \\
\text{covers}(R', R'') & \Leftrightarrow & (1) \wedge \neg(2) \wedge (3) \wedge (4) & \text{and its converse:} \\
\text{covered\_by}(R', R'') & \Leftrightarrow & (1) \wedge \neg(2) \wedge (3) \wedge (4); & \\
\text{inside}(R', R'') & \Leftrightarrow & \neg(1) \wedge \neg(2) \wedge (3) \wedge (4) & \text{and its converse:} \\
\text{contains}(R', R'') & \Leftrightarrow & \neg(1) \wedge (2) \wedge \neg(3) \wedge (4); & \\
\text{equal}(R', R'') & \Leftrightarrow & (1) \wedge \neg(2) \wedge \neg(3) \wedge (4). &
\end{array}$$

All of them stay invariant under several transformations (e.g., scaling and rotation).

Ordinal relations can be defined in terms of the above concepts as:

$$\begin{array}{lcl}
\text{is\_part\_of}(R', R'') & \Leftrightarrow & \text{inside}(R', R'') \vee \text{covered}(R', R''); \\
\text{consists\_of}(R', R'') & \Leftrightarrow & \text{contains}(R', R'') \vee \text{covers}(R', R'').
\end{array}$$

As to direction relations, the cardinal directions provide a good idea of the concept and of the various possibilities. Unlike the previous cases, the direction of an *observed* object is always expressed relative to another object, called the *reference* (and denoted in the following by overlined symbols). Let us denote North as $N$, South as $S$, West as $W$ and East as $E$, and the position of the reference as $I$. A 4-direction model $D_4 = \{N, S, W, E\}$ or, more often, the more flexible 8-direction model $D_8 = \{N, NE, E, SE, S, SW, W, NW\}$ can be exploited to locate

[1]A binary relationship is an *ordering relationship* if it is reflexive, antisymmetric and transitive.

[2]An evolution of this model that considers also the complement of the region (and hence a $3 \times 3$ matrix), called *9-Intersection* model, was successively developed [16, 18].

observed objects with respect to the reference [24]. All relationships are transitive; $I$ is obviously symmetric as well, while all the others are pairwise mutually opposite: $N$–$S$, $W$–$E$, $NW$–$SE$, $NE$–$SW$. Differently from topological relations, there is no complete agreement in the literature on their definition; here we will refer to the projection-based approach of [40].

Let us start from single points. Considering the reference point placed in the origin of a Cartesian axes system, $N$ consists of the positive $y$ axis, $S$ of the negative $y$ axis, $W$ of the negative $x$ axis and $E$ of the positive $x$ axis; the rest are associated to the four quadrants. More generally, given a reference point $(\overline{x}, \overline{y})$, the directions of an observed point $(x, y)$ with respect to it are defined by the following areas:

| | $W$ | $Y$ | $E$ |
|---|---|---|---|
| $N$ | $NW$ $x < \overline{x} \wedge y > \overline{y}$ | $N_r$ $x = \overline{x} \wedge y > \overline{y}$ | $NE$ $x > \overline{x} \wedge y > \overline{y}$ |
| $X$ | $W_r$ $x < \overline{x} \wedge y = \overline{y}$ | $I$ $x = \overline{x} \wedge y = \overline{y}$ | $E_r$ $x > \overline{x} \wedge y = \overline{y}$ |
| $S$ | $SW$ $x < \overline{x} \wedge y < \overline{y}$ | $S_r$ $x = \overline{x} \wedge y < \overline{y}$ | $SE$ $x > \overline{x} \wedge y < \overline{y}$ |

where the relations denoted by a $r$ subscript are restricted to half-lines; more relaxed versions thereof, that involve whole rows or columns of the above schema, are shown as headings and are defined as follows:

$$\begin{aligned} N &= NW \vee N_r \vee NE, & S &= SW \vee S_r \vee SE, \\ W &= NW \vee W_r \vee SW, & E &= NE \vee E_r \vee SE, \\ X &= W_r \vee I \vee E_r, & Y &= N_r \vee I \vee S_r. \end{aligned}$$

One can step from single points to regions by working on the sets of points that make up the regions, and combining them by means of universally and existentially quantified formulæ. For any relation $\mathscr{R} \in D_8$,

- Its *strong* version is defined as

$$\mathscr{R}_s(R, \overline{R}) \quad \Leftrightarrow \quad \forall p \in R, \forall \overline{p} \in \overline{R} : \mathscr{R}(p, \overline{p});$$

- Its *weak* version $\mathscr{R}_w$ is a relaxed version that admits some overlapping of the two bounding boxes, provided that the opposite does not hold for some pair of points;
- Its *bounded* versions $\mathscr{R}_{sb}$ and $\mathscr{R}_{wb}$ are as above, but requiring that the bounding box of the observed region falls within the projection of the reference;
- Its *just* version $\mathscr{R}_j$ relaxes the strong version allowing and requiring that at most the bounding box boundaries touch each other;
- Its corresponding general version is defined as:

$$\mathscr{R}(R', R'') \quad \Leftrightarrow \quad \mathscr{R}_s(R', R'') \vee \mathscr{R}_w(R', R'') \vee \mathscr{R}_j(R', R'').$$

It can be immediately noticed that, some of these relations being defined in terms of others, the former imply the latter.

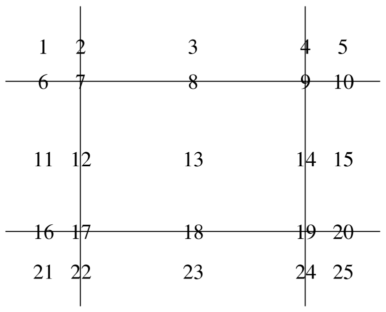


**Fig. 5.2** Partition of the plane into 25 zones with respect to a reference rectangle

*Example 5.1* (Some direction relations for regions)

| | | | |
|---|---|---|---|
| weak $N$ | $: N_w(R, \overline{R})$ | $\Leftrightarrow$ | $\exists p \forall \overline{p} : N(p, \overline{p}) \wedge \forall p \exists \overline{p} : N(p, \overline{p}) \wedge \exists p, \overline{p} : S(p, \overline{p})$; |
| strong bounded $N$ | $: N_{sb}(R, \overline{R})$ | $\Leftrightarrow$ | $N_s(R, \overline{R}) \wedge \forall p \exists \overline{p} : NW(p, \overline{p}) \wedge \forall p \exists \overline{p} : NE(p, \overline{p})$; |
| weak bounded $N$ | $: N_{wb}(R, \overline{R})$ | $\Leftrightarrow$ | $N_w(R, \overline{R}) \wedge \forall p \exists \overline{p} : NW(p, \overline{p}) \wedge \forall p \exists \overline{p} : NE(p, \overline{p})$; |
| just $N$ | $: N_j(R, \overline{R})$ | $\Leftrightarrow$ | $(\forall p, \overline{p} : N(p, \overline{p}) \vee X(p, \overline{p})) \wedge \exists p, \overline{p}$ $: X(p, \overline{p}) \wedge \exists p, \overline{p} : N(p, \overline{p})$; |
| weak $NE$ | $: NE_w(R, \overline{R})$ | $\Leftrightarrow$ | $\exists p \forall \overline{p} : NE(p, \overline{p}) \wedge \exists p, \overline{p} : S(p, \overline{p}) \wedge \forall p \exists \overline{p} : NE(p, \overline{p})$, |

where points denoted $p$ belong to region $R$ and points denoted $\overline{p}$ belong to the reference $\overline{R}$.

**Minimum Bounding Rectangles** An interesting discussion about spatial relationships between two rectangles in the space is provided in [40]. Indeed, the document components are often represented in terms of their minimum bounding box, and thus those relationships can be usefully exploited as spatial descriptors for the document components organization. Given a reference rectangle $\overline{R}$, the straight lines passing from its four sides partition the plane into 25 possible zones such that any observed point must fall within one of them, as depicted in Fig. 5.2:

- 1 rectangle (zone 13);
- 4 points (zones 7, 9, 17, 19);
- 4 segments (zones 8, 12, 14, 18);
- 8 half-lines (zones 2, 4, 6, 10, 16, 20, 22, 24);
- 8 unlimited areas (zones 1, 3, 5, 11, 15, 21, 23, 25).

This allows the maximum flexibility in representing the position of a single point with respect to $\overline{R}$. In particular, $\overline{R}$ occupies exactly the limited elements of the partition: zone 13 for its inner part and zones 7, 8, 9, 12, 14, 17, 18, 19 for its boundary.

More often, the (bounding box of the) observed object is another rectangle as well (excluding the case of perfectly horizontal or vertical segments), not just a point. In this case, a quick analysis of the schema in Fig. 5.2 reveals that 169 combinations of mutual positions of the reference rectangle with respect to the observed one are possible. Each specific situation is represented by just enumerating the zones of the schema that overlap the compared rectangle. All topological and direction relations between the two rectangles can be described in this way. If a whole page layout is to be described, clearly there is no specific reference frame to be considered, but each frame in turn becomes the reference. This would introduce redundancy because the relation between any pair of frames would be described twice depending on which is the reference in turn. This problem can be overcome by defining an ordering on the frames (e.g., top-down, left-to-right based on the frame top-left corner) and comparing each of them only to those following it in the ordering. When the frames do not overlap each other, this also results in a simplification of the representation, since the symmetric elements of the partition become useless (in the above example, zones 1, 2, 3, 8, 11, 12 and 13 wouldn't be ever exploited).

*Example 5.2* (Spatial relations expressed using the 25-partitioned plane)
Topological relations:

inside $\Leftrightarrow$ zone13 $\wedge \neg$ zone8 $\wedge \neg$ zone12 $\wedge \neg$ zone14 $\wedge \neg$ zone18;
equal $\Leftrightarrow$ zone8 $\wedge$ zone12 $\wedge$ zone14 $\wedge$ zone18 $\wedge \neg$ zone3 $\wedge$
$\neg$ zone11 $\wedge \neg$ zone15 $\wedge \neg$ zone23;
covered_by $\Leftrightarrow$ zone13 $\wedge$
$\neg$ zone3 $\wedge \neg$ zone11 $\wedge \neg$ zone15 $\wedge \neg$ zone23 $\wedge$
(zone8 $\vee$ zone12 $\vee$ zone14 $\vee$ zone18) $\wedge$
($\neg$ zone8 $\vee \neg$ zone12 $\vee \neg$ zone14 $\vee \neg$ zone18)

Direction relations:

$N_s$ $\Leftrightarrow$ (zone1 $\vee$ zone2 $\vee$ zone3 $\vee$ zone4 $\vee$ zone5) $\wedge$
$\neg$ zone6 $\wedge \neg$ zone8 $\wedge \neg$ zone10;
$N_{sb}$ $\Leftrightarrow$ zone3 $\wedge \neg$ zone2 $\wedge \neg$ zone4 $\wedge \neg$ zone8;
$N_{wb}$ $\Leftrightarrow$ zone3 $\wedge$ zone8 $\wedge$ zone13 $\wedge \neg$ zone2 $\wedge$
$\neg$ zone4 $\wedge \neg$ zone12 $\wedge \neg$ zone14 $\wedge \neg$ zone18.

### *5.1.2 Logical Structure Description*

Parallel to the visual structure of a document, although strictly related to it for the foregoing reasons, is its logical structure. Since the document may include several kinds of components at different levels of complexity, playing different roles and possibly interrelated in many different ways to each other, a sufficiently expressive representation must be available to processing systems intended to handle them as true documents and not just as computer files. XML is often exploited for this purpose, as a powerful and flexible language.

**DOM (Document Object Model)** The *DOM* is a platform-independent representation to model the logical structure, content and style of documents, and the way in which they can be built, accessed, navigated and manipulated by adding, deleting or modifying elements and/or content. The DOM does not express the relevance or structure of information in a document. It is a logical model that just specifies representational and behavioral requirements, leaving each specific implementation in

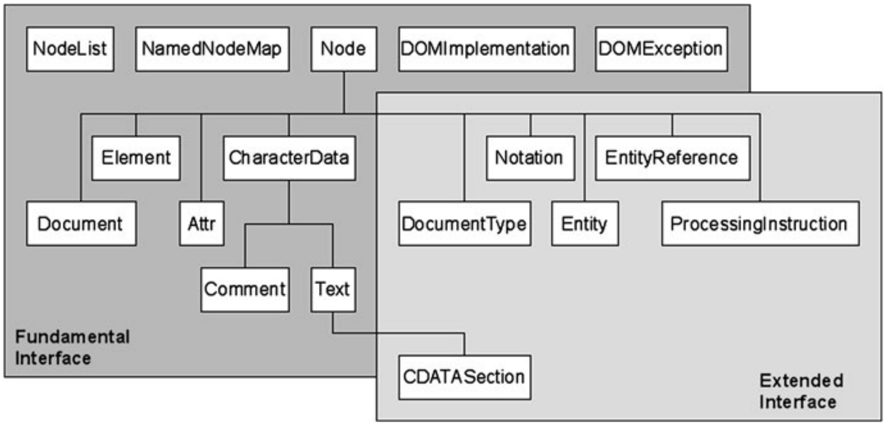


**Fig. 5.3** DOM interface hierarchy

any language free to support them in any convenient way, as long as *structural isomorphism* is preserved: any two DOM implementations will produce the same structure model representation of any given document. Being object oriented, it models documents using objects (each endowed with identity, structure and behavior) by specifying:

- Classes (*interfaces*), an abstract specification of how to access and manipulate documents and components thereof in an application's internal representation;
- Their semantics, including properties (*attributes*) and behavior (*methods*);
- Their relationships and collaborations among each other and with objects.

It is particularly suited to represent and handle XML/HTML documents in object oriented environments. Indeed, the DOM representation of a document (called its *structure model*) has a forest-shaped structure which may consist of many trees whose nodes are objects (rather than data structures). Some types of nodes may have child nodes, others are leaves. Figure 5.3 shows the DOM interface hierarchy.

**DOMImplementation** provides methods for performing operations that are independent of any particular DOM instance. **DOMException** instances are raised when attempting invalid operations. **Node** is the primary interface, representing a single node in the document tree (for which several sub-types, or specializations, are available). It exposes methods for dealing with children, inherited also by its sub-classes that cannot have children. It includes a *nodeName* attribute inherited by all its descendants to declare the name of their instances. **NodeList** handles ordered lists of Nodes (e.g., the children of a Node); **NamedNodeMap** handles unordered sets of nodes referenced by their name attribute.

**Document** is the (mandatory and sole) root of the tree, representing the whole document and providing access to all of its data. **Element** represents an element in an HTML/XML document, possibly associated with attributes represented by **Attr** instances, whose allowed values are typically defined in a DTD. Attr objects are not considered part of the document tree, but just properties of the elements they are associated with. Thus, in fact they are not children of the element they describe, and do not have a separate identity from them. **CharacterData** is an abstract interface (it has no actual objects) to access character data, inherited by Text, Comment, and CDATASection. **Comment** refers to the characters between `<!--` and `-->`. **Text** nodes can only be leaves, each representing a continuous (i.e., not interleaved with any Element or Comment) piece of textual content of an Element or Attr.

**DocumentType** allows dealing with the list of entities that are defined for the document. Each Document has at most one such object whose value is stored in the *doctype* attribute. **Notation** nodes have no parent, and represent notations in the DTD that declare either the format of an unparsed entity (by name) or processing instruction targets. **Entity** represents a (parsed or unparsed) entity in an XML document. **EntityReference** objects represent references to entities whose corresponding Entity node might not exist. If it exists, then the subtree of the EntityReference node is in general a copy of its subtree. Entity and EntityReference nodes, and all their descendants, are read-only. **ProcessingInstruction** represents processor-specific information in the text of an XML document. **CDATASection** objects contain (in the *DOMString* attribute inherited by Text) text whose characters would be regarded as markup. They cannot be nested and are terminated by the `]]>` delimiter.

**DocumentFragment** is a 'lightweight' version of a Document object, useful for extracting (e.g., copying/cutting) or creating a portion of document tree that can be inserted as a sub-tree of another Node (e.g., for moving around document pieces), by adding its children to the children list of that node.

*Example 5.3* (DOM node types for XML and HTML) Internal nodes, along with possible types of their children nodes:

**Document**
Element (at most one), ProcessingInstruction, Comment, DocumentType (at most one)
**DocumentFragment**
Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
**EntityReference**
Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
**Element**
Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
**Attr**
Text, EntityReference
**Entity**
Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference

Leaf nodes:
**DocumentType**, **ProcessingInstruction**, **Comment**, **Text**, **CDATASection**, **Notation**.

The DOM standard was defined by the W3C to establish a multi-environment framework and avoid incompatibility among browsers. Different levels of specifications are available, each requiring additional features with respect to the previous levels. The current level is 2, although pieces of level 3 are progressively becoming W3C recommendations. *Level 1* [1] consists of two modules:

**Core** (mandatory) provides a low-level set of fundamental interfaces that can represent any structured document; it also defines (optional) extended interfaces for representing XML documents.

**HTML** provides additional, higher-level interfaces that are used with those defined in the Core to provide a more convenient view of HTML documents.

An implementation including the extended XML interfaces does not require the HTML module, and XML interfaces are not required for implementations that only deal with HTML documents. *Level 2* [2] is made up of 14 modules: Core, XML, HTML, Views, Style Sheets, CSS, CSS2, Events, User interface Events, Mouse Events, Mutation Events, HTML Events, Range, Traversal. An implementation is 'Level 2 compliant' if it supports the Core; it can be compliant to single modules as well (if it supports all the interfaces for those modules and the associated semantics).

Several APIs for the various programming languages have been developed to handle DOM representations. This provides programmers with an interface to their proprietary representations and functionalities (they may even access pre-DOM software), and authors with a means to increase interoperability on the Web. DOM APIs keep the whole document structure in memory, which allows random access to any part thereof, but is quite space demanding.[3]

## 5.2 Pre-processing for Digitized Documents

Albeit new documents are nowadays produced directly in digital formats, where the type of components (e.g., text, graphic, picture, formula, etc.) is explicit in the representation, the huge number of legacy (paper) documents produced in past centuries cannot be ignored. The typical way in which they are transformed into a computer-readable form is by means of a digitization process (e.g., scanning), both because it is cheaper and because it preserves the original characteristics of the document. In such cases, they come in the form of raster images,[4] and hence represented in non-structured digital formats, although their content is not actually perceived by a human user as a single graphical item. Thus, the distance between the document representation and the information it contains is longer than in the case of natively digital documents, and additional work must be carried out by a *pre-processing* phase to bridge the gap from the pixel-level grain size to the component-level one.

[3] Another model, called *SAX* (*Simple API for XML*), processes the documents linewise. This avoids the need to completely load them into memory, but bounds processing to proceed forward only (once an item is passed over, it can be accessed again only by restarting processing from the beginning).

[4] A minimum resolution for allowing significant processing is 300 dpi, but thanks to the recently improved performance of computers the current digitization standards are becoming 400 dpi, with a trend towards 600 dpi.

Usually, such a phase acts on a simplified representation of the document image, that allows reducing computational (time and space) requirements while still preserving sufficient detail to provide effective results. The original details, of course, are not lost, and can still be exploited when needed by tracing back the simplified image (or portions thereof) to the source one.

One possible kind of simplification consists in decreasing the image color depth. Often just a black&white representation of the image is considered, that is sufficient for global processing purposes. Ideally, black should denote pixels that are relevant to the processing objectives, independently of their specific meaning, while white should express their irrelevance.[5] A way for doing this is using thresholding techniques that turn each pixel into either black or white depending on its passing a significance value based on the distribution of colors or gray levels in the original image (see Sect. 4.3.3). Later processing phases can (even selectively) switch back to the gray-level or color representation, if needed or useful for their aims.

As another way for simplification, the image size can be reduced by scaling it down. A pixel in the reduced image will take on a color that depends on the pixels in the original image that collapsed onto it (e.g., the most frequent or the average). If not excessive, while still preserving the significant components in the documents, scaling has a number of desirable side-effects, such as the reduction of computational requirements for processing the document, and the removal of small (undesired) specks introduced by the use, age or photocopying/digitization process of the document. For A4-sized images at 300 dpi, reducing the resolution to 75 dpi (i.e., scaling down each dimension to $1/4$th, and hence the whole image to $1/16$th, where each $4 \times 4$ region collapses onto a single pixel) has proven to be a good trade-off [4]. Again, whenever needed one can restore the original size by scaling up the entire image or just parts thereof and reading the corresponding area from the original image.

The document image often suffers from undesired noise introduced by the digitization process itself. This makes the distance from the available representation to the intended one even longer, and requires the processing techniques to be aware of the possible kinds of noise and of the ways for reducing it, when present.

**Document Image Defect Models** A study on *document image defects* was carried out in [7], focusing on *symbols* (possibly combined into more complex shapes) as the fundamental elements to be recognized in a document image. Symbols can be affected both by geometrical *deformations* and by additional *imaging defects* due to the processing technologies and techniques. It may happen that a (combination of) transformation(s) turns an original symbol to a shape that is closer to another, different symbol. An attempt to model such defects resulted in the following categorization:

**Resolution** of the image quantization, depending on both the size of the symbol and the acquisition resolution, and affecting the symbol shape smoothness;

[5]Switching from the color space to a logic one, they can be interpreted as denoting a pixel being important, in terms of **True** or **False**.

**Blur** affecting the symbol shape sharpness;
**Threshold** for binarization that can result in considering noise as actual content (if too low) or in missing content (if too high);
**Sensitivity** of the digitization receptors, also depending on the symbol intensity;
**Jitter** image pixels are not digitized as a perfect grid, but may be displaced and overlap to other cells of the theoretical grid;
**Skew** rotation of a symbol;
**Width** and **Height** horizontal and vertical stretching of the symbol, respectively;
**Baseline** a symbol being placed above or below the conventional baseline;
**Kerning** horizontal displacement of the symbol.

A model of defects as a support to image processing should ensure *completeness* (being able to express all perceived defects), *calibration* (adaptivity of the underlying distribution to any image population), *simulation* (being able to generate likely defected images), *enumeration* (supporting the automatic computation of all possible symbolic prototypes implied by a given feature extraction system).

## 5.2.1 Deskewing

A common source of handicaps for a successful layout analysis comes from the process and means by which the document image has been obtained. Indeed, photocopying or scanning a document page might introduce an undesired rotation of the image due to imperfect document alignment during digitization. Such an odd orientation is called the *skew angle* of a document image [5]: according to trigonometric conventions, it is negative when the image is rotated clock-wise, or positive in the opposite case. When this happens, identifying and evaluating the angle by which the rotation took place, in order to restore the original orientation by applying an inverse rotation, turns out to be fundamental for an effective application of the further processing steps (e.g., segmentation), and in some cases might even preclude the very feasibility of the document processing task.

The skew angle typically affects the orientation of the text lines or rectangular pictures in the document, that are horizontal in the original image but not in the digitized one, which thus provides a valid indicator of the problem [15]. An estimation $\hat{\theta}$ of the actual angle $\theta$ can be obtained by building $H_\theta$, the horizontal *projection profile* (i.e., the histogram of the black pixels for each row) of the document image $I$, for different possible rotations $\theta$ [21]. For the sake of efficiency, it can be useful to apply such an estimate only on a previously selected sample region $R \subseteq I$ instead of the whole document image. As shown in Fig. 5.4, at the correct rotation, when lines are horizontal (no skew), the histogram shape is characterized by neat peaks having approximately the same width as the characters' height, while the profile shows smoother and lower peaks when the skew angle is large (the larger the angle, the smoother and lower the peaks, suggesting a uniform distribution). As a drawback, this might introduce errors when there is no single orientation in the document. Indeed, when several local orientations are available, one could come up

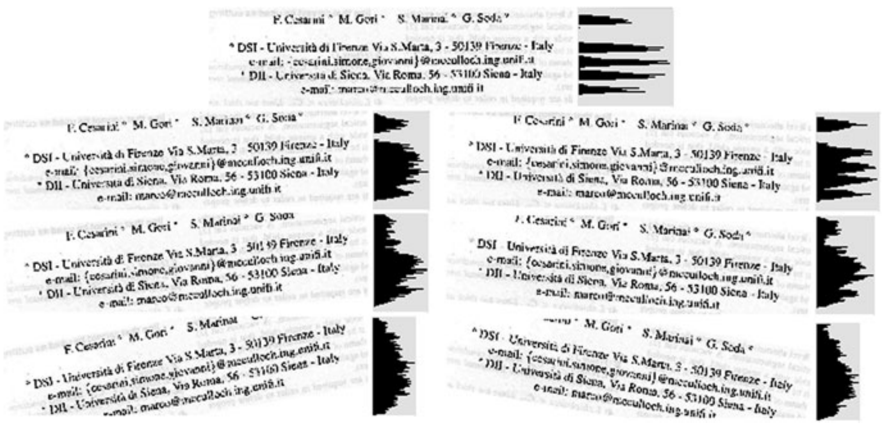


**Fig. 5.4** Different skew angles (positive on the *left*, negative on the *right*) for a fragment of scanned document and corresponding horizontal black pixel histograms. From *top* to *bottom*: 0° (no skew), ±2.5°, ±5°and ±10°

with an angle that does not represent the most frequent (or dominant) orientation in the document. Mathematically, this concept is captured by the variance; thus, locating the best rotation, and hence the correct skew angle, corresponds to identifying the global maximum of the real-valued *energy alignment measure* function [5]

$$A(\theta) = \sum_{i \in R} H_\theta^2(i)$$

within a suitable neighborhood of the horizontal. Another way for determining the skew of a document image comes as a side-effect of the *docstrum* [38] technique, to be described later in this chapter, that also allows handling different orientations in the same document.

Once the skew angle has been identified, its effect can be compensated by inverse rotation. Rather than considering each pixel in the original image and transposing it into a corresponding position of the deskewed one, a better solution is considering each pixel of the deskewed image and assigning it the value of the corresponding pixel of the original one located by inverse rotation. Indeed, for rotations that are not a multiple of the right angle, an approximation of the pixel coordinates to the closest integers is needed, which in the former approach would produce pixels in the original image that collapse on the same pixel of the deskewed one, leaving as a consequence some deskewed pixels blank (see Sect. 4.4.1).

### *5.2.2 Dewarping*

*Dewarping* is the term usually exploited to denote a recent area of interest for digitized document pre-processing, aimed at compensating the noise introduced by the flat (2D) representation of spatially spread (3D) documents. In fact, when scanning an open book the pages often cannot be completely flattened on the scanner bed, and in the spine region the page shape appears deformed so that document components, e.g., text lines, become curved rather than straight. The problem is even worse if documents are photographed rather than scanned because this effect is amplified and additional perspective distortions are introduced. In fact, this latter option is being given more and more attention, due both to the availability of cheap and compact high-resolution cameras and to its causing a reduced stress on the document under digitization (an extremely important requirement in the case of historical and fragile items).

Dewarping is particularly crucial for ensuring a sufficiently good outcome from application of OCR to the digitized document. Indeed, although an option could be developing OCR systems that can directly deal with the problem and fix it, a more straightforward approach consists in specific pre-processing of the page to remove this kind of noise before applying standard OCR technologies. The approaches proposed so far as potential solutions to restore the original flat version of a warped document have focused either on the hardware side (by exploiting dedicated acquisition devices that can better capture the 3D nature of the digitized object) or on the software one (by working on a standard picture of the item obtained using a traditional means). The latter option is cheaper and ensures wider applicability, for which reason it is being given much attention. In particular, two kinds of approaches exist for identifying the deformations to be corrected, one based on the geometrical features of the overall document image, and one based on the distortions detected on specific document components, such as text lines (a quite significant and straightforward indicator of the problem). Techniques attempted to carry out the image dewarping task include splines, snakes, linear regression, grid modeling and even fuzzy sets.

**Segmentation-Based Dewarping** A dewarping technique that exploits segmentation of the document image and linear regression was proposed in [25]. Starting from a black&white image, it consists of two macro-steps whose algorithms are sketched below. In the following, elements having the same superscript, subscript or bars are to be interpreted as associated to the same object. The former step preliminarily identifies words and text lines:

1. Find the most frequent height $h$ among the connected components of black pixels in the page (since most of them will correspond to single characters, it is likely to represent the average character height);
2. Remove the non-text and noise components (in [25], all components having height or width less than $h/4$, or height greater than $3h$);
3. Apply horizontal smoothing (see Sect. 5.3.2) with threshold $h$ (the resulting connected components should correspond to words);
4. Consider the set $P$ of smoothed connected components (*words*) in the page (assume a word $w$ has bounding box coordinates $(x_l, y_t, x_r, y_b)$, such that $x_l < x_r$ and $y_t < y_b$) and set a threshold $T$ (=$5h$ in [25]) for linking adjacent words in a line;

```
5. while $P \neq \emptyset$:
   (a) Initialize a new line $L \leftarrow \{\overline{w}\}$, where $\overline{w}$ is the uppermost word in the page;
   (b) Set the current pivot word $w^p \leftarrow \overline{w}$;
   (c) $O = \{w^o \in P | [y_t^p, y_b^p] \cap [y_t^o, y_b^o] \neq \emptyset\}$ (words whose horizontal projection overlaps $w^p$);
   (d) while $\exists w' \in O$ whose distance from the right bound of $w^p$ is $x_l' - x_r^p < T$:
       (i) $L \leftarrow L \cup \{w^r\}$, where $w^r$ is the closest such word;
       (ii) $w^p \leftarrow w^r$;
   (e) $w^p \leftarrow \overline{w}$;
   (f) while $\exists w' \in O$ whose distance from the left bound of $w^p$ is $x_l^p - x_r' < T$:
       (i) $L \leftarrow L \cup \{w^r\}$, where $w^r$ is the closest such word;
       (ii) $w^p \leftarrow w^r$;
   (g) $P \leftarrow P \setminus L$;
   (h) $L \leftarrow L \setminus \{w \in L | x_r - x_l < 2h\}$ (too small a word width is insufficient to provide a significant estimate of the word's actual slope);
   (i) Merge the two leftmost words in $L$ if the first one has width less than $T$.
```

The latter step estimates the slope of single words, where the deformation is likely to be quite limited (a simple rotation), and based on that restores the straight page image by translating each pixel of each word therein to its correct position:

```
1. Initialize an empty (white) upright page image $\overline{I}$
2. for each smoothed connected component (word) $w$, determine its upper and lower baselines $y_\top = a_\top x + b_\top$ and $y_\perp = a_\perp x + b_\perp$, respectively, with corresponding slopes $\theta_\top = \arctan a_\top$ and $\theta_\perp = \arctan a_\perp$, by linear regression of the set of top and bottom pixels in each of its columns
3. for each line,
   (a) Assign to its leftmost word $\overline{w}$ the slope $\overline{\theta} = \min(|\overline{\theta}_\top|, |\overline{\theta}_\perp|)$ (supposed to be the most representative), and to each of the remaining words $w^c$ the slope $\theta_\top^c$ or $\theta_\perp^c$, whichever is closer to $\overline{\theta}$
   (b) Straighten each word $w^c$ in turn according to $\overline{w}$ and to the immediately previous word $w^p$ by moving each of its pixels $(x, y)$ to $\overline{I}(x, r + d)$, such that
       $r = (x - x_l) \cdot \sin(-\theta) + y \cdot \cos\theta$   applies rotation, and
       $d = \overline{y}_* - y_*^c$   applies a vertical translation that restores the lines straight,
       where
       $y_* = \begin{cases} (a_\top x_l + b_\top) \cdot \cos\theta & \text{if } |\theta_\top - \theta^p| < |\theta_\perp - \theta^p|, \\ (a_\perp x_l + b_\perp) \cdot \cos\theta & \text{otherwise.} \end{cases}$
```

That is, each word is rotated according to the closest slope to that of the previous word and moved so that it becomes aligned to either the upper or the lower baseline of the first word in the line.

If the original image $I$ is not in black&white,

```
1. Compute a binarization $B$ of $I$
2. $M \leftarrow \emptyset$ (initialization of a model to be used for dewarping)
```

3. **for** each black pixel $B(x, y)$ in $B$, $M \leftarrow M \cup \{\langle (x, y'), (d, \theta, x_l)\rangle\}$, where $(x, y')$ are the straightened coordinates computed as before for pixel $(x, y)$, and $(d, \theta, x_l)$ are the parameters used for such a roto-translation, associated to the word to which $B(x, y)$ belongs
4. **for** each pixel $\overline{I}(x, y)$ of the dewarped page, $\overline{I}(x, y) = I(x, \frac{y-\overline{d}-(x-\overline{x_l})\sin(-\overline{\theta})}{\cos(\overline{\theta})})$ where $\langle(\overline{x}, \overline{y}), (\overline{d}, \overline{\theta}, \overline{x_l})\rangle = \arg\min_{\langle (x', y'), (d', \theta', x'_l)\rangle \in M}(|x - x'| + 2 \cdot |y - y'|)$.

That is, an inverse roto-translation is computed to find in the original image the value to be assigned to each dewarped pixel, according to the parameters associated to its closest black pixel in $B$.

### 5.2.3 Content Identification

The blocks singled out by segmentation may contain graphical or textual information. To properly submit them to further processing (e.g., text should be acquired in the form of encoded alphanumeric characters, while graphical components could be input to an image processing subsystem), their kind of content must be identified. Rough content categories that can be sensibly distinguished are: text, horizontal or vertical line, raster image and vector graphic. Interesting results for this task, on A4 document images whose resolution was scaled down from 300 dpi to 75 dpi, were obtained by applying supervised Machine Learning techniques, and specifically decision tree learning [31] based on the following numeric features [4]:

- Block height ($h$);
- Block width ($w$);
- Block area ($a = w \times h$);
- Block eccentricity ($w/h$);
- Total number of black pixels in the block ($b$);
- Total number of black-white transitions in the block rows ($t$);
- Percentage of black pixels in the block ($b/a$);
- Average number of black pixels per black-white transition ($b/t$);
- Short run emphasis (F1);
- Long run emphasis (F2);
- Extra long run emphasis (F3).

Measures F1, F2 and F3, in particular, are to be interpreted as follows [54]: F1 gets large values for blocks containing many short runs, which happens when the text is made up of small-sized characters (e.g., in newspaper articles); F2 gets large values for blocks containing many runs having medium length, which happens when quite large characters (say, 24 pt) are used (e.g., newspaper subtitles); finally, F3 gets large values for blocks containing few runs, all of which very long, which means that the characters used for writing the text have a very large size (this is typical in the main titles of newspaper pages).

### *5.2.4 Optical Character Recognition*

The textual content of documents is of outstanding importance for a number of reasons: it explicitly provides high-level information on the document content, it is the main medium considered for document indexing and retrieval, and last but not least, it allows referencing to several kinds of useful background knowledge. Unfortunately, so long as the document is represented as an image, the characters are considered aggregates of pixels, just as any other pictorial image therein. To have access to the above opportunities, it is necessary to switch from the image level to the explicit representation of text as such, expressed by means of text encoding standards (e.g., ASCII or UNICODE) that allow its proper processing. In this way, what is written on paper becomes a set of pixels, whose shape is analyzed in order to identify characters, then words, then sentences, and so on.

The branch of Pattern Recognition aimed at the automatic identification of image components corresponding to known shapes of text symbols and at their translation into a sequence of encoded characters is known as *Optical Character Recognition* (*OCR*). It is a member of a wider family of approaches aimed at *Character Recognition* that can be organized in the following taxonomy [14]

**On-line** act while the user writes on special input devices.
**Off-line** start only after the whole set of characters to be recognized is available.

- **Magnetic** (*MCR*) recognize characters written with magnetic ink (e.g., those used in bank environments for card authentication).
- **Optical** (*OCR*) retrieve characters in scanned or photographed documents.
  - **Handwritten** concerned with calligraphic texts written by persons.
  - **Printed** referred to the typographic fonts used in the printing industry.

Thus, for the purpose of this book, OCR is the focus of attention, and specifically the Printed branch of OCR systems, which is more tractable than the corresponding Handwritten counterpart, due to the many calligraphic styles that are to be faced in the latter with respect to the more standardized fonts available in the former.

Being able to automatically read printed text has been long since an interest of technologists. The first, mechanical attempts to design one such a system date back to the end of the 1920s and the beginning of the 1930s. Since then, several improvements were reached, particularly in the 1950s, that made this technology sufficiently reliable for mass application, at least in controlled environments. Since 1965 in the US, and later in other countries all over the world, mail services exploit OCR systems to recognize mail addresses by reading the ZIP code of the destination written on letters and printing it on the letters themselves as a barcode used by sorting out machines to forward the mail to the corresponding postal office. The first system able to read any normal kind of printed font was developed in the 1970s by R. Kurzweil, that a few years later sold his activity to Xerox. OCR of text written in languages based on the Latin alphabet, or derivations thereof, reaches nowadays success percentages around 99%, leaving little room for improvement (because the residual errors are due to many different causes). Thus, human intervention is still required for proof-checking and/or ensuring that the OCR result is error-free. Conversely, research is still very active on recognition of languages based on non-Latin scripts and characterized by a large number of symbols (e.g., Arabic, ideograms, Indian), and of handwritten documents, particularly those including cursive calligraphy. This is a wide research area by itself, whose thorough discussion goes beyond the scope of this book. Specialized literature can be referenced for a more specific landscape of the field [13, 26, 32].

Recognition of a character starting from its image representation relies on low-level graphical features, such as its *blobs* (connected components of pixels) or their *outlines* (i.e., their contours). However, working on the original pixel-level description of such features would be unpleasant for several reasons: it would be memory- and time-consuming, due to the number of details to be taken into account, and it would be error-prone, due to the high specificity of each single item compared to the others (even similar ones). Thus, the character images are usually reduced to black&white or gray-level and described by an abstract representation that focuses on the most relevant and promising characteristics, singled out by a feature extraction technique. Roughly, two main kinds of such features exist. *Statistical* ones focus on quantitative aspects, and are typically expressed as attribute-value arrays. These include, among others:

**zoning** the character image is partitioned into an equally spaced grid, considering the total number of black pixels for each resulting cell;
**projection histograms** reporting the number of black pixels on each row and column of the image.

*Structural* ones are more centered on a qualitative perspective, and typically represented using graph-like structures. These specifically focus on discontinuity points in which the curves abruptly change, and describe them in ways that range from single pixels neighborhood to spatial relationships between noteworthy subparts of the character shapes, such as loops, curls, end points, junction points and the way in which they are spatially combined to form the character.

Several algorithms have been proposed to accomplish this task, some of which purposely developed to act on images that do not specifically represent text (pictures, 2D or 3D graphic drawings, etc.). For the actual recognition, Machine Learning techniques have been often attempted. In particular, a traditional sub-symbolic approach used for recognizing known shapes in images are *Artificial Neural Networks* (*ANN*s) [31]: in the OCR case, these networks are provided with manuscripts to be processed, and their intended output consists of the recognized characters, obtained after a variable number of steps (depending on the degree of learning of the network). Low accuracy in pure recognition is sometimes tackled with the use of support dictionaries of the target languages. After a first 'perceptual' interpretation of the results, and a preliminary grouping of characters in words, the dictionaries are checked for verification of such words: if the recognized word is not present, and the word in the dictionary that is closest to it lies within a given range of modifications required to make them equal, then the latter is returned instead of the former.

**Tesseract** *Tesseract* [49, 50] is a multi-platform OCR engine which recently gained attention. It was born as a proprietary project, developed in C language by Hewlett-Packard (HP) laboratories between 1984 and 1994 as a potential supplementary software to be provided with HP flatbed scanners in place of other commercial software. In 1995, it was submitted to the UNLV (University of Nevada, Las Vegas) Accuracy test, and ranked in the top three OCR engines [43]. Nonetheless, it never became a final product, and always remained at the prototype stage. Subsequently, it underwent significant changes, among which a porting to C++. In its base version, Tesseract does not provide layout analysis features, it can process only single-column TIFF (and some versions of BMP) images, and returns the extracted characters in a plain text (TXT) file. In 2005, Google obtained that it were released as open source,[6] and undertook an effort for updating and improving it.

Although many solutions embedded in Tesseract are different from mainstream OCR techniques, it is interesting to have a look into it to see how an OCR engine works. Given the input image, it applies the following processing steps:

1. The contours (*outlines*) of the connected components (*blobs*) in the image are found (output is the same for both black-on-white and white-on-black characters); groups of nested components are considered separately.
2. Blobs whose bounding box is significantly larger or smaller than the median character height are temporarily removed. Then, proceeding from left to right, the sequence of adjacent blobs that draw a reasonably smooth curve is clustered to make up a line, whose baseline, meanline, descender and ascender lines are recorded meanwhile. Thus, even without explicit preprocessing, skewed or even warped lines (which are frequent in scanned or photocopied volumes) are identified and described by a (least square fitted) spline interpolation. Finally, the removed blobs are put back in place, and those that represent overlapping characters from different lines are split according to the identified baselines.
3. Lines are split into words based on character spacing; first fixed pitch text is identified and split into single characters, then proportional text spacing is handled by a fuzzy evaluation of blank pixels between the baseline and meanline (postponing critical cases after word recognition).
4. Each word undergoes the recognition process, after which, in case of success, it is passed to an adaptive classifier to improve performance on subsequent recognitions. First fixed pitch text is split according to the identified pitch; then proportional text is processed as follows:
   (a) Blob outlines are approximated polygonally. For those that include several characters, *chop points* (concave vertices opposite to other concave vertices or line segments) are located, and different combinations of chops are attempted according to a backtracking technique, trying the most promising chop points first, until a satisfactory split is obtained.
   (b) It often happens that single characters are broken into several blobs due to either excessive chopping, or the original character marks being very thin

[6]Code available at http://code.google.com/p/tesseract-ocr/.

(e.g., because of low printing or scanning quality). For instance, the round component of letters in Times font (o, d, b, etc.) might be split in two curves (a left and a right one) due to the thinner thickness on the top and bottom. For these cases an *association* step exploits the A* algorithm on an optimized representation to search pieces to be joint together.

Each character is classified by extracting a set of features (horizontal/vertical position, angle) from its normalized outline and approximately matching each such feature against a table that provides the corresponding set of possible classes. After matching the whole set of features, a selected list of best candidate classes (those most frequently found) is returned by a *class pruner*. Each such class is associated to some character prototypes, expressed by a mathematical combination of *configurations*, whose distance from the current character is evaluated: the closest prototype for overall and configuration-specific distances determines the final class for the character.

5. Light linguistic analysis is carried out. The best words available for the given segmentation are chosen among various categories (each having a different priority/weight in determining the final preference): top frequent word, top dictionary word, top numeric word, top upper case word, top lower case (with optional initial upper) word, top classifier choice word. When different segmentations yield different number of characters, two measures for comparing a pair of words are used: *confidence* (based on the normalized distance from the prototype) and *rating* (additionally based on the total outline length of the unknown character, that should not change depending on the number of components).
6. A new pass is performed through the page to try and solve unrecognized words using the adaptive classifier trained with the words recognized later during the previous step. To deal with many different fonts, the strictness of the static classifier is relaxed, and a font-sensitive classifier is trained on the single documents, assuming they include a limited number of fonts.
7. Fuzzy spaces and small-cap text are handled.

The character prototypes are learnt by clustering the segments (described by horizontal/vertical position, angle, and length) that make up the polygonal approximations of training characters. Since this classification technique can natively deal with damaged characters, only undamaged ones are needed in the training phase, which allows significantly reducing the number of training instances (tens of thousands instead of more than a million) with respect to other techniques. Both the static and the adaptive classification exploit the same classifier features, although the normalization is different in the latter to improve distinction between upper and lower case letters and tolerance to noise specks.

Advantages of Tesseract include high portability, the open source approach and full Unicode (UTF-8) compliance. Several languages are supported (although not comparable to commercial products), among which English (the first to be included), French, Italian, German, Spanish, Dutch. Good recognition results are obtained for resolutions between 200 and 400 dpi (larger resolutions do not necessarily improve performance). Its fundamental disadvantage consists in its having never had a complete end-user oriented development: it can handle only single-page, single-column input (it has no page layout analysis features, although recent development included an algorithm for page segmentation [45, 51]), it does not provide output formatting, no native User Interface is available, diacritical marks are not perfectly handled, only left-to-right scripts are supported and, last but not least, no complete and organized documentation is available for its potential developers.[7]

**JTOCR** *JTOCR* is an open source[8] wrapper for Tesseract written in Java language, that provides a multi-platform GUI front-end and overcomes several limitations of that OCR engine. The additional facilities include file drag-and-drop, pasting of images from the clipboard, localized user interface, integrated scanning support (on Windows only), watch folder monitor to support batch processing, custom text replacement in post-processing. JTOCR also extends the set of accepted formats for input images to other widely used and lighter formats: PDF, TIFF, JPEG, GIF, PNG, and BMP. JPEG, due to the use of lossy compression, might cause problems in character recognition. Conversely, GIF saves space at the expenses of color depth. In this respect, PNG represents a good tradeoff between image file size and recognition quality. Another interesting feature is that JTOCR can process multi-page TIFF images at once by separately calling Tesseract for each page transparently to the user.

Although originally intended for Vietnamese (its former name was *VietOCR*), it works also with other languages if proper resources are provided in the `tessdata` directory, and a corresponding tag is added in the `ISO639-3.xml` configuration file. For instance, English can be added including the following tag:

```
<entry key="eng">English</entry>
```

according to which the tool looks in the aforementioned directory for a file whose name starts with the same characters as specified in the `key` attribute (in the above example, `eng[...]`). After doing this, the new dictionary can be exploited.

[7]The source code of Tesseract is structured in several directories:

| | |
|---|---|
| `ccmain` | main program |
| `training` | training functionalities |
| `display` | a utility to view and operate on the internal structures |
| `testing` | test scripts (also contains execution results and errors) |
| `wordrec` | lexical recognition |
| `textord` | organization of text in words and lines |
| `classify` | character recognition |
| `ccstruct` | structures for representing page information |
| `viewer` | client-side interface for viewing the system (no server side is yet available) |
| `image` | images and image processing functionalities |
| `dict` | language models (including extension by addition of new models) |
| `cutil` | management of file I/O and data structures in C |
| `ccutil` | C++ code for dynamic memory allocation and data structures. |

[8]Code available at http://vietocr.sourceforge.net/.

*Example 5.4* (Sample `ISO639-3.xml` configuration file for JTOCR)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM
                     "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>ISO 639-3 Standard</comment>
  <entry key="eng">English</entry>
  <entry key="vie">Vietnamese</entry>
  <entry key="fra">French</entry>
  <entry key="spa">Spanish</entry>
  <entry key="deu">German</entry>
  <entry key="ita">Italian</entry>
  <entry key="nld">Dutch</entry>
  <entry key="por">Portuguese</entry>
</properties>
```

The Graphical User Interface includes an area to display the input image, and another area to show the processing output. It allows zooming in and out the image (each step applies an enlargement/reduction factor of 1.5), to select, using a mouse, the excerpt from which the text is to be read, and to crop and send it to Tesseract.[9] The actual recognition is started by calling the `tesseract.exe` executable and passing it as parameters the image to be processed (converted to TIFF), the text file that will receive the output and the desired language. Then, the output file produced by Tesseract is read and displayed on screen. Note that calling the Tesseract executable (`.exe`) instead of its Dynamic Link Library (`.dll`) version allows for smooth future updating because it will suffice to replace the old with the new engine, without any side-effect on the wrapper itself.

## 5.3 Segmentation

After pre-processing has reduced the document to standard conditions, further analysis steps can start. A fundamental task in layout analysis is *segmentation* that, given the source (raster or vector) document representation, returns a partition of its area into portions of content representing significant pieces of the layout. Segmentation consists of three steps [44]: *simplification* (that removes from the image information that is superfluous or irrelevant for the specific task, e.g., using various kinds

[9]Version 0.9.2 of JTOCR, the latest available at the time of writing, has a bug in the crop operation (method `jMenuItemOCRActionPerformed` of class `gui.java`). Let us represent a rectangle having top-left corner at coordinates $(x, y)$, width $w$ and height $h$ as a 4-tuple $(x, y, w, h)$. Given a selection $(x_S, y_S, w_S, h_S)$ of the original image at zoom ratio $p$, the excerpt is identified as $(\frac{x_S}{p}, \frac{y_S}{p}, \frac{w_S}{p}, \frac{h_S}{p})$. Clearly, no displacement is taken into account. To fix the problem, the offset $(x_o, y_o) = (\frac{x_P - x_I}{2}, \frac{y_P - y_I}{2})$ between the container panel $(x_P, y_P, w_P, h_P)$ and the image $(x_I, y_I, w_I, h_I)$ has to be considered, and hence the correct formula is $(\frac{x_S - x_o}{p}, \frac{y_S - y_o}{p}, \frac{w_S}{p}, \frac{h_S}{p})$.

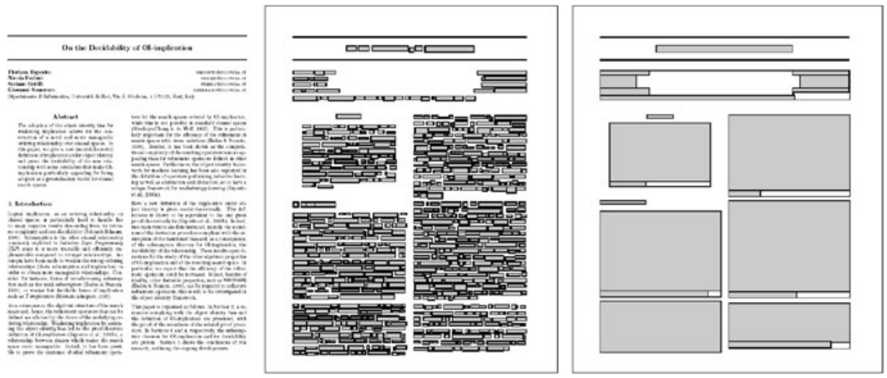


**Fig. 5.5** A PDF document, its basic blocks and the target frames

of image filters), *feature extraction* (that gathers the data needed for the actual partitioning) and *decision* (that actually partitions the image). The result of such a process is a set of content blocks that should be consistent and significant (and possibly meaningful enough to deserve further and specialized processing). Thus, the quality of the segmentation outcome can determine the quality and even the feasibility of the whole document processing activity. In the low-level representation available in the source file, the elements that play the role of basic blocks are characterized by a single type of content (text or graphics). Conversely, the identified aggregates might include basic elements having homogeneous content type, or might mix text and graphic elements conceptually related to each other.

For easy handling, blocks and frames are usually represented by their *minimum bounding box*, i.e., the smallest rectangle in which they can be enclosed. Since the number of aggregates is usually less than the number of basic blocks, the segmented page can be handled more efficiently than the original one. Figure 5.5 shows a document, along with its basic components and the top-level aggregates, that can be identified. An interesting indicator for a document image is its *spread factor* [4], defined as the ratio between the average distance among regions in the document and the average height of the regions themselves. In documents having a simple structure, with a few, sparse content regions, this ratio is greater than 1.0, while in complex documents, made up of many, crowded regions of content, it is less than 1.0. Thus, this measure can be useful in assessing the complexity of the document, estimating the computational effort needed to process it, and in some cases properly setting accordingly the processing parameters of the subsequent steps (if any).

### *5.3.1 Classification of Segmentation Techniques*

Several methods have been proposed in the literature to identify the layout structure of documents. They can be categorized according to various dimensions, depend-

**Table 5.1** A classification of some layout analysis algorithms according to processing direction and granularity level

| | Direction | Representation | Layout |
|---|---|---|---|
| RLSA | Top-down | Pixel | Manhattan |
| RLSO | Bottom-up | Pixel, Block | Non-Manhattan |
| X–Y trees | Top-down | Pixel | Manhattan |
| CLiDE | Bottom-up | Block | Manhattan |
| docstrum | Bottom-up | Block | Non-Manhattan |
| Background analysis | Top-Down | Block | Manhattan |

ing on the particular perspective of interest. Table 5.1 summarizes some relevant techniques that will be discussed in the following paragraphs.

One distinction concerns the grain-size of the matter on which the algorithms are run. It clearly depends on the (representation of the) source document that can be provided either in the form of a digitized document, where the pixels in the bitmap can be considered as the basic blocks, or in the form of a born-digital document (such as a word processing file, a PostScript or PDF file, etc.), where the basic blocks are explicit. The two cases are very different as regards the abstraction level of the input, and hence are typically faced by different algorithms, although the former can be reported to the latter by considering connected components in the bitmap as basic blocks. A limitation of many classical methods is their being designed to deal only with digitized documents and hence their being inapplicable to born-digital documents which are nowadays pervasive.

As to the behavior that determines the direction in which the layout tree is built, two broad categories can be distinguished:

**Bottom-up** techniques start from elementary items (the smallest elements) in the document description (e.g., the pixels in case of digitized documents) and progressively group them into higher-level components [29, 38, 53];

**Top-down** ones start from the entire document page and progressively split it into high-level components [8, 35, 42] and downward until basic blocks are found.

The latter are typically more efficient, but less effective in handling documents with fancy layout. Of course, one might consider hybrid techniques that partly proceed by grouping components and partly proceed by splitting components. For instance,

- Intermediate groupings can be identified bottom-up based on perceptual criteria such as overlapping, adjacency or (type and/or size) similarity;
- Compound objects, such as paragraphs, columns, figures, tables and formulæ, can be identified top-down.

In fact, many algorithms assume that the document under processing has a so-called *Manhattan* (or *taxicab*) layout in which significant content blocks are surrounded by perpendicular background areas [6]. This assumption holds for many typeset documents, and significantly simplifies the processing activity, but cannot be assumed in general. Top-down and most bottom-up methods are able to process

Manhattan layout documents only; conversely, bottom-up techniques are generally considered better candidates for handling the non-Manhattan case, since basic components grouping can identify significant aggregates ignoring high level regularities. However, usually documents do show a Manhattan layout, and non-Manhattan behavior is limited to particular document areas/components. For this reason, in order to improve efficiency, a hybrid approach can be exploited: first, Manhattan zones in the page are identified top-down, and then a bottom-up technique is applied separately to each of them (or just to those in which suitable heuristics indicate non-Manhattan behavior) to refine the outcome.

Another feature according to which classifying the various techniques is the amount of explicit knowledge used during the layout analysis process. In this perspective, three levels of knowledge about the layout structure of a document can be distinguished [34]:

- Generic (e.g., the baselines of the characters in a word are co-linear);
- Class-specific (e.g., text never appears aside images);
- Publication-specific (e.g., characters are at most 22 dpi).

Knowledge used in bottom-up approaches is usually less specific to the single document than that used in top-down processing which, on the contrary, typically derives from the relationships between the geometrical and the logical structure of particular document classes.

The overall layout analysis outcome can be improved by leveraging peculiar features of some documents, such as the presence of horizontal or vertical lines, or the co-existence of different text sizes (and hence spacings) within the same document. In born-digital documents, these features are explicitly represented, and hence can be straightforwardly exploited; in digitized documents, they are implicit and require proper pre-processing to be derived.

### *5.3.2 Pixel-Based Segmentation*

Some algorithms work on digitized images directly at the pixel-level, ignoring any possible structure of pixel aggregates. These strategies aim at obtaining directly high-level components, without first identifying intermediate pixel aggregates that can play a role in the document (e.g., single characters or straight lines).

**RLSA (Run Length Smoothing Algorithm)** A classical and efficient segmentation technique is the *Run Length Smoothing* (sometimes called *Smearing*) *Algorithm* (RLSA) [55]. Given a sequence of black and white pixels, a *run* is defined as a sequence of adjacent pixels of the same kind, delimited by pixels of the opposite color. The run length is the number of pixels in a run, and 'smoothing' a run means changing the color of its pixels so that they become of the same color as the pixels delimiting the run. RLSA identifies runs of white pixels in the document image and fills them with black pixels whenever they are shorter than a given threshold. In particular, the RLSA works in four steps, each of which applies an operator:

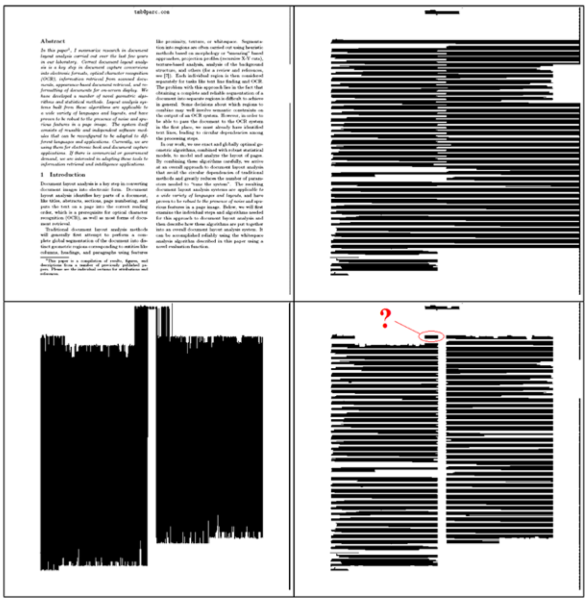


**Fig. 5.6** Application of the first three steps of RLSA to a sample document. *Top-left*: Original paper; *Top-right*: Horizontal smoothing; *Bottom-left*: Vertical smoothing; *Bottom-right*: Logical AND. Note the presence of a false block in the ANDed image

1. Horizontal smoothing, carried out by rows on the image with threshold $t_h$;
2. Vertical smoothing, carried out by columns on the image with threshold $t_v$;
3. Logical AND of the images obtained in steps 1 and 2 (the outcome has a black pixel in positions where both input images have one, or a white pixel otherwise);
4. New horizontal smoothing with threshold $t_a$ on the image obtained in step 3, to fill white runs inside the discovered blocks.

A top-down strategy based on RLSA consists of repeatedly applying it, working at each round with progressively smaller thresholds on the zones identified as blocks in the previous round. Much work in the literature is based on RLSA, exploiting it or trying to improve its performance by modifying it [11, 46, 52].

The result of the first three steps of RLSA on a sample document is shown in Fig. 5.6, where two shortcomings of the technique are evident. First, due to the presence of thin black lines produced on the right border of the image by scanning or photocopying, the horizontal smoothing covers most of the right margin of the page. Although in many cases this is fixed by step 3, spurious blocks not corresponding to any actual content in the original document, as in this case in the first column to the right of the 'Abstract' heading, can be returned nevertheless. A further shortcoming of this technique lays in its inability to handle documents having non-Manhattan layout, as shown in Fig. 5.7.

The assessment of suitable thresholds is a hot problem, directly affecting the overall effectiveness of the technique. Manually setting such thresholds is not trivial, both because it is not a typical activity that (even expert) humans are used to carry out on documents, and because there is no unique threshold that can equally fit all documents. Studies on RLSA suggest to set large thresholds, in order to keep the number of black pixels in the smoothed images sufficiently large to prevent the

AND operator from dropping again most of them (indeed, this is why an additional horizontal smoothing is provided for). The original paper uses $t_h = 300$, $t_v = 500$ and $t_a = 30$ for document images scanned at 240 dpi, based on the assumption that $t_h$ and $t_v$ should be set to the length in pixels of long words, while $t_a$ should be so wide as to cover a few characters. However, the generic claim that large thresholds are better is not so helpful in practice. Hence, a strong motivation for the development of methods that can automatically assess proper values for the $t_h$, $t_v$ and $t_a$ parameters, possibly based on the specific document at hand; e.g., [41] proposes to set

$$t_h = 2 \cdot mcl, \qquad t_v = mtld,$$

where

- $mcl \in [\lceil M \cdot 3.24 \rceil, \lceil M \cdot 6.3 \rceil]$ is the mean character length of the document, computed according to the maximum $M = \max_i H_h^b(i)$ of the histogram of horizontal black runs $H_h^b(i)$, with suitable tolerance;
- $mtld = \arg\max_{i \in [\lceil 0.8 \cdot mcl \rceil, 80]} H_v^w(i)$ is the mean text line distance of the document, located as the position of the global maximum of the vertical white runs histogram $H_v^w(i)$ in the specified range;

using a mix of statistical considerations on the histograms of horizontal/vertical black/white run lengths and empirical definitions of multiplicative constants.

**RLSO (Run-Length Smoothing with OR)** *RLSO* [23] is a variant of the RLSA that performs:

1. Horizontal smoothing of the image, carried out by rows with threshold $t_h$;
2. Vertical smoothing of the image, carried out by columns with threshold $t_v$;
3. Logical OR of the images obtained in steps 1 and 2 (the outcome has a black pixel in positions where at least one of the input images has one, or a white pixel otherwise).

Each connected component in the resulting image is considered a frame, and exploited as a mask to filter the original image through a logical AND operation in order to obtain the frame content. The result on a sample document having non-Manhattan layout, along with the corresponding frames extracted by ANDing the original image with the smoothed one, is shown in Fig. 5.7.

Compared to RLSA, RLSO has one step less (no final horizontal smoothing is performed, since the OR operation, unlike the AND, preserves everything from the two smoothed images), and requires shorter thresholds (and hence fills fewer runs) to merge original connected components (e.g., characters) into larger ones (e.g., frames). Thus, it is more efficient than RLSA, and can be further sped up by avoiding the third step and applying vertical smoothing directly on the horizontally smoothed image obtained from the first step. This does not significantly affect, and may even improve, the quality of the result (e.g., adjacent rows having inter-word spacings vertically aligned would not be captured by the original version). However, the OR causes every merge of components to be irreversible, which can be a problem when

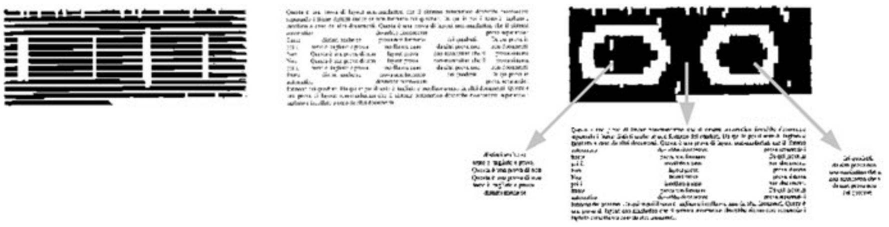
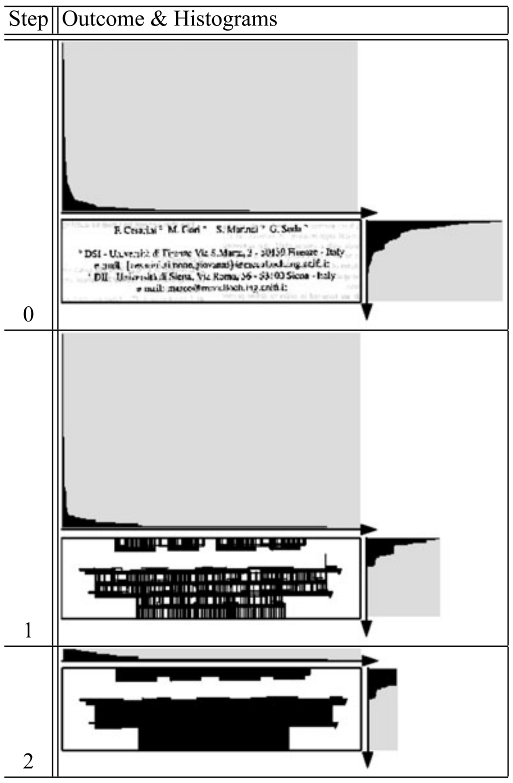


**Fig. 5.7** Application of RLSA (on the *left*) and of RLSO (on the *right*) to the non-Manhattan digitized document in the *middle*. *Below*, the three text frames identified by RLSO

**Fig. 5.8** Iterated RLSO on a document with different spacings. Each step shows the segmentation outcome and the horizontal and vertical cumulative histograms (scaled down to 10% of the original values) according to which the thresholds are automatically assessed



logically different components are very close to each other and might be erroneously merged if the threshold is too high. Conversely, too low thresholds might result in an excessively fragmented layout. Thus, as for RLSA, the choice of proper horizontal/vertical thresholds is a very important issue for effectiveness.

A technique to automatically assess document-specific thresholds on pages using a single font size was proposed in [22] and is illustrated in Fig. 5.8. It is based on the distribution of white run lengths in the image, represented by *cumulative* histograms where each bar reports the number of runs having length larger or equal than the

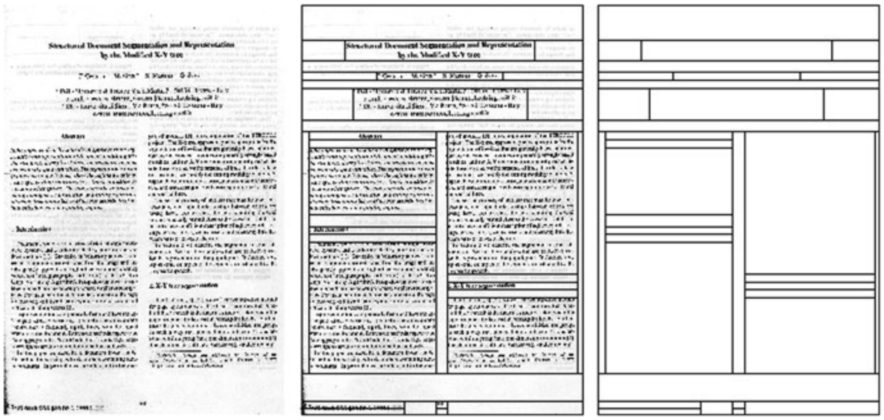


Fig. 5.9 X–Y-tree partitions derived from the horizontal and vertical splits of a document

associated value. To introduce some tolerance, the histogram bars are scaled down to 10%. The slope in a bar $b$ of a histogram $H(\cdot)$ can be computed with respect to the next bar $b+1$ as:

$$\frac{H(b+1)-H(b)}{(b+1)-(b)} = H(b+1)-H(b)$$

(excluding the last bar that would be a meaningless threshold). The shape of the graphic is monotonically decreasing, and its derivative (that is either negative or null) is proportional to the number of runs introduced by a given length (the larger the absolute value, the more runs). Larger and prolonged slopes correspond, respectively, to frequent and homogeneous spacings that are likely to separate different components; by contrast, flat regions (except leading and trailing ones) denote candidate thresholds. A cautious approach requires a 0-slope flatness and takes as threshold the initial run length of the first candidate region. Depending on the specific task, a larger (negative) slope value can be adopted.

Iterated application of RLSO to the smoothed image resulting from the previous round, each time computing the thresholds, allows identifying progressively larger aggregates of components in documents including different spacings, where a single threshold that is suitable for all components cannot be found. In Fig. 5.8, application to (0) yields (1), and application to (1) yields (2). Then, the horizontal histogram has only the (useless) leading and trailing 0-slope regions, while vertically there is one more flat zone that would merge the two frames together.

**X–Y Trees** A well-known top-down strategy for Manhattan layouts, proposed in [36], repeatedly applies interleaved horizontal and vertical splits between disjoint components, as shown in Fig. 5.9. It is extremely sensitive to skew. The resulting partition of the page is represented in an *X–Y tree* [36], a spatial data structure where nodes correspond to rectangular blocks, and node children represent the locations of the horizontal or vertical subdivisions of the parent.

The horizontal (respectively, vertical) split of a block is based on its vertical (respectively, horizontal) projection profile, represented as a binary string reporting 0 for scanlines that contain only white pixels or 1 otherwise. The two-dimensional page-segmentation problem is turned into a series of one-dimensional string-parsing problems performed applying *block grammars*, whose rules define three kinds of syntactic attributes:

**atoms** strings of 1s or 0s. A *black* atom is a maximal all-one substring. It is the smallest indivisible partition of the current block profile. A *white* atom is an all-zero substring. Classes of atoms are defined according to their length.

**molecules** strings of alternating black and white atoms. A *black* molecule is a sequence of black and white atoms followed by a black atom. A *white* molecule is a white atom separating two black molecules. Classes of molecules are defined according to their black/white pattern.

**entities** molecules to which a class label (e.g., title, authors, figure caption) has been assigned, possibly depending on an ordering relationship. Classes of entities are defined in terms of classes of molecules or of other entities.

The grammar productions derive from publication-dependent typographic specifications for the layout organization, defining the size and number of atoms within an entity, or the number and order of allowed occurrences of entities on a page. They are automatically coded starting from a purposely developed tabular representation.

*Example 5.5* (Sample rules of a block grammar for an X–Y tree)

$\texttt{c ::= \{0\}}^{2-5}$

A sequence of two to five 0s defines an atom of class *c*.

$\texttt{C ::= \{cac\}}^{0-1}\texttt{b}$

At most one sequence of atoms `cac`, followed by an atom `b`, defines a molecule of class C.

```
TITLE ::= C
HEADING ::= TITLE S AUTHORS
```

A molecule of class `C` defines an entity `TITLE`. Two entities `TITLE` and `AUTHORS` separated by an entity of class `S` define an entity `HEADING`.

Grammars as well are organized in a tree structure whose levels correspond to levels of the X–Y tree. Since several alternative grammars may be available for application at a given level, this is actually an AND/OR tree.[10] If the grammars at a given level cannot assign a unique label to a block, the lower-level grammars are exploited to identify labels for its sub-blocks.

A modification of this technique has been proposed in [12]. Two splitting clues based on the average character width/height are defined: *cutting spaces* are horizontal (vertical) white rectangles having the same height (width) as the region to

[10] A tree where the offspring of a node can be partitioned into groups such that the elements in the same group are considered in AND and groups are considered in OR.

be split, and a width (height) larger than a threshold that avoids too fine-grained splits; *cutting lines* are sufficiently thin horizontal (vertical) lines that have a width (height) representing a sufficient portion of the overall width (height) of the region to be split, and a sufficiently wide surrounding space. The split outcome is represented in a *Modified X–Y tree* (or *M-X–Y tree*), that includes cutting lines as leaves and is extended with *node attributes* (expressing the cut direction for internal nodes, or the kind of region content for leaf nodes) and *arc attributes* (that express the nearest blocks to a given node by means of Below and Right_ of relations, this way defining an adjacency graph associated to the tree).

### 5.3.3 Block-Based Segmentation

Other algorithms work on basic components that are blocks, rather than simple pixels. Such blocks often correspond to single characters or (fragments of) words, depending on the source format of the document. They can be derived from a pixel-level representation as the connected components therein, or can be readily available in the case of born-digital documents. In both cases, an abstract representation of each basic block as the least rectangle that encloses it (*minimum bounding box*) is usually exploited as a surrogate of the block itself.

**The DOCSTRUM** The *docstrum* (acronym of 'document spectrum') [38] is a document page representation based on global structural features, useful for carrying out various kinds of layout analysis tasks. It is obtained by means of a bottom-up technique in which connected components in the page (described by attributes related to their bounding box and pixel features, and represented by their centroid) are clustered according to a $k$-Nearest Neighbor approach. For any component $c_i$, its $k$ closest components $c_j$, $0 < j \leq k$ are selected, and the segment $D_{ij}$ that connects the corresponding centroids is considered, in terms of the Euclidean distance $d$ and angle $\phi$ between those centroids. In the case of textual components, typical neighbors of a letter will be its adjacent letters, belonging to the same or adjacent words in the same line, and the letters in the adjacent lines above and below that are closest to the vertical of its centroid. Thus, the use of $k = 5$ (as in Fig. 5.10) can be interpreted as the search for the closest neighbors in the four cardinal points, plus an additional 'safety' neighbor, which suffices for typical text analysis. This causes the docstrum to draw long horizontal lines, each made up of several segments, in correspondence to lines of textual content, in turn connected to each other by other almost perpendicular segments. Different values could be set for $k$, with consequent different computational demand, for specific purposes (e.g., smaller values focus on single lines more than on line blocks, while larger values are needed to deal with documents where between-line distances are wider than within-line ones).

The plot of all points $(d, \phi)$, representing such segments as polar coordinates in the *distance* $\times$ *angle* plane, is the docstrum. It is worth noting that, since the nearest-neighboring relation is not symmetric (if component $c'$ is in the $k$ nearest neighbors of component $c''$, the opposite does not necessarily hold), the angle orientation should be expressed in the whole $[0°, 360°]$ range to distinguish, based on

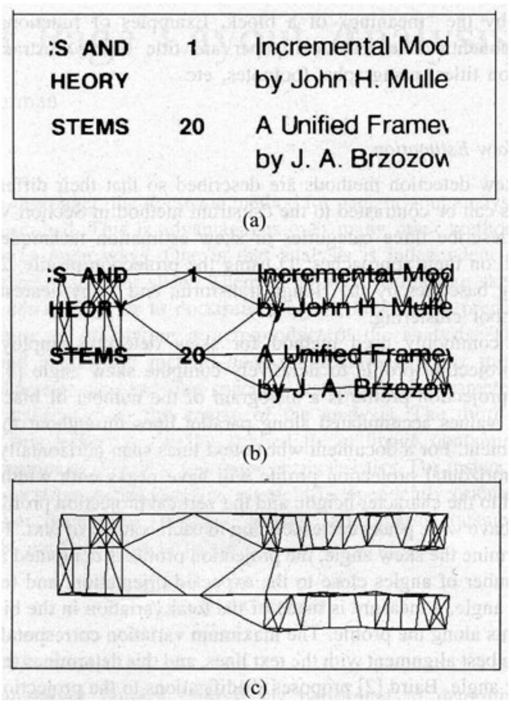


**Fig. 5.10** DOCSTRUM segments between components using 5 nearest neighbors (reprint from [38] ©1993 IEEE). From top to bottom: (**a**) the original document fragment; (**b**) the fragment with the segments superimposed; (**c**) the segments alone. The emerging horizontal lines are evident

the directionality of the segment, which is the reference component and which is the neighbor. However, segments are usually considered undirected, and hence angles range in the [0°, 180°[ interval only. This simplification clearly causes loss of potentially significant or useful information. Although (being distances always positive) the plot would in theory take just half the plane, for readability it is made symmetric by replicating each point by its 180° mirrored counterpart.

The docstrum is typically cross-shaped (see Fig. 5.11): in case of no skew angle, four clusters (sometimes collapsing into just two), pairwise symmetric with respect to the origin, are placed along the $x$ axis, and two clusters, again symmetric with respect to the origin, are placed along the $y$ axis. On the $x$ axis, the inner clusters (having larger cardinality) indicate between-character spacing, while the outer ones (having much smaller cardinality) represent between-word spacing. The peak distance in these two clusters represents the most frequent inter-character spacing, which is likely to correspond to the within-line spacing. Conversely, the $y$ axis clusters denote between-line spacing. Statistics on the plot can support further analysis:

**Spacing and Rough Orientation Estimation** This is carried out by plotting the histograms of the docstrum points' distances and angles, respectively.

**Identification of text lines and Precise Orientation Estimation** Based on their centroids rather than baselines, a transitive closure is carried out on the segments and they are fit to lines that minimize the squared errors.

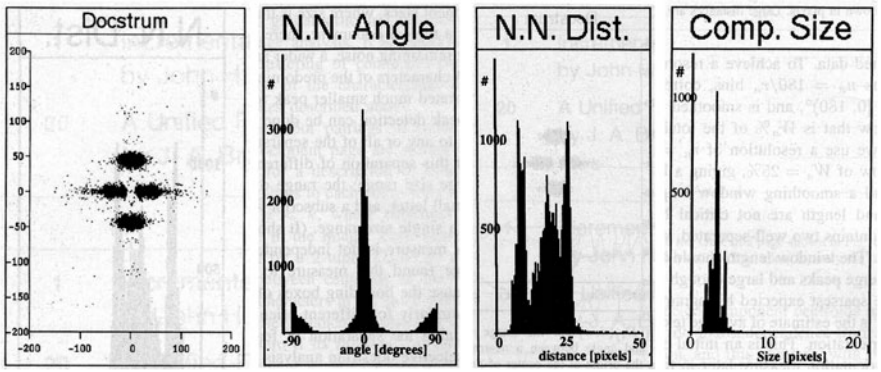


**Fig. 5.11** Graphics for a document spectrum. From *left* to *right*: the docstrum, the histograms for angles and distances, and the histogram for the size of components (reprint from [38] ©1993 IEEE)

**Structural Block Determination** Text lines are grouped by progressively merging two partial groupings whenever two members, one from each, fulfill some requirements on parallelism, perpendicular and parallel proximity, and overlap.

**Filtering** Segments connecting components whose distance, angle or size are outside given ranges can be filtered out, and possibly processed separately.

**Global and Local Layout Analysis** Each group of components representing a connected component of the nearest neighbor segments graph can be analyzed separately, determining its own orientation and processing it independently.

The docstrum is preferably applied after removing salt-and-pepper noise (in order to filter out insignificant components) and graphical components from the page image. It should also be selectively applied on component subsets having homogeneous size, not to affect subsequent analysis based on average and variance statistics.

**The CLiDE (Chemical Literature Data Extraction) Approach** *CLiDE* [47] is a project purposely oriented towards chemistry documents processing, in order to extract information from both the text and the graphic. Here, we will focus on its layout analysis strategy, that is general and based on the definition of a novel distance among elements and on the exploitation of well-known greedy algorithms for the bottom-up detection and representation of the layout hierarchy. It works assuming that the page skew is inside a $\pm 5°$ tolerance range.

The layout components of a document are arranged in an $n$-ary tree where each level denotes a kind of component and the depth of a level is directly proportional to the granularity degree of the kind of component it represents. The sequence of levels (with an indication of the reading order for the corresponding components, top-down $\downarrow$ or left-to-right $\rightarrow$), from the root to the leaves, is:

page - strip ($\downarrow$) - column ($\rightarrow$) - block ($\downarrow$) - line ($\downarrow$) - word ($\rightarrow$) - char ($\rightarrow$)

Such a tree is built bottom-up as a minimal-cost spanning tree[11] for an undirected graph (called the *page graph*) whose nodes are the connected components of the page image and whose edges are weighted with the distance between the components they link. In particular, the definition of distance between two components $c'$ and $c''$ is peculiar to the approach:

$$D\big(c', c''\big) = \max\big(d_x\big(c', c''\big), d_y\big(c', c''\big)\big),$$

where

$$d_x\big(c', c''\big) = \begin{cases} 0 & \text{if } L(c', c'') < R(c', c''), \\ L(c', c'') - R(c', c'') & \text{otherwise,} \end{cases}$$

$$d_y\big(c', c''\big) = \begin{cases} 0 & \text{if } B(c', c'') > T(c', c''), \\ T(c', c'') - B(c', c'') & \text{otherwise,} \end{cases}$$

with

$$L\big(c', c''\big) = \max\big(c'.x_l, c''.x_l\big), \qquad R\big(c', c''\big) = \min\big(c'.x_r, c''.x_r\big),$$
$$B\big(c', c''\big) = \max\big(c'.y_b, c''.y_b\big), \qquad T\big(c', c''\big) = \min\big(c'.y_t, c''.y_t\big)$$

and $(c.x_l, c.y_t)$, $(c.x_r, c.y_b)$ denote, respectively, the coordinates of the top-left and bottom-right corners of the bounding box of a component $c$.

The tree is computed according to the algorithm by Kruskal because it ensures that exactly the closest connections between components are chosen, and that there is a one-to-one correspondence between the layout components of the page and the subtrees of the outcoming tree. Actually, such a tree is in some sense 'flattened' as follows. The algorithm is run until the next smallest distance to be considered exceeds a given threshold (the authors suggest the double or triple of the last included distance). At that point a forest is available, in which each tree can be considered as a single component in the next level, denoted by a new compound node. Then the algorithm is restarted on the compound nodes (that are much fewer than before, which improves efficiency). Figure 5.12 shows the first three levels of aggregation on a sample text.

Several heuristics are exploited to improve the algorithm efficiency (a problem mainly present in the initial stages of the process, due to the very large number of graph nodes), and allow reaching linear time with respect to the number of components:

- Using a *path compression* technique [48] that directly links each node to its root when building the sub-trees, in order to speed up the retrieval of the root for each node in subsequent steps;
- Representation in the graph of *probably necessary* distances (i.e., edges) only, excluding for instance characters/words having vertical intersection less than a

[11] A tree built from a graph using all of its nodes and just a subset of its edges (*spanning tree*) such that the sum of weights of the subset of edges chosen to make up the tree is minimum with respect to all possible such trees. In Kruskal's algorithm [48], it is built by progressively adding the next unused edge with minimum weight, skipping those that yield cycles, until a tree is obtained.

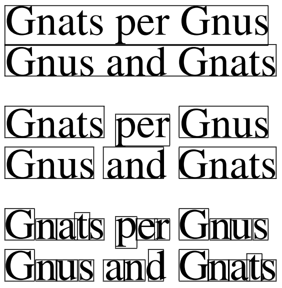


**Fig. 5.12** The first three levels of aggregation of the tree built by CLiDE: character (at the *bottom*), word (in the *middle*) and line (at the *top*). Each bounding box in higher levels represents a compound node associated to the subtree of corresponding elements in the immediately lower level

threshold (or no intersection at all) or horizontal distance larger than a threshold (typically the width of a character), and blocks that are separated by a graphic line;

- Sorting of the components by increasing horizontal or vertical distance (that makes easier the selection of probably necessary distances).

Other heuristics are adopted for dealing with particular cases. For instance, two word blocks are considered as belonging to the same line if their vertical intersection is at least 70%, to tackle cases in which the bounding boxes of adjacent lines overlap because the between-line spacing is minimum. Another problem is the presence, in the same document, of several font sizes (and hence different spacings), for which specific heuristics are needed.

**Background Analysis** An algorithm to identify the layout structure of a document page, proposed in [10], is based on the analysis of the page background, and identifies the white rectangles that are present in the page by decreasing area, reducing to the '*Maximal White Rectangle problem*':

**Given**

- A set of rectangles in the plane $C = \{r_0, \dots, r_n\}$, all placed inside a contour rectangle $r_b$
- A function defined on rectangles $q : R^4 \to R$ such that if $r$ is contained in $r'$ then $q(r) \le q(r')$ (e.g., the rectangle area)

**Find** a rectangle $r$ contained in $r_b$ that maximizes $q$ and does not overlap any rectangle in $C$.

The algorithm exploits a representation in terms of rectangular bounds and contained obstacles (with the respective position inside the bound), based on the following data structures:

**Rectangle** a 4-tuple $(x_L, y_T, x_R, y_B)$ representing the coordinates of its top-left and bottom-right corners in the plane, $(x_L, y_T)$ and $(x_R, y_B)$, respectively;

**Bound** a pair $(r, O)$ made up of a Rectangle $r$ and a Set of obstacles $O$, each of which is, in turn, a Rectangle overlapping $r$;

**Priority Queue** a Set structure providing a direct access operator (*dequeue*) for extraction of the element having the maximum value for a scoring function $q$.

In particular, the following temporary structure is needed:

- $Q$: PriorityQueue of Bound
  whose elements are of the form $(r, O)$, organized according to a quality function $q$ consisting of the area of $r$:

$$q : R^4 \to R, \quad q(r) = q(x_L, y_T, x_R, y_B) = |x_L - x_R| \cdot |y_T - y_B|.$$

Starting from an initial bound $(\overline{r}, \overline{O})$, where rectangle $\overline{r}$ delimits the whole page under processing and $\overline{O}$ is the set of basic blocks of content that are present in the page, the problem is solved as follows:

```
1. Enqueue $(\overline{r}, \overline{O})$ in $Q$ based on $q(\overline{r})$
2. while $Q \neq \emptyset$ do
   (a) $(r, O) \leftarrow$ dequeue($Q$)
   (b) if $O = \emptyset$ then
       (i) return $r$
   (c) else
       (i) $p \leftarrow$ pivot($O$)
       (ii) Enqueue $(r_A, O_A)$ in $Q$ based on $q(r_A)$
            where $r_A = (r.x_L, r.y_T, r.x_R, p.y_T)$; $O_A = \{b \in O | b \cap r_A \neq \emptyset\}$
       (iii) Enqueue $(r_B, O_B)$ in $Q$ based on $q(r_B)$
            where $r_B = (r.x_L, p.y_B, r.x_R, r.y_B)$; $O_B = \{b \in O | b \cap r_B \neq \emptyset\}$
       (iv) Enqueue $(r_L, O_L)$ in $Q$ based on $q(r_L)$
            where $r_L = (r.x_L, r.y_T, p.x_L, r.y_B)$; $O_L = \{b \in O | b \cap r_L \neq \emptyset\}$
       (v) Enqueue $(r_R, O_R)$ in $Q$ based on $q(r_R)$
            where $r_R = (p.x_R, r.y_T, r.x_R, r.y_B)$; $O_R = \{b \in O | b \cap r_R \neq \emptyset\}$
```

The underlying idea resembles a branch-and-bound method and consists in iteratively extracting from the queue the element having the largest area: if the corresponding set of obstacles is empty, then it represents the maximum white rectangle still to be discovered, otherwise one of its obstacles is chosen as a *pivot* (the original proposal suggests to choose the central one with respect to the current bound) and the contour is consequently split into four regions (those resting above, below, to the right and to the left of the pivot, i.e., $r_A$, $r_B$, $r_L$, $r_R$, respectively), some of which partially overlap, as shown in Fig. 5.13. For each such region, the quality function $Q$ is evaluated, and the obstacles that fall in it are identified, this way creating a new structure to be inserted in the priority queue. The procedure has a linear complexity in the number of obstacles in the bound, that must be scanned to obtain the obstacle sets $O_A$, $O_B$, $O_L$ and $O_R$. This technique ensures that white areas are never split into smaller parts, and hence the first empty bound that comes out of the queue is actually the largest.

*Example 5.6* (Definition of a sub-area in the Background Analysis algorithm) Consider the case represented in the top-left frame of Fig. 5.13, and assume the pivot is block R represented in solid black. Applying the required split to such a pivot, the upper region includes obstacles R1 and R2, the right-hand-side region only R2, the lower one only R3, and the left-hand-side one R1 and R3.

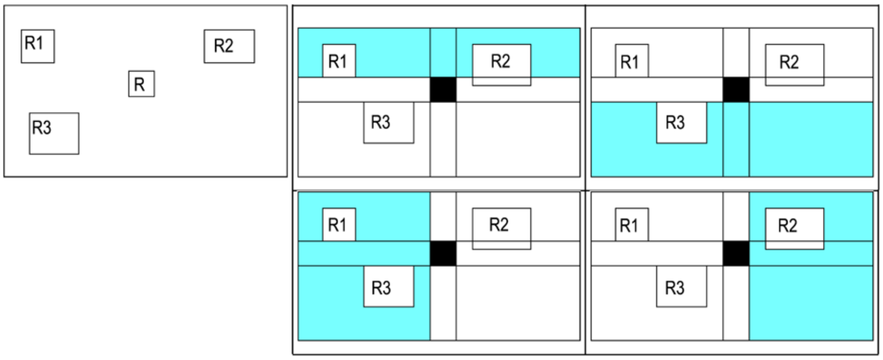


**Fig. 5.13** Bound splitting according to a pivot in the maximal white rectangle algorithm

In the original algorithm, after a maximal background rectangle $m$ is found, the authors suggest saving such a white space into a supporting structure

- Background: Set of Rectangle (output),

adding it as a new obstacle to the set of obstacles $\overline{O}$ and restarting the whole procedure from the beginning until a satisfactory background is retrieved. However, in this way all previously performed splits would get lost, and the new computation would not take advantage of them. To improve performance, [19] proposes just adding $m$ as an additional obstacle to all bounds in the current queue to which it overlaps, and continuing the loop until the stop criterion is met. This yields the following algorithm:

```
1. Background $\leftarrow \emptyset$
2. Enqueue $(\overline{r}, \overline{O})$ in $Q$ based on $q(\overline{r})$
3. **while** $Q \neq \emptyset$ **do**
   (a) $(r, O) \leftarrow$ dequeue$(Q)$
   (b) **if** $O = \emptyset$ **then**
       (i) Background $\leftarrow$ Background $\cup \{r\}$
       (ii) Add $r$ as a new obstacle to all elements in $Q$ overlapping it
   (c) **else**
       (i) $p \leftarrow$ pivot$(O)$
       (ii) Enqueue $(r_A, O_A)$ in $Q$ based on $q(r_A)$
            where $r_A = (r.x_L, r.y_T, r.x_R, p.y_T)$; $O_A = \{b \in O | b \cap r_A \neq \emptyset\}$
       (iii) Enqueue $(r_B, O_B)$ in $Q$ based on $q(r_B)$
            where $r_B = (r.x_L, p.y_B, r.x_R, r.y_B)$; $O_B = \{b \in O | b \cap r_B \neq \emptyset\}$
       (iv) Enqueue $(r_L, O_L)$ in $Q$ based on $q(r_L)$
            where $r_L = (r.x_L, r.y_T, p.x_L, r.y_B)$; $O_L = \{b \in O | b \cap r_L \neq \emptyset\}$
       (v) Enqueue $(r_R, O_R)$ in $Q$ based on $q(r_R)$
            where $r_R = (p.x_R, r.y_T, r.x_R, r.y_B)$; $O_R = \{b \in O | b \cap r_R \neq \emptyset\}$
4. **return** Background
```

The authors of [19] also suggest that choosing as a pivot a 'side' block (i.e., the top or bottom or leftmost or rightmost block), or even better a 'corner' one (one which is

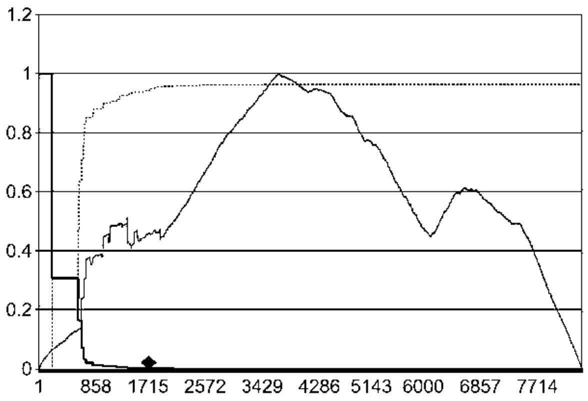


**Fig. 5.14** Typical curves of evolution in the background analysis algorithm, useful for defining a stop criterion (reprint from [19] ©2008 Springer)

at the same time both top or bottom and leftmost or rightmost), results in a quicker retrieval of the background areas with respect to choosing it at random or in the middle of the bound. Indeed, in this way at least one white area can be immediately separated from the margin of the current bound.

The algorithm works top-down since it progressively splits page fragments with the aim of identifying the background. If the white rectangles are stored in the *Background* structure as they are retrieved, when the queue becomes empty their union represents the page background, i.e., the 'negative' of the layout structure. Once the background has been found, the geometrical structure of the page can be obtained by opposition. However, it is clear that taking the whole procedure to its natural end (retrieval of all the background) would just return the original blocks. If somehow larger aggregates (*frames*) are to be found, the procedure must be stopped before retrieving useless background pieces, such as inter-word or inter-line ones. A fixed minimum threshold on the size of the white rectangles to be retrieved can be set (as soon as a smaller white rectangle is retrieved, the whole procedure can be stopped), but it is still too rigid for dealing with several different kinds of documents.

A further contribution of [19] consisted in an empirical study of the algorithm behavior to help in defining such a stop criterion depending on the specific document. The values of three variables were traced in each step of the algorithm until its natural conclusion, as shown in Fig. 5.14 for a sample document:

1. Size of the queue (black line), normalized between 0 and 1;
2. Ratio of the last white area retrieved over the total white area of the current page under processing (bold line);
3. Ratio of the white area retrieved so far over the total white area of the current page under processing (dashed line).

Parameter 3 is never equal to 1 (due to the minimum size threshold), but becomes stable before reaching 1/4 of the total steps of the algorithm. This generally holds for all documents except those having a scattered appearance (the *Spread Factor* introduced in Sect. 5.3 can be a useful indicator for this). Correspondingly, the trend of parameter 2 decreases up to 0 in such a point, marked in the figure with a black diamond. This suggests that the best moment to stop executing the algorithm is just

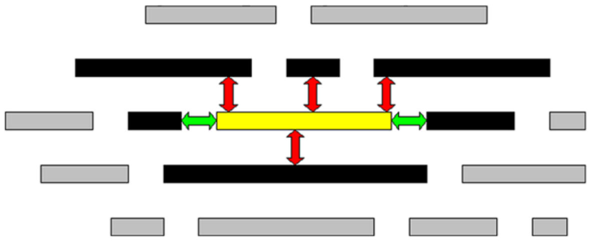


**Fig. 5.15** Adjacency between blocks for RLSO on born-digital documents. The *central block* is the reference, and *black blocks* are the adjacent blocks considered by the algorithm. *Gray blocks* are not considered adjacent

there, since before the layout is not sufficiently detailed, while after useless white spaces are found (e.g., those inside columns and sections), as shown by the black line in the graphic. Specifically, parameter 2 is more easily implemented, due to the fixed limit 0 (whereas the limit of parameter 3 depends on the specific document). It is worth noting that this value is reached very early, and before the size of the structure that contains the blocks waiting to be processed starts growing dramatically, thus saving lots of time and space resources.

**RLSO on Born-Digital Documents** The principles underlying RLSO can be transposed to the case of born-digital documents [23], where basic blocks (often characters, fragments of words, images or geometrical elements such as rectangles and lines) play the role of black runs in digitized documents, and the distance between adjacent components that of white runs. However, while each black pixel has at most one white run and one next black pixel in each horizontal/vertical direction, a block may have many adjacent blocks on the same side, according to their projections. Thus, the notions of *adjacency* and *distance* between blocks deserve further attention. The latter is computed as in CLiDE. As to the former, the principle is that, if the projection of a block in one direction meets several blocks whose projections mutually overlap, only the closest of them is taken as adjacent, while the farther ones are discarded. This is shown in Fig. 5.15, where the black blocks are considered adjacent to the block in the middle, and the arrows denote their distance. Note that the two top blocks are not considered adjacent to the reference block, although they could be joined to it by a straight arrow that would not overlap any other block.

Basic blocks are progressively merged into higher-level components as follows:

1. For each basic block, build a corresponding frame containing only that block;
2. $H \leftarrow$ list of all possible triples $(d_h, b_1, b_2)$ where $b_1$ and $b_2$ are horizontally adjacent basic blocks of the document and $d_h$ is their horizontal distance;
3. Sort $H$ by increasing distance;
4. While $H$ is not empty and the first element of the list, $(d', b'_1, b'_2)$, has distance below a given horizontal threshold $t_h$ $(d' < t_h)$
   (a) Merge in a single frame the frames to which $b'_1$ and $b'_2$ belong
   (b) Remove the first element from $H$;
5. $V \leftarrow$ list of all possible triples $(d_v, b_1, b_2)$ where $b_1$ and $b_2$ are vertically adjacent basic blocks of the document and $d_v$ is their vertical distance;
6. Sort $V$ by increasing distance;

7. While $V$ is not empty and the first element of the list, $(d'', b''_1, b''_2)$, has distance below a given vertical threshold $t_v$ $(d'' < t_v)$
   (a) Merge in a single frame the frames to which $b''_1$ and $b''_2$ belong
   (b) Remove the first element from $V$.

Steps 2–4 correspond to horizontal smoothing in RLSO, and steps 5–7 to vertical smoothing. Each frame obtained can be handled separately.

Efficiency of the procedure can be improved by preliminarily organizing the basic blocks into a structure where they are stored top-down in 'rows' and left-to-right inside 'rows', so that considering in turn each of them top-down and left-to-right is sufficient to identify all pairs of adjacent blocks while limiting the number of useless comparisons. Given a block, its horizontal adjacent is the nearest subsequent block on the same row whose initial coordinate is greater than the final coordinate of the block being considered, while its vertical adjacents are those in the nearest subsequent row whose vertical projections overlap to it (as soon as the first non-overlapping block having initial coordinate greater than the final coordinate of the block being considered is encountered, the rest of the row can be ignored).

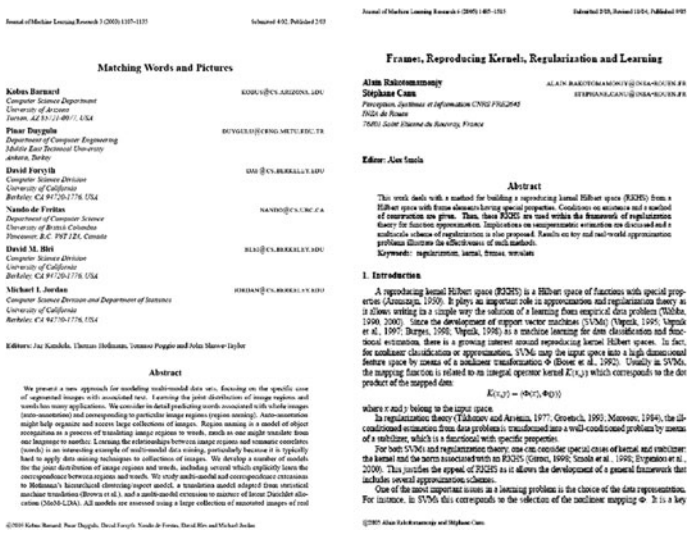
Although similar to that proposed in CLiDE, this method does not build any graph, and improves efficiency by considering adjacent components only. The "nearest first" grouping is mid-way between the minimum spanning tree technique of CLiDE and a single-link clustering technique [9], where the number of clusters is not fixed in advance, but automatically derives from the number of components whose distance falls below the given thresholds. Compared to the ideas in DOCSTRUM, here the distance between component borders, rather than centers, is exploited. Indeed, the latter option would not work in this case since the basic blocks can range from single characters to (fragments of) words, and this lack in regularity would affect the proper nearest neighbor identification. For the same reason, there is no fixed number of neighbors to be considered, but all neighbors closer than the thresholds are taken into account.

## 5.4 Document Image Understanding

*Document Image Understanding* (or *interpretation*) is "the formal representation of the abstract relationships indicated by the two-dimensional arrangement of the symbols" [33] in a document. The identification of the logical structure of a document can be cast as the association of a logical component to (some of) the geometrical components that make up the document layout. The outcome of such a process is fundamental for distinguishing which components are most significant and hence deserve further processing. For instance, the title, authors, abstract and keywords in a scientific paper could be directly exploited to file the document and fill the corresponding record fields. First of all, the whole document can be associated to a document class, which is such a fundamental and common task to deserve being considered a separate step by itself (known as *document image classification*). Indeed, a single collection usually includes several types of documents, having different geometrical structures that are to be handled differently from each other. In practice, the type of a document is often indicated by the geometrical structure of its first page. Humans are able to immediately distinguish, e.g., a bill from an advertisement, or a letter from a (newspaper or scientific) article, without actually reading their content, but based exclusively on their visual appearance.

After assessing a document class, the document image understanding activity can be completed by locating and labeling the logical components which appear in the various pages, that often correspond to frames identified during the layout analysis step. If different levels of logical components are present, it can be useful to proceed top-down, by looking for the larger aggregates first and then for the simpler structures therein. Assuming that it is possible to identify logical components based on the visual appearance only, just as humans do, the objective can be attained by comparing the geometrical structure of each page against models of logical components, that represent the invariant features of such components in the layout of documents belonging to the same class. This task must necessarily be carried out after the classification step because the kind of logical components that one can expect to find in a document strongly depends on the document class. For instance, in a commercial letter one might expect to find a sender, possibly a logo, an address, a subject, a body, a date and a signature; many of these components make no sense in a scientific paper, where, in contrast, one might look for the title, authors and their affiliations, abstract, possibly keywords and bibliographic references.

Form-based documents are characterized by neatly defined areas of the page in which the various information items are to be placed, which significantly reduces the effort needed to identify and separate them. Conversely, many other kinds of documents (such as scientific papers) show an extreme variability in the geometrical structure of the pages and in the arrangement of the components therein (e.g., the title and headings may take just one or several lines, the number of authors and their affiliations can range from one to many items, and the paragraph length can vary significantly). Figure 5.16 presents an outstanding example of this variability: although belonging to the same class of documents (scientific papers taken from the same series), the front page of the paper on the left contains a part of the first section, while that of the document on the right is not even able to include the whole abstract. Because of this, while geometrical models that search for a certain kind of information in a well defined area of the page can be of use in the case of forms, rigid algorithms based on fixed rules are of little help to identify the logical structure in the latter kind of documents, where the purely spatial approach to recognition of components shows its limits. A deep analysis of human behavior easily reveals that in those cases people support spatial considerations by a more complex kind of knowledge concerning the relationships among components. Indeed, what helps them in defining the proper role of a given component, and the type of document as well, is the way it is related to the other components around the page and to the page itself, more pregnantly than its intrinsic properties and attributes. For instance, it is more useful to know, say, that the keywords are a left-aligned small text block placed between the authors and the abstract, than trying to capture general regularities in its size and/or placement in the page. For this reason, a number of systems based on Artificial Intelligence techniques have been developed to carry out this task with suitable flexibility.



Journal of Machine Learning Research 3 (2003) 1107–1135 Submitted 4/02; Published 2/03

Matching Words and Pictures

Kobus Barnard KOBUS@CS.ARIZONA.EDU
*Computer Science Department*
*University of Arizona*
*Tucson, AZ 85721-0077, USA*

Pinar Duygulu DUYGULU@CENG.METU.EDU.TR
*Department of Computer Engineering*
*Middle East Technical University*
*Ankara, Turkey*

David Forsyth DAF@CS.BERKELEY.EDU
*Computer Science Division*
*University of California*
*Berkeley, CA 94720-1776, USA*

Nando de Freitas NANDO@CS.UBC.CA
*Department of Computer Science*
*University of British Columbia*
*Vancouver, B.C. V6T 1Z4, Canada*

David M. Blei BLEI@CS.BERKELEY.EDU
*Computer Science Division*
*University of California*
*Berkeley, CA 94720-1776, USA*

Michael I. Jordan JORDAN@CS.BERKELEY.EDU
*Computer Science Division and Department of Statistics*
*University of California*
*Berkeley, CA 94720-1776, USA*

Editors: Jaz Kandola, Thomas Hofmann, Tomaso Poggio and John Shawe-Taylor

Abstract

We present a new approach for modeling multi-modal data sets, focusing on the specific case of segmented images with associated text. Learning the joint distribution of image regions and words has many applications. We consider in detail predicting words associated with whole images (auto-annotation) and corresponding to particular image regions (region naming). Auto-annotation might help organize and access large collections of images. Region naming is a model of object recognition as a process of translating image regions to words, much as one might translate from one language to another. Learning the relationships between image regions and semantic correlates (words) is an interesting example of multi-modal data mining, particularly because it is typically hard to apply data mining techniques to collections of images. We develop a number of models for the joint distribution of image regions and words, including several which explicitly learn the correspondence between regions and words. We study multi-modal and correspondence extensions to Hofmann's hierarchical clustering/aspect model, a translation model adapted from statistical machine translation (Brown et al.), and a multi-modal extension to mixture of latent Dirichlet allocation (MoM-LDA). All models are assessed using a large collection of annotated images of real

©2003 Kobus Barnard, Pinar Duygulu, David Forsyth, Nando de Freitas, David Blei and Michael Jordan.

Journal of Machine Learning Research 6 (2005) 1485–1515 Submitted 3/03; Revised 11/04; Published 9/05

Frames, Reproducing Kernels, Regularization and Learning

Alain Rakotomamonjy ALAIN.RAKOTOMAMONJY@INSA-ROUEN.FR
Stéphane Canu STEPHANE.CANU@INSA-ROUEN.FR
*Perception, Systèmes et Information CNRS FRE2645*
*INSA de Rouen*
*76801 Saint Etienne du Rouvray, France*

Editor: Alex Smola

Abstract

This work deals with a method for building a reproducing kernel Hilbert space (RKHS) from a Hilbert space with frame elements having special properties. Conditions on existence and a method of construction are given. Then, these RKHS are used within the framework of regularization theory for function approximation. Implications on semiparametric estimation are discussed and a multiscale scheme of regularization is also proposed. Results on toy and real-world approximation problems illustrate the effectiveness of such methods.
Keywords: regularization, kernel, frames, wavelets

1. Introduction

A reproducing kernel Hilbert space (RKHS) is a Hilbert space of functions with special properties (Aronszajn, 1950). It plays an important role in approximation and regularization theory as it allows writing in a simple way the solution of a learning from empirical data problem (Wahba, 1990, 2000). Since the development of support vector machines (SVMs) (Vapnik, 1995; Vapnik et al., 1997; Burges, 1998; Vapnik, 1998) as a machine learning for data classification and functional estimation, there is a growing interest around reproducing kernel Hilbert spaces. In fact, for nonlinear classification or approximation, SVMs map the input space into a high dimensional feature space by means of a nonlinear transformation $\Phi$ (Boser et al., 1992). Usually in SVMs, the mapping function is related to an integral operator kernel $K(x,y)$ which corresponds to the dot product of the mapped data:

$$K(x,y) = \langle \Phi(x), \Phi(y) \rangle$$

where $x$ and $y$ belong to the input space.

In regularization theory (Tikhonov and Arsénin, 1977; Groetsch, 1993; Morozov, 1984), the ill-conditioned estimation from data problem is transformed into a well-conditioned problem by means of a stabilizer, which is a functional with specific properties.

For both SVMs and regularization theory, one can consider special cases of kernel and stabilizer: the kernel and the norm associated with an RKHS (Girosi, 1998; Smola et al., 1998; Evgeniou et al., 2000). This justifies the appeal of RKHS as it allows the development of a general framework that includes several approximation schemes.

One of the most important issues in a learning problem is the choice of the data representation. For instance, in SVMs this corresponds to the selection of the nonlinear mapping $\Phi$. It is a key

©2005 Alain Rakotomamonjy and Stéphane Canu.

**Fig. 5.16** Two very different layouts from the same class of documents (reprint from [19] ©2008 Springer)

## 5.4.1 Relational Approach

Looking back at the aforementioned definition of Document Image Understanding, one immediately notes the explicit reference to a 'formal representation of abstract relationships'. A typical framework that allows expressing and handling relations in Computer Science refers to formal logic, and specifically to the *First-Order Logic* (*FOL* for short) setting, where predicate expressions, rather than arrays of numeric values, are the main representation tool. This allows describing observations of any complexity, once a proper language has been defined, because the number and structure of description components is not fixed in advance as in attribute-value formalisms, but may vary according to the needs of each specific observation. In addition to such an improved expressive power, FOL provides several other advantages. It is human-understandable and human-writable because its formulæ exploit abstract symbols directly referred to human concepts. It is able to exploit background knowledge (if available) and to carry out different inference strategies, such as induction, deduction, abduction and abstraction, that allow tackling several aspects of complexity in the domain (although at the cost of heavier computational requirements).

In FOL, unary predicates are typically exploited for representing boolean object properties (e.g., 'red(X)'), while $n$-ary predicates can express values of non-boolean object properties (e.g., 'length(X, Y)') and/or relationships among objects (e.g., 'above(X, Y)'). Works that successfully adopted such an approach to the problem of Document Image Understanding have focused on descriptors that allow expressing the kind of document components (block, frame, page), their properties

**Table 5.2** First-Order Logic descriptors for a document layout structure

| Descriptor | Interpretation |
|---|---|
| page($C$) | component $C$ is a page |
| block($C$) | component $C$ is a block |
| frame($C$) | component $C$ is a frame |
| type_$T$($C$) | the content type of component $C$ is $T \in$ {text, image, hor_line, ver_line, graphic, mixed} |
| page_no($P$, $N$) | $P$ is the $N$th page in the document |
| width($C$, $W$) | $W$ is the width of component $C$ |
| height($C$, $H$) | $H$ is the height of component $C$ |
| hor_pos($C$, $X$) | $X$ is the horizontal position of the centroid of component $C$ |
| ver_pos($C$, $Y$) | $Y$ is the vertical position of the centroid of component $C$ |
| part_of($C'$, $C''$) | component $C'$ contains component $C''$ |
| on_top($C'$, $C''$) | component $C'$ is above component $C''$ |
| to_right($C'$, $C''$) | component $C''$ is to the right of component $C'$ |
| $A$_aligned($C'$, $C''$) | $C'$ and $C''$ have an alignment of type $A \in$ {left, right, center, top, bottom, middle} |

(position, size, type) and spatial relationships (mutual direction and alignment). Table 5.2 reports a possible selection of predicates to describe the layout structure of a generic document, that includes nine boolean properties (type_$T$ is indeed a representative of six predicates, expressing different kinds of content), five non-boolean (in this case, numeric) properties, and nine relationships (six of which collapsed in the $A$_aligned predicate). Numeric properties can be expressed either as absolute values in some unit of measure (e.g., pixels or dots), or in relative values with respect to the page (e.g., percentage of the page size).

A FOL classifier consists of a logical theory made up of definitions in the form of logical implications, whose premise defines the requirements for an object to be assigned to the class specified in the consequence. It may include at the same time several class definitions, and can exploit also definitions of other sub-concepts useful to describe those classes, plus possible inter-relationships between them, that can be combined at classification time by automatic reasoners in charge of carrying out the classification. Given a set of facts expressing the layout of a document by means of the same descriptors, the reasoner applies the rules in the theory to produce a decision.

*Example 5.7* (Sample FOL definitions for Document Image Understanding) A definition for class *Springer-Verlag Lecture Notes* paper is:

$$\begin{array}{l}
\text{part\_of}(D, P_1) \wedge \text{page}(P_1) \wedge \text{page\_no}(P_1, 1) \wedge \\
\text{part\_of}(P_1, F_1) \wedge \text{frame}(F_1) \wedge \\
\text{hor\_pos}(F_1, X_1) \wedge X_1 \in [0.333, 0.666] \wedge \text{ver\_pos}(F_1, Y_1) \wedge Y_1 \in ]0.666, 1] \wedge \\
\text{width}(F_1, W_1) \wedge W_1 \in ]0.625, 1] \wedge \text{height}(F_1, H_1) \wedge H_1 \notin ]0.006, 0.017] \wedge \\
\text{part\_of}(P, F_2) \wedge \text{frame}(F_2) \wedge \\
\text{hor\_pos}(F_2, X_2) \wedge X_2 \in [0.333, 0.666] \wedge \text{ver\_pos}(F_2, Y_2) \wedge Y_2 \in [0, 0.333] \wedge \\
\text{to\_right}(F_1, F_2) \wedge \text{on\_top}(F_2, F_1) \\
\quad \Rightarrow \text{sv\_ln}(D)
\end{array}$$

To be read as:

"A document $D$ belongs to class 'Springer-Verlag Lecture Notes paper' if its first page $P_1$ contains two components $F_1$ and $F_2$, both of which are frames, where: both have horizontal position placed in the middle third of the page; $F_1$ has vertical position in the lower third of the page and $F_2$ has vertical position in the upper third of the page; $F_1$ has width within 62.5% and 100% of the page width (i.e., very large) and height not comprised between 0.6% and 1.7% of the page width (i.e., not very very small); $F_2$ is to right of $F_1$ and $F_2$ is on top of $F_1$".

A definition for the *title* logical component in the previous class is:

$$\begin{array}{l}
\text{part\_of}(D, P_1) \wedge \text{page}(P_1) \wedge \text{page\_no}(P_1, 1) \wedge \\
\text{part\_of}(P_1, C) \wedge \text{frame}(C) \wedge \text{type\_text}(C) \wedge \\
\text{hor\_pos}(C, X) \wedge X \in [0.333, 0.666] \wedge \text{ver\_pos}(C, Y) \wedge Y \in [0, 0.333] \wedge \\
\text{part\_of}(P_1, F_2) \wedge \text{frame}(F_2) \wedge \text{type\_text}(F_2) \wedge \text{ver\_pos}(F_2, Y_2) \wedge Y_2 \in [0.333, 0.666] \wedge \\
\text{part\_of}(P_1, F_3) \wedge \text{frame}(F_3) \wedge \text{height}(F_3, H_3) \wedge H_3 \in ]0.057, 0.103] \wedge \\
\text{hor\_pos}(F_3, X_3) \wedge X_3 \in [0.333, 0.666] \wedge \text{ver\_pos}(F_3, Y_3) \wedge Y_3 \in [0, 0.333] \wedge \\
\text{on\_top}(F_3, F_2) \wedge \text{on\_top}(C, F_2) \wedge \text{on\_top}(C, F_3) \\
\quad \Rightarrow \text{title}(C)
\end{array}$$

"A component $C$ is a 'title' if it is of type text, placed in the top-center region of the first page $P_1$ of a document $D$, on top of two other frames $F_2$ and $F_3$, such that $F_3$ is also on top of $F_2$, where $F_2$ is also of type text and placed vertically in the middle of the page, while $F_3$ is also placed in the top-center region of the page and has height between 5.7% and 10.3% of the page height".

It is easy to recognize $F_3$ as representing the authors frame, and $F_2$ the abstract heading.

Although FOL techniques allow a domain expert to write by hand suitable classifiers for documents and their components, actually writing such things is not easy, due to the subtleties that often characterize the application domain to be described and to the inability of non-trained people to formalize their knowledge. For this reason, automatic learning of such classifiers has been attempted, with positive results. The branch of Machine Learning that deals with FOL descriptions is known as *Inductive Logic Programming* (*ILP* for short) [31, 37].

**INTHELEX (INcremental THEory Learner from EXamples)** *INTHELEX* [20] is an ILP system that learns hierarchical logic theories from positive and negative examples. It is fully incremental (it can both refine an existing theory and learn one from scratch), and able to learn simultaneously multiple concepts/classes, possibly related to each other. It can remember all the processed examples, so to guarantee validity of the learned theories on all of them, and is able to exploit additional inference strategies (deduction, abduction, abstraction) with respect to pure induction.

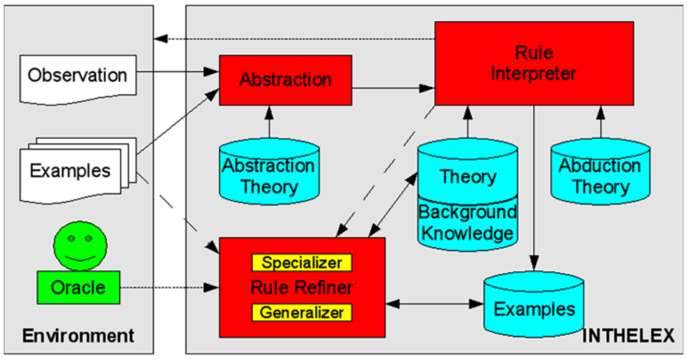


**Fig. 5.17** Architecture and data flow in INTHELEX

Its architecture is depicted in Fig. 5.17. It is a general-purpose system, but has been successfully applied to several tasks related to document processing, and in particular to document image understanding. Indeed, the foregoing features allow it to tackle various kinds of domain complexity, many of which are present in the document case: incrementality, in particular, is a key option when the set of documents to be handled is continuously updated and extended. The definitions shown in Example 5.7 were actually learned by INTHELEX from real-world examples of different types of scientific papers, although the representation formalism used by the system is slightly different (theories are expressed as Prolog programs that can be readily exploited by any Prolog interpreter as classifiers for the learned concepts).

An initial theory can be provided by an expert, by another system, or even be empty. Such a starting theory can be refined whenever a new set of examples (observations whose class is known) of the concepts to be learned, possibly selected by an expert, becomes available from the environment, so that it is able to explain them (dashed arrows in the schema). Otherwise, when new observations of unknown class are available, the set of inductive hypotheses in the theory is applied to produce corresponding decisions (class assignments). If an oracle points out an incorrectness in such a decision, the cause is identified and the proper kind of correction chosen, starting the theory revision process according to a data-driven strategy (dashed arrows in the schema).

As for *induction*, two refinement operators are provided to revise the theory:

**Generalization** for definitions that reject positive examples. The system first tries to generalize one of the available definitions of the concept the example refers to, so that the resulting revised theory covers the new example and is consistent with all the past negative examples. If such a generalization is found, then it replaces the chosen definition in the theory, or else a new definition is chosen to compute generalization. If no definition can be generalized in a consistent way, the system checks whether the example itself can represent a new alternative (consistent) definition of the concept. If so, such a definition is added to the theory.

**Specialization** for definitions that explain negative examples. The theory definitions that concur in the wrong explanation are identified, and an attempt is made to specialize one of them by adding to it one or more conditions which characterize all the past positive examples and can discriminate them from the current negative one. In case of failure, the system tries to add the negation of a condition, that is able to discriminate the negative example from all the past positive ones.

If all refinement attempts fail, the example is added to the theory as a (positive or negative, as appropriate) exception. New incoming observations are always checked against the exceptions before applying the rules that define the concept they refer to.

Multistrategy operators are integrated in INTHELEX according to the *Inferential Theory of Learning* framework [30]:

**Deduction** allows recognizing known concepts that are implicit in the descriptions in order to explicitly add and use them. Such concepts are those specified in the learned theory or in a *Background Knowledge* provided by some expert. Differently from the learned theory, the background knowledge is assumed to be correct and hence cannot be modified.

**Abduction** aims at completing the description of partial observations by hypothesizing facts that, together with the given theory and background knowledge, could explain them (as defined by Peirce). According to the framework proposed in [28], the concepts about which assumptions can be made are specified as a set of *abducibles*, while a set of *integrity constraints* (each corresponding to a complex condition that must be fulfilled) biases what can be abduced.

**Abstraction** is a pervasive activity in human perception and reasoning, aimed at removing superfluous details by shifting the description language to a higher level one. According to the framework proposed in [56], it is implemented by various operators, each in charge of a particular kind of abstraction: replacing several components by a compound object, decreasing the granularity of a set of values, ignoring whole objects or just part of their features, neglecting the number of occurrences of some kind of object.

Some examples of how multistrategy can help in the document image analysis domain are the following: deduction can identify in a purely geometrical description known logical components, or derived spatial relationships starting from just basic ones; abduction can hypothesize the presence of layout components that are missing in the description because of an error of the author or due to bad digitization; abstraction can eliminate insignificant components such as small specks, or denote frequent combinations of components by a single predicate, or simplify numeric attributes translating them into discrete intervals.

### 5.4.2 Description

The class of a document, and the role played by its components, represent precious knowledge to be exploited in later document management. Information *about* a document, as opposed to that expressed by the document content, is referred to as *metadata* (etymologically, "data about data"). Since metadata define a context and provide semantics for the objects they describe, they are a fundamental means for allowing a shared interpretation of a resource by different users. Although their importance is traditionally acknowledged in the context of archives and library catalogs, they are extremely relevant for automatic document processing as well, which is why they are often explicitly embedded into the documents, or attached to them. Indeed, *semantic interoperability* between different applications requires to establish a shared network of data whose meaning and value is universally agreed upon. Of course, the involved applications must agree on how to represent and interpret the metadata, for which reason the availability of standardized conventions is a critical requirement.

**DCMI (Dublin Core Metadata Initiative)** The *DCMI*, or simply *Dublin Core*, draws its name from the American town where it was first conceived in 1995, and from its purpose: providing authors and publishers of information objects with (1) a core of fundamental elements to describe any digital material they produce, and (2) a set of shared tools for accessing those resources through a computer network. The project, developed under the auspices of the American On-line Computer Library Center (OCLC), involved librarians, archivists, publishers, researchers, software developers, and members of the IETF (Internet Engineering Task Force) working groups. Supported by the NISO (National Information Standards Organization), it was acknowledged as ISO 15836 standard in 2003, and recently revised in 2009 [3].

The original Core defines 15 basic description *Elements* to be included in or referenced by the digital objects they describe. Being very general, such a set is undergoing an extension, by means of so-called *Qualifiers*. Two kinds of qualifiers exist: *Element Refinements* express more specific interpretations of Elements; *Encoding Schemes* specify shared conventions for expressing the Element values (e.g., controlled vocabularies or formal identification systems). Not to spoil the general schema, Refinements should be ignored by applications that do not recognize them. Below is a list of the basic Elements along with the current recommendations for Refinements (preceded by →) and Schemes, where applicable:

**Title** → Alternative
*A name given to the resource* (typically, the one by which it is formally known).

**Creator** *An entity primarily responsible for making the resource.*

**Subject** *The topic of the resource* (typically keywords, key phrases, or classification codes expressing the matter of the resource). Schemes: LCSH, MeSH, DDC, LCC, UDC.

**Description** → Table of Contents, Abstract
*An account of the resource* (may include, among others, a graphical representation or a free-text account of the resource).

**Publisher** *An entity responsible for making the resource available.*

**Contributor** *An entity responsible for making contributions to the resource.*

**Date** → Created, Valid, Available, Issued, Modified, Date Accepted, Date Copyrighted, Date Submitted
*A point or period of time associated with an event in the life cycle of the resource.*

May be used to express temporal information at any level of granularity. Schemes: DCMI Period, W3C-DTF.[12]

**Type** *The nature or genre of the resource.* Scheme: DCMI Type Vocabulary.
Includes terms that describe general categories, functions, genres or level of aggregation by content.

**Format** $\rightarrow$ Extent, Medium
*The file format, physical medium, or dimensions of the resource.*
Examples of dimensions include size and duration. It can be used to specify the software or hardware needed to display or process the resource. Scheme: IMT[13] (not used for Refinements).

**Identifier** $\rightarrow$ Bibliographic Citation
*An unambiguous reference to the resource within a given context.* Scheme: URI (not used for Refinements).

**Source** *A related resource from which the described resource is derived* in whole or in part. Scheme: URI.

**Language** *A language of the resource.* Schemes: ISO 639-2, RFC 3066 (superseded by RFC 4646).[14]

**Relation** $\rightarrow$ Is Version Of, Has Version, Is Replaced By, Replaces, Is Required By, Requires, Is Part Of, Has Part, Is Referenced By, References, Is Format Of, Has Format, Conforms To
*A related resource.* Scheme: URI.

**Coverage** $\rightarrow$ Spatial, Temporal
*The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.*
Spatial information may be a named place or a location specified by its geographic coordinates (schemes: DCMI Point, ISO 3166, DCMI Box, TGN[15]). Temporal topic may be a named period, date, or date range (schemes: DCMI Period, W3C-DTF). A jurisdiction may be a named administrative entity or a geographic place to which the resource applies. Where appropriate, named places or time periods are preferred to numeric identifiers such as sets of coordinates or date ranges.

**Rights** $\rightarrow$ Access Rights, Licence (scheme: URI)
*Information about rights held in and over the resource* or a reference to the service that provides such an information.
Typically includes a statement about various property rights associated with the resource, including Intellectual Property Rights (IPR) (e.g., the copyright). If missing, no assumption can be made about the rights of the resource.

The 'entities' referred to in the definitions of Creator, Publisher and Contributor can be, e.g., persons, organizations, or services, and are usually indicated by their names.

[12] A simplified profile of the ISO 8601 standard, that combines in different patterns groups of digits *YYYY*, *MM*, *DD*, *HH*, *MM*, *SS* expressing year, month, day, hour, minute and second, respectively.

[13] Internet Media Type, formerly MIME types.

[14] The typical two-character language codes, optionally followed by a two-character country code (e.g., `it` for Italian, `en-uk` for English used in the United Kingdom).

[15] Thesaurus of Geographic Names.

For Identifier, Source and Relation, the use of a Digital Object Identifier (*DOI*), or an International Standard Book Number (*ISBN*) can be a suitable alternative to a URI or URL.

Qualifiers have the following properties:

*Name* The unique token assigned to it;
*Label* The human-readable label assigned to it;
*Definition* A statement that represents its concept and essential nature;
*Comment* Additional information associated with it (if available);
*See Also* A link to more information about it (if available).

Originally conceived for the description of Web resources, the Core has been subsequently adopted by several communities, among which museums, commercial agencies and organizations all over the world, which made it a *de facto* standard, in addition to a *de jure* one. Indeed, its strengths include the fostering of an integrated approach to information, easy comprehension and exploitation of its elements, and straightforward application to different languages by simply translating the elements' names (which has been done already for more than 20 languages). Its generality permits the description of a wide variety of resources in different formats, which is a key issue for supporting semantic interoperability. As a drawback, it can turn out to be too vague for suitably describing specific application domains; in such cases the schema can be personalized and extended by integrating and evolving the data structure with different and more appropriate meanings. This provides a lot of flexibility to the indexer when recording the features of a resource, allowing him to create detailed specifications when needed, at the expenses of interoperability (although exploiting the same schema, different personalizations would not share its interpretation).

## References

1. Document Object Model (DOM) Level 1 Specification—version 1.0. Tech. rep. REC-DOM-Level-1-19981001, W3C (1998)
2. Document Object Model (DOM) Level 2 Core Specification. Tech. rep. 1.0, W3C (2000)
3. Dublin Core metadata element set version 1.1. Tech. rep. 15836, International Standards Organization (2009)
4. Altamura, O., Esposito, F., Malerba, D.: Transforming paper documents into XML format with WISDOM++. International Journal on Document Analysis and Recognition **4**, 2–17 (2001)
5. Baird, H.S.: The skew angle of printed documents. In: Proceedings of the Conference of the Society of Photographic Scientists and Engineers, pp. 14–21 (1987)
6. Baird, H.S.: Background structure in document images. In: Advances in Structural and Syntactic Pattern Recognition, pp. 17–34. World Scientific, Singapore (1992)
7. Baird, H.S.: Document image defect models. In: Baird, H.S., Bunke, H., Yamamoto, K. (eds.) Structured Document Image Analysis, pp. 546–556. Springer, Berlin (1992)
8. Baird, H.S., Jones, S., Fortune, S.: Image segmentation by shape-directed covers. In: Proceedings of the 10th International Conference on Pattern Recognition (ICPR), pp. 820–825 (1990)
9. Berkhin, P.: Survey of clustering Data Mining techniques. Tech. rep., Accrue Software, San Jose, CA (2002)

10. Breuel, T.M.: Two geometric algorithms for layout analysis. In: Proceedings of the 5th International Workshop on Document Analysis Systems (DAS). Lecture Notes in Computer Science, vol. 2423, pp. 188–199. Springer, Berlin (2002)
11. Cao, H., Prasad, R., Natarajan, P., MacRostie, E.: Robust page segmentation based on smearing and error correction unifying top-down and bottom-up approaches. In: Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 392–396. IEEE Computer Society, Los Alamitos (2007)
12. Cesarini, F., Marinai, S., Soda, G., Gori, M.: Structured document segmentation and representation by the Modified X–Y tree. In: Proceedings of the 5th International Conference on Document Analysis and Recognition (ICDAR), pp. 563–566. IEEE Computer Society, Los Alamitos (1999)
13. Chaudhuri, B.: Digital Document Processing—Major Directions and Recent Advances. Springer, Berlin (2007)
14. Chen, Q.: Evaluation of OCR algorithms for images with different spatial resolution and noise. Ph.D. thesis, University of Ottawa, Canada (2003)
15. Ciardiello, G., Scafuro, G., Degrandi, M., Spada, M., Roccotelli, M.: An experimental system for office document handling and text recognition. In: Proceedings of the 9th International Conference on Pattern Recognition (ICPR), pp. 739–743 (1988)
16. Egenhofer, M.J.: Reasoning about binary topological relations. In: Gunther, O., Schek, H.J. (eds.) 2nd Symposium on Large Spatial Databases. Lecture Notes in Computer Science, vol. 525, pp. 143–160. Springer, Berlin (1991)
17. Egenhofer, M.J., Herring, J.R.: A mathematical framework for the definition of topological relationships. In: Proceedings of the 4th International Symposium on Spatial Data Handling, pp. 803–813 (1990)
18. Egenhofer, M.J., Sharma, J., Mark, D.M.: A critical comparison of the 4-intersection and 9-intersection models for spatial relations: Formal analysis. In: Proceedings of the 11th International Symposium on Computer-Assisted Cartography (Auto-Carto) (1993)
19. Esposito, F., Ferilli, S., Basile, T.M.A., Di Mauro, N.: Machine Learning for digital document processing: from layout analysis to metadata extraction. In: Marinai, S., Fujisawa, H. (eds.) Machine learning in Document Analysis and Recognition. Studies in Computational Intelligence, vol. 90, pp. 105–138. Springer, Berlin (2008)
20. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T.M., Di Mauro, N.: Incremental multistrategy learning for document processing. Applied Artificial Intelligence: An International Journal **17**(8/9), 859–883 (2003)
21. Fateman, R.J., Tokuyasu, T.: A suite of lisp programs for document image analysis and structuring. Tech. rep., Computer Science Division, EECS Department—University of California at Berkeley (1994)
22. Ferilli, S., Basile, T.M.A., Esposito, F.: A histogram-based technique for automatic threshold assessment in a Run Length Smoothing-based algorithm. In: Proceedings of the 9th International Workshop on Document Analysis Systems (DAS). ACM International Conference Proceedings, pp. 349–356 (2010)
23. Ferilli, S., Biba, M., Esposito, F., Basile, T.M.A.: A distance-based technique for non-Manhattan layout analysis. In: Proceedings of the 10th International Conference on Document Analysis Recognition (ICDAR), pp. 231–235 (2009)
24. Frank, A.U.: Qualitative spatial reasoning: Cardinal directions as an example. International Journal of Geographical Information Systems **10**(3), 269–290 (1996)
25. Gatos, B., Pratikakis, I., Ntirogiannis, K.: Segmentation based recovery of arbitrarily warped document images. In: Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), pp. 989–993 (2007)
26. Impedovo, S., Ottaviano, L., Occhinegro, S.: Optical character recognition—a survey. International Journal on Pattern Recognition and Artificial Intelligence **5**(1–2), 1–24 (1991)
27. Kainz, W., Egenhofer, M.J., Greasley, I.: Modeling spatial relations and operations with partially ordered sets. International Journal of Geographical Information Systems **7**(3), 215–229 (1993)

28. Kakas, A.C., Mancarella, P.: On the relation of truth maintenance and abduction. In: Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence (PRICAI), pp. 438–443 (1990)
29. Kise, K., Sato, A., Iwata, M.: Segmentation of page images using the area Voronoi diagram. Computer Vision Image Understanding **70**(3), 370–382 (1998)
30. Michalski, R.S.: Inferential theory of learning. Developing foundations for multistrategy learning. In: Michalski, R., Tecuci, G. (eds.) Machine Learning. A Multistrategy Approach, vol. IV, pp. 3–61. Morgan Kaufmann, San Mateo (1994)
31. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
32. Mori, S., Suen, C.Y., Yamamoto, K.: Historical review of OCR research and development. Proceedings of the IEEE **80**(7), 1029–1058 (1992)
33. Nagy, G.: Twenty years of document image analysis in PAMI. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(1), 38–62 (2000)
34. Nagy, G., Kanai, J., Krishnamoorthy, M.: Two complementary techniques for digitized document analysis. In: ACM Conference on Document Processing Systems (1988)
35. Nagy, G., Seth, S., Viswanathan, M.: A prototype document image analysis system for technical journals. Computer **25**(7), 10–22 (1992)
36. Nagy, G., Seth, S.C.: Hierarchical representation of optically scanned documents. In: Proceedings of the 7th International Conference on Pattern Recognition (ICPR), pp. 347–349. IEEE Computer Society Press, Los Alamitos (1984)
37. Nienhuys-Cheng, S.H., de Wolf, R. (eds.): Foundations of Inductive Logic Programming. Lecture Notes in Computer Science, vol. 1228. Springer, Berlin (1997)
38. O'Gorman, L.: The document spectrum for page layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(11), 1162–1173 (1993)
39. O'Gorman, L., Kasturi, R.: Document Image Analysis. IEEE Computer Society, Los Alamitos (1995)
40. Papadias, D., Theodoridis, Y.: Spatial relations, minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science **11**(2), 111–138 (1997)
41. Papamarkos, N., Tzortzakis, J., Gatos, B.: Determination of run-length smoothing values for document segmentation. In: Proceedings of the International Conference on Electronic Circuits and Systems (ICECS), vol. 2, pp. 684–687 (1996)
42. Pavlidis, T., Zhou, J.: Page segmentation by white streams. In: Proceedings of the 1st International Conference on Document Analysis and Recognition (ICDAR), pp. 945–953 (1991)
43. Rice, S.V., Jenkins, F.R., Nartker, T.A.: The fourth annual test of OCR accuracy. Tech. rep. 95-03, Information Science Research Institute, University of Nevada, Las Vegas (1995)
44. Salembier, P., Marques, F.: Region-based representations of image and video: Segmentation tools for multimedia services. IEEE Transactions on Circuits and Systems for Video Technology **9**(8), 1147–1169 (1999)
45. Shafait, F., Smith, R.: Table detection in heterogeneous documents. In: Proceedings of the 9th IAPR International Workshop on Document Analysis Systems (DAS). ACM International Conference Proceedings, pp. 65–72 (2010)
46. Shih, F., Chen, S.S.: Adaptive document block segmentation and classification. IEEE Transactions on Systems, Man, and Cybernetics—Part B **26**(5), 797–802 (1996)
47. Simon, A., Pret, J.C., Johnson, A.P.: A fast algorithm for bottom-up document layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence **19**(3), 273–277 (1997)
48. Skiena, S.S.: The Algorithm Design Manual, 2nd edn. Springer, Berlin (2008)
49. Smith, R.: A simple and efficient skew detection algorithm via text row accumulation. In: Proceedings of the 3rd International Conference on Document Analysis and Recognition (ICDAR), pp. 1145–1148, IEEE Computer Society, Los Alamitos (1995)
50. Smith, R.: An overview of the Tesseract OCR engine. In: Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), pp. 629–633. IEEE Computer Society, Los Alamitos (2007)

51. Smith, R.: Hybrid page layout analysis via tab-stop detection. In: Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR), pp. 241–245. IEEE Computer Society, Los Alamitos (2009)
52. Sun, H.M.: Page segmentation for Manhattan and non-Manhattan layout documents via selective CRLA. In: Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR), pp. 116–120. IEEE Computer Society, Los Alamitos (2005)
53. Wahl, F., Wong, K., Casey, R.: Block segmentation and text extraction in mixed text/image documents. Graphical Models and Image Processing **20**, 375–390 (1982)
54. Wang, D., Srihari, S.N.: Classification of newspaper image blocks using texture analysis. Computer Vision, Graphics, and Image Processing **47**, 327–352 (1989)
55. Wong, K.Y., Casey, R., Wahl, F.M.: Document analysis system. IBM Journal of Research and Development **26**, 647–656 (1982)
56. Zucker, J.D.: Semantic abstraction for concept representation and learning. In: Proceedings of the 4th International Workshop on Multistrategy Learning (MSL), pp. 157–164 (1998)

# Part III
# Content Processing

Once the significant logical components in a document have been found, they must be processed according to the type of their content in order to extract the information they carry. In some sense, each component undergoes a sequence of steps that (although specific for each component type) is analogous to that formerly applied to the whole document from an organizational viewpoint. In particular, the main form of knowledge communication among humans is *natural language*, that represents a challenge for automatic processing because of its inherent ambiguity and strict connection to semantics. *Text*, as the written representation of language, is the key towards accessing information in many documents, which in turn would enforce global knowledge sharing.

Text processing represents a preliminary phase to many document content handling tasks aimed at extracting and organizing information therein. The computer science disciplines devoted to understanding language, and hence useful for such objectives, are *Computational Linguistics* and *Natural Language Processing*. They rely on the availability of suitable linguistic resources (corpora, computational lexica, etc.) and of standard representation models of linguistic information to build tools that are able to analyze sentences at various levels of complexity: morphologic, lexical, syntactic, semantic. Chapter 6 provides a survey of the main Natural Language Processing tasks (tokenization, language recognition, stemming, stopword removal, Part of Speech tagging, Word Sense Disambiguation, Parsing) and presents some related techniques, along with lexical resources of interest to the research community.

The initial motivation and, at the same time, the final objective for creating documents is information preservation and transmission. In this perspective, the availability of huge quantities of documents, far from being beneficial, carries the risk of scattering and hiding information in loads of noise. As a consequence, the development of effective and efficient automatic techniques for the identification of interesting documents in a collection, and of relevant information items in a document, becomes a fundamental factor for the practical exploitation of digital document repositories. Chapter 7 is concerned with the management of document contents. Several Information Retrieval approaches are presented first, ranging from term-based indexing to concept-based organization strategies, for supporting document search. Then, tasks more related to the information conveyed by documents are presented: Text Categorization (aimed at identifying the subject of interest of a document), Keyword Extraction (to single out prominent terms in the document text) and Information Extraction (that produces a structured representation of the features of noteworthy events described in a text).

# Chapter 6
# Natural Language Processing

The main and most ancient form of knowledge communication among human beings is *natural language*. Despite being included in the list of earliest objectives of Artificial Intelligence [21], computer applications nowadays are still far from able to handle it as humans do. Some reasons why it so hard a matter to work on automatically are to be found in its high and inherent ambiguity (due to historical evolution), and in its strict connection to semantics, whereas computer systems work at the syntactic level and require formal representations. Since *text* is the written representation of language, and hence its permanent counterpart, it is clear that the ability to properly process text is the key towards accessing information in many documents, which in turn would enforce global knowledge sharing. The term 'text' usually denotes a single information unit that can be a complete logical unit (e.g., an article, a book) or just a part thereof (e.g., a chapter, a paragraph). From the perspective of computer science representation, on the other hand, each physical unit (e.g., a file, an e-mail message, a Web page) can be considered as a document. In this chapter, however, 'document' and 'text' are to be intended as synonyms, and as consisting of sequences of characters.

One research field in charge of exploring this matter from a theoretical perspective is *computational linguistics* whose task is to "give computer systems the ability of generate and interpret natural language" [12] by developing a computational theory of language, using "tools [...] of artificial intelligence: algorithms, data structures, formal models for representing knowledge, models of reasoning processes, and so on" [1]. Being placed half-way between humanistic and computer sciences, it must find a tradeoff among very different conceptions of the language and of its processing. Here the language understanding perspective is of interest, involving the extraction of information implicitly contained in texts and its transformation into explicit knowledge resources, structured and accessible both by human users and by other computational agents.

The operational support to such an objective comes from the field of *Natural Language Processing* (*NLP*) [20] based on three kinds of elements [8]:

Representation **models** and **standards** that ensure interoperability among different language processing systems and consistency in its processing;

**Resources** consisting of large databases that represent and encode morphologic, lexical, syntactic and semantic information;

**Tools** for text analysis at various levels (morphologic, lexical, syntactic, semantic) and, based on the outcome of such an analysis, for subsequent knowledge extraction, acquisition and structuring.

Tools heavily rely on the availability of suitable resources to carry out their tasks, and can cooperate with each other to perform a global attack to the problem by referring to common models and exploiting shared standards. Several industrial initiatives in this field witness that some techniques are now mature, although the convergence of a number of factors have caused that, against a large amount of work made for English, much less exists for other languages.

## 6.1 Resources—Lexical Taxonomies

The availability of linguistic resources is crucial to support significant advances in NLP research and to ensure applicability of the results to real world. The main, and probably the most important need is for large *corpora* (a *corpus* is a collection of documents, usually—but not necessarily—consistent and domain-related). Indeed, they serve both as a support for automatically building/learning NLP tools, and as testbeds on which the different tools can be applied and evaluated. Recently, the Web as an enormous *corpus* of readily available documents has partly solved this problem, but has also imposed the need to face real texts, much more varied and complex than the controlled laboratory ones.

Another important kind of resources are dictionaries and *thesauri*,[1] to provide different kinds of lexical information (which is the base for most techniques aimed at extracting and organizing information in textual documents). A *word* in a language is a *form* (a string on a finite alphabet of symbols) endowed with a *sense* (an element belonging to a given set of meanings) in that language. The *vocabulary* of a language is the set of such pairs (*form*, *sense*), while a *dictionary* is a list that contains such pairs ordered alphabetically.

Since words in a text are used to explicitly encode concepts, the need for associating them to machine-processable definitions of concepts and relations has become more and more urgent. *Ontologies*,[2] the state-of-the-art Computer Science solution to formal representation of knowledge in a given domain, might be too complex to be of practical use. *Computational lexica* that arrange words and their underlying concepts in suitable taxonomies are a surrogate of ontologies (some say they *are* a form of ontology) representing a tradeoff between expressiveness and efficiency.

[1] A *thesaurus* is a dictionary based on semantic models of organization, that reports for each word, in addition to its meaning, also a list of related words (e.g., its synonyms and antonyms).

[2] A "formal, explicit specification of a shared conceptualization" [13], i.e., a vocabulary to model the type of objects, concepts, and their properties and relations, in a domain. In philosophy, a systematic account of Existence.

**WordNet** *WordNet*[3] was defined by its author, the famous psychologist G.A. Miller, as "an on-line lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, and adjectives are organized into synonym sets, each representing one underlying concept. Different relations link the synonym sets" [22]. Each set of synonyms is called a *synset*. Hence, WordNet can be a precious support to several activities involved in the automatic processing of text. Its organization model of terms is different than that traditionally used to compile dictionaries: instead of having a list of words ordered by morphologic and syntactic features, forms and senses are the vertices of graphs whose edges represent (i.e., are labeled with) several kinds of semantic and lexical relations. Overall, it can be seen as a semantic network that reproduces the mental representation of the lexicon and determines the meaning of single terms.

Relations allow inferring indirect relationships among terms and concepts that are not directly linked to each other. The most important are the following:

**Synonymy** and **antinomy** are both symmetric. Two words are considered as *synonyms* not just because they have a meaning in common: it is required that replacing each other in a sentence does not change the sentence meaning. Thus, synonymy is bound to link terms belonging to the same syntactic category (e.g., replacing a name with a verb, even when syntactically correct, would change the sentence semantics). Since two terms can be synonyms in some contexts (e.g., 'board' and 'plank' in carpentry) but not in others, synonymy is a real-valued (rather than boolean) property, continuously ranging in [0, 1]. *Antinomy* links pairs of opposite terms and is exploited to organize attributes and adverbs.

**Hyperonymy** and **hyponymy** (inverses of each other) are both transitive. They represent, respectively, generalization and specialization, i.e., the two directions of an *is_a* relationship that links a more general synset (the *hypernym*) to a subclass thereof (the *hyponym*). They induce a heterarchy on the set of synsets. Many metrics for the evaluation of similarity among synsets are based on the (length of the) *is_a* paths that connect them (e.g., 'man'–'male'–'person'–'organism'–'living thing'–'object'–'entity').

**Meronymy** and **holonymy** (inverses of each other) are both transitive and antisymmetric. They link synsets based on the concept of part–whole composition (e.g., 'finger'–'hand'), as in the *part_of* relationship, but also of member (e.g., 'student'–'class') and substance (e.g., 'oxygen'–'water') composition.

Additionally, morphologic relationships between terms having different declension, gender and number, but that can be associated to the same concept, allow grouping terms in normal forms having the same root.

The syntactic category of a word expresses the set of linguistic contexts in which it can be exploited (e.g., the term 'run' can be a noun or a verb, depending on where/how it is used). Specifically, four categories are used to express concepts, and are taken into account by WordNet:

[3]Website http://wordnet.princeton.edu.

**Nouns** (N) both common nouns and widely used proper (geographic or person) names are included, organized in a heterarchy, that represents a semantic network inside of which inferential activities can be carried out. Starting from the leaves that correspond to purely concrete terms the abstraction level of the terms progressively increases as one approaches the root ('entity') whose immediate offspring consists of 25 primitive terms that denote abstract concepts.

**Verbs** (V) are important because they express semantic relationships between the concepts underlying their subjects and objects. They are very flexible, but also very prone to polysemy, because they can take on different meanings according to the terms to which they are associated (e.g., 'to get' + preposition). Like nouns, they are organized in a heterarchy rooted in 15 conceptual categories, each expressing a different set of semantically similar actions. In addition to specialization and synonymy, verbs also participate in the logical *entailment* relationship that specifies the existence of semantic connections such as temporal sequence (to carry out an action, another one must have been previously carried out), strict inclusion (an action can be carried out only if another is being carried out), cause–effect relation (if an action is carried out, as its effect another one must take place as well); e.g., 'to move' is a logical consequence of 'to fall' because if something is falling it must also be moving (but not *vice versa*).

**Adjectives** (Aj) (including also nouns and phrases frequently used as modifiers) are divided into two classes. *Ascriptive* ones, expressing intrinsic properties of objects (e.g., 'good', 'rich'), are organized in a network of *dipoles*, i.e., antinomous pairs, whose poles are connected to other adjectives or dipoles by synonymy (some might have an intermediate or graduated meaning between the two poles of a dipole). Semantically similar attributes tend to cluster around a central item (a pole in a dipole). Ascriptive adjectives that do not have antonyms can assume as indirect antonyms those of similar adjectives. *Nonascriptive* adjectives (or *pertainyms*), derived from nouns and expressing external properties of objects (e.g., 'polar', 'lateral'), are connected to the corresponding nouns. With antinomy being inapplicable, they are organized like nouns, according to composition and generalization. Color adjectives are a special case. Highly polysemous adjectives whose meaning depends on the noun to which they are associated are handled by doubling the poles and associating to them different meanings.

**Adverbs** (Av) (also called *open-class words*) are organized in dipoles as well.

Thus, WordNet is more than a digital mix of a dictionary and a thesaurus, oriented to classical term-based search methods. It is a data structure that supports conceptual information search. In its database (whose structure is reported in Table 6.1) each concept has an identifier (to reduce redundancy compared to classical thesauri), each lemma is associated to the identifiers of the meanings that it can take on (this way defining synsets), and the relationships connect concepts and/or terms to each other. The success of WordNet is such that several initiatives have produced versions of it for many other languages and for specific domains as well.

**WordNet Domains** *WordNet Domains* is an extension of WordNet that classifies each synset on the grounds of the semantic area(s) (or *domains*) to which it be-

**Table 6.1** WordNet database tables in Prolog format. A pair $w = (\#, i)$ denotes a word sense, i.e., a term with its associated synset

**s**$(\#, i, w, ss\_type, sense\_number, tag\_count)$ $w$ is the $i$th word defining synset #, also reporting information on the grammatical type and frequency.

**sk**$(\#, i, s)$ $s$ is the sense key for word $(\#, i)$ (present for every word sense).

**g**$(\#, d)$ $d$ is a textual description/definition (*gloss*) of synset #.

**syntax**$(\#, i, m)$ $m$ is the syntactic marker for word $(\#, i)$ if one is specified.

**hyp**$(\#_1, \#_2)$ (only for nouns and verbs) $\#_2$ is a hypernym of $\#_1$ (and $\#_1$ is a hyponym of $\#_2$).

**ins**$(\#_1, \#_2)$ noun $\#_1$ is an instance of noun $\#_2$ (and $\#_2$ has_instance $\#_1$).

**ent**$(\#_1, \#_2)$ verb $\#_2$ is an entailment of verb $\#_1$.

**sim**$(\#_1, \#_2)$ (only for adjectives contained in adjective clusters) adjective $\#_2$ has similar meaning to (and hence is a satellite of) adjective $\#_1$ (which is the cluster head).

**mm**$(\#_1, \#_2)$ noun $\#_2$ is a member meronym of noun $\#_1$ (and $\#_1$ is a member holonym of $\#_2$).

**ms**$(\#_1, \#_2)$ noun $\#_2$ is a substance meronym of noun $\#_1$ (and $\#_1$ is a substance holonym of $\#_2$).

**mp**$(\#_1, \#_2)$ noun $\#_2$ is a part meronym of noun $\#_1$ (and $\#_1$ is a part holonym of $\#_2$).

**der**$(\#_1, \#_2)$ there exists a reflexive lexical morphosemantic relation between terms $\#_1$ and $\#_2$ representing derivational morphology.

**cls**$(\#_1, i_1, \#_2, i_2, class\_type)$ $\#_1$ is a member of the class represented by $\#_2$. If either $i$ is 0, the pointer is semantic.

**cs**$(\#_1, \#_2)$ verb $\#_2$ is a cause of verb $\#_1$.

**vgp**$(\#_1, i_1, \#_2, i_2)$ verb synsets that are similar in meaning and should be grouped together.

**at**$(\#_1, \#_2)$ the adjective is a value of the noun (attribute relation). For each pair, both relations are listed (i.e., each # is both a source and target).

**ant**$(\#_1, i_1, \#_2, i_2)$ the two words are antonyms. For each pair, both relations are listed (i.e., each $(\#, i)$ pair is both a source and target word).

**sa**$(\#_1, i_1, \#_2, i_2)$ (only for verbs and adjectives) Additional information about $(\#_1, i_1)$ can be obtained by seeing $(\#_2, i_2)$ (but not necessarily *vice versa*).

**ppl**$(\#_1, i_1, \#_2, i_2)$ adjective $(\#_1, i_1)$ is a participle of verb $(\#_2, i_2)$, and *vice versa*.

**per**$(\#_1, i_1, \#_2, i_2)$ adjective $(\#_1, i_1)$ pertains to the noun or adjective $(\#_2, i_2)$, or adverb $(\#_1, i_1)$ is derived from adjective $(\#_2, i_2)$.

**fr**$(\#, f\_num, w\_num)$ specifies a generic sentence frame for one or all words in verb #.

longs [19]. At the time of writing, it is in its 3.2 version, based on WordNet 2.0.[4] It is organized as a hierarchical structure, called *WordNet Domains Hierarchy* (WDH), in which 168 semantic categories are grouped into 45 basic domains that, in turn, refer to 5 top-level classes that lastly depend on a *Factotum* class that includes all terms that cannot be classified in other categories. Table 6.2 reports the first levels of the hierarchy. Each basic category is also considered as a child of itself, which ensures that the union of terms belonging to child nodes corresponds to the set of terms

[4]Since backward compatibility is not guaranteed in WordNet versioning, and hence the synset IDs may change, WordNet Domains might not be aligned with the latest WordNet version. This can be a problem when both are to be used, since either one gives up in using the latest WordNet version, or one has to install two different WordNet versions on the same system.

**Table 6.2** Top-level classes and corresponding basic domains in WordNet domains

| Top-level class | Basic domains |
|---|---|
| Humanities | History, Linguistics, Literature, Philosophy, Psychology, Art, Paranormal, Religion |
| Free Time | Radio–TV, Play, Sport |
| Applied Science | Agriculture, Food, Home, Architecture, Computer Science, Engineering, Telecommunication, Medicine |
| Pure Science | Astronomy, Biology, Animals, Plants, Environment, Chemistry, Earth, Mathematics, Physics |
| Social Science | Anthropology, Health, Military, Pedagogy, Publishing, Sociology, Artisanship, Commerce, Industry, Transport, Economy, Administration, Law, Politics, Tourism, Fashion, Sexuality |

belonging to the parent node. Thus, for instance, Telecommunication has children Post, Telegraphy, Telephony (in addition to Telecommunication itself).

Clearly, a term that is associated to different synsets might have different domains for each synset. For instance, the word ‘mouse’ has two meanings (i.e., belongs to two synsets) in WordNet: The rodent (synset 02244530, associated to domain Animals) and the pointing device (synset 03651364, associated to domain Computer Science). However, even a single synset can be associated to several domains (e.g., synset 00028764, ‘communication’, is associated to domains Linguistics and Telecommunication).

*Example 6.1* (Complete WordNet Domains sub-tree for **Applied_ Science**)

- Agriculture
  - Animal_Husbandry
    - Veterinary
- Food
  - Gastronomy
- Home
- Architecture
  - Town_Planning
  - Buildings
  - Furniture
- Computer_Science
- Engineering
  - Mechanics
  - Astronautics
  - Electrotechnology
  - Hydraulics
- Telecommunication
  - Post
  - Telegraphy
  - Telephony
- Medicine
  - Dentistry
  - Pharmacy
  - Psychiatry
  - Radiology
  - Surgery

To go beyond categories that are just syntactic labels, they were mapped onto the *Dewey Decimal Classification* (*DDC*) system [10], a taxonomy originally developed for the classification of items in a library environment, but then adopted for cataloging Internet resources as well. The characteristic of DDC is that each subject in the hierarchy is bound to have at most ten sub-categories (whence the attribute ‘decimal’). If children are denoted by digits $0 \dots 9$, each path in the taxonomy, uniquely identifying a subject starting from the root, can be expressed in the form of a decimal number, called a *subject code*. The core DDC consists of the first three levels of specialization (whose top-level subjects are reported in Table 6.3), and hence can be expressed by just three digits, where the hundreds identify ten

**Table 6.3** Dewey decimal classification system main categories

| | |
|---|---|
| 000 | Computer science, Information, General works |
| 100 | Philosophy, Psychology |
| 200 | Religion |
| 300 | Social sciences |
| 400 | Language |
| 500 | Science |
| 600 | Technology |
| 700 | Arts, Recreation |
| 800 | Literature |
| 900 | History, Geography, Biography |

broad areas of interest in human knowledge, each of which is divided in sub-areas denoted by the tens that, in turn, can be divided into sub-sub-categories indicated by the units. This yields 1000 possible subjects in the core, not all of which are currently exploited. In case more specialized factorization is required, any level of sub-categories can be defined by appending more digits (i.e., adding levels to the taxonomy). To distinguish the core from the extended part, subject codes are represented as decimal numbers *xxx.xxxx*... where the integer part refers to the core and is mandatory, while the decimal part (after the decimal point) is optional. This allows accommodating any future expansion and evolution of the body of human knowledge, and represents one of the success factors for DDC with respect to other systems that allow any number of sub-categories, but must rely on more complex codes. Each subject code is associated to a lexical description that, along with its position in the hierarchy, fully determines its semantics. Specific publications are available to guide librarians in assigning to any given text the proper subject code.

There is no one-to-one correspondence between a DDC category and a WDH domain because the latter was defined to fulfill the following requirements:

**Disjunction** the interpretation of all WDH labels should not overlap;
**Basic coverage** all human knowledge should be covered by the Basic Domains;
**Basic balancing** most Basic Domains should have a comparable degree of granularity.

However, each WDH domain is associated to precise DDC categories, in order to provide it with an explicit semantics and an unambiguous identification [2].

**Senso Comune** A recent project, aimed at providing a collaborative platform to build and maintain an open "machine-readable dictionary that provides semantic information in a formal way" of Italian language, is Senso Comune [23]. It joins, but neatly distinguishes, the lexical taxonomy and linguistic ontology aspects. The linguistic source is a cutting-edge Italian dictionary specifically oriented towards term usage [9]. The association between the senses and the reference ontology is based on the intuition that objects are usually represented by nouns, qualities by adjectives and events by verbs (although this heavily depends on the language at hand).

While WordNet starts from an analysis of the language lexicon, Senso Comune adopts an opposite approach, starting from the following ontological structure:

- **Entity** maximum generality concept;
  - **Concrete** entities spatially and temporally determined;
    - **Object** spatially determined concrete entities with autonomous existence, having no temporal parts but properties that may change in time;
    - **Event** concrete entities spreading in time, associated with actors and having temporal parts;
  - **Abstract** entities not determined spatially and temporally;
    - *Meaning* the association of a word form to a corresponding meaning in an ontology (and, in some cases, to contexts in which it occurs);
    - *MeaningDescription* a set of glosses, phraseology and notes describing a meaning, provided by users (*UserMeaningDescription*) or by a dictionary (*DictionaryMeaningDescription*);
    - *UsageInstance* phraseology that concurs to make up meaning descriptions, provided by users (*UserUsageInstance*) or dictionary (*DictionaryUsageInstance*);
    - *MeaningRelation* an association between pairs of meanings: *Synonymy*, *Troponymy* (different modes in which an action may occur), *Hyponymy*, *Antinomy*, *Meronymy*;
  - **Quality** features, attributes of entities (whose existence depends on the entities, although they are not parts thereof),

where sublists denote sub-classes, boldface items are the general part, and italics ones refer to the lexical part.

Concepts in the ontology are expressed in a syntactic restriction of the *DL-Lite* description logics[5] [7], considered a good tradeoff between effectiveness (it allows expressing inclusion dependencies, existential quantification on roles, negation) and efficiency (comparable to that of relational DBMSs due to limitations imposed on the use of universal quantification, disjunction, and enumeration).

## 6.2 Tools

Natural language is based on several kinds of linguistic structures of increasing complexity, layered on top of each other to make up sentences. Dealing with written text, at the bottom there is the morphologic level,[6] then comes the lexical one, followed by the syntactic one and, lastly, by the semantic one. Since each level is characterized by its own kinds of possible ambiguities, a *divide and conquer* approach suggests tackling them separately by different specialized tools, each exploiting the

[5]*Description Logics* are fragments of First-Order Logic, having a formal semantics, and typically exploited to define concepts (i.e., classes) and roles (i.e., binary relations) in ontologies.

[6]Considering spoken text, a further preliminary level is the phonetic one.

results of the lower level(s) and providing support to the higher one(s), as shown in next subsections. Each phase extracts some kinds of components or structure, and represents them in a format that is suitable to the application of other processing tools and techniques. They can work according to statistical–mathematical approaches or to symbolic ones. The latter are closer to the way in which humans process linguistic information, but are sometimes too rigid compared to the typical difficulty of fixing used language into strict rules. The former are more robust in this respect, but can be applied only provided that large quantities of linguistic resources are available.

This stratified perspective allows carrying out only the steps that are sufficient for a given objective, saving resources whenever further elements are not needed. For instance, considering the syntactic-logical structure of sentences would require handling an extremely large number of details on, and of complex inter-relationships among, the elements that make up the text. In many applications (e.g., search by key terms), by restricting to the most significant terms only, and by considering them in isolation, a good trade-off can be reached between effectiveness and computational requirements, this way allowing the algorithms to work on a less complex problem. A *Bag of Words* (*BoW*) is a classical compact representation of a text, consisting of the set of terms appearing in it (or of a suitable subset thereof). Despite being a significant simplification, BoWs are not free from the tricks of natural language, in this case due to the well-known ambiguity of some terms outside of a specific context, such as *synonymy* (different words having the same meaning—e.g., 'committee' and 'board') and *polysemy* (words having different meanings—e.g., 'board' as a plank or as a committee).

### 6.2.1 Tokenization

The first step to be carried out is splitting the text into basic elements, known as *tokenization*. Based only on structural considerations with respect to the symbols of the alphabet in which it is written, any text can be preliminarily split into elementary components, called *tokens*, each of which can be associated to a category that is significant for the aims of subsequent processing (on different kinds of tokens further specific processing steps can be applied). Typical categories are words (sequences made up of letters only), numbers and other types of values, punctuation marks and various kinds of symbols. The level of detail by which this step is carried out depends both on the kind of processing that is to be subsequently applied to the text and on the amount of computational resources that one is willing to spend: sometimes it is useful to store particular features of the tokens (e.g., the presence of capital letters, or the adjacency to other kinds of tokens such as punctuation); in other cases a simpler approach can be adopted, that consists in just ignoring punctuation and numbers, and in taking into account only contiguous strings of alphabetical characters (many search engines on the Internet work in this way).

A formalism for expressing patterns that denote some kind of token is that of *regular expressions*. They specify how to build valid sequences of symbols taken from an alphabet $\mathscr{A}$ according to the following definition:

- Any symbol of $\mathscr{A}$ is a regular expression; then,
- Given two regular expressions $R$ and $S$, the following constructions are regular expressions as well:

  $RS$ or $R \cdot S$ denoting concatenation of (sequences of) symbols (all the symbols in $R$ followed by all symbols in $S$);

  $(R|S)$ alternative (sequences of) symbols (either $R$ or $S$ is to be included in the sequence);

  $[R]$ optionality (possibility of not including $R$ in the sequence);

  $\{R\}_n^m$ iteration (concatenation of $R$ with itself $k$ times, $n \le k \le m$).

In iteration, a $*$ superscript indicates any number of repetitions, while subscript 0 can be omitted. Moreover, to express composition of several alternative regular expressions $R_1, \dots, R_l$, the simpler notation $(R_1|R_2|\dots|R_l)$ is exploited instead of $((\dots(R_1|R_2)|\dots)|R_l)$.

*Example 6.2* Regular expression defining real numbers in decimal notation:

$$[(+|-)]\{(0|1|2|3|4|5|6|7|8|9)\}_1^*.\{(0|1|2|3|4|5|6|7|8|9)\}_1^*,$$

where $\mathscr{A} = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$.

Further constraints can be set on the outcome of the pattern (e.g., if the pattern represents a date, the number of days must be consistent with the month, and in the case of February with the year as well, to handle leap years[7]).

### 6.2.2 Language Recognition

Tokens that can be classified as 'words' follow the morphologic rules of the language used, and hence they must be analyzed in detail from a linguistic viewpoint. Because the inflection rules are different among languages, the complexity of morphologic analysis can significantly vary according to the language (e.g., the variety of inflection suffixes available in Italian makes handling this language much more difficult than English). Thus, if the application is intended to work in multi-lingual environments, and the specific language in which each single text is written is not known in advance, an intermediate step of *language recognition* is required, that indicates the proper linguistic tools to be exploited in the next steps.

[7]To improve approximation between the conventional year and the actual movement of the Earth, a year is leap, i.e., includes February 29th, if it can be divided by 4, unless it can also be divided by 100 (in which case it is not), unless it can be divided by 400 as well (in which case it is).

A first way for recognizing the language in which a text is written is counting the number of typical words that appear in the text from each language, and taking the language for which such a number is maximum. These typical words are usually identified as those most frequently exploited in the language, such as articles, prepositions, pronouns, and some adverbs. Attention must be paid to words that are typical for several languages, and hence are not discriminant between them (e.g., 'a' which is an article in English, a preposition in Italian and Spanish, etc.). This approach is straightforward (because these lists of words are generally available as lexical resources, being useful also for successive processing phases such as stopword removal—see Sect. 6.2.3) and safer (since actual words of each language are identified), but might turn out to be inapplicable or might be misled in cases of short texts written in telegraphic style, where just those peculiar words are usually omitted.

Another, more sophisticated approach, is based on the analysis of the frequency of typical sequences of letters. A sequence of $n$ consecutive letters in a text is called an *$n$-gram*[8] ($n = 1$ yields *monograms*, $n = 2$ *digrams*, $n = 3$ *trigrams*, etc.). Theoretical studies and practical experiments reported in [24] have shown that the frequency distribution of $n$-grams is typical for each language, so that it can be exploited for language recognition. Obviously, the larger the $n$, the finer the approximation of the language: it turned out that, using trigrams, a very close approximation to the sound of a language is obtained.[9] Thus, a text can be assigned to the language whose $n$-gram distribution best matches the $n$-gram frequency found in the text.

### 6.2.3 Stopword Removal

A first way to reduce the (size of the) BoW for a given text, while limiting the consequences on effectiveness, is the so-called *stopword removal* step, consisting in

[8]This term is often used also to denote a sequence of $n$ consecutive *words* in a text, but here the letter-based interpretation is assumed.

[9]The cited experiment adopted a generative perspective on this subject. First, casual sequences of equiprobable letters and spaces were generated, showing that no particular language was suggested by the outcome. Then a first-order approximation of English was produced by generating sequences where the probability of adding a character to the sequence was the same as the frequency computed for that character on actual English texts. Again, this was of little help for hypothesizing from the sequence the language from whose frequency distribution it was generated. Switching to a second order approximation, where each character was extracted according to a different distribution depending on the previous extracted character, started showing some hints of English here and there. Indeed, it was able to represent phenomena such as the 'qu' pair, where the 'u' must necessarily follow the 'q', that the single-letter approximation could not express. Finally, a third-order approximation, in which the probability distribution for extracting the next symbol depended on the previous two characters, showed a significant improvement in the outcome, where English was quite evident in the overall flavor of the sound, and several monosyllable words were actually caught. Comparable results were obtained for other languages as well, e.g., Latin.

the elimination of the common-use terms appearing with high frequency in a language, or of terms that are in any case not considered fundamental to distinguish the text from others, or to identify its subject, or to understand its content. Most such terms correspond to *function words*, i.e., words in a language that have little meaning by themselves (e.g., articles, adverbs, conjunctions, pronouns and prepositions, auxiliary verbs), as opposed to *content words* to which a specific independent meaning can be assigned (e.g., nouns, verbs and adjectives). Thus, the latter are to be preserved, being considered as potentially significant (often independently of their spread in the collection and of the number of occurrences).

Stopwords can be determined explicitly, by listing the terms to be removed, or implicitly, by specifying which grammatical functions are considered meaningless. The latter option is more compact because all words having the specified functions will be automatically removed, and more general because it is language-independent, but requires the system to know the category of each word in the text. The former, conversely, requires each single word to be evaluated for inclusion in, or exclusion from, the list of terms to be removed, which implies more effort and is prone to errors, but ensures more flexibility and allows extending or modifying the stopword list at need. Moreover, if *a-priori* knowledge on the documents is available, supplementary domain-dependent stopword lists could be provided. Indeed, if all the texts considered concern the same area of knowledge, even specific terms that in general would be characteristic and/or discriminant might become quite meaningless. Of course, a mixed approach is possible, that removes all words belonging to some grammatical categories or appearing in given lists.

### 6.2.4 Stemming

*Morphologic normalization* of terms is another way for further reducing the BoW size for a text. It is aimed at replacing each term with a corresponding standardized form that is independent on variations due to *inflection* (conjugation of verbs or declension of nouns and adjectives), and that will be used in subsequent processing steps as a representative of the terms that derive from it (which can be thought of as introducing some degree of generalization in the text representation). In addition to the dimensional reduction in text representation obtained by including in the BoW just one representative for several variations of each term, some consider this normalization process as introducing a rough kind of semantics in the description as a desirable side-effect. The underlying rationale is that words have been defined to denote concepts, and that they carry such concepts independently of the inflection needed to properly include them in meaningful sentences. Thus, ignoring morphologic variations of the same words in some sense corresponds to going back to the underlying concepts.

There are two approaches to obtain such a result: Terms that can be associated to the same morphologic root (called their *stem*) can be represented by the root itself (*stemming*) or by the corresponding lemma (*lemmatization*); e.g., 'comput-er', 'comput-ational', 'comput-ation' are represented by the root 'comput' for stemming, or by the basic form 'compute' for lemmatization. Stemming is often preferred because it does not require the ability to retrieve the basic form of words, but in its simpler forms just consists in suffix removal.

**Suffix Stripping** *Suffix stripping* is a famous stemming algorithm, devised by M.F. Porter for English [25], that iteratively identifies and removes known prefixes and suffixes without exploiting any dictionary. Its advantage of being very simple has three drawbacks: it might generate terms that are not valid words in the language (e.g., computer, computational, computation $\to$ comput), it might reduce to the same representative words having different meaning (e.g., organization, organ $\to$ organ) and it cannot recognize morphologic deviations (e.g., go, went).

The algorithm exploits rewriting rules that progressively transform words into new forms. Any word satisfies the regular expression schema $[C]\{VC\}_m[V]$, where $C$ and $V$ denote non-empty sequences of consonants only and of vowels only, respectively (in addition to **a**, **e**, **i**, **o**, **u**, also **y** preceded by a consonant is considered a vowel). As for regular expressions, square brackets denote optional pieces and curly brackets denote iterations, whose number is indicated by the subscript (called the *measure* of the word). Rules are of the form

```
(C) S -> R
```

meaning that suffix `S` is replaced by `R` if the word meets the condition `C`. Rules are grouped in lists according to the step in which they are to be applied (as reported in Table 6.4); at most one rule per list/step is applied, and precisely the one whose `S` has longest match on the word (if any). The condition is a logical expression based on $m$ or on patterns to be checked on the word, where:

- $*$ denotes any sequence of characters;
- An uppercase letter denotes the presence (in uppercase or lowercase) of that letter in the word;
- $v$ denotes a vowel;
- $d$ denotes a double consonant;
- $o$ denotes a sequence *consonant–vowel–consonant* where the latter consonant is not **w**, **x** nor **y**.

When the condition is the same for all elements in a list, it is reported once and for all at the top of the list.

Informally, each of the five steps that make up the Porter stemmer is in charge of performing a particular reduction of the term, and precisely:

1. Reduces final '-s' in nouns and verbs inflection (e.g., books $\to$ book); reduces past participles and ing-forms (e.g., disabled $\to$ disable, meeting $\to$ meet); normalizes final '-i' in terms ending by '-y' and containing a vowel (e.g., happy $\to$ happi; sky $\to$ sky) and removes final '-e' (to provide a homogeneous input to step 4 as from step 2).

**Table 6.4** Synopsis of the rules underlying Porter's Suffix Stripping algorithm

| Step 1 | Step 2 | Step 3 | Step 4 | Step 5 |
|---|---|---|---|---|
| Step 1(a)<br>• SSES $\rightarrow$ SS<br>• IES $\rightarrow$ I<br>• SS $\rightarrow$ SS<br>• S $\rightarrow$<br>Step 1(b)<br>• $(m > 0)$ EED $\rightarrow$ EE<br>• $(*v*)$ ED $\rightarrow$<br>• $(*v*)$ ING $\rightarrow$<br>If rule 2 or 3 in Step 1(b) is successful:<br>• AT $\rightarrow$ ATE<br>• BL $\rightarrow$ BLE<br>• IZ $\rightarrow$ IZE<br>• $(*d \wedge \neg(*L \vee *S \vee *Z))$ $\rightarrow$ single letter<br>• $(m = 1 \wedge *o)$ $\rightarrow$ E<br>Step 1(c)<br>• (*v*) Y $\rightarrow$ I | $(m > 0)$<br>• ATIONAL $\rightarrow$ ATE<br>• TIONAL $\rightarrow$ TION<br>• ENCI $\rightarrow$ ENCE<br>• ANCI $\rightarrow$ ANCE<br>• IZER $\rightarrow$ IZE<br>• ABLI $\rightarrow$ ABLE<br>• ALLI $\rightarrow$ AL<br>• ENTLI $\rightarrow$ ENT<br>• ELI $\rightarrow$ E<br>• OUSLI $\rightarrow$ OUS<br>• IZATION $\rightarrow$ IZE<br>• ATION $\rightarrow$ ATE<br>• ATOR $\rightarrow$ ATE<br>• ALISM $\rightarrow$ AL<br>• IVENESS $\rightarrow$ IVE<br>• FULNESS $\rightarrow$ FUL<br>• OUSNESS $\rightarrow$ OUS<br>• ALITI $\rightarrow$ AL<br>• IVITI $\rightarrow$ IVE<br>• BILITI $\rightarrow$ BLE | $(m > 0)$<br>• ICATE $\rightarrow$ IC<br>• ATIVE $\rightarrow$<br>• ALIZE $\rightarrow$ AL<br>• ICITI $\rightarrow$ IC<br>• ICAL $\rightarrow$ IC<br>• FUL $\rightarrow$<br>• NESS $\rightarrow$ | $(m > 1)$<br>• AL $\rightarrow$<br>• ANCE $\rightarrow$<br>• ENCE $\rightarrow$<br>• ER $\rightarrow$<br>• IC $\rightarrow$<br>• ABLE $\rightarrow$<br>• IBLE $\rightarrow$<br>• ANT $\rightarrow$<br>• EMENT $\rightarrow$<br>• MENT $\rightarrow$<br>• ENT $\rightarrow$<br>• $(*S \vee *T)$ ION $\rightarrow$<br>• OU $\rightarrow$<br>• ISM $\rightarrow$<br>• ATE $\rightarrow$<br>• ITI $\rightarrow$<br>• OUS $\rightarrow$<br>• IVE $\rightarrow$<br>• IZE $\rightarrow$ | Step 5(a)<br>• $(m > 1)$ E $\rightarrow$<br>• $(m = 1 \wedge \neg *o)$ E $\rightarrow$<br>Step 5(b)<br>• $(m > 1 \wedge *d \wedge *L)$ $\rightarrow$ single letter |

2. Reduces 'double' suffixes to single ones (e.g., organization → organize; organizer → organize);
3. Somewhere in between steps 2 and 4: eliminates some simple suffixes (e.g., '-ful', '-ness': hopeful → hope; goodness → good) and reduces some 'double' ones (e.g., '-ic-' into '-ic': electriciti → electric; electrical → electric);
4. Eliminates simple suffixes ('-ant', '-ence', etc.);
5. Performs final stem adjustment (e.g., '-e': probate → probat; cease → ceas).

Suffixes are not removed when the length of the stem (i.e., its measure, $m$) is below a given threshold, empirically derived. The rules reporting 'single letter' as a substitution (in Steps 1(b) and 5(b)) reduce the final double letter to a single one.

*Example 6.3* (Application of the Porter stemmer to two sample words)

| | Step 1 | Step 2 | Step 3 | Step 4 | Step 5 |
|---|---|---|---|---|---|
| Generalizations | Generalization | Generalize | General | Gener | |
| Oscillators | Oscillator | Oscillate | | Oscill | Oscil |

Tests carried out by its author have shown that this algorithm reduces by about one third the size of a vocabulary of 10000 words. Its transposition to languages having a more complex morphology and/or characterized by many more suffixes, however, may not be as straightforward and efficient as for English.

### 6.2.5 Part-of-Speech Tagging

Lexical normalization is usually accompanied by a *Part-of-Speech* (*PoS*) *tagging* step that labels the words according to their grammatical function (which is typically hinted by the suffixes lost in normalization). This is a preliminary step for many activities. For instance, a syntactic analysis of sentences cannot be carried out without knowing the grammatical role of their components, and identifying the meaning of a component often cannot be carried out if grammatical role ambiguities are not solved. Indeed, just because some words can play different grammatical roles, PoS tagging usually cannot be performed by focusing on single items (e.g., 'run' as a verb means walking quickly while as a noun means a sequence of similar objects), but a sequence of neighbor terms that are present in the text (including those that have a low significance for interpretation purposes, and that are usually considered as stopwords) must be taken into account.

PoS taggers are obtained according to two approaches, statistical or rule-based. The latter has the advantage of being human-readable (and hence errors can be easily fixed), easily extensible, and compact, but is usually considered less reliable than the former. Conversely, statistical approaches have proven to be very performant, yielding high accuracy rates, but at the cost of a large amount of data to be stored for obtaining meaningful statistics, and of not being easily modified by human experts. A successful technique for automatically building statistical PoS taggers are Hidden Markov Models (*HMM*s) that can work either on labeled or on unlabeled data. In the following, a widely-known rule-based approach that has proven to reach similar accuracy as statistical approaches will be described.

**Rule-Based Approach** Brill's algorithm [4] builds a *rule-based* PoS tagger by first obtaining a rough set of rules from a *corpus* of texts, then applying it and using its errors to refine the rules and improve their performance using two techniques. The former (which alone turned out to reduce error rate below 8%) consists in tagging unknown words as proper nouns if they are capitalized, or as the most common tag among known words ending with the same three letters otherwise. Moreover, the tagger learns patch templates that change the initial tag $t$ of a word to a new tag $t'$ if a given condition is met, where conditions can be as follows:

- The word is in context $C$;
- The word has property $P$;
- Another word in a neighboring region $R$ of the word has property $P$,

and the property is one of the following: article, 'had', preposition, modal, singular noun, proper noun, 3rd singular nominal pronoun, objective personal pronoun, infinitive 'to', verb, past participle verb, past verb. Although these templates might not be fully correct, if any of them is useless, no instantiations of it will pass the conditions to be included in the final patch list.

The tagger is trained on 90% of the given *corpus* and applied to a further 5%, annotating the frequency of tagging errors as triples (*wrong*, *correct*, *amount*). Then, for each such triple the specific error occurrences are located, and the patch template that maximizes error reduction (computed as the amount of initial of errors that are now correctly recognized, minus the new errors introduced by application of the patch) is instantiated. As soon as a new patch is found, it is applied to improve the corpus tagging, before continuing the process. In operation, first the basic tagger is applied, then each patch in turn is tried for changing a word tag from $t$ to $t'$ if that word was tagged $t'$ at least once in the training corpus.

This takes linear time in the number of words to be tagged, and has shown to be ten times faster than the fastest stochastic tagger. However, one strength of stochastic taggers is the possibility of automatically building them using Machine Learning. Thus, Brill proposed several techniques for learning rules for his tagger as well [6], called *Transformation-Based Error-Driven Learning*.

In the supervised learning version, an initial annotator is built (using a previously available set of rules, or even at random), whose outcome is compared to the correct one on a manually annotated corpus. Then, from such a comparison, patches to be subsequently applied to the initial annotator output are learned according to the following space of allowed transformations [5]:

- The tag given to the word in a specific position (one or two places) before or after the current word;
- The tag given to any word within two or three positions before or after the current word;
- The tags of both the previous and the following words;
- The tags of the two words immediately before or after the current one;
- The current or the previous word being capitalized;

using error reduction as the evaluation function (in contrast to HMM learning that tries to maximize probability). In each learning iteration, the transformations that show the best error reduction are added. Since the algorithm is data-driven, only actually observed patterns are attempted, which reduces time requirements.

In the unsupervised version, neither likelihood information nor any other kind of knowledge is provided. The initial state annotator associates each word with the logical OR of all possible tags for it, as can be obtained by analyzing a dictionary. Instead of changing word tags, here the transformation templates reduce uncertainty, and hence are based on a context that specifies a tag or a specific word in the position immediately before or after the current word. As to the scoring function, error reduction cannot be considered since in this case the correct tags are not known. Instead, the previous tagging is used as a training set, leveraging the distribution of unambiguous words therein. The score of transforming a set of tags $A$ to one of its elements $b \in A$ in context $C$ is computed according to how frequently a candidate tag appears as measured by unambiguously tagged words with respect to all others in the context:

$$in(b, C) - \frac{f(b)}{f(R)} in(R, C),$$

where $f(t)$ is the number of words unambiguously tagged as $t$ in the previous iteration; $in(t, C)$ denotes how many times a word tagged as $t$ appears in context $C$ in the tagging obtained from the previous iteration; and

$$R = \arg \max_{c \in A, c \neq b} \frac{f(b)}{f(c)} in(c, C).$$

Learning stops when no positive scoring for this function is found.

Lastly, a *weakly supervised* strategy is proposed, where a limited supervisor intervention is permitted, to mix benefits coming from the exploitation of large unannotated resources with a limited human effort. Since a transformation-based system is a processor rather than a classifier, it can be applied to any initial state annotator. In particular, the initial state can be obtained by unsupervised learning: the initial state annotator and an unannotated text are provided to the unsupervised learner, and the result is input to the supervised learner, that provides ordered adjustment patches for that. Now, applying the result to the annotated corpus, a second ordered set of patches is learned. On new texts, first the unsupervised transformation, and then the supervised patches are applied.

### 6.2.6 Word Sense Disambiguation

Starting from the lexical level, the issue of semantic assessment of sentences comes into play. For instance, the BoW of a text could be further reduced by replacing synonymous terms by a single standard representative of the underlying concept. Unfortunately, as already pointed out, many words are polysemous, which means that they can be synonyms of different words in different contexts. Identifying their correct semantics is the objective of *Word Sense Disambiguation* (*WSD*), defined as the "association of a given word in a text or discourse with a definition or meaning (*sense*) which is distinguishable from other meanings potentially attributable to that word" [15]. More specifically, assuming that a previous PoS tagging step has resolved ambiguities about different possible syntactic categories for that word, WSD is concerned only with distinguishing its correct sense. An effective WSD step is clearly fundamental for all processing tasks involving text understanding, but can be profitably exploited in several other applications as well.

WSD works on an entire text or on selected words (called a *context*), for each of which the set of possible senses is known. A dictionary or thesaurus can easily provide such an initial set, to be possibly restricted based on the domain, text or discourse under consideration. A variety of techniques has been developed to assign one such sense to each word occurrence, working under two main approaches: *knowledge-driven*, exploiting information from external knowledge sources (such as lexica, encyclopedias, or special-purpose resources developed to support WSD), or *data-driven* (also known as *corpus-based*), leveraging information coming from previously disambiguated instances of the word in given collections of texts (based on the text, context or discourse in which the word appears, on the domain or situation to which the text refers, etc.). The two approaches are not incompatible, since the available resources can suggest a proper context, and knowledge about the context can point out the proper resources to be exploited.

The WSD task can take advantage by the consistency that can be generally appreciated (at least locally) in well-written texts, so that words belonging to the same piece of text are likely to refer to the same concept, or to strictly related ones. This feeling can be captured by different assumptions:

**One Sense per Collocation** the meaning in which a word is used is constant in all occurrences of a given collocation. A *collocation*, intended by J.R. Firth (in 'Modes of Meaning', 1951) as a 'habitual' or 'usual' co-occurrence, can be operationally identified as the probability that an item co-occurs, within a given distance, with other specific items [14] being greater than chance [3], or as the co-occurrence of two words in some defined relation [27].

**One Sense per Discourse** all occurrences of a word in a text refer to the same sense. Although the sense of a polysemous word depends on the context of use, it seems extremely likely (98%) that all of its occurrences in a discourse share the same sense [11]. This would allow disambiguating just one occurrence and then assigning that sense to all the others. Unfortunately, other studies found many more cases (33–45%) of multiple-senses per discourse, arguing that this hypothesis is likely (although not guaranteed) to hold for *homonymy* (unrelated meanings of a word), but not for polysemy (different but related meanings of a word) [16]. Thus, systematic polysemy of a word in a discourse can represent at best a hint to guide the tagging of occurrences of that word.

More formally, WSD aims at assigning a sense $\bar{s}$, chosen from a set of candidates $S$, to a word $w$, based on a set of *features* (boolean conditions) $\mathbf{f} = \{f_i\}$. The performance of a WSD classifier, called a *disambiguator*, is evaluated by comparison to the baseline accuracy obtained by always choosing for each word its most common sense. A classical tool to perform WSD are *decision list* classifiers, consisting of ordered lists each of whose items, of the form $f \to s$ predicts a specific sense $s$ for word $w$ if the feature $f$, based on collocations of $w$, is true (e.g., "word $w'$ occurs in a window of size $k$ around $w$", or "word $w'$ immediately precedes $w$", or "word $w'$ immediately follows $w$", etc.). It is clear that the One Sense per Collocation assumption is implicitly made by this model.

Since manually writing disambiguators is a complex task, there is a strong motivation to automatically induce them, which is usually performed according to the following approaches:

**Supervised learning** exploiting a (large) corpus annotated with word senses;

**Dictionary/Thesaurus based** exploiting a dictionary of word senses as a surrogate of the supervisor in supervised learning;

**Semi-supervised learning** (*bootstrapping*) exploiting a small set of labeled data and a large unlabeled corpus.

Supervised methods require the set of features to be preliminarily known. One technique is *Decision lists learning*, aimed at distinguishing between two possible senses ($S = \{s', s''\}$). It works as follows:

1. For each feature $f_i$ compute its score: $score(f_i) = \log \frac{p(s'|f_i)}{p(s''|f_i)}$;
2. Sort features by score.

Another approach consists in building *Naive Bayes classifiers*. It performs the assignment based on conditional probability, as follows:

$$\begin{aligned}\overline{s} &= \arg\max_{s\in S} p(s|\mathbf{f}, w) = \arg\max_{s\in S} \frac{p(s|w)p(\mathbf{f}|s, w)}{p(\mathbf{f}|w)} \\ &\quad \text{the denominator being independent of } s \\ &= \arg\max_{s\in S} p(s|w)p(\mathbf{f}|s, w) \\ &\quad \text{assuming features } f_i \text{ are conditionally independent given the word sense } s \\ &= \arg\max_{s\in S} p(s|w)\prod_i p(f_i|s),\end{aligned}$$

where probabilities can be estimated by frequency counts $|\cdot|$:

$$p(s|w) = \frac{|s, w|}{|w|} \quad \text{and} \quad p(f_i|s) = \frac{|f_i, s|}{|s|}.$$

The assumption of conditional independence is not always true (features usually take context into account, which is ignored in the above equation). Moreover, unseen words require some kind of smoothing to be handled.

**Lesk's Algorithm** A dictionary-based approach was proposed by Lesk [18]. It assigns the sense to a word by counting overlaps between terms that appear in the definitions of the various word senses. Given a dictionary $D$, and a word $w$ in a sentence made up of a set of content words $C$ (the context), its best sense $\overline{s}$ is found as follows:

```
1. $\overline{s} \leftarrow$ most frequent sense for $w$ in $D$
2. max $\leftarrow 0$
3. **for all** $s_i \in S$ **do**
   (a) overlap$_i \leftarrow |\text{signature}(s_i) \cap C|$
   (b) **if** overlap$_i >$ max **then**
       (i) $\overline{s} \leftarrow s_i$
       (ii) max $\leftarrow$ overlap$_i$
4. **return** $\overline{s}$
```

where $S$ is the set of all senses for $w$ as specified by $D$. In the original algorithm signature$(\cdot)$ is the signature of content words in the definition, but this often yields a zero overlap. Alternatively, signature$(\cdot)$ can be taken as the set of all content words in the definition of the sense in $D$.

**Yarowsky's Algorithm** Yarowsky [29] proposed a semi-supervised technique to learn a decision list for each ambiguous word in the training corpus, based on microcontexts:

1. Choose a few seed collocations for each sense and label those words;
2. **while** coverage of the corpus is not sufficient **do**
   (a) Train a supervised classifier on the labeled examples;
   (b) Label all examples with the current classifier, and store into a new labeled data set the labels about which the current classifier is confident above a threshold;
   (c) Optionally, propagate labels throughout the discourse to fix mistakes and obtain additional examples.

After collecting contexts for the word to be disambiguated, a few representative occurrences for each sense of that word in those contexts are manually labeled as training examples. Then, they are used to learn a classifier to be applied on the remaining contexts (called the *residual*), leveraging the uneven distribution of collocations with respect to the ambiguous occurrences to be classified. At each round, new collocations might be discovered: if they take the reliability of some previous assignments under the threshold, this causes such assignments to be withdrawn. One-sense-per-collocation is implicitly assumed in the decision list classifier,[10] and One-sense-per-discourse is exploited in the optional propagation step.

The suggested learning algorithm [28] uses as contexts windows (sets of neighbor words) of different size $\pm k$ ($k \in [2, 10]$), and words pairs at offsets $\{-1, -2\}$, $\{-1, +1\}$, or $\{+1, +2\}$ with respect to the word to be disambiguated. It is argued that the optimal value for $k$ depends on the kind of ambiguity (for local ambiguities $k \in [3, 4]$ is enough, while semantic or topic-based ambiguities require $k \in [20, 50]$), and that, for different ambiguous words, different distance relations are more efficient. Furthermore, the use of additional (e.g., PoS) information is also considered (e.g., 'first noun to the left').

The final decision list classifier is ordered by decreasing predictive features based on log-likelihood ratio to find the most reliable evidence for disambiguation: item $f_i \rightarrow s'$ precedes item $f_j \rightarrow s''$ in the list if

$$\log \frac{p(s'|f_i)}{p(s''|f_i)} \geq \log \frac{p(s''|f_j)}{p(s'|f_j)}.$$

In ambiguous cases, the first matching feature is used.

As to the initial labeling, it can start from a very limited number of words, even two (depending on how much burden one wants to put on the automatic procedure). The actual labeling can be carried out manually, by focusing on the most frequent collocations, or by relying on dictionary definitions (e.g., WordNet provides indications on the frequency of sense usage for each word).

### 6.2.7 Parsing

Syntactic or logical parsing of sentences allows obtaining their structure. This is a quite difficult task, due to the intrinsic complexity of natural language, and also the

[10] In cases of binary ambiguity, the estimation is that in a given collocation a word is used with only one sense with a 90–99% probability [27].

resulting information turns out to be in many cases very complex to be represented effectively and handled efficiently. Nevertheless, this level of text description is fundamental to carry out enhanced NLP tasks, such as text understanding or some kinds of information extraction. As a trivial example, the sentences "The dog bit the man" and "The man bit the dog" contain exactly the same words, and hence would be considered just equivalent by a BoW-based approach, although reporters know very well that the former is not news, while the latter is. To gain this level of understanding, one must be able to distinguish the subject and the object of the sentence.

The grammars exploited in NLP are usually aimed at identifying suitable combinations of (some variation of) the following kinds of relevant aggregates (called *constituents*):

**Noun Phrase** a (meaningful) combination of articles, pronouns, nouns and adjectives;
**Verb Phrase** a (meaningful) combination of verbs and adverbs;
**Prepositional Phrase** usually a (series of) Noun Phrase(s) introduced by preposition(s);
**Clause** a sub-component of a sentence having a complete meaning by itself, such as a subordinate or incidental;
**Sentence** a complete period in the text, involving some combination of the previous structures and of other grammatical items such as conjunctions and exclamations;

at the syntactic level, and, within a sentence, its logical components such as *Subject*, *Predicate*, *Direct* and (different types of) *Indirect Objects*, plus information on the associations of clauses with these items.

**Link Grammar** *Link Grammar* [26] is a formal context-free grammatical system based on the analysis of different kinds of *links* connecting pairs of words in a sentence. It is purely lexical (i.e., it does not make explicit use of constituents and syntactic categories, as phrase structure grammars do); anyway, constituents emerge as particular types of links that join some subsets of words to each other. The grammar definition being distributed and split as information associated to each single word, it can be easily extended, and allows knowing exactly which words can be connected directly to which others. A probabilistic version of the model, based on an algorithm for determining the maximum-likelihood estimates of the parameters, was proposed in [17].

Words in the allowed vocabulary represent terminal symbols of the grammar. They are associated to *linking requirements*, expressing how they can be correctly exploited to build sentences, and are grouped by similar requirements. Each requirement is an expression that composes a number of *connectors* (link types to be *satisfied* by linking them to matching connectors in other words), using logic operators AND and XOR, plus parentheses as needed for specifying the desired evaluation priority. Sometimes the empty formula (), implicitly satisfied, can be useful. This formalism is easy to read by humans and allows compactly expressing many combinations of possible exploitations of a word.

A connector begins with one or more capital letters, possibly followed by lowercase letters (called *subscripts*) or $*$'s, and terminated by $-$ (imposing a link to a word on the left) or + (imposing a link to a word on the right). It *matches* another connector having opposite sign (placed in the proper direction according to the +/− specification) if, after lining them up on the first character and considering them as followed by infinite ∗'s, the two strings match (with a ∗ matching any lowercase letter). A *multiconnector*, denoted by a @ prefix, is allowed to connect to one or more links (e.g., sequences of adjectives).

A *linkage* is a set of links that successfully recognize a sentence belonging to the language. A sequence of words is a valid *sentence* if the following constraints are fulfilled:

**Satisfaction** the local linking requirements of each word are satisfied;
**Planarity** the links do not cross (i.e., the corresponding graph is planar);
**Connectivity** the words and links form a connected graph;
**Ordering** scanning the requirements formula of a word, the linked words have monotonically increasing distance from that word, i.e., the more 'external' connectors in the scanning direction are satisfied by closer words (in that direction) than more 'internal' ones (hence, the AND is not commutative);
**Exclusion** at most one link is specified for each pair of words.

*Example 6.4* (Sample link grammar outcome) Parsing the sentence "*This is a test of the link grammar approach*" produces:

```
    +------------------------Xp-----------------------+
    |                     +-----------Js----------+   |
    |                     |  +---------Ds---------+   |
    |           +-Ost--+  |  |     +------AN------+   |
    +--Wd--+Ss*b+ +-Ds-+Mp+  |     |     +---AN---+   |
    |      |    | |    |  |  |     |     |        |   |
LEFT-WALL this is a test of the link grammar approach .
         p     v      n           n     n       n
```

where the following connectors appear:

**AN** noun-modifiers to following nouns;
**D** determiners to nouns;
**J** prepositions to their objects;
**M** nouns to various kinds of post-noun modifiers;
**O** transitive verbs to their objects, direct or indirect;
**S** subject nouns to finite verbs;
**W** subjects of main clauses to the wall;
**X** punctuation symbols either to words or to each other,

with constituent tree:

```
(S (NP This)
   (VP is
       (NP (NP a test)
           (PP of
               (NP the link grammar approach))))
   .)
```

where **S** denotes Sentences, **NP** Noun Phrases, **VP** Verb Phrases and **PP** Prepositional Phrases.

The first application of the model to English consisted of a vocabulary of about 25000 words specifying 800 requirement definitions. In its latest version, maintained in cooperation with the Abiword project,[11] it was extended to 60000 word forms, featuring enhanced handling of capitalization, numeric expressions and punctuation, and improved for coverage (of syntactic constructions), robustness (to portions of the sentence that it cannot understand) and flexibility (guessing the category of unknown words based on context and spelling). Its implementation, in C language, carries out an exhaustive search using dynamic programming techniques, look-up in a dictionary of links for each word, and exploits suitable data structures and heuristics to ensure a performance that is cubic ($O(n^3)$) in the number $n$ of words that make up the sentence to be parsed.

## References

1. Allen, J.F.: Natural Language Understanding. Benjamin-Cummings, Redwood City (1994)
2. Bentivogli, L., Forner, P., Magnini, B., Pianta, E.: Revising WordNet domains hierarchy: Semantics, coverage, and balancing. In: Proceedings of COLING 2004 Workshop on Multilingual Linguistic Resources, pp. 101–108 (2004)
3. Berry-Rogghe, G.: The computation of collocations and their relevance to lexical studies. In: Aitken, A.J., Bailey, R.W., Hamilton-Smith, N. (eds.) The Computer and Literary Studies, pp. 103–112. Edinburgh University Press, Edinburgh (1973)
4. Brill, E.: A simple rule-based part of speech tagger. In: HLT '91: Proceedings of the Workshop on Speech and Natural Language, pp. 112–116 (1992)
5. Brill, E.: Some advances in transformation-based part of speech tagging. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI), vol. 1, pp. 722–727 (1994)
6. Brill, E.: Unsupervised learning of disambiguation rules for part of speech tagging. In: Natural Language Processing Using Very Large Corpora Workshop, pp. 1–13. Kluwer, Amsterdam (1995)
7. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-lite family. Journal of Automated Reasoning **39**(3), 385–429 (2007)
8. Calzolari, N., Lenci, A.: Linguistica computazionale—strumenti e risorse per il trattamento automatico della lingua. Mondo Digitale **2**, 56–69 (2004) (in Italian)
9. De Mauro, T.: Grande Dizionario Italiano dell'Uso. UTET, Turin (1999) (in Italian)
10. Dewey, M., et al.: Dewey Decimal Classification and Relative Index. Edition 22. OCLC Online Computer Library Center (2003)
11. Gale, W., Church, K., Yarowsky, D.: One sense per discourse. In: Proceedings of the ARPA Workshop on Speech and Natural Language Processing, pp. 233–237 (1992)
12. Grishman, R.: Computational Linguistic—An Introduction. Studies in Natural Language Processing. Cambridge University Press, Cambridge (1986)
13. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2), 199–220 (1993)
14. Halliday, M.: Categories of the theory of grammar. Word **17**, 241–292 (1961)

[11] http://www.abisource.com/projects/link-grammar.

15. Ide, N., Véronis, J.: Introduction to the special issue on Word Sense Disambiguation: The state of the art. Compuational Linguistics **24**(1), 1–40 (1998)
16. Krovetz, R.: More than one sense per discourse. In: Proceedings of SENSEVAL Workshop, pp. 1–10 (1998)
17. Lafferty, J., Sleator, D.D., Temperley, D.: Grammatical trigrams: A probabilistic model of link grammar. In: Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language (1992)
18. Lesk, M.: Automatic sense disambiguation using machine-readable dictionaries: How to tell a pine cone from an ice cream cone. In: Proceedings of the 5th International Conference on Systems Documentation (SIGDOC), pp. 24–26 (1986)
19. Magnini, B., Cavaglià, G.: Integrating subject field codes into WordNet. In: Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC), pp. 1413–1418 (2000)
20. Manning, C.D., Schutze, H.: Foundations of Statistical Natural Language Processing. MIT Press, New York (1999)
21. McCarthy, J., Minsky, M.L., Rochester, N., Shannon, C.E.: A proposal for the Dartmouth Summer research project on Artificial Intelligence. Tech. rep., Dartmouth College (1955)
22. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.J.: Introduction to WordNet: An on-line lexical database. International Journal of Lexicography **3**(4), 235–244 (1990)
23. Oltramari, A., Vetere, G.: Lexicon and ontology interplay in Senso Comune. In: Proceedings of OntoLex 2008 Workshop, 6th International Conference on Language Resources and Evaluation (LREC) (2008)
24. Pierce, J.R.: Symbols, Signals and Noise—The Nature and Process of Communication. Harper Modern Science Series. Harper & Brothers (1961)
25. Porter, M.: An algorithm for suffix stripping. Program **14**(3), 130–137 (1980)
26. Sleator, D.D., Temperley, D.: Parsing English text with a link grammar. In: Proceedings of the 3rd International Workshop on Parsing Technologies (1993)
27. Yarowsky, D.: One sense per collocation. In: Proceeding of ARPA Human Language Technology Workshop, pp. 266–271 (1993)
28. Yarowsky, D.: Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In: Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, pp. 88–95 (1994)
29. Yarowsky, D.: Unsupervised Word Sense Disambiguation rivaling supervised methods. In: Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, pp. 189–196 (1995)

# Chapter 7
# Information Management

The ultimate objective of automatic document processing is capturing the various aspects of the information conveyed by documents, so that they can be semantically interpreted and exploited for satisfying the user needs. Clearly, a fundamental requirement is that users must be able to search large repositories of documents and obtain as a result a list of items that closely fit their purposes, in spite of the many levels of complexity and ambiguity that are inborn in their representation. More advanced applications aim at the explicit extraction and exploitation of information, such as automatic summarization or categorization of documents. The two perspectives are closely related because the extraction of suitable information can obviously enable or improve a focused information retrieval, and effective information retrieval can filter relevant documents in which searching the information to be extracted. Additionally, both can take advantage from a deep understanding of the document semantics that resembles as closely as possible their human interpretation. A further related concern is how to represent the document-related knowledge in a formal and standard way that allows different applications to share their information and to consistently process it. The mainstream research has so far devoted most effort to develop techniques concerned with text, as the easiest way to approach explicit semantics. Accordingly, this chapter will deal mostly with the textual perspective, although a few hints to the emerging field of image-based information retrieval will be provided.

## 7.1 Information Retrieval

The spread of electronic documents in the last years has significantly increased the number and size of the repositories in which they are stored. As a consequence, final users experience dramatic troubles in retrieving those that are actually relevant to satisfy their *information needs*. An outstanding example is the search for interesting documents on the Web using traditional query-based search engines. On the other hand, the aim of making up document collections is to enforce the control and management of the information they contain, and in order to pursue such an objective the ability to retrieve the desired information in an easy way is essential. Manual search in this context is often infeasible, especially when information is available in non-structured form (e.g., texts in natural language or images). Hence, the need to develop automatic and clever systems for document indexing and for the effective and efficient retrieval of interesting information from them, which is the fundamental motivation for the research in the field of *Information Retrieval* (*IR*). IR, that is specifically concerned with problems related to the organization and representation of information, as a key to properly accessing it, is aimed at improving the activities of management, organization and search of documents.[1]

Currently, users search documents and information using almost exclusively textual *queries* in (some simplified form of) natural language. Unfortunately, the ambiguity and imprecisions that are typical of natural language represent by themselves an obstacle, which is made still more difficult by the fact that, currently, both the documents in the repository and the textual queries that express the information needs of the user are often reduced to simple sets of terms, possibly weighted, used as a surrogate of the related semantic content (a trend that is going to be overcome by more recent methods that try to interpret the document collection on which the search is to be performed and to capture the underlying semantics). For instance, this chapter, although clearly not providing any useful information on winter sports, would be retrieved as a result of a "winter sports" query just because of the presence of these words in the text.

### 7.1.1 Performance Evaluation

The outcomes of search sessions have a direct impact on the activities of those who are going to use them, boosting their productivity if good, or possibly even preventing them from reaching their objectives if bad. Thus, information retrieval is a ticklish activity in which the *relevance* of the results takes on a crucial importance. Ideally, what the automatic IR systems consider as relevant or irrelevant should match as much as possible what is actually such for the user. In other words, the objective is avoiding that a document that is relevant to the user is not retrieved by the system or, conversely, that a document proposed as relevant by the system is not significant to the user. In this perspective, useful parameters to evaluate the effectiveness of the results are the following:

**TP** (*True Positives*) relevant documents recognized as such by the system;
**FN** (*False Negatives*) relevant documents not considered as such by the system;
**FP** (*False Positives*) documents erroneously considered relevant by the system;
**TN** (*True Negatives*) documents that the system correctly considers as irrelevant.

[1] A possible specification/refinement/evolution of these objectives involves the possibility of querying a document base by issuing the request as a question in natural language (a field known as *Question Answering*).

More precisely, the number of documents for each such category in a search result is of interest.

The classical measures used in IR and based on these parameters[2] are *Recall* ($R$) that expresses how good is the system in retrieving relevant documents:

$$R = \frac{\text{number of relevant documents retrieved}}{\text{total number of relevant documents in the collection}} = \frac{TP}{TP + FN} \in [0, 1]$$

and *Precision* ($P$) that represents the degree of relevance of the retrieved documents with respect to the overall result of the query:

$$P = \frac{\text{number of relevant documents retrieved}}{\text{total number of documents retrieved}} = \frac{TP}{TP + FP} \in [0, 1].$$

These measures catch two different, and often opposite, desirable features of the retrieval effectiveness. Thus, improving performance of either of the two usually results in a worse performance of the other. To have a single value that characterizes the overall performance of a system (which is also more comfortable for evaluation), a tradeoff between them must be taken into account. With this objective, the *Fallout* measure, or *F-measure* (also called *F-score*), was defined as:

$$F = \frac{(1 + \beta^2) \cdot P \cdot R}{(\beta^2 \cdot P + R)} = \frac{(1 + \beta^2) \cdot TP}{(1 + \beta^2) \cdot TP + \beta^2 \cdot FN + FP},$$

where $\beta$ is a non-negative real number expressing the relative weight given to recall with respect to precision (e.g., $\beta = 2$ assigns recall twice the importance of precision, while $\beta = 0.5$ does the opposite). The traditional setting uses $\beta = 1$ to assign to both parameters the same importance, thus yielding the harmonic mean of precision and recall, often denoted as *F1-measure*:

$$F = \frac{2 \cdot P \cdot R}{P + R}.$$

[2]Another measure typically used to evaluate performance of Machine Learning systems is *Accuracy* (*Acc*), representing the overall ratio of correct choices made by the system:

$$Acc = \frac{TP + TN}{TP + FN + FP + FN} \in [0, 1]$$

or its complement, *Error Rate* (*ER*):

$$ER = 1 - Acc \in [0, 1].$$

This measure, that is intuitive and sufficiently significant when the number of positive and negative instances is balanced, becomes tricky when these quantities are very different from each other. Indeed, when negative instances are an overwhelming majority (say 99%), as is the case for IR (where the number of documents in the collection that are not significant to the query are almost the totality), a trivial system saying that everything is irrelevant would reach 99% accuracy, although it is clear that its performance is absolutely insignificant.

**Fig. 7.1** Term–document matrix in the vector space model

$$\begin{array}{c|c|c|c|c|}
 & d_1 & d_2 & \dots & d_n \\
\hline
t_1 & w_{11} & w_{12} & \dots & w_{1n} \\
\hline
t_2 & w_{21} & w_{22} & \dots & w_{2n} \\
\hline
\dots & \dots & \dots & & \dots \\
\hline
t_m & w_{m1} & w_{m2} & \dots & w_{mn} \\
\hline
\end{array}$$

## *7.1.2 Indexing Techniques*

Effective IR is heavily founded on clever strategies to index documents so that those that are relevant to a given query can be quickly and precisely identified. In turn, two peculiarities distinguish the different indexing techniques and determine their strengths and weaknesses: the parameters according to which they organize a given collection of documents in order to make easy the subsequent retrieval of some of them, and the representation adopted to compactly store the collection itself. The landscape of solutions proposed in the literature to tackle these two issues is very wide. Some of the most interesting will be introduced in the following. Most of them are based, directly or indirectly, on the vector space model representation.

**Vector Space Model** The *Vector Space Model* [24] is a mathematical model for the compact representation of a set of texts. A collection of $n$ documents, in which $m$ different terms appear overall, is represented by means of a *Term–Document Matrix* $A_{m \times n}$ in which each column represents a document, and each row is associated to a term, describing its distribution across documents. Thus, each matrix element $w_{ij}$ expresses the 'relevance' of the $i$th term in the $j$th document (see Fig. 7.1). Such a relevance, in the form of a numerical weight, is typically assigned as a function of the frequency by which the former appears in the context of the latter, and depends on both the importance of the word in the considered text (the column of the matrix), and the degree by which the word carries information on the general domain of discourse. This representation defines a high-dimensional geometrical space, whose dimensions are the terms, in which a text $d_j$ is represented as an $m$-dimensional column-vector:

$$\mathbf{d}_j = (w_{1j}, w_{2j}, \dots, w_{mj}),$$

where each component acts as a coordinate whose value is the weight $w_{ij}$ for the term/dimension $t_i$. If the $i$th term is not present in the $j$th document, $w_{ij}$ is usually null. In this case, since typically only a very small portion of all indexed terms appears in each single document, the Term–Document matrix $A$ is very sparse, which allows for optimizations in its storage and handling that try to compensate for its huge size. In order to deal fairly with documents having different length, where the weights might be biased, a possible solution is to normalize the document vectors so that all have unit length. That is, the norm of the document $\mathbf{d}$ must be equal to 1:

$$\|\mathbf{d}\| = 1.$$

The values in $A$ are computed according to suitable *weighting functions* that are usually based on the number of occurrences of the terms in single documents and/or in the whole collection. In order to have a more realistic estimation of the importance of a term in a document, it is wise to consider the term not only in the local context (i.e., in the document under consideration), but also in the general context of the entire document collection. Indeed, although it is sensible to consider a term as presumably relevant when it is very frequent, it is nevertheless true that, if it is equally frequent in most documents of the collection, it is not actually very discriminative (think, e.g., of articles and prepositions). Hence, to improve the quality of the results, a good weighting scheme should try and balance a *local factor* $L(i, j)$ (sometimes called *tf factor*, from *T*erm *F*requency) that provides a measure of how relevant term $t_i$ is to document $d_j$ (i.e., how well it could represent its content) and a *global factor* $G(i)$ (sometimes referred to as *idf factor*, from *I*nverse *D*ocument *F*requency) that provides a measure of the spread of term $t_i$ among all documents in the collection. A weighting function that takes into account both parameters is obtained by their product:

$$w_{ij} = L(i, j) \cdot G(i).$$

Several functions have been proposed, both for local and for global weighting. They variously combine the following parameters:

- $tf_{ij}$ number of occurrences of term $t_i$ in document $d_j$;
- $\max_i tf_{ij}$ maximum number of occurrences among all terms in document $d_j$;
- $gf_i$ total number of occurrences of term $t_i$ in the whole document collection;
- $n$ number of documents that make up the collection;
- $df_i$ number of documents in the collection in which term $t_i$ appears.

Typical functions used for local weighting are the following:

**Binary**

$$L(i, j) = \begin{cases} 1 & \text{if term } t_i \text{ is present in document } d_j; \\ 0 & \text{otherwise;} \end{cases}$$

**Normal**

$$L(i, j) = tf_{ij};$$

**Normalized**

$$L(i, j) = 0.5 + 0.5 \frac{tf_{ij}}{\max_i tf_{ij}}.$$

Binary weighting ignores the number of occurrences, and takes into account just the presence or absence of terms. Normalized weighting is based on the assumption that longer texts are likely to include more occurrences of a term, which would penalize shorter ones, and hence reports absolute occurrence values to a standard range that affects only half of the total weight.

For the global weighting one can use:

**Fixed**

$$G(i) = 1;$$

**Logarithmic**

$$G(i) = \log \frac{n}{df_i};$$

**Probabilistic**

$$G(i) = \log \frac{n - df_i}{df_i}.$$

Fixed weighting just ignores the spread of the term along the entire collection. Logarithms are used in the other two cases to obtain a more comfortable mathematical behavior of the function.

The most widely-known *weighting scheme* is called *TF-IDF*, and is based on the following considerations:

- The more frequently a term occurs in a document, the more it can be presumed to be important:

  $$L(i, j) = tf_{ij}.$$

  The term frequency can be normalized considering the whole corpus of documents:

  $$L(i, j) = \frac{tf_{ij}}{\max\{tf_{ij}\}}.$$

- Terms that appear in many documents are less significant:

  $$G(i) = idf_i = \log_2 \frac{n}{df_i},$$

  i.e., the inverse average frequency of term $t_i$ across the collection.

The overall *TF-IDF* weight associated to a term is thus computed as

$$w_{ij} = tf_{ij} \cdot idf_i = tf_{ij} \cdot \log_2 \frac{n}{df_i} \quad \text{or}$$
$$w_{ij} = \frac{tf_{ij}}{\max\{tf_{ij}\}} \cdot idf_i = \frac{tf_{ij}}{\max\{tf_{ij}\}} \cdot \log_2 \frac{n}{df_i}.$$

The *log-entropy* weighting scheme is named after being obtained as the product between a local factor with logarithmic trend and a global one of entropic kind:

$$L(i, j) = \log(tf_{ij} + 1), \qquad G(i) = 1 - \sum_j \frac{p_{ij} \cdot \log(p_{ij})}{\log n},$$

where $p_{ij} = \frac{tf_{ij}}{gf_i}$. The local factor is the logarithm of the number of occurrences of term $t_i$ in document $d_j$, in order to smooth the effects of large differences in frequencies. The global factor is the entropy (representing the 'noise' in the collection) that takes into account the distribution of terms in all documents.

After years in which most effort on NLP was devoted to increasing performance in terms of efficiency, the need to improve effectiveness to tackle real-world cases led more recent research towards the extension and enrichment of the purely lexical representation, based on term occurrences, with conceptual or semantic information, so that documents are described (also or only) according to the concepts expressed therein. For instance, replacing terms in a BoW by the corresponding concept (if identified with sufficient accuracy, which could require a preliminary disambiguation step), or by a hypernym thereof (in the attempt to trade generality for polysemy), yields the so-called *Bag of Senses* (*BoS*) on which the same techniques presented in this section for BoWs can be applied. The new representation would also allow identifying and removing (*prune*) concepts that have little significance to the collection (e.g., because too frequent or too infrequent), this way enhancing the document representation and further reducing its size.

### 7.1.3 Query Evaluation

The simplest term-based *query evaluation* technique is *Text Matching* that returns the list of documents that contain all words in the query, ordered by decreasing cumulative number of occurrences of such terms. If $q_1, \dots, q_k$ are the query terms, the score for the $j$th document is computed as follows:

$$\sum_{i=1}^{k} o(i, j),$$

where the $o(i, j)$ function returns the number of occurrences of term $q_i$ in the $j$th document. Thus, documents containing more occurrences of the query terms are returned first in the final ranking. Although very simple, this technique requires long computational times, especially on large document collections. A more complex technique is *clustering* that groups similar documents (meaning that they tend to be relevant for the same kind of search), this way improving the retrieval speed because only a representative of each group of documents is considered, and all corresponding documents are handled by means of their representative.

A vector space defines a multi-dimensional space whose dimensions are the terms, and where each document can be associated to a point in the space by considering its weights as coordinates for the corresponding axes. This allows translating the vague notion of 'similarity' between any two documents, according to the terms they contain, in more formal geometrical notions of closeness. Given two documents $d'$ and $d''$, a straightforward way to carry out a comparison is using the Euclidean distance between the points/vectors representing them, $\mathbf{d}'$ and $\mathbf{d}''$. Another strategy is computing a linear combination of the values in $\mathbf{d}'$ with those in $\mathbf{d}''$. However, these solutions would depend on the exact position of the two documents, and hence on the specific weights that describe them, which could be unfair for documents having very different lengths and number of distinct terms. Thus, the measure most widely used in the literature for this purpose is *cosine similarity*, based on the cosine of the angle between the two vectors:

$$CosSim\big(\mathbf{d}', \mathbf{d}''\big) = \frac{\mathbf{d}' \cdot \mathbf{d}''}{\|\mathbf{d}'\| \cdot \|\mathbf{d}''\|} = \frac{\sum_{i=1}^{m} (w_i' \cdot w_i'')}{\sqrt{\sum_{i=1}^{m} (w_i')^2}\sqrt{\sum_{i=1}^{m} (w_i'')^2}},$$

where

- $m$ is the number of terms;
- $\mathbf{d} = (w_1, w_2, \dots, w_m)$ denotes a document vector and $w_i$ is the value associated to its $i$th element;
- $\|\mathbf{d}\|$ is the norm of $\mathbf{d}$.

Note that, assuming that all document vectors are normalized to unit length, the cosine similarity formula is simplified to $\mathit{CosSim}(\mathbf{d}', \mathbf{d}'') = \mathbf{d}' \cdot \mathbf{d}''$.

The problem of document comparison becomes especially relevant when either of the two is a query string $q$, and the aim is retrieving the documents in a collection that are most 'similar' to it. To be able to apply the foregoing techniques, $q$ must be represented as a *pseudo-document*, i.e., as a vector of weights $\mathbf{q}$ using the same schema, and the same global statistics, exploited to build the vector space in which the search is to be carried out. In particular, if documents in the Term–Document matrix are represented as vectors of $m$ elements, $\mathbf{q}$ must be $m$-dimensional as well have a one-to-one correspondence between its elements and those of the document vectors (so that corresponding elements in both refer to the same term). Each element value is a function of the corresponding term occurrences, with the positions of terms that do not appear in the query set at 0 and ignoring query terms that are not indexed in the Term–Document matrix. Moreover, for a fair comparison, the same weighting function used for the indexing process must be applied. This allows locating $\mathbf{q}$ as a point in the same space just as any other document $\mathbf{d}_j$ in the collection, and hence comparing them for similarity. After computing the value $\mathit{CosSim}(\mathbf{d}_j, \mathbf{q})$ for all documents $\mathbf{d}_j$ $(j = 1, \dots, n)$ in the collection, suitable operations can be carried out on the resulting list of values. Documents can be ranked by decreasing similarity to the query, returning the top items (e.g., those whose value is above a given threshold) as the query result, and/or only documents that contain all query terms can be selected.

**Relevance Feedback** Knowing which documents in a collection are relevant, and which ones are irrelevant, to a given query may help in better focusing the search results. Of course, if such documents were completely known, the search problem would become void, but this is clearly not the case. The principle underlying the *relevance feedback* approach is that just sample subsets of these items can be sufficient to improve the retrieval task, and that such sample subsets can be provided, implicitly or explicitly, by the very user that issued the query. In particular, given a user query (represented as a term vector) $\mathbf{q}$ and an indication of a set of documents $P$ that the user considers relevant and a set of documents $N$ that he considers irrelevant for that query, a famous formula to obtain a modified query $\mathbf{q}'$ that takes into account all this information was proposed by Rocchio [23]:

$$\mathbf{q}' = \alpha \cdot \mathbf{q} + \beta \cdot \frac{\sum_{\mathbf{p} \in P} \mathbf{p}}{|P|} - \gamma \cdot \frac{\sum_{\mathbf{n} \in N} \mathbf{n}}{|N|},$$

where the value of each coordinate (i.e., term) in the original query is changed by adding the average of the corresponding values in the positive samples and subtracting the average of the corresponding values in the negative samples. In other words, the vector defined by the terms in the original query is moved towards (as if it were attracted by) the centroid of the positive samples, and far from (as if it were repelled by) the centroid of the negative samples. Parameters $\alpha$, $\beta$ and $\gamma$ weight the importance given, respectively, to the original query terms, to the positive samples and to the negative samples. It is usual to weight the positive samples more than the negative ones, assuming that the user's indications tend to be biased towards interesting items (typical weights might be $\alpha = 1$, $\beta = 0.75$, $\gamma = 0.15$ [18]). At the extreme, $\gamma = 0$ totally ignores negative samples and completely attracts the query towards the centroid of the positive ones, which is a very common setting in recent practice. The technique can be iterated, gaining feedback each time from the last query results, and has been shown to improve mainly recall (but also precision).

## 7.1.4 Dimensionality Reduction

Since the size of a typical Term–Document matrix is very large, a compelling need is to reduce its dimensionality. While some traditional ways for reducing the dimensionality of attribute–value representations focused on the identification and removal of less significant features (*feature selection*) or on the computation of aggregate features that express the same information as the original ones, here the aim is to develop brand new features based on the conceptual content of the documents. Instead of representing documents as points in a space whose dimensions are words, advanced *dimensionality reduction* techniques represent documents (and related queries) as points in a space whose dimensions are the underlying concepts automatically identified. Thus, the parameters of interest become:

- $n$ the number of documents;
- $m$ the number of terms;
- $k$ the number of concepts according to which restructuring the space.

These techniques are purposely designed to overcome the main problem of retrieval approaches that compute the relevance of a document with respect to a query based on the strict presence or absence, in the former, of the terms contained in the latter. Indeed, while words (especially if considered independent of each other, as in BoWs) are not reliable indicators for a document's content and semantics, due to their intrinsic ambiguity and to the author's taste in choosing them,[3] concepts represent a more stable reference. They allow retrieving the indexed documents not based on their literal correspondence with a query, but on analogous meanings, this way returning even documents that do not contain the same words as the query. Moreover, while words (at least, function words) cannot be deliberately ignored without affecting the correctness of the statistical analysis, less important concepts can be stripped out, this way restructuring the space so that more significant similarities can emerge.

[3] Because of *synonymy*, a person issuing a query might use different words than those that appear in an interesting document to denote the same concepts, so the document would not be retrieved; because of *polysemy*, uninteresting documents concerning the alternate meanings might be retrieved.

**Latent Semantic Analysis and Indexing** The phrase *Latent Semantic* denotes a framework aimed at capturing the concepts expressed in a given document collection by relying on deep, and thus hidden (whence the term *latent*), relations among words. Rather than generalizations of terms (e.g., dog $\rightarrow$ mammal), concepts are abstract representations of sets of interrelated terms, called *semantic domains* (e.g., {dog, pet, mammal, ... }). Such relations cannot emerge by simple considerations about frequency, co-occurrence or correlation in term usage because the semantic structure that characterizes each sentence is not just a many-to-many association among terms and concepts (it is hidden to the words chosen to express it and depends on the whole set of documents). They can be evaluated using statistical techniques that remove noisy elements and focus on the underlying concepts. Latent Semantic represents both words and documents in the same space of concepts, which allows assessing (semantic) similarity not only among texts, but also among words, and between words and texts as well, opening a number of possible applications.

*Latent Semantic Analysis* (*LSA*) is a mathematical technique to automatically extract and infer, from a document collection, relationships concerning the contextual exploitation of words (defined as unique and separated character strings that make up significant aggregates such as sentences or paragraphs) in some passages of the discourse [17]. The basic idea is that the contexts in which a word appears, or does not appear, act as constraints to determine the similarity among the meanings of groups of words. Interestingly, the semantics that emerges from the LSA is based only on the terms and their distribution in the document collection, without exploiting any external resource, such as dictionaries, human intervention (which would be inefficient and costly) or perception of the real world outside. Although LSA adopts a simple BoW representation of texts, several comparisons of its outcomes to the cognitive behavior of human mind revealed a close resemblance, improving as long as their semantic spaces become closer [16]. For this reason, LSA has application to several cognitive tasks, such as classification of texts according to a subject, information retrieval, evaluation of text consistency, evaluation of human cognitive behavior in various contexts (e.g., answers to questions of psychiatrists, learning degree of students on educational texts, etc.). Of course, such an apparent resemblance is not a guarantee that statistics is a suitable technique underlying human psychology: thus, there is still no agreement about the LSA being a simulation or an emulation of the human knowledge.

LSA takes the Vector Space representation of the *corpus* (it was observed that the log-entropy weighting scheme, or the TF-IDF scheme based on a normalized local and a logarithmic global factor, are more suitable for LSA), and computes the *Singular Value Decomposition* (*SVD*) of the Term–Document matrix[4] $A$ (a kind of

[4]In the rest of this section, not to distract the reader from the main matter, most subsidiary information, such as recalls of mathematical concepts, comments and implementation hints, are provided as footnotes. First of all, it may be useful to recall some notations that will be used in the following:

- $A^T$, *transpose* of matrix $A$;
- $\text{diag}(a_1, \ldots, a_n)$, *diagonal* matrix $n \times n$;

factorial analysis typically exploited for estimating the rank of a matrix and in the canonical analysis of correlation). Given a matrix $A_{m\times n}$ in which (without loss of generality) $m > n$ and $\mathrm{rank}(A) = r$, $SVD(A)$ yields three matrices $U$, $W$ and $V$ such that:

$$A = U \times W \times V^T,$$

where:[5]

- $W_{r\times r} = \mathrm{diag}(\sigma_1, \dots, \sigma_r)$ such that[6] $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$; scalars $\sigma_i$ are the *singular values*, $\lambda_i = \sigma_i^2$ are the non-negative eigenvalues of matrix $A \times A^T$;
- $U_{m\times r}$ and $V_{n\times r}$ are orthogonal ($U \times U^T = V \times V^T = I$); the $r$ columns of $U$ (respectively, $V$) are the eigenvectors associated to the $r$ eigenvalues of $A \times A^T$ (respectively, $A^T \times A$), and refer to the left (respectively, right) singular values.

Another form to express the SVD of $A$ is the so-called *dyadic decomposition*:

$$A = \sum_{i=1}^{r} u_i \cdot \sigma_i \cdot v_i^T,$$

where the triples $(u_i, \sigma_i, v_i)$ are called *singular triples* and:

- $u_i$ is the $i$th column vector of $U$;
- $v_i^T$ is the $i$th row vector of $V$.

SVD is defined for any matrix and is unique, excluding degeneracies due to equal singular values.

In the case of the Term–Document matrix $A$:

- $r$ represents the number of concepts found in the collection;
- $U$ expresses the degree of similarity between terms and concepts;
- $W$ is the matrix of concepts;
- $V$ expresses the degree of similarity between documents and concepts.

---

- $I_n$, *identity* matrix of order $n$;
- $\mathrm{rank}(A)$, *rank* of matrix $A$;
- $A^{-1}$, *inverse* matrix of $A$.

Moreover, given a *square* matrix $M_{n\times n}$ (having the same number of rows and columns):

- A number $\lambda$ is an *eigenvalue* of $M$ if there exists a non-null vector $v$ such that $Mv = \lambda v$;
- Such a vector $v$ is an *eigenvector* of $M$ with respect to $\lambda$;
- $\sigma(M)$ denotes the set of eigenvalues of $M$.

*Singular values* are a generalization of the concept of eigenvalues to rectangular matrices.

[5] $U$ describes the entities associated to the rows as vectors of values of derived orthogonal coefficients; $V$ describes the entities associated to the columns as vectors of values of derived orthogonal coefficients, and $W$ contains scale values that allow going back to the original matrix.

[6] Some methods to carry out the SVD do not return the diagonal of $W$ ordered decreasingly. In such a case it must be sorted, and the columns of $U$ and the rows of $V^T$ must be permuted consequently.

The vector of a document corresponds to a weighted average of the vectors of the terms it contains, and *vice versa* the vector of a term consists of a weighted average of the vectors of the documents that contain that term. In some sense, the meaning of a word is the average meaning of all passages in which it appears, and, conversely, the meaning of a passage is the average meaning of the words it contains. Thus, a dimensional reduction of these components is carried out, selecting the largest (i.e., the first) $k < r$ diagonal elements of $W$, and setting the others at 0. This results in the elimination of the $(r - k)$ less significant columns of $U$ and $V$ (and in lighter memory requirements). A crucial issue is how to determine a number of concepts $k$ that removes just the insignificant semantic domains. Indeed, it has a direct impact on performance: small values of $k$ (compared to $n$), making LSA operate in a small-sized space of concepts, enhance the similarity among documents; on the contrary, for large values of $k$, the documents tend to be less similar to each other, which may mislead LSA in identifying the proper relationships among them.

Now, the truncation of the original matrix $A$ with parameter $k$ is the $(m \times n)$ matrix $A_k$, having rank $k$, defined as:

$$A_k = U_k \times W_k \times V_k^T,$$

where

- $U_k$ is the $m \times k$ matrix made up of the first $k$ columns from $U$;
- $V_k$ is the $n \times k$ matrix made up of the first $k$ columns from $V$ (and hence $V_k^T$ is made up of the first $k$ rows from $V^T$);
- $W_k = \text{diag}(\sigma_1, \ldots, \sigma_k)$.

Thus, applying SVD to $A$ allows deriving the semantic structure of the document collection by using matrices $U$, $V$ and $W$. Then, the original Term–Document matrix $A$ is approximated by $A_k$, using matrices $U_k$, $V_k$ and $W_k$ obtained by extracting the first $k < r$ singular triples.[7]

*Latent Semantic Indexing* (*LSI*) [10] is an application of LSA to automatic document indexing and retrieval. First of all, the query must be transformed into an $m$-dimensional pseudo-document as discussed in Sect. 7.1.3. Then, the query and all document vectors must undergo a dimensional reduction from $m$ (number of terms) to $k$ (number of concepts). For a vector $d$,

$$d_k = d^T \times U_k \times W_k^{-1},$$

where

- $d^T$ is the $(1 \times m)$ transpose of the $(m \times 1)$ vector $d$;
- $U_k$ is the $(m \times k)$ matrix, obtained during document indexing;
- $W_k^{-1}$ is the inverse matrix of $W_k(k \times k)$, obtained during the indexing step;[8]
- $d_k$ is the $(1 \times k)$ vector corresponding to $d$.

[7]This allows the conversion of $m$-dimensional vectors into $k$-dimensional ones, that is needed during the document retrieval phase (and by some index updating techniques for the LSI).

[8]$W_k$ being a diagonal matrix, its inverse consists just of the inverse of each element thereof. That is, for $i = 1, \ldots, k : W_k^{-1}(i,i) = \frac{1}{W_k(i,i)}$.

Now, both the query and all documents being represented in a $k$-dimensional space, the comparison techniques presented in Sect. 7.1.3 can be applied to assess their similarity, replacing the number of terms by the number of concepts ($m = k$) in the corresponding formulæ. The higher the resulting degree, the more relevant the corresponding document. In particular, Cosine Similarity can be used. Thus, a document can be returned even if it matches only a subset of the query terms.

The LSI proved to be much more effective than traditional techniques that exploit vector spaces, and even more after dimensionality reduction. Albeit reduced with respect to the original, its dimensional space preserves and enhances the semantic relations. Another advantage of the LSI is the availability of incremental methodologies to update it (i.e., add new terms and/or documents to an existing LSI index) without recomputing the whole index from scratch. Indeed, applying from scratch LSI at each update, taking into account both the old (already analyzed) and the new documents, would be computationally inefficient. Two techniques have been developed: Folding-In [4] is a much simpler alternative that uses the existing SVD to represent new information but yields poor-quality updated matrices, since the semantic correlation information contained in the new documents/terms is not exploited by the updated semantic space; SVD-Updating [22] represents a trade-off between the former and the recomputation from scratch.

**Concept Indexing** *Concept Indexing* (*CI*) [14, 15] is a dimensionality reduction technique developed to support document categorization and information retrieval.[9] It is computationally more efficient than LSI, for both time and space requirements. The new dimensionality $k$ refers to a set of concepts underlying the document collection. Each concept is determined as a group $S_i$ ($i = 1, \dots, k$) of similar docu-

[9]Concept Indexing is not to be mistaken for *Conceptual Indexing* [28], proposed in the same period, and aimed at tackling, in addition to the well-known problem of classical indexing techniques due to synonymy and polysemy spoiling the search results, the shortcoming due to the typical attitude to define hierarchical topic/concept categories, so that a document placed across two topics must be cast under either of the two, and is not found when searching under the other. Based on a generality relation that allows assessing whether a concept is *subsumed* by another, expressed by a set of basic facts (called *subsumption axioms*) and reported in a—possibly hand-made—dictionary or thesaurus (such as WordNet), it can determine whether a text is more general than another according to the generality relationships among their constituents. Thus, new texts can be mapped onto such a taxonomy, and the taxonomy itself can be progressively built and extended. Conceptual Indexing recognizes the conceptual structure of each passage in the *corpus* (i.e., the way in which its constituent elements are interlinked to produce its meaning). Four components are needed for this technique: A *concept extractor* that identifies terms and sentences to be indexed (and documents where they appear); a *concept assimilator* that analyzes the structure and meaning of a passage to determine where it should be placed in the conceptual taxonomy (and to which other concepts it should be linked); a *conceptual retrieval* system that exploits the conceptual taxonomy to associate the query to the indexed information; a *conceptual navigator* that allows the user to explore the conceptual taxonomy and to browse the concepts and their occurrences in the indexed material.

ments, and represented by its *centroid* (called a *concept vector*, of which $S_i$ is said to be the *supporting set*):

$$\mathbf{c}_i = \frac{1}{|S_i|} \sum_{\mathbf{d} \in S_i} \mathbf{d},$$

i.e., a group centroid is a single pseudo-document that summarizes the content of the actual documents belonging to that group, having as coordinates the averages of their corresponding coordinates. The larger the value of a coordinate, the more important that term to that concept (thus, those with largest value can be considered as *keywords* for that group/concept). The closeness between a document **d** and the centroid of the $i$th group can be used to categorize a document with respect to the concepts; e.g., using cosine similarity:

$$CosSim(\mathbf{d}, \mathbf{c}_i) = \frac{\mathbf{d} \cdot \mathbf{c}_i}{\|\mathbf{c}_i\|} = \frac{\frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} CosSim(\mathbf{d}, \mathbf{x})}{\sqrt{\frac{1}{|S_i|^2} \sum_{y,z \in S_i} CosSim(\mathbf{y}, \mathbf{z})}}.$$

In CI, document vectors are normalized to unit length ($\|\mathbf{d}\| = 1$). This clearly does not imply that centroids have unit length as well, but ensures that $\|\mathbf{c}_i\| \leq 1$.

CI identifies the groups of similar documents according to two approaches:[10]

**Unsupervised** automatically determines such classes when no prior knowledge other than the document collection to be indexed is available (as in LSI, but faster), and can point out small subsets of homogeneous documents;

**Supervised** exploits, if available, a set of documents whose class is known as a guidance to improve computation of the reduced space (and has been shown to be more effective than other traditional classification algorithms, such as $k$-NN).

The unsupervised approach was proposed first, and exploits a clustering technique having linear complexity in both time and space with respect to the size of the input information. Given the number $k$ of desired concepts, a variant of the $k$-means clustering technique is exploited to directly partition the set of documents (i.e., the columns in the Term–Document matrix) into $k$ groups. A different, progressive computation of the clusters (that allows controlling some interesting features such as a balanced cardinality among them) is obtained using *recursive bisection*, starting from two clusters and repeatedly splitting one cluster until $k$ are obtained. Specifically, the decision on which cluster to split is carried out according to the maximum *aggregate dissimilarity*:

$$|S_i|^2 \left(1 - \|\mathbf{c}_i\|^2\right)$$

among the current clusters (the squared length of a cluster centroid measures the average pairwise similarity between the documents in the cluster, so its complement expresses the pairwise dissimilarity).

[10]A quick reference to Machine Learning techniques, including $k$-NN, clustering and $k$-means, is provided in Appendix B; a more extensive treatise can be found in [20].

As an extension of the unsupervised approach, the supervised version exploits available knowledge about the class membership of sample documents to guide document clustering. Each cluster is biased to contain only samples from the same class. If $l$ different classes are indicated for the samples, the following possibilities are available with respect to the rank $k$ of the reduced dimensional space (i.e., the number of target concepts):

**if** $k = l$ after clustering, nothing needs to be done;
**else** $(k \neq l)$ the $l$ classes provided by the user are exploited to cluster the whole set of documents into $l$ clusters, and then the clusters are taken to $k$ by

- **if** $k > l$ repeatedly splitting some clusters according to a partitional clustering algorithm (this ensures that each final cluster will contain only samples from the same class);
- **if** $k < l$ repeatedly merging some clusters according to an agglomerative clustering algorithm (in this case samples from different classes will be joined in the same cluster, which is likely to spoil the classification performance).

Then, in both cases, the centroid vectors of the clusters are scaled (normalized) to unit length ($\|\mathbf{c}_i\| = 1$) to become the axes of the reduced space, and each document is expressed as a function thereof. Given the matrix $C_{k\times m}$ whose rows are the centroid vectors, an $m$-dimensional document vector $\mathbf{d}$ becomes $k$-dimensional by projection onto this space: $\mathbf{d}_k = \mathbf{d} \cdot C$. Accordingly, the whole collection $A$ is reduced to $A_k = A \times C$, and a query $\mathbf{q}$ is reduced to $\mathbf{q}_k = \mathbf{q} \cdot C$. The $i$th coordinate of a reduced document vector indicates how close that document is to the concept represented by the $i$th cluster. On such reduced representations, the cosine similarity can be applied as usual.

### 7.1.5 Image Retrieval

The possibility of digitally creating, manipulating and handling not only documents, but images as well, has made much easier their integration. This caused that, year after year, documents, even those produced in a non-professional way, were progressively enriched with graphic elements (not to mention the fact that, when coming from a digitization process, the document itself is an image). Thus, while informative content was initially concentrated in the document text, nowadays ignoring graphic elements brings the risk of neglecting very important information. Although managing them is quite computationally demanding, the current availability of more powerful hardware makes it possible to search even in pictures and graphics useful information for classification and indexing. Indeed, lots of subjects of interest and disciplines can get enormous advantage from the ability to manage image content: from medicine to security, to art, etc. Unfortunately, the state-of-the-art in such a field is still much behind that in text processing. *CBIR* (acronym for *Content-Based Image [Indexing and] Retrieval*) [7, 9] is the research area aimed at automatically extracting from images significant semantic and content-related information, possibly to be expressed in linguistic form in order to be retrieved also by text-based search engines.

Two main categories of approaches to image retrieval exist. *Text-based* ones transpose the search in a corresponding term-based search on keywords or textual descriptions of the image taken, e.g., from the file names, captions, surrounding text or explicitly provided by human experts (some search engines work in this way). This solution allows efficient retrieval thanks to the performance of current DBMSs, but indexing is difficult and unreliable. In particular, manual annotation of images by human experts is a disadvantageous approach because costly, time-consuming and very subjective (much more than text because the gap between the syntactic aspects and their interpretation is much wider). Hence, the motivation for research on automatic linguistic indexing of images, aimed at automatically producing textual descriptions of the concepts expressed by images. Although fundamental for CBIR and for Image Understanding in general, however, it is an extremely complex task if not limited to restricted domains.

This has strongly oriented the research towards *image-based* solutions, directly working on 'characteristic' information (also known as *signatures*) automatically extracted from the pixel level. Images are described by proper sets of (primitive or derived) features, and are compared based on suitable distances on such sets. Thus, given a repository of images and a query image, elements of the former can be ranked by decreasing similarity with respect to the latter. The features to be extracted from an image depend on the specific objectives and kind of data involved in the current task, but in general can be divided into *low-level* and *high-level* ones (the former often help to improve accuracy in the extraction of the latter). Some examples are:

**Color** (low-level) Expresses the distribution of colors in the image. A possible representation is the color histogram that, statistically, denotes the combined probability of intensities of the channels provided for by the adopted color space. RGB is the most widespread color space, but other color spaces, such as colorimetric ones, can be more suitable, because more uniform to human perception.

**Texture** (low-level) In general, *texture* can be defined as a part of an image showing sufficient perceptual regularity (some kind of *pattern*) to be considered as a single area, although made up of elements that are not chromatically uniform. Strategies for texture characterization can be divided into three categories: *statistical* ones exploit statistics concerning the gray levels of the pixels (typically based on a histogram or matrix of occurrences), but usually ignore their spatial organization; *structural* ones consider the texture as consisting of tiles organized on a surface according to rules defined by a grammar, but are difficult to be implemented; *spectral* ones are based on the properties of the Fourier spectrum, considering the periodicity of gray levels and identifying peaks of energy in it.

**Shape** (high-level) In many applications, image analysis can be guided quite effectively based only on the identification of shapes (often corresponding to noteworthy objects) and their positions. This happens, for instance, in domains (such as medical images) in which color and texture may appear similar but in fact characterize different objects. The main problems are due to noisy images and to the possible presence of several overlapping shapes in the image, that might be hard to distinguish.

As already pointed out (see Sect. 5.3), the feature extraction problem is strictly related to that of segmentation [5], intended as the partitioning of an image into significant areas.

## 7.2 Keyword Extraction

*Keywords* are words (or phrases) in a text that, due to their particular significance therein, can be deemed as 'keys' to access (i.e., to understand) the information it conveys (just like keys in cryptography allow decoding messages). The possibility of assigning keywords to a text relies on the assumption that its content can be characterized by a restricted set of terms expressing the relevant concepts. *Keyword Extraction* (*KE*) is the branch of Document Processing aimed at identifying such terms, based on a formal analysis of texts [13]. It has many practical applications: the content of a large *corpus* can be compactly expressed by just its keywords; a text can be automatically summarized by selecting only the passages containing keywords (likely to be more important), or classified into categories corresponding to the keywords; in IR, the relevance of a document can be associated to the density of some keywords, or documents containing the query terms as keywords can be ranked best. Analysis of keywords also allows studying intra- and inter-text consistency, and assessing the information density of a text or component according to the number of keywords it contains.

KE techniques usually rank the terms in the available texts according to an estimation (a *weight*) of their being keywords, and then select as keywords the first $k$ items in such a ranking. The number of items to be selected can be fixed (usually $k = 10$), or based on considerations on the specific values (e.g., all items having weight above a given threshold), or on their distribution in the ranking (e.g., a given percentage of items from the top, or the top items until an item is found whose weight difference from the previous item in the rank is above a given threshold). KE can take advantage from other text processing techniques. For instance, the set of candidate terms can be reduced using stopword removal and stemming. A further profitable restriction can be obtained by removing the content words that, although potentially expressing some meaning, uniformly occur in the given *corpus*, and hence are not useful to characterize the content of a single document with respect to the others (a typical problem in specialized collections that deal with a particular subject). However, such a reduction cannot be carried out *a priori* as the other ones. Also the logical structure of documents, if known (being explicit, as in HTML pages, or detected by Document Image Understanding techniques), can be leveraged. In a basic structure-based approach, considering $n$ types of logical blocks, each associated to a weight $I_j$ ($j = 1, \dots, n$) expressing its importance (e.g., in scientific papers, terms in the title, abstract or bibliographic references can be weighted more, and yet more those in an explicit keyword list provided by the authors, if any), the evaluation of a term $t$ being a keyword can be computed as

$$w(t) = \sum_{j=1}^{n} tf_j \cdot I_j,$$

where $tf_j$ is the frequency of $t$ in blocks of type $j$. If a term appears in several blocks, the weights coming from the different blocks are added to obtain an overall evaluation [21].

KE is clearly placed half-way between the lexical level (keywords are terms actually appearing in a document) and the semantic level (keywords should express the content of the document), and encompasses both perspectives. Although a text can be considered as a description of the underlying information content, it goes without saying that the meanings of its single constituent words cannot be ‘summed up’ or ‘averaged’ to obtain the overall meaning. They just represent clues, whose relevance depends on their mutual relationships as well; e.g., *concordance* (as the grammatical agreement between different but related parts of a sentence, that affects their form) is important to point out such relationships. Thus, a complete linguistic analysis of the text is desirable. Several KE techniques exist, suitable to different application domains that, according to their focusing just on the surface clues or on the underlying concepts, can be grouped into syntactic and semantic approaches.

Although each subject requires the use of specialized words, the setting based on purely terminological aspects might be too superficial. Being more strongly and straightforwardly related to the semantic domain, or context, discussed in the document, the concepts expressed in a text are a more reliable and faithful representation of its content. Hence, semantic approaches are very interesting candidates to a more effective solution of the KE task. Unfortunately, many terms, being polysemous, denote several concepts, which calls for some kind of word sense disambiguation (a certainly non-trivial task) before being able to identify the most relevant concepts underlying the given text(s). After assigning a weight to each concept in the document under processing, those with highest ranking can be selected, and the corresponding terms in the document (up to $k$) can be extracted as keywords.

However, most work in KE so far has focused on approaches based only on morphologic and, sometimes, syntactic (that, resolving ambiguity in the sentence structure, allow in principle more effective outcomes) aspects of texts. Considering the lexical level, they can exploit statistics on the (co-)occurrence of terms in the available set of documents as expressed by a vector space representation. For instance, the $k$ terms in a document having larger weight in the corresponding column vector of the weighted Term–Document matrix can be selected as keywords. Straightforward approaches to assess such weights are:

**Basic** Consists in simply counting the occurrences of the words in the document, according to the intuition that more important concepts are usually repeated more frequently than secondary ones.

**TF-IDF** Suggests choosing as keywords for a document the terms appearing more frequently in that document but not in the others, based on the same considerations as in Sect. 7.1.2 (the discriminative power of a term is diminished by its being uniformly spread through the collection). This represents a good tradeoff between complexity and effectiveness. For such an approach to be profitable, the document collection should preferably cover different subjects, in order to avoid that the IDF value (and hence the TF-IDF as well) of a potential keyword is close to 0, preventing it from being considered a key.

More complex approaches exist, some of which are recalled below.

**TF-ITP** Differently from the TF-IDF approach, the *TF-ITP* (*Term Frequency-Inverse Term Probability*) technique [2] disregards the frequency of terms in the whole specific collection. Instead, it considers, more in general, their inverse probability, i.e., the probability that a term appears in a document in the given language. If term $t$ appears $tf$ times in a document and its probability is $tp$, its TF-ITP weight is defined as

$$w(t) = \frac{tf}{tp}.$$

The term probability estimates are reported in a dictionary, built from a large document *corpus* (possibly on a reduced set of terms, after stopword removal, stemming and elimination of words appearing just once in the *corpus*). If a term that is not present in the dictionary is found in a document, it is assigned a default probability value (the smallest probability found in the dictionary is suggested in the original proposal).

**Naive Bayes** The *Naive Bayes* approach proposed in [27] casts KE as a classification problem: given a document, find what terms belong to class 'keywords' and what belong to class 'ordinary words'. A probabilistic setting is adopted, and the parameters for applying such a distinction to forthcoming texts are automatically learned from a set of training documents. For the model that decides whether a term is a key to be effective, significant features of the previously identified keywords must be exploited. Due to psychological attitudes reflected in the way in which humans construct natural language utterances, keywords are usually placed towards the beginning or the end of a text, sentence or paragraph. Thus, taking into account the word positions in the document in addition to their TF-IDF weights, the statistical evaluation function to identify key terms is more accurate and its outcome is more reliable. The probability that a term is a key given its TF-IDF value ($T$), its distance from the previous occurrence of the same word ($D$), and its position with respect to the entire text ($PT$) and to the sentence ($PS$) in which it occurs, is computed according to Bayes' Theorem, as:

$$\begin{aligned} p(key|T, D, PT, PS) &= \frac{p(T, D, PT, PS|key) \cdot p(key)}{p(T, D, PT, PS)} \\ &\quad \text{assuming that all feature values are independent} \\ &= \frac{p(T|key) \cdot p(D|key) \cdot p(PT|key) \cdot p(PS|key) \cdot p(key)}{p(T, D, PT, PS)}, \end{aligned}$$

where

- $p(key)$ is the prior probability of that term being a key;
- $p(T|key)$ is the probability of the term having TF-IDF value $T$ when it is a key;
- $p(D|key)$ is the probability of the term having distance $D$ from its previous occurrence when it is a key;
- $p(PT|key)$ is the probability of the term having relative position $PT$ from the beginning of the text when it is a key;
- $p(PS|key)$ is the probability of the term having relative position $PS$ from the beginning of the sentence when it is a key; and
- $p(T, D, PT, PS)$ is the probability of the term having TF-IDF value $T$, distance $D$ from its previous occurrence and distance $PT$ from the beginning of the text and $PS$ from the beginning of the sentence.

Assuming that $p(key)$ is equal for all terms, it can be ignored in the above formula, and the conditional probabilities on the right-hand-side become:

$$p(X|key) = \frac{p(X, key)}{p(key)} \sim p(X, key)$$

that can be estimated by the corresponding relative frequency on the training set, thus yielding:

$$p(key|T, D, PT, PS) = \frac{p(T, key) \cdot p(D, key) \cdot p(PT, key) \cdot p(PS, key)}{p(T, D, PT, PS)}.$$

The independence assumption trivially does not hold because the position of a term in a sentence affects its position in the paragraph and, in turn, this affects its position in the whole text; nevertheless, the Bayesian approach seems to work well even in these cases.

The list of weights computed as above for all candidate terms $t$ (intended as the probability that $t$ is a keyword)[11] is finally sorted by decreasing probability, and ties are resolved by considering the TF-IDF weight alone or, as a last resort, the word position in the text.

**Co-occurrence** When the set of documents is huge and a quick response is needed, or a training corpus on the subject under consideration is not available, techniques based on an analysis of the overall collection are inapplicable. A technique for KE from a single document containing $n$ terms, without additional knowledge, was proposed in [19], leveraging frequent co-occurrence of words (where co-occurrence is limited to terms *in the same sentence*). Specifically, the normalized relative frequency (i.e., the frequency of words in the document divided by the text length, so that their sum is 1) is used, as an approximation of term probability.

Preliminarily, a set $G$ of $k < n$ frequent words on which to focus in computing co-occurrence is selected (empirically, 30% of the initial words: $|G| \leq n \cdot 0.3$). Low-frequency terms (and hence their co-occurrences) are ignored because their precise

[11] Attention must be paid if $p(D|key)$, $p(PT|key)$ or $p(PS|key)$ is zero, in which case the probability would be 0 as well, or if $p(T, D, PT, PS) = 0$, in which case a division by zero would occur.

probability of occurrence would be hard to estimate. Since $G$ acts as a kind of base according to which expressing the importance of co-occurrences, it should be orthogonal. In practice, terms in $G$ are hardly orthogonal because if two terms in $G$ often occur together, a term frequently co-occurring with any of them would consequently often co-occur with the other as well. To fix this, instead of just selecting orthogonal terms (which would yield a sparse matrix), terms in $G$ can be clustered, and clusters can be used instead of single terms (balanced clusters yield better results). Two techniques have shown to perform sensible term grouping according to their use, role and meaning:

**Similarity-based** clustering based on similar distribution of co-occurrence among terms, evaluated statistically by the *Jensen–Shannon divergence measure*:

$$J\left(t',t''\right)=\frac{1}{2}\left[D\left(t'\left\|\frac{t'+t''}{2}\right.\right)+D\left(t''\left\|\frac{t'+t''}{2}\right.\right)\right],$$

where $D(p \parallel q)$ denotes the KL divergence.

**Pairwise** clustering based on frequent co-occurrence, using *mutual information* to measure the degree of relevance:

$$I\left(t',t''\right)=\log_2\frac{p(t',t'')}{p(t')\cdot p(t'')}.$$

A pair of terms $(t',t'')$ in $G$ is clustered if $J(t',t'')>0.95\cdot\log 2$ or $I(t',t'')>\log 2$ (again, thresholds are determined empirically). Let $\mathcal{C}$ be the resulting set of clusters (if clustering is not applied, $\mathcal{C}$ consists of just the singletons of terms in $G$).

While pure term frequency alone can be misleading and insufficient to properly point out significant keywords, the distribution of co-occurrence between terms in $G$ and the other terms is a better clue to assess the degree of relevance of terms.[12] Indeed, terms that appear independently of those in $G$ should have a co-occurrence distribution that is similar to the unconditional one. Conversely, those that are actually (lexically, semantically or otherwise) related to terms in $G$ should exhibit a larger co-occurrence value, and a biased distribution, and hence might be relevant. The typical statistical test used for evaluating the significance of biases between observed and expected frequencies is $\chi^2$:

$$\chi^2(t)=\sum_{C\in\mathcal{C}}\frac{(f(t,C)-n_{\{t\}}p_C)^2}{n_{\{t\}}p_C},$$

where

- $f(t,X)$ is the number of co-occurrences of $t$ with any term in $X$;
- $n_X$ is the total number of terms in sentences in which any term in $X$ appears (i.e., the number of terms co-occurring with any term in $X$)—to take into account that

[12]A matrix $M_{n\times k}$ in which $m_{tg}$ reports in how many sentences term $t$ in the text co-occurs with term $g\in G$ is useful to represent such information. Having $k<n$, $M$ is not square, and no diagonal can be considered.

terms appearing in long sentences are more likely to co-occur with many terms than those appearing in short ones; in a simplified version, $n_{\{t\}} = f(t, G)$;
- $p_C = n_C / n$ is the unconditional (expected) probability of a $C \in \mathcal{C}$;

and hence $n_{\{t\}} p_C$ is the expected co-occurrence frequency of a term $t$ with terms in cluster $C$ and the actual frequency $f(t, C)$ is considered as a sample value. Thus, if $\chi^2(t) > \chi^2_\alpha$, the null hypothesis $H_0 =$ "occurrence of terms in $G$ is independent from occurrence of $t$" is rejected (indicating a bias) with significance level $\alpha$. The weight of each term as a keyword is then measured as the robustness of the bias values:

$$\begin{aligned} w(t) &= \chi^2(t) - \max_{C \in \mathcal{C}} \left( \frac{(f(t, C) - n_{\{t\}} p_C)^2}{n_{\{t\}} p_C} \right) \\ &= \sum_{C \in \mathcal{C}} \frac{(f(t, C) - n_{\{t\}} p_C)^2}{n_{\{t\}} p_C} - \max_{C \in \mathcal{C}} \left( \frac{(f(t, C) - n_{\{t\}} p_C)^2}{n_{\{t\}} p_C} \right) \end{aligned}$$

subtracting the maximum to avoid that adjuncts of a $g \in G$ (e.g., 'modern' for $g =$ 'art') get large $\chi^2$ values just because of their co-occurrence, although they are not important by themselves.

## 7.3 Text Categorization

*Text Categorization* (*TC*) is the task of assigning a given document the *subject* (or subjects) it refers to. In some sense, it can be considered the content analysis counterpart of document image classification in layout analysis (indeed, it is also known as *Text Classification*). The subjects, called *categories*, are represented as simple uninterpreted labels to which no additional form of knowledge or meaning is attached, and are usually pre-defined. As a classification task, it can be formally represented by a function $\Phi$ (called the *oracle*) that, given a document and a category, says whether their association is correct (i.e., whether that document belongs to that category), and hence represents the way in which documents would be classified by an expert. More precisely,

$$\Phi : D \times C \to \{T, F\},$$

where $D$ is the set of documents, $C = \{c_1, \dots, c_n\}$ is the set of categories, and the codomain consists of the truth values **True** ($T$) and **False** ($F$). Two strategies can be adopted:

**Single-label** each document is assigned to exactly one category;
**Multi-label** each document can be assigned to zero or many known categories.

The latter often results in a ranking that expresses, for each category, to what degree of confidence it can be associated to the given document. The former applies only if the given categories are mutually independent. For instance, in the binary case, any document must be assigned either to a category $c$ or to its complement $\overline{c}$, and the oracle becomes $\Phi : D \to \{T, F\}$. Applying a binary classification to each class $c_i$, a multi-label approach can be reduced to a set of single-label ones. Thus, single-label methods have received more attention.

Applications of TC range from automatic indexing (for retrieving documents according to their category) to document organization (based on the category of membership), information filtering (to determine if a document is more or less interesting with respect to the user's needs), Word Sense Disambiguation (to find and point out the presence in a document of an ambiguous term because of synonymy or polysemy) and spam filtering (to automatically classify incoming e-mail messages, sort them in folders and eliminate undesired ones).

Since the oracle is usually not available, and manual categorization is often infeasible due to the huge number of documents and to the typically subjective evaluations of humans,[13] a significant amount of research has been spent in automatically learning an approximation of the oracle on the grounds of a (possibly very large) set of documents whose categories are known. One such document $\overline{d}$ is called a *positive example* for $c$ if $\Phi(\overline{d}, c) = T$, or a *negative example* for $c$ if $\Phi(\overline{d}, c) = F$. The problem can be formally expressed as follows:

- Given a pre-defined set of categories $C$, a set of documents $D$, and the truth values **True** $(T)$ and **False** $(F)$,
- Find a function $\Phi' : D \times C \to \{T, F\}$, called a *classifier*, that approximates the target function $\Phi : D \times C \to \{T, F\}$.

A binary classifier for $c_i$ is, thus, a function $\Phi'_i : D \to \{T, F\}$ that approximates the target function $\Phi_i : D \to \{T, F\}$.

Any of the several Machine Learning approaches proposed in the literature [20] can be applied, some of which are briefly recalled in Appendix B. An extensive survey of Machine Learning approaches to TC can be found in [25]. Documents and sentences can be represented according to their syntactic structure, or as simple sets of terms associated with weights that denote their relevance to the document itself (usually based on their presence or frequency in the document). In the latter case, the same representation techniques (e.g., stopword removal, stemming and BoW) and weighting schemes (e.g., TF-IDF) as those used for document indexing can be borrowed from IR, but when the number of terms is very large a dimensionality reduction that focuses on a reduced set of terms is crucial for efficiency (usually selecting terms with high frequency in the collection). Thus, terms act as features to describe the observations (texts) to be used for learning or classification.

Outside of the ML area, an adaptation of the *Rocchio* approach, borrowed from IR, can be applied. It models each class $c_i$ (for $i = 1, \dots, n$) as a set of weights $(w_1^i, \dots, w_m^i)$, with

$$w_k^i = \beta \cdot \sum_{d_j \in N_i} \frac{w_{kj}}{|N_i|} - \gamma \cdot \sum_{d_j \in P_i} \frac{w_{kj}}{|P_i|}$$

[13]Some studies revealed that the overlapping ratio between classifications provided by different experts with the same skills and experience is just around 30%.

for $k = 1, \dots, m$, where $P_i$ is the set of positive examples, $N_i$ is the set of negative examples (i.e., the closeness to $P_i$ is rewarded, while the closeness to $N_i$ is penalized), and factors $\beta$ and $\gamma$ weight their relative importance (setting $\gamma = 0$ would just result in the centroid of $P$). This technique is very efficient, but its effectiveness needs to be improved by different kinds of refinements (e.g., restricting $N_i$ to near-positive examples only, that are more informative to guide the learning step).

**A Semantic Approach Based on WordNet Domains** A well-founded and ready-to-use reference taxonomy of subject categories is provided by WordNet Domains (see Sect. 6.1). This consideration led to the semantic-based Text Categorization technique proposed in [1]. Indeed, in order to exploit WordNet Domains, a preliminary, explicit mapping of the terms appearing in the text to the underlying concepts as expressed in WordNet is necessary. A very important aspect of this approach is its working on just the document to be categorized, without any need for a training set because all the background knowledge to perform the task comes from the WordNet and WordNet Domains resources. In fact, it can be exploited also in more specific contexts, as long as a thesaurus corresponding to WordNet and a taxonomy corresponding to WordNet Domains are available. In any case, an integration of such specialized resources with WordNet and WordNet Domains is advisable, in order to reuse the knowledge conveyed by the latter and to keep a connection with the general context.

A first problem is that a term typically corresponds to several possible concepts in WordNet, due to polysemy. A preliminary parsing of the sentences in the text may help in reducing this ambiguity, allowing to select for each term a limited subset of possible syntactic categories to focus on, which in turn allows the overall procedure to be more focused. However, this does not completely solve the problem since the resulting set seldom reduces to a singleton. Instead of performing explicit word sense disambiguation, the relevance and likelihood for the various senses (i.e., synsets) of each term exploited in the document is determined using a peculiar *density function* defined levelwise, bottom-up by linguistic aggregates, as follows:

- Given a term $t$, and the set $s(t)$ of the corresponding synsets in WordNet, the weight of a synset $s$ with respect to $t$ is

$$w(s,t) = \frac{1}{|s(t)|}$$

  if $s \in s(t)$ (i.e., the relevance of synsets is uniformly distributed), or 0 otherwise. To manage cases of words, or concepts, or both, that are not included in WordNet, the term itself is always added as an extra concept identifier to $s(t)$, which also avoids a possible division by zero (albeit introducing some noise).
- Given a sentence $S$ made up of terms $\{t_1, \dots, t_n\}$, with $s(S) = \bigcup_{j=1}^{n} s(t_j)$ the set of synsets related to terms appearing in $S$, the weight of a synset $s$ with respect to $S$ is defined as

$$w(s,S) = \sum_{i=1}^{n} \frac{w(s,t_i)}{|S|} = \sum_{i=1}^{n} \frac{w(s,t_i)}{n}$$

if $s \in s(S)$, or 0 otherwise. That is, it consists of the weight of all synsets included in the sentence (normalized by the number of terms in the sentence).

- Given a document $D$ made up of sentences $\{S_1, \dots, S_n\}$, with $s(D) = \bigcup_{j=1}^{n} s(S_j)$ the set of synsets related to terms appearing in $D$, the weight of a synset $s$ with respect to $D$ is defined as

$$w(s, D) = \sum_{j=1}^{n} w(s, S_j) \cdot w(S_j, D),$$

where coefficients $w(S_j, D)$, for $j = 1, \dots, n$, allow weighting differently each sentence in the document. Such factors can be determined starting from a set of values $w_j > 0$, each expressing the importance of sentence $S_j$, normalized so that their sum on all sentences in $D$ is 1, obtaining:

$$w(S_j, D) = \frac{w_j}{\sum_{k=1}^{n} w_k}.$$

A strategy to assign the basic importance values is considering the document structure, if available (e.g., a title might get a larger weight than that of a sentence in a paragraph), but any other rationale that reflects the user needs or the domain peculiarities can be exploited.

The weight of a category is computed by summing the weights of all synsets referred to that category, which is expected to be more significant than the pure number of their occurrences. Then, after ranking these cumulative weights, the proper categories for the document can be determined as those exceeding a given threshold. In fact, an apparent difference in value usually occurs between the first few items in the ranking and the subsequent ones.

## 7.4 Information Extraction

In some sense, *Information Extraction* (*IE*) is to Document Image Understanding as TC is to Document Image Classification. In its typical setting, IE aims at identifying occurrences of a particular class of events or relationships in a free text, and at extracting their relevant features in a pre-defined structured representation (the *template*). The template fields are called *slots*, and the information items extracted from the text to fill such slots are called their *instances*. Regarding the text as a sequence of *tokens*, slot instances usually consist of sub-sequences thereof, called *fragments*. Two approaches can be considered:

**Single-slot** is the easiest. Slots are considered as independent of each other, so that no global considerations are needed to fill them;

**Multi-slot** assumes that different slots are interrelated, and hence requires the ability to find and interpret suitable relationships among (sometimes very distant) text components (e.g., in price listings, the ‘price’ slot instance must refer exactly to the ‘item’ slot one).

*Text Mining* is a subtask of IE, aimed at the acquisition of new knowledge elements on a given domain. Due to the complexity of the task, IE systems are usually dedicated to very specialized domains (e.g., particular kinds of news or advertisements in newspapers).

Although IE is a neatly different task than IR (the latter aiming at the identification of interesting documents in a collection, the former at the identification of relevant information items in a document), it exploits the same evaluation measures (*precision*, *recall* and *fallout*), computed on a slot-basis. Differently from IR, however, a slot can be wrong because *spurious* (when filled although the correct information is not actually in the text) or *incorrect* (when the correct information is elsewhere in the text). Moreover, several (combinations of) situations in slot filling make the outcome of these measures sometimes unreliable.

Any IE system must include some form of the following functionality (listed in a typical, although not mandatory, order) [8]:

**Pre-processing** Portions of the document that are likely to contain the desired information are extracted (referred to as *text zoning*) and split into sentences (identification of sentence boundaries might not be trivial), possibly filtering out the less promising ones in order to improve efficiency of the next steps.

**NLP** Morphologic and lexical analysis is useful to normalize and categorize terms/tokens. In particular, the ability to identify proper names (a complex task that cannot be reduced to just selecting capitalized words) is crucial in IE because they often correspond to slot instances. However, syntactic information is needed to capture the textual relationships that underlie slot instances identification. IE usually focuses on small or medium-size grammatical aggregates (e.g., noun phrases and verb phrases) that can be reliably identified by current parsers. Further aggregation in higher-level ones is carried out only when a significant degree of confidence is guaranteed because wrong outcomes would introduce additional misleading noise.

**Semantic functionality** WSD (obtained using a *lexical disambiguator*) and coreference resolution (including *anaphora* and different forms referring to the same sense) allow denoting each entity with a single identifier and avoiding the recognition of only apparently different objects in different templates. Discourse analysis and inferences on the text content might be required for this [12].

**Template Generation** Produces the expected result in the desired form (filling template slots).

Candidate slot instances are located in the text using domain-specific patterns or rules, often manually provided by experts. Since this manual approach is difficult, costly and imprecise, the application of Machine Learning techniques is desirable. In the simplest approach, *rote* learning [11], a dictionary of specific sequences of tokens to be exactly (case insensitive) searched for is associated to each slot. Although trivial, this approach can deal with several real cases. If a slot-filling rule selects $n$ fragments in the training text(s), of which $c$ are correct, the probability that it provides a correct (respectively, wrong) answer is $\frac{c}{n}$ (respectively, $1 - \frac{c}{n}$), but is usually estimated using (some variation of) the *Laplacian*, as discussed in the following.

If, given a large number of trials $n$, an event $e$ occurred $n_e$ times, its theoretical probability $\overline{p}$ can be approximated by the *relative frequency* $\hat{p} = n_e/n$. Since this formula might yield odd results when either $n_e$ or $n$ are zero (the latter case raising a division by zero), safer ways for estimating $\overline{p}$ are based on the *Laplacian* law of succession that proved to have a good correlation to the actual probability [6]:

$$\frac{n_e + 1}{n + 2},$$

or on $m$-estimates [20],

$$\frac{n_e + mp}{n + m},$$

where $p$ is a prior estimate of $\overline{p}$ (assumed, when lacking further information, to be uniform for all possible $k$ outcomes of event $e$: $p = 1/k$) and $m$ is a constant (called the *equivalent sample size* because it can be interpreted as augmenting the $n$ actual observations by an additional $m$ virtual samples distributed according to $p$) that combines $\hat{p}$ and $p$ determining how much influence the observed data have with respect to $p$. If $m = 0$, the $m$-estimate is equivalent to the simple relative frequency.

**WHISK** *WHISK* [26] is a system that automatically induces rules for multi-slot IE in any kind of text. In structured text it exploits the available structure (e.g., tags in HTML documents). Free text (e.g., newspaper articles) is pre-processed and annotated with grammatical (e.g., **SUBJ{**...**}** delimits the subject of a sentence, @**Passive ...** denotes a verb in passive form, @*stem* denotes the stem of a verb, etc.) and possibly semantic (e.g., @**PN[**...**]PN** delimits a person name, etc.) information. Semi-structured text, intended as an intermediate form between these two extremes, typically characterized by ungrammatical and/or telegraphic style (e.g., readers' advertisements on newspapers, or technical logs and records), is annotated with features such as @**start** (to denote the beginning of the text), @**blankline**, @**newline**.

Rules learned by WHISK exploit regular expressions that capture relationships between different facts in the text, based on the following formalism:

* skips all elements in the text until the first occurrence of the next element in the pattern is found (to limit its applicability);

'...' a specific string (a *literal*) to be found (case insensitive) in the text (might include syntactic or semantic annotations);

(...) a sequence to be extracted as a slot instance (called an *extractor*);

***F** similar to * but requires the next piece of the regular expression to be in the same grammatical field as the previous one.

Since patterns including only literals would be too rigid for handling natural language, more flexibility is obtained by using *semantic classes*, placeholders that represent several specific literals, each pre-defined (e.g., *Digit* and *Number*) or defined separately as a disjunction of literals in the form $C = (l_1|\dots|l_n)$.

Rules are made up of three components:

**ID::** a numeric unique identifier of the rule;

**Pattern::** a regular expression to be matched on the given text, that defines the context of relevant fragments and exactly delimits their extraction boundaries, this way avoiding the need for any kind of post-processing;

**Output::** $\mathbf{N}$ $\{\mathbf{S}_1\ \$V_1\} \ldots \{\mathbf{S}_n\ \$V_n\}$ the structure of a template $\mathbf{N}$, where $V_i$ is the instance of slot $\mathbf{S}_i$, and there is a one-to-one correspondence between the $i$th slot in Output and the $i$th extractor in Pattern.

After a rule succeeds, the template instance is stored and text scanning is continued from the next character. When it fails, the slot instances found thus far are retained and scanning restarts after that point on the remaining input.

WHISK adopts a supervised setting. Given a new tagged instance (the *seed*) and a training set $T$, rules are induced top-down, starting from the empty rule *(*)*...*(*)* having as many extractors as the template slots. It is the most general rule that covers everything but is always wrong (because the * in the first slot matches everything), and is progressively specialized by anchoring slots to text elements (which increases the set of correctly covered instances) one by one from left to right. An 'element' can be a specific literal (any word, number, punctuation or structure tag) taken by the seed or a semantic class including it, and the addition of several elements might be needed to cover the seed. For each slot, two alternative anchorings are tried, one adding elements inside the slot extraction boundaries (...) and the other adding elements both before and after it, and the one with greatest coverage of the partially anchored slots on $T$ is carried on. Since the anchored rule, although correct on the seed, might be wrong on previous training instances, further elements are added to specialize it until it makes no errors on $T$ or no improvement is obtained in the Laplacian estimate of the rule error, $L = \frac{n_e+1}{n+1}$ (a slight variation of the original one). The same estimate is used to select rule specializations, both when anchoring a slot and during consistency enforcement with respect to $T$. If the Laplacian error of the rule before specialization exceeds a pre-defined tolerance it is discarded, otherwise all elements in the seed are tried in turn, and the one that yields the rule with minimum Laplacian error is retained (ties are resolved heuristically, preferring semantic classes to words, and syntactic tags to literals, to have more general rules). For domains with no tags, terms near extraction boundaries are preferred (possibly within a window of given size). Using this strategy, a better rule that might be obtained by two sub-optimal specializations instead of the single best one cannot be reached.

To reduce human intervention, WHISK adopts an interactive approach that asks for the expert assessment only on sample cases deemed as more informative to guide the rule learning activity. Given a set of untagged instances $U$, it repeatedly selects a set of still untagged instances and asks the user to completely tag them. The initial set is chosen at random, while instances to be included in each next set are drawn from the following categories (with user-defined relative proportions): instances covered by an existing rule (for checking purposes, and hence to increase precision), probable near-misses that would be covered by a minimal generalization of an existing rule (to increase recall), and instances not covered by any rule (to explore new portions of the solution space). After tagging, these instances are removed from $U$, added to the set of training instances $T$ and exploited to test the current rules (those contradicted by a new tagged instance are removed as inconsistent) and to learn new ones (when an instance–tag pair is found for which the tag is not extracted by any rule, it becomes a seed to grow a new rule according to the above strategy). To avoid overfitting, pre-pruning (stopping as soon as the percentage of errors on the training set falls below a given threshold) and post-pruning (dropping rules with Laplacian expected error greater than a threshold) strategies are applied.

**A Multistrategy Approach** The approach to single-slot IE proposed in [11] aims at "finding the best unbroken fragment of text to fill a given slot in the answer template", assuming that each text includes a single instance for the slot. The problem is cast as a classification task in which, given a fragment of text (and possibly its context), the objective is saying (or estimating) whether it properly fits the slot or not. It aims at improving flexibility and effectiveness by exploiting different strategies and representations to capture different aspects of the problem, and combining several learning components.

To compute the probability that a sequence $H_{i,k}$ of $k$ consecutive tokens starting at token $i$ is a suitable filler for a slot $s$, given training data $D$, the probabilities of the instance having a given length and position, containing specific terms and having a certain structure are considered:

$$p(H_{i,k}|D) = p(\text{position})\,p(\text{length})\,p(\text{terms})\,p(\text{structure}).$$

For estimating $p(\text{position} = i)$, a histogram is computed by considering bins having a fixed width; then, bin midpoint positions are associated to the corresponding histogram value, and other positions to the linear interpolation of the two closest histogram values. $p(\text{length} = k)$ is estimated as the ratio of slot instances having length $k$ over all slot instances. Assuming that position and length are independent, the first two factors yield prior expectations for $H_{i,k}$:

$$p(\text{position} = i)\,p(\text{length} = k) = p(H_{i,k}).$$

$p(\text{terms})$ is evaluated using evidence in the training data $D$ that specific tokens occur before ($B$), in ($I$) and after ($A$) a slot instance, limited to a pre-specified context window of size $w$:

$$p(D|H_{i,k}) = p(B|H_{i,k}) \cdot p(I|H_{i,k}) \cdot p(A|H_{i,k}),$$

where single probabilities are computed using $m$-estimates, and exploiting as a denominator the number of training documents in which a term appears (as in IDF), rather than the number of training instances (as probability theory would require). Altogether, these factors correspond to Bayes' formula:

$$p(H_{i,k}|D) = \frac{p(D|H_{i,k})\,p(H_{i,k})}{p(D)}$$

and associate an estimate to any fragment; to allow rejection of all fragments in a document when a field is not instantiated in it, a minimum confidence threshold $\gamma \cdot \mu$ is set, where $\gamma$ is defined by the user and $\mu$ is the log-probability of the smallest field instance confidence. In addition to suffering from low-frequency tokens, this formula ignores the contribution of syntax. To fix this, the additional factor $p$(structure) is included, expressing the probability that the fragment has the proper structure. A probabilistic grammar of the slot instances is learned,[14] and the probability that the fragment under consideration belongs to such a grammar is considered.

The grammar alphabet, instead of characters, consists of high-level symbols, that can be specific tokens (e.g., 'Prof', '.', etc.), or abstractions thereof according to some feature (e.g., 'capitalized', 'upper_case', 'lower_case', '$n$_digit_long', '$n$_character_long', 'punctuation', etc.). The original text can be turned into a sequence of such symbols using *alphabet transducers* in the form of *decision lists* (i.e., sequences of pattern/emission rules). To better fit the training data, such transducers can be learned as well for each field, assuming that each token can be generalized independently of its context, using a greedy covering approach: iteratively, the set of features to be taken into account is compared against the training instance, and a rule corresponding to the most promising symbol according to some strategy (even spread, information gain, $m$-estimates) is appended to the list, removing the training instances that it covers, until few training instances remain uncovered or a fixed decision list length is reached.

Relations among tokens or symbols, very important in IE, can be taken into account using a symbolic relational inductive learner. *SRV* is a special-purpose (for IE) rule learner that exploits instances of the target field as positive examples, while negative examples are limited to fragments having length between the smallest and the largest positive example of the target field (to reduce the candidates). Examples are described according to token features, propositional—such as length, type (e.g., 'numeric'), typography (e.g., 'capitalized'), PoS (e.g., 'noun'), and lexical content (e.g., person_name')—or relational ones—such as adjacent tokens (e.g., 'previous' and 'next') or grammatical links (e.g., as provided by the link grammars described in Sect. 6.2.7). Rules learned by SRV may include five predicates:

**length**$(R, N)$ where $R$ is a relational operator and $N$ is an integer;

**some**$(X, P, F, V)$ the token linked by a path $P$ of relational constraints to a token $X$ in the fragment must have value $V$ for feature $F$ (if $P$ is empty, the test applies to $X$);

**every**$(F, V)$ all tokens in the fragment must have value $V$ for feature $F$;

**position**$(X, F, R, N)$ feature $F$ of a token $X$ bound by a **some** literal in the same rule must fulfill relation $R$ for value $N$;

**relpos**$(V_1, V_2, R, N)$ token $V_1$ precedes token $V_2$ at a distance that fulfills relation $R$ for value $N$.

Paths are initially empty or contain only one element, and can be progressively extended (if useful) adding further relational features.

[14] In the case of regular grammars, the corresponding Finite State Automata are learned by starting from the maximally specific case (called the *canonical acceptor*) and progressively merging states according to considerations that depend on the specific algorithm.

SRV adopts a top-down, greedy strategy: starting from an empty rule that covers everything, antecedent literals are added until no negative example is covered. The choice of the literal to be added is guided by the *information gain* metric: given a set of examples $S$, its information depends on the balance of positive examples $P(S)$ and negative ones $N(S)$ in it:

$$I(S) = -\frac{\log_2 P(S)}{P(S) + N(S)}$$

and hence the gain obtained by adding a literal to a rule that covers a set $S$ of examples, resulting in a new rule covering $S' \subseteq S$, is given by $G(A) = P(S')(I(S) - I(S'))$. To reduce the computational effort, literals can be preliminarily associated to the number of positive and negative examples they cover. A rule is committed as soon as it does not cover any negative example (to avoid wasting time in cases when this cannot be obtained, literals covering few examples are discarded *a priori*). The final classifier is obtained performing a $k$-fold cross-validation, and merging the rules learned in each fold. Rules in such a classifier are treated as independent predictors, each having a confidence given as usual by $p = c/n$ (see the introductory paragraphs to this section). Then, given a fragment and the set $P$ of confidence values for all matching rules, the combined confidence score is computed as $\overline{P} = 1 - \prod_{p \in P}(1 - P)$.

## 7.5 The Semantic Web

The techniques presented in previous sections provide precious information to be added to the documents as *metadata* (literally, from ancient Greek, 'data about the data' contained in the document). Keyword Extraction provides core terms in the document, Text Categorization indicates its subject and area of interest, Information Extraction highlights features of important events expressed therein. With respect to metadata associated to document image understanding, concerning extraction of relevant text from the document, here an added value is represented by the relationships to the underlying informative content. In addition to the potential improvement of Information Retrieval effectiveness, since documents can be searched at the semantic level rather than just at the syntactic or lexical one, this kind of knowledge is becoming more and more important in the context of the *Semantic Web*, a new perspective on the WWW in which computer machines and programs are supposed to understand each other when cooperating and exchanging data and services. An excerpt of the original paper can better explain these concepts [3]:

> [...] Most of the Web's content today is designed for humans to read, not for computer programs to manipulate meaningfully. [...] The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users. [...] The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [...] The essential property of the World Wide Web is its universality. [...] Information varies along many axes. One of these is the difference between information produced primarily

for human consumption and that produced mainly for machines. [. . . ] The Semantic Web aims to make up for this. [. . . ]

After all, this is the ultimate objective at which the whole field of automatic document processing is aimed. Obviously, true autonomous semantic capabilities are still to come for the present computers, and the performance of human experts in providing this kind of high-level information is still out of reach. Nevertheless, proper exploitation of suitable tools and background knowledge can ensure satisfactory results for many purposes, at least in restricted environments, and intense research in all fields of Digital Document Processing allows good expectations for the future.

## References

1. Addis, A., Angioni, M., Armano, G., Demontis, R., Tuveri, F., Vargiu, E.: A novel semantic approach to create document collections. In: Proceedings of the IADIS International Conference—Intelligent Systems and Agents (2008)
2. Begeja, L., Renger, B., Saraclar, M.: A system for searching and browsing spoken communications. In: HLT/NAACL Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval, pp. 1–8 (2004)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
4. Berry, M.W., Dumais, S.T., O'Brien, G.W.: Using linear algebra for intelligent information retrieval. SIAM Review **37**(4), 573–595 (1995)
5. Boulgouris, N.V., Kompatsiaris, I., Mezaris, V., Simitopoulos, D., Strintzis, M.G.: Segmentation and content-based watermarking for color image and image region indexing and retrieval. EURASIP Journal on Applied Signal Processing **1**, 418–431 (2002)
6. Cestnik, B.: Estimating probabilities: A crucial task in machine learning. In: Proceedings of the 9th European Conference on Machine Learning (ECAI), pp. 147–149 (1990)
7. Chen, Y., Li, J., Wang, J.Z.: Machine Learning and Statistical Modeling Approaches to Image Retrieval. Kluwer, Amsterdam (2004)
8. Cowie, J., Wilks, Y.: Information Extraction. In: Dale, R., Moisl, H., Somers, H. (eds.) Handbook of Natural Language Processing, pp. 241–260. Marcel Dekker, New York (2000)
9. Deb, S.: Multimedia Systems and Content-Based Image Retrieval. IGI Publishing (2003)
10. Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society of Information Science **41**(6), 391–407 (1990)
11. Freitag, D.: Machine learning for information extraction in informal domains. Machine Learning **39**, 169–202 (2000)
12. Grishman, R.: Information extraction: Techniques and challenges. In: International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology. Lecture Notes in Computer Science, vol. 1299, pp. 10–27. Springer, Berlin (1997)
13. Hunyadi, L.: Keyword extraction: Aims and ways today and tomorrow. In: Proceedings of the Keyword Project: Unlocking Content through Computational Linguistics (2001)
14. Karypis, G., Han, E.H.S.: Concept indexing: A fast dimensionality reduction algorithm with applications to document retrieval & categorization. Tech. rep. TR 00-016, University of Minnesota—Department of Computer Science and Engineering (2000)
15. Karypis, G., Han, E.H.S.: Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In: Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM), pp. 12–19 (2000)
16. Landauer, T.K., Dumais, S.T.: A solution to Plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge. Psychological Review **104**, 111–140 (1997)

17. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Processes **25**, 259–284 (1998)
18. Manning, C.D., Raghavan, P., Schutze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
19. Matsuo, Y., Ishizuka, M.: Keyword extraction from a single document using word co-occurrence statistical information. International Journal on Artificial Intelligence Tools **13**(1), 157–169 (2004)
20. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
21. O Kit Hong, F., Bink-Aghai, R.P.: A Web prefetching model based on content analysis (2000)
22. O'Brien, G.W.: Information management tools for updating an SVD-encoded indexing scheme. Tech. rep. CS-94-258, University of Tennessee, Knoxville (1994)
23. Rocchio, J.: Relevance feedback in information retrieval. In: Salton, G. (ed.) The SMART Retrieval System, pp. 313–323. Prentice-Hall, New York (1971)
24. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Communications of the ACM **18**(11), 613–620 (1975)
25. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys **34**(1), 1–47 (2002)
26. Soderland, S.: Learning information extraction rules for semi-structured and free text. Machine Learning **34**, 233–272 (1999)
27. Uzun, Y.: Keyword extraction using Naive Bayes (2005)
28. Woods, W.A.: Conceptual indexing: A better way to organize knowledge. SML Technical Report Series. Sun Microsystems (1997)

# Appendix A
# A Case Study: DOMINUS

Quo vadis, Domine?

*DOMINUS* (DOcument Management INtelligent Universal System) [87] is a prototype framework for digital library management that pervasively embeds document-related Artificial Intelligence solutions, and in particular Machine Learning and knowledge-based approaches, throughout the whole spectrum of activities. It is of interest here because most techniques and algorithms presented in this book have found their place and position in it. Hence, it shows how several different methods, technologies and tools can be brought to cooperation in order to cover the whole workflow, roles and functionality that are involved in managing and exploiting documents throughout their life cycle. It represents a basic platform providing general-purpose functionality, to be tailored and extended in order to satisfy specific needs related to the particular context and environment in which it is to be used.

## A.1 General Framework

DOMINUS allows managing a set of digital libraries that can cooperate and/or be delivered in a unique, consistent and co-ordinate fashion. Each library may be public or private, this way defining biases for the users regarding the access to the documents they contain. In private libraries, security and privacy of document storage and transmission is supported.

### *A.1.1 Actors and Workflow*

DOMINUS provides different kinds of roles for the users that can be involved in its framework. Each role, in turn, implies a set of prerogatives regarding document processing, management and access. A single user can take on several roles in several

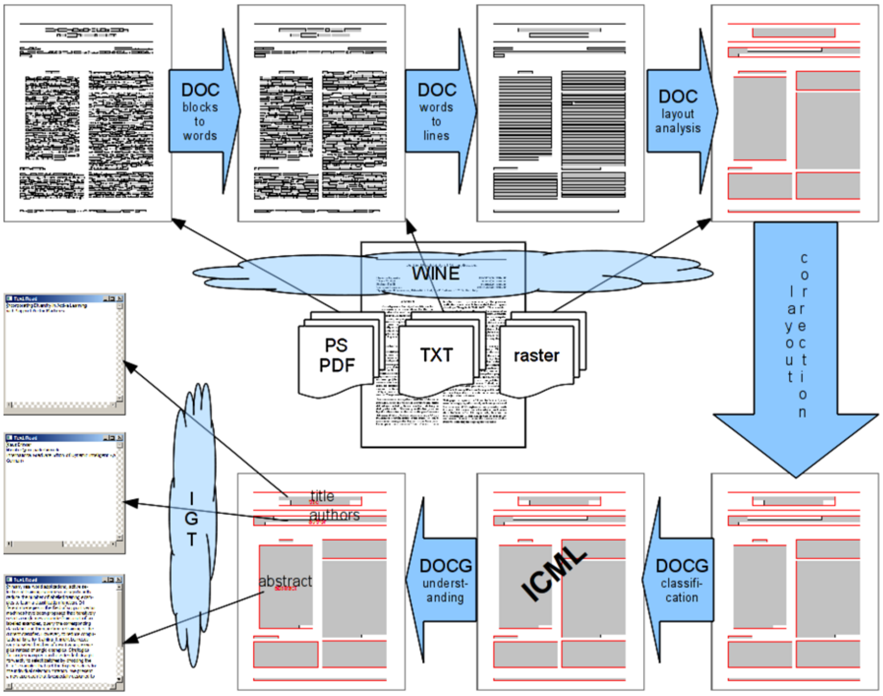


**Fig. A.1** DOMINUS document processing flow

digital libraries enclosed in the system, and the same role in a given library can be played by several users. All kinds of users are required to register to the system in order to exploit its functionality because user traceability is deemed as fundamental to ensure a correct and controlled workflow. *Administrators* control the general system and have the power to accept, include, suspend or expel digital libraries. Users that are in charge of managing a single library are called *librarians*. They must be enabled to play this role by an administrator or by another librarian of the same library, and have the power to accept, include, suspend or expel documents and users of the library. Each librarian can also enable other users to act as *authors* and/or *maintainers* of their library, and plays himself such roles. Authors can submit documents to a library. Maintainers are technicians that supervise the document flow inside the library, check that the automatic processing is correctly carried out, and, in case it is, validate the intermediate or final results, or, in case it is not, make the proper corrections and then restart it from the last safe accomplishment. The basic role, *end-user*, has just consultation purposes. A query panel is available to end-users, in which they can combine different search techniques, mix search results, get help and suggestions for their search, bookmark and organize the documents of interest. Any newly registered user starts as an end-user, having access to public libraries only; private libraries may provide them access by enabling specific rights.

A typical document workflow in DOMINUS (whose main steps are depicted in Fig. A.1) starts with the document submission by an author, which causes basic

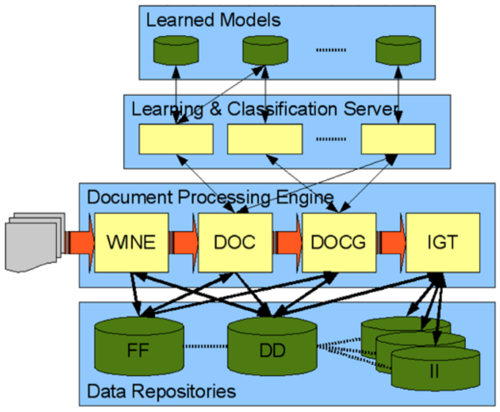


**Fig. A.2** Architecture of the DOMINUS document processing system

entries to be inserted in the database and a new folder to be created in the 'under processing' section of the library, where all the intermediate files produced by the workflow will be stored. The document itself is saved in such a folder, and then undergoes a preliminary step in which a standard internal representation is produced, independent of its actual source format. On such a normalized representation the layout analysis algorithms are applied, to identify the geometrical components and their inter-relationships, that are saved in another file. The resulting description is input to a classifier that assigns the document to one of the classes of interest to that library and, based on this, tags the document components according to the semantic role they play in that class. This allows extracting the content from the most significant ones only (which should improve the performance and quality of information extraction), in order to further process it. Pictures and graphics are cropped and saved in a sub-folder, while text is stored in the database for filing purposes. Subsequently, the document indexing and information extraction phases are started. Interesting objects are identified in images, while several kinds of Natural Language Processing techniques are applied to text (text categorization, keyword extraction, lexical and semantic indexing).

### *A.1.2 Architecture*

The overall architecture of DOMINUS is reported in Fig. A.2. Four main components can be identified, each of which could, in principle, be installed on a different computer in case the system has to be deployed in a distributed environment. Actually, several (logic or database) modules inside these components can be detached to work on separate computers, as well.

The *Document Processing Engine* is in charge of carrying out the various tasks that are needed to extract useful information from an input document and to make it available to final users for search and consultation purposes. It is activated whenever a new digital document is submitted to the system. A first step is carried out by a pre-processing module, called WINE (Wrapper for the Interpretation of Non-uniform Electronic document formats), that produces a standard internal XML representation of the document, independent of the source format in which it was provided. Such a representation initially describes the document as a set of pages made up of basic blocks directly extracted from the document source, and is extended with further (meta-)information as long as it is gathered by the system during the subsequent processing steps. For some sources, an intermediate vectorial format is produced.

Once the document's basic representation is obtained, the layout analysis step can be started. It is carried out by a module called DOC (Document Organization Composer) that is in charge of producing the frame-level representation of the document pages. Due to the possibly large number of basic blocks returned by WINE (e.g., in PS/PDF documents they consist of just word fragments), a preliminary aggregation is needed to enhance performance of the layout analysis process. Composite blocks corresponding to whole words are identified first, and then (since even after this step the number of word blocks can still be quite large) a further aggregation of words into lines is carried out. Based on such a line-block representation, DOC groups the semantically related blocks into the surrounding frames, each corresponding to a homogeneous and significant logical component in the document, and updates accordingly the XML representation of the document.

The next step is carried out by the DOCG (Document Objects Content Gatherer) module. To determine which frames are significant and should be further processed, Document Image Classification and Understanding are performed on the document description, and the resulting outcome is saved as meta-information in the XML representation of the document, that now contains the original document information, enriched with layout aggregations, class information and components annotation. From selected frames, textual and graphical information is automatically extracted, stored and passed on for further processing. After the information is extracted from the significant components, a final document categorization and indexing step, based on several and complementary techniques, is carried out by the IGT (Indexer of Graphics and Text) module in order to pave the way for an effective content-based information retrieval by the end users.

Many of the tasks carried out by the processing modules just described involve the production and exploitation of intermediate information that is to be permanently stored in the system for future reference. This is done through various interrelated *Data Repositories*. All information concerning the users, the collections and the documents' features and text is stored into the DOMINUS Database ($D^2$), a classical relational database. The source document, along with its internal representations and the extracted components (e.g., the cropped figures) are stored as separate files, each in the suitable format, according to a hierarchical folder organization, based on the different collections managed by the system, that is globally identified as the File Folders ($F^2$). Since different indexing techniques can be applied on a given collection, each on different, and possibly overlapping, subsets of documents, several separate Information Indexing ($I^2$) databases are generated, one for each index, that can be related to the main database because they use the same keys and ID's for the items they describe.

A fundamental role in the overall document processing architecture is played by the *Learning & Classification Server*, a suite of Machine Learning tools that come into play in different moments and with different objectives to support several tasks and modules of the Document Processing Engine (e.g., document classification, component labeling, layout correction, block aggregation, etc.), based on a number of automatically *Learned Models*. More specifically, these tools are exploited in two modalities. The 'learning' mode is used to continuously adapt the domain knowledge whenever new experimental evidence reveals inadequacies of the learned models or changes in the context. Indeed, in typical digital libraries new documents continuously become available over time and are to be integrated in the collection. For this reason, a predominant role is played by incremental learning techniques. The 'classification' mode is exploited to apply the learned models to specific cases in order to obtain useful information. The learning steps are carried out off-line, so that the system is in continuous operation even during the learning phase (albeit with the old models). Only after termination of the learning task the old models are replaced by the new ones, transparently to the users. The inborn complexity of the document domain, and the need to express relations among components, suggest the exploitation of symbolic first-order logic as a powerful representation language for some learning tasks.

## A.2 Functionality

The following list of functionality currently provided by DOMINUS reflects the sequence of processing steps that are applied on input documents from their submission up to indexing and information extraction. Each step is associated to the ratio of past documents on which the system automatically identified the proper solution, and to a threshold, such that, when processing a new document in that step, it is immediately carried on to the next step if the ratio is above the threshold (unless any problem arises), or it is suspended until an explicit confirmation by the maintainer is obtained if the ratio falls below the threshold (a final confirmation might be in any case required before committing the overall result).

### *A.2.1 Input Document Normalization*

Normalization of an input document obviously depends on the source format. The preprocessing step performed by WINE takes as input a generic digital document and produces a corresponding vectorial description. At the moment, it deals with documents in TXT, PS/PDF and raster formats, that represent the current *de facto* standards for document interchange. For TXT sources, sequences of characters delimited by spaces are considered as basic blocks. For PS or PDF documents, a proprietary PostScript program[1] is run that rewrites basic PostScript operators so that, when accessing the source, drawing instructions are redirected to strings in a text file, producing logical atoms that describe basic blocks (raster images are cropped and saved as autonomous objects). For raster sources nothing is done, and processing starts directly with the layout analysis step.

### *A.2.2 Layout Analysis*

Different techniques concur to identify the frames in a page, defined in DOMINUS as groups of objects completely surrounded by white space. Preliminarily, two steps are carried out to reduce the number of blocks that are input to the layout analysis algorithms, in order to improve their efficiency and effectiveness. If the basic blocks extracted from the document correspond just to fragments of words, they are grouped into word-level aggregates based on overlapping or adjacency. Then, a further aggregation of words into lines is performed (simple techniques based on the mean distance between blocks work on single-column documents, but are sometimes unable to correctly handle multi-column ones). Then, the Manhattan-shaped components are extracted first, using the RLSA on raster documents or (an improvement of) the Background Structure analysis algorithm on documents described by basic blocks. Lastly, on each Manhattan frame obtained this way, the RLSO (in its two versions for raster and block-based documents, respectively) is applied to retrieve non-Manhattan sub-parts (if any). After identifying the background structure of the document, the foreground is obtained by computing its complement. Two kinds of aggregates are identified: blocks filled with a homogeneous kind of content, and rectangular frames made up of blocks of the former type.

The modifications to the Background Structure analysis algorithm deserve some attention. First of all, horizontal/vertical lines are deemed as explicit separators, and hence their surrounding white spaces are *a priori* considered as background. Second, to avoid retrieving inter-word or inter-line background, white blocks whose height or width is below a given size are ignored. Third, extraction of background pieces is stopped as soon as the area of the new white rectangle retrieved, $W(R)$, represents a ratio of the total white area in the document page, $W(D)$, less than a given threshold $\delta$, i.e.,

$$\frac{W(R)}{W(D)} = \frac{W(R)}{A(D) - \sum_{i=1,\dots,n} A(R_i)} < \delta,$$

where $A(D)$ is the area of the document page under consideration and $A(R_i)$, for $i = 1, \dots, n$, are the areas of the blocks identified thus far in the page.

Due to the fixed stop threshold, the automatically recognized layout structure might not be fully satisfactory: important background pieces might be missing (so

[1]Existing tools for performing this task, such as *PSTOEDIT* (www.pstoedit.net) or *JPedal* (www.jpedal.org), were discarded due to not being fully compliant with the requirements.

that different frames are merged) or superfluous ones might have been included (resulting in single frames having been split). In other words, background areas are sometimes considered as content ones and/or *vice versa*. DOMINUS allows the maintainer to manually fix the problems by forcing some algorithm steps forward or backward with respect to the stop condition. This is possible since the system maintains three structures that keep track of: all white rectangles found ($W$), all black rectangles found ($B$) and all rectangles that it has not analyzed yet ($N$—if no threshold is given all the rectangles are analyzed and $N$ will be empty at the end of processing). When the user asks to go forward, the system extracts and processes further rectangles from $N$; conversely, if the user asks to go back, the system correspondingly moves blocks between $W$, $B$ and $N$.

However, many forward steps (during which the system would probably restore insignificant white rectangles) might be needed before retrieving very small missing background rectangles (e.g., the gap between two frames), and rectangles whose size is below the minimum threshold would not be retrieved at all. For this reason, DOMINUS allows the maintainer to directly point and remove superfluous pieces of retrieved background, or to explicitly add missing ones. Such manual corrections are logged and used by the incremental learning component INTHELEX to learn and refine layout correction models to improve performance on future documents by identifying and fixing such wrong background choices. Each example describes the added or removed block and all the blocks around it, along with their size and position in the document and the spatial relationships among them, both before and after the manual correction is made.

**Kernel-Based Basic Blocks Grouping** To automatically infer rewriting rules that suggest suitable parameter settings to group basic/word blocks into lines, RARE (Rule Aggregation REwriter) uses a kernel-based method. Each block is described as a feature vector of the form:

$$[i, p, x_i, x_f, y_i, y_f, c_x, c_y, h, w]$$

whose parameters are to be interpreted as follows:

- $i$ is the identifier of the considered block;
- $p$ is the number of page in which the block is placed;
- $(x_i, y_i)$ and $(x_f, y_f)$ are the coordinates of the top-left and bottom-right corners of the block, respectively;
- $(c_x, c_y)$ are the coordinates for the centroid of the block;
- $h, w$ are the block height and width, respectively.

Based on this description, given a block $O$ and its top-left, top, top-right, right, bottom-right, bottom, bottom-left and left *Close Neighbor* blocks $\{C_k\}_{k\in\{1,2,\dots,8\}}$ (thus, $O$ has at least one close neighbor and at most eight), an instance from which learning rewriting rules is represented using a template $[O, C_k]$:

$$[n, p, x_i, x_f, y_i, y_f, c_x, c_y, x_{ki}, x_{kf}, y_{ki}, y_{kf}, c_{kx}, c_{ky}, d_x, d_y]$$

for each close neighbor, where $n$ is an identifier for the new block, $d_x$ and $d_y$ express the distances between the two blocks, and the other parameters are those of the original blocks. An example is a set of instances, and is labeled as positive for the target concept “the two blocks can be merged” if and only if at least one $C_k$ is adjacent to $O$ and belongs to the same line in the original document, or as negative otherwise. It is worth noting that not every pair of adjacent blocks is necessarily a positive example, since they could belong to different frames in the document. As a consequence of the adopted representation, the problem is cast as a *Multiple Instance Problem* and solved by applying the *Iterated-Discrim* algorithm [77] in order to discover the relevant features and their values to be encoded in the rewriting rules.

### *A.2.3 Document Image Understanding*

On the final layout structure, Document Image Understanding is carried out, based on first-order logic models available in the Learning and Classification Server, and previously learned by INTHELEX from sample documents manually annotated by the maintainers. New incoming documents are first classified according to their first-page layout, to determine how to file them in the digital repository and what kind of processing they should undergo next. This step is crucial in digital libraries, where many different layout structures for the documents, either belonging to different classes or even to the same class, can be encountered. Then, each frame identified therein is associated to a tag expressing its role. While rules defining classes are independent on each other, rules to identify logical components might be interrelated, since it is typical that one component is defined and identified in relation to another one (e.g., the ‘authors’ frame might be defined as the frame placed just under the ‘title’ and above the ‘abstract’), which is a much more complex problem.

In case of failure or wrong classification, the maintainer can point out the correct outcome, which causes the theories to be suitably updated. It could be even the case that not only single definitions turn out to be faulty and need revision, but whole new document classes are to be included in the collection as soon as the first document for them becomes available. This represents a problem for most other systems that require not only all the information on the application domain, but also the set of classes for which they must learn definitions, to be completely determined when the learning process starts. Lastly, if documents of new, unknown classes are submitted, the system can autonomously extend the set of classes that are taken into account.

### *A.2.4 Categorization, Filing and Indexing*

After Document Image Understanding, DOMINUS focuses on a subset of logical components deemed as particularly significant. Graphical components are processed in order to identify known objects that can contribute to the understanding of the document content. Textual components are ‘read’, using the Tesseract OCR engine in the case of digitized documents, or extracting the text in the case of natively digital sources, and the extracted text is exploited for filing a record of the document. A full set of NLP techniques is also applied, including both standard (tokenization, language recognition, stopword removal, PoS tagging, stemming) and advanced ones (syntactic and semantic analysis), preliminary to further processing: while the former are often sufficient for document indexing purposes, the latter may support more precise and focused information extraction. Various kinds of linguistic resources, among which WordNet and WordNet Domains, are exploited as well.

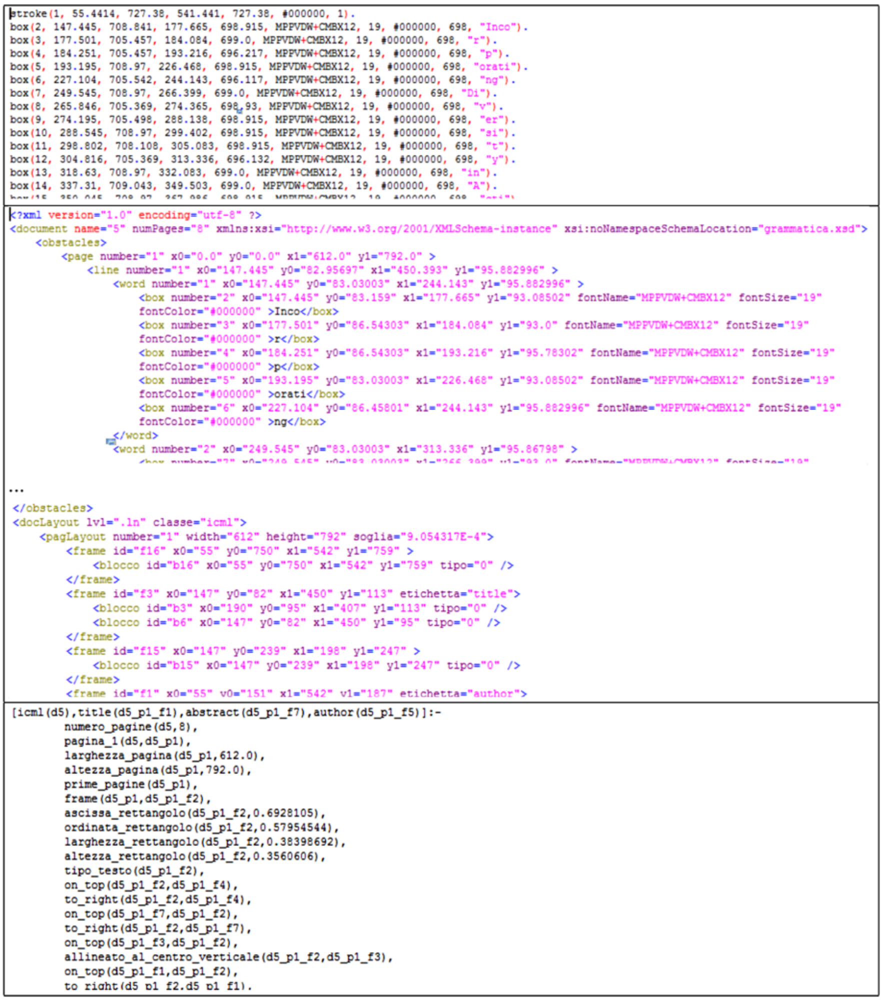
As to *Information Retrieval*, DOMINUS includes both classical term-based and advanced concept-based indexing techniques (using word stems for enhanced flexibility): a Vector Space Model based on TF-IDF, Latent Semantic Indexing (LSI) based on the log-entropy weighting scheme and Concept Indexing. In LSI, the minimum number of top-ranked documents needed to cover the whole set of terms is used as the reduction parameter $k$, and incremental updating methods are exploited, applying the most efficient techniques first to ensure a prompt response. Folding-In is applied each time a new document is submitted, while SVD-Updating is run after a number of new submissions to improve the overall index quality. Indexing from scratch is performed only occasionally, when the current index quality is tampered by several partial updatings. *Cosine similarity* is exploited to rank the outcome.

*Keyword Extraction* is included both for extracting key terms to be used as metadata for the document, and to perform keyword-based information retrieval. Two approaches are currently implemented, both taking into account the document logical structure to weight differently the terms: a global one based on term occurrence statistics that works on the whole collection, and a conceptual one based on WordNet and WordNet Domains that works on single documents. *Text Categorization*, either supervised (using a co-occurrence based technique) or unsupervised (using Concept Indexing), is exploited for a better organization of documents and their subject-based retrieval.

A GUI is provided that allows managing and mixing these techniques, their parameters and their results, in order to better fit the user needs. A module to suggest refined queries according to user relevance feedback is also supported, suggesting a different set of terms that turned out to be more characteristic of the documents that he singled out as interesting.

## A.3 Prototype Implementation

The variety of tasks needed to carry out the several processing steps provided by DOMINUS have required the exploitation of different technologies in its prototype implementation. Java is used as the main programming language, allowing maximum portability among platforms and easy connection to other technologies; in particular, the Java Server Faces have been chosen as a framework for the Web Application component that allows remote control of the system functionality. C++ is exploited for low-level tasks, such as communication with external modules. Prolog supports the implementation of most learning systems, and particularly of those



```
stroke(1, 55.4414, 727.38, 541.441, 727.38, #000000, 1).
box(2, 147.445, 708.841, 177.665, 698.915, MPPVDW+CMBX12, 19, #000000, 698, "Inco").
box(3, 177.501, 705.457, 184.084, 699.0, MPPVDW+CMBX12, 19, #000000, 698, "r").
box(4, 184.251, 705.457, 193.216, 696.217, MPPVDW+CMBX12, 19, #000000, 698, "p").
box(5, 193.195, 708.97, 226.468, 698.915, MPPVDW+CMBX12, 19, #000000, 698, "orati").
box(6, 227.104, 705.542, 244.143, 696.117, MPPVDW+CMBX12, 19, #000000, 698, "ng").
box(7, 249.545, 708.97, 266.399, 699.0, MPPVDW+CMBX12, 19, #000000, 698, "Di").
box(8, 265.846, 705.369, 274.365, 698.93, MPPVDW+CMBX12, 19, #000000, 698, "v").
box(9, 274.195, 705.498, 288.138, 698.915, MPPVDW+CMBX12, 19, #000000, 698, "er").
box(10, 288.545, 708.97, 299.402, 698.915, MPPVDW+CMBX12, 19, #000000, 698, "si").
box(11, 298.802, 708.108, 305.083, 698.915, MPPVDW+CMBX12, 19, #000000, 698, "t").
box(12, 304.816, 705.369, 313.336, 696.132, MPPVDW+CMBX12, 19, #000000, 698, "y").
box(13, 318.63, 708.97, 332.083, 699.0, MPPVDW+CMBX12, 19, #000000, 698, "in").
box(14, 337.31, 709.043, 349.503, 699.0, MPPVDW+CMBX12, 19, #000000, 698, "A").
```

```
<?xml version="1.0" encoding="utf-8" ?>
<document name="5" numPages="8" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="grammatica.xsd">
    <obstacles>
        <page number="1" x0="0.0" y0="0.0" x1="612.0" y1="792.0" >
            <line number="1" x0="147.445" y0="82.95697" x1="450.393" y1="95.882996" >
                <word number="1" x0="147.445" y0="83.03003" x1="244.143" y1="95.882996" >
                    <box number="2" x0="147.445" y0="83.159" x1="177.665" y1="93.08502" fontName="MPPVDW+CMBX12" fontSize="19"
                    fontColor="#000000" >Inco</box>
                    <box number="3" x0="177.501" y0="86.54303" x1="184.084" y1="93.0" fontName="MPPVDW+CMBX12" fontSize="19"
                    fontColor="#000000" >r</box>
                    <box number="4" x0="184.251" y0="86.54303" x1="193.216" y1="95.78302" fontName="MPPVDW+CMBX12" fontSize="19"
                    fontColor="#000000" >p</box>
                    <box number="5" x0="193.195" y0="83.03003" x1="226.468" y1="93.08502" fontName="MPPVDW+CMBX12" fontSize="19"
                    fontColor="#000000" >orati</box>
                    <box number="6" x0="227.104" y0="86.45801" x1="244.143" y1="95.882996" fontName="MPPVDW+CMBX12" fontSize="19"
                    fontColor="#000000" >ng</box>
                </word>
                <word number="2" x0="249.545" y0="83.03003" x1="313.336" y1="95.86798" >

...
</obstacles>
<docLayout lvl=".ln" classe="icml">
    <pagLayout number="1" width="612" height="792" soglia="9.054317E-4">
        <frame id="f16" x0="55" y0="750" x1="542" y1="759" >
            <blocco id="b16" x0="55" y0="750" x1="542" y1="759" tipo="0" />
        </frame>
        <frame id="f3" x0="147" y0="82" x1="450" y1="113" etichetta="title">
            <blocco id="b3" x0="190" y0="95" x1="407" y1="113" tipo="0" />
            <blocco id="b6" x0="147" y0="82" x1="450" y1="95" tipo="0" />
        </frame>
        <frame id="f15" x0="147" y0="239" x1="198" y1="247" >
            <blocco id="b15" x0="147" y0="239" x1="198" y1="247" tipo="0" />
        </frame>
        <frame id="f1" x0="55" y0="151" x1="542" y1="187" etichetta="author">
```

```
[icml(d5),title(d5_p1_f1),abstract(d5_p1_f7),author(d5_p1_f5)]:-
        numero_pagine(d5,8),
        pagina_1(d5,d5_p1),
        larghezza_pagina(d5_p1,612.0),
        altezza_pagina(d5_p1,792.0),
        prime_pagine(d5_p1),
        frame(d5_p1,d5_p1_f2),
        ascissa_rettangolo(d5_p1_f2,0.6928105),
        ordinata_rettangolo(d5_p1_f2,0.57954544),
        larghezza_rettangolo(d5_p1_f2,0.38398692),
        altezza_rettangolo(d5_p1_f2,0.3560606),
        tipo_testo(d5_p1_f2),
        on_top(d5_p1_f2,d5_p1_f4),
        to_right(d5_p1_f2,d5_p1_f4),
        on_top(d5_p1_f7,d5_p1_f2),
        to_right(d5_p1_f2,d5_p1_f7),
        on_top(d5_p1_f3,d5_p1_f2),
        allineato_al_centro_verticale(d5_p1_f2,d5_p1_f3),
        on_top(d5_p1_f1,d5_p1_f2),
        to_right(d5_p1_f2,d5_p1_f1).
```

**Fig. A.3** Internal representations in DOMINUS: Vector format (*top*), XML (*middle*), Horn clause (*bottom*)

based on relational representations. CLIPS (C-Language Integrated Production System) was used to implement the expert system components that are in charge of performing knowledge-intensive tasks. MySQL is used as a DataBase Management System. A PostScript program is exploited to extract information from PS and PDF files. XML serves as a standard representation formalism for the information to be passed through the system. GnuPG is in charge of the security-related aspects of document management. Moreover, DOMINUS exploits different internal representations for the documents, an excerpt of which is shown in Fig. A.3.

The vector format describing the original digital document as a set of pages, each of which is composed of *basic blocks*, is based on the following descriptors:

**box**$(i, x_l, y_t, x_r, y_b, f, d, c, r, s)$ a piece of text, represented by its bounding box;
**stroke**$(i, x_l, y_t, x_r, y_b, c, t)$ a graphical (horizontal/vertical) line;
**fill**$(i, x_l, y_t, x_r, y_b, c)$ a closed area filled with one color;
**image**$(i, x_l, y_t, x_r, y_b)$ a raster image;
**page**$(n, w, h)$ page information,

where $i$ is the block identifier; $(x_l, y_t)$ and $(x_r, y_b)$ are, respectively, the top-left and bottom-right coordinates of the block ($x_l = x_r$ for vertical lines and $y_t = y_b$ for horizontal lines); $f$ is the text font and $d$ its size; $c$ is the color of the text, line or area in RGB (#rrggbb) format; $r$ denotes the row in which the text appears; $s$ is the portion of document text contained in the block; $t$ is the thickness of the line; $n$ is the page number; $w$ and $h$ are the page width and height, respectively.

The *XML representation* initially contains the same information as the vector representation and is extended as long as the document is processed, to include the information gained by the various steps. It is organized in a hierarchical structure:

**document** the whole document (attributes include its name, number of pages and class)
 **obstacles** the basic layout blocks found in the document source
  **page** a page in the document (attributes: page number, bounding box coordinates)
   **line** a text line in the document (attributes: ID, bounding box coordinates)
    **word** a word in the document (attributes: ID, bounding box coordinates)
     **box** corresponding to a *box* item in the vector representation
   **stroke** corresponding to a *stroke* item in the vector representation
   **image** corresponding to an *image* item in the vector representation
 **layout** the result of document image analysis (attributes include the document class)
  **page** a page in the document (attributes include the layout analysis threshold)
   **frame** a frame in the document page (attributes include its logical role)
    **block** a content block in the document page (attributes include its content type)

A third representation, exploited by the First-Order Logic learning and classification systems, expresses the document layout as a Horn clause made up of predicates applied to terms. Once the layout components of a document have been discovered, the description of their features and spatial/topological relationships (plus page information in multi-page documents) is built using the following page descriptors:

**page_**$n(d, p)$ $p$ is the $n$th page in document $d$;
**first_page**$(p)$ true if $p$ is the first page of the document;
**last_page**$(p)$ true if $p$ is the last page of the document;
**in_first_pages**$(p)$ true if page $p$ belongs to the first $1/3$ pages of the document;
**in_middle_pages**$(p)$ true if page $p$ is in the middle $1/3$ pages of the document;
**in_last_pages**$(p)$ true if page $p$ belongs to the last $1/3$ pages of the document;
**number_of_pages**$(d, n)$ document $d$ has $n$ pages;
**page_width**$(p, w)$ page $p$ has width $w$ (expressed in dots);
**page_height**$(p, h)$ $p$ has height $h$ (expressed in dots);

frame descriptors:

**frame**$(p, f)$ $f$ is a frame of page $p$;
**type_**$t(f)$ $t$ is the type of content of frame $f$
 ($t \in \{$**text**, **graphic**, **mixed**, **vertical_line**, **horizontal_line**$\}$);
**width**$(f, w)$ frame $f$ has width $w$ (expressed as a ratio of the page width);
**height**$(f, h)$ frame $f$ has height $h$ (expressed as a ratio of the page height);

**x_coordinate**$(f, x)$ the centroid of frame $f$ has horizontal coordinate $x$;
**y_coordinate**$(f, y)$ the centroid of frame $f$ has vertical coordinate $y$;

and topological relation descriptors:

**on_top**$(f_1, f_2)$ frame $f_1$ is placed above frame $f_2$;
**to_right**$(f_1, f_2)$ frame $f_1$ is placed on the right of frame $f_2$;
**left_aligned**$(f_1, f_2)$ frames $f_1$ and $f_2$ are left-aligned;
**center_aligned**$(f_1, f_2)$ frames $f_1$ and $f_2$ are center-aligned horizontally;
**right_aligned**$(f_1, f_2)$ frames $f_1$ and $f_2$ are right-aligned;
**top_aligned**$(f_1, f_2)$ frames $f_1$ and $f_2$ are top-aligned;
**middle_aligned**$(f_1, f_2)$ frames $f_1$ and $f_2$ are center-aligned vertically;
**bottom_aligned**$(f_1, f_2)$ frames $f_1$ and $f_2$ are bottom-aligned.

To improve efficiency, horizontal and vertical frame size and position are automatically discretized by the learning system, using abstraction operators, into a set of intervals denoted by symbolic descriptors, according to the ratio of the whole page size they represent:

**pos_upper**$(f)$, **pos_middle**$(f)$, **pos_lower**$(f)$ vertical position of the frame in the page;
**pos_left**$(f)$, **pos_center**$(f)$, **pos_right**$(f)$ horizontal position of the frame in the page;
**width_**$X(f)$ horizontal size of the frame (7 intervals);
**height_**$X(f)$ vertical size of the frame (11 intervals).

## A.4 Exploitation for Scientific Conference Management

The DOMINUS framework can be exploited in several different domain-specific applications. This section focuses on the case of *Scientific Conference* organization, a complex and multi-faceted activity where the *Chairs* play the role of Librarians. DOMINUS, as a Conference Management System, can support some of the more critical and knowledge-intensive tasks, among which are submission of abstracts and papers and review assignment and management.

Let us present a possible scenario. A candidate Author opens the submission page on the Internet, where (after registering, or after logging in if already registered) he can browse his hard disk and submit a file in one of the accepted formats. After uploading, the paper undergoes the processing steps provided by DOMINUS. Layout analysis and document classification allow automatically checking that the submission fulfills the acceptable style standards (a single conference might allow different styles for the submitted papers, e.g., full paper, poster, demo), otherwise the Author can be immediately warned. The layout components of interest are then identified (e.g., title, author, abstract and references in a full paper) and their text is read, stored and used to automatically file the submission record (e.g., by filling its title, authors and abstract fields), asking the Author just for a confirmation. If the system is unable to carry out any of these steps, such an event is notified to the conference Chairs that can manually fix the problem and let the system proceed.

After the submission deadline, the papers are automatically categorized based on their content, which paves the way for a suitable reviewer assignment that matches the paper topic against the reviewers' expertise, in order to find the best associations. The actual assignment is carried out by an expert system, exploiting a mix of sources expressing the similarity between papers and reviewers from different perspectives.

**GRAPE** *GRAPE* (Global Review Assignment Processing Engine) [76] is an expert system for review assignment that can take advantage of both the papers' content and the reviewers' expertise and preferences (as expressed in the *bidding* phase, if provided for). Let $\{p_i\}_{i=1,\dots,n}$ denote the set of papers submitted to the conference and $\{r_j\}_{j=1,\dots,m}$ the set of reviewers. Let $T(\cdot)$ denote the number of topics associated to an entity, among $t$ conference topics. The assignment must fulfill at least the following constraints:

1. Each paper is assigned to exactly $k$ reviewers (usually, $k \in \{2, 3, 4\}$);
2. Each reviewer has roughly the same number of papers to review (the mean number of reviews per reviewer is $nk/m$), except for specific reviewers $r_j$ that must review at most $h_j$ papers;
3. Papers should be reviewed by domain experts;
4. Reviewers should be assigned to papers based on their expertise and preferences.

Two measures are to be maximized in order to fulfill constraints 3 and 4: reviewer *gratification* and article *coverage*. Both are based on the *confidence* degree between a paper and a reviewer. Additionally, the former also depends on the number of assigned papers that were actually bid by the reviewer, and the latter on the *expertise degree* of the reviewers the paper was assigned to (related, for a reviewer $r_j$, to the number of topics in which he is expert, and computed as $T(r_j)/t$).

As to the confidence assessment between a paper $p_i$ and a reviewer $r_j$, one parameter is the number of topics in common $|T(p_i) \cap T(r_j)|$ declared by the Authors as the subject of their paper and by the Reviewers as their area of expertise (if any of the two is empty, the corresponding coverage degree will be always zero). However, practical experience proved that such a self-assessment is not always trustworthy, for which reason additional parameters have been added, among them the subject category and keywords automatically identified by the system, and a similarity computed for each paper-reviewer pair according to Latent Semantic Analysis. For a paper, this information is extracted using the text in the title, abstract and bibliographic references,[2] as the most significant indicators of the subject and research field the paper is concerned with. For a reviewer, it is extracted from its curriculum (including the list of his previous publications).

In a first phase, the system progressively assigns reviewers to papers with the fewest candidate reviewers. At the same time, it 'prefers' assigning papers to reviewers with few assignments, to avoid having reviewers with few or no assigned papers. This can be viewed as a search for review assignments that keep low the average number of reviews *per* reviewer and maximize the coverage degree of the papers. In a second phase, the system tries to assign a paper $p_i$ which has not yet

[2] A small expert system is exploited to identify the various components of bibliographic references, independently of the particular style exploited for listing them.

been assigned $k$ reviewers to a reviewer $r_j$ with a high confidence level between $r_j$ and $p_i$ or, if it is not possible, to a reviewer with a high level of expertise. Each reviewer is then presented with the list of all papers by decreasing similarity, where the first $h$, provisionally assigned to him, are highlighted. Now he can bid them based on this suggestion, instead of having an unordered list of papers as usual in the current practice. After collecting all the reviewers' bids, GRAPE is restarted to search for a new solution that tries to change previous assignments by taking into account these bids as well. Then, the final solution is presented to the reviewers.

GRAPE being a rule-based system, it is easy to add new rules in order to change/improve its behavior, and to describe background knowledge, such as further constraints or conflicts, in a natural way (e.g., a reviewer could be preferably assigned to papers in which he is cited, but must not review papers submitted by people from his Institution).

# Appendix B
# Machine Learning Notions

Many approaches developed for the various steps of document processing rest on Machine Learning techniques. Indeed, the complexity of the tasks they involve requires a great deal of flexibility and adaptation, that standard algorithms based on static parameters cannot provide. Obviously, Machine Learning is a widespread research area by itself, for which many valid books are available (e.g., [139]) and new publications improve everyday the state-of-the-art. Thus, this appendix cannot be anything else than a brief recall of the main types of possible approaches to Machine Learning and, therein, a quick description of some techniques that are referenced in the book chapters, in order to provide the reader with an immediate support and pointers to more thorough publications.

## B.1 Categorization of Techniques

*Machine Learning* aims at obtaining some representation (a *model*) of interesting concepts (often called *classes*) in a given domain, starting from data and knowledge coming from that domain, in order to exploit them for future applications, most often prediction of the correct class for new observations (*classification*). Given such a general objective, there are several dimensions along which Machine Learning approaches and techniques can be categorized. Some of them are orthogonal, and hence give rise to a number of possible combinations. Others are organized in a general-to-specific taxonomy, with respect to the degree of expressive power or other parameters.

A first neat distinction is to be made between *supervised* and *unsupervised* learning techniques. The former work on a set of *training examples*, i.e., descriptions of *observations* taken from the domain of interest labeled with the proper classes, and produce (implicit or explicit) descriptions for those classes. The latter take unlabeled observations, and search for regularities that can be considered as relevant for future applications. For instance, *clustering* aims at grouping the available observations so that each group contains similar and homogeneous observations, while observations belonging to different groups are quite different.

From another perspective, *symbolic* and *numerical* (or *sub-symbolic*) approaches can be distinguished according to the representation of observations, and the processing thereof, being based on logic and abstract (human-understandable) concepts rather than values to be processed by mathematical or statistical operations (more problematic for human interpretation). The latter usually exploit *attribute-value* descriptions, where a fixed number of descriptors is defined, for each of which a single value can be filled; the former typically allow descriptions of variable length, in the form of logic formulæ. Symbolic approaches, in particular, can work at the level of propositional logic, or can adopt first-order logic as a representation language, which introduces the ability to express relationships among objects, at the cost of higher computational complexity.

*Conceptual* learning aims at obtaining intensional descriptions of concepts, while *lazy* techniques just store interesting data and perform predictions simply by comparing the new observations to them.

## B.2 Noteworthy Techniques

In the following, some Machine Learning techniques referenced in the book are briefly introduced.

**Artificial Neural Networks** *Artificial Neural Networks* (*ANN*s) [139] are a traditional sub-symbolic approach inspired from the behavior of biological neural networks in the human brain. Just like a network of interlinked neurons is present in the brain, where each neuron receives chemical inputs from its predecessors and sends accordingly a chemical output to other successor neurons, in an artificial neural network each element is interconnected to other elements of the same kind, and can output a signal whose intensity is mathematically computed as a function of the intensity of the signals coming from its input neurons. A subset of neurons directly receive their input stimuli from the outside world, as the values of features or attributes that describe an observation to be classified; another subset of neurons, each corresponding to a class, provide their output to the external world; the rest are internal neurons, organized in a (possibly multi-layered) set of connections. ANNs are built by defining their structure and assigning a standard weight to all connections thereof. Then, for each learning example provided, made up of an input description and the corresponding correct class, first the description is used for classification, and then a backpropagation mechanism traces back the connections that were activated to produce the classification and reinforces or smooths them, according to the predicted class being correct or wrong, respectively. Such a change represents a learning of new information by the network. Thus, the single artificial neurons do not change the operations they embed, but the overall network behavior changes because of modifications of the weights on the relationships among them. ANNs are very interesting because, once learned, they can be directly implemented onto electronic chips and plugged into devices that exploit them. Unfortunately, human interpretation of a learned ANN is a still unsolved problem.

**Decision Trees** A *decision tree* is a branching structure in which the root is the starting point, each internal node corresponds to a test over an attribute, each branch represents a possible outcome (or set of outcomes) for the test, and the leaves bear class information. Given an observation (in the form of a feature vector) described according to the same attributes as the tree, starting from the root and routing the tree by repeatedly carrying out the tests in the nodes and following the branch labeled with the corresponding results, one gets the class to be assigned to the observation at hand. Machine Learning techniques for automatically building decision trees starting from a set of observations already tagged with the correct classes (called the *training set*) usually place in the nodes that are closer to the root the attributes that turn out to be more discriminative for partitioning the training set instances into different classes, according to the outcome of class-driven information measures applied to the training set.

***k*-Nearest Neighbor** *k-Nearest Neighbor* (*k-NN*) is a lazy supervised technique that reduces the learning step to just storing the available examples. When a new observation is to be classified, it is compared to all the examples by means of some kind of distance (or similarity) measure suitable for the adopted representation (e.g., multi-dimensional Euclidean distance for vectors of numeric values). The examples are then sorted by increasing distance (or decreasing similarity), and the first $k$ of the ranking (the 'nearest neighbors' of the observation, according to the distance perspective) are considered. The majority class thereof is thus assigned to the observation in question. One issue is the choice of $k$, usually taken as (an approximation of) the square root of the number of training examples. Another question is how to handle ties between classes in the first $k$ neighbors: if only two possible classes are considered, it suffices taking $k$ to be an odd integer to ensure a single winner class.

**Inductive Logic Programming** *Inductive Logic Programming* (*ILP*) [145] is the intersection between Inductive Learning (that defines the task) and Logic Programming (that provides the representation) aiming at the identification of logic theories that explain the given examples. Theories contain hypotheses, one for each concept to be learned, that in turn are made up of *clauses*, i.e., first-order logic formulæ in the form of implications:

$$p_1 \wedge \cdots \wedge p_n \quad \Rightarrow \quad c_1 \vee \cdots \vee c_m$$

to be interpreted as "if all the premises $p_i$ are true, then at least one of the conclusions $c_j$ is true". Both the premises and the conclusions are *literals*, i.e., possibly negated atoms, where an *atom* is a logic predicate applied to arguments that are *terms* (variables, constants or functions applied to other terms as arguments). Terms usually denote objects: constants stand for specific objects, variables for generic ones, and functions represent objects by reference to other objects. Predicates with only one argument represent properties of that object, while predicates with many arguments express relationships among those objects. Particularly interesting are *Horn* clauses in which exactly one literal is present in the conclusion, and hence there is no uncertainty about what is true once the premises have been verified.

**Naive Bayes** Given a set of possible classes $\{c_i\}_{i=1,\dots,n}$ for objects described by a set of $m$ features $\{f_j\}_{j=1,\dots,m}$, it might be interesting to assess the probability that an object $o$ having feature values $\{v_j\}_{j=1,\dots,m}$ belongs to each class. Mathematically, this is expressed by the conditional probability $p(c(o) = c_i | f_1(o) = v_1 \wedge \cdots \wedge f_m(o) = v_m)$. Although these values are generally unknown, the Bayes theorem[1] allows expressing them according to the inverse conditional probability, which yields the *Naive Bayes* classifier formula (omitting the functions for the sake of readability):

$$\begin{aligned} p(c_i | v_1 \wedge \cdots \wedge v_m) &= \frac{p(v_1 \wedge \cdots \wedge v_m | c_i) \cdot p(c_i)}{p(v_1 \wedge \cdots \wedge v_m)} \\ &\quad \text{assuming statistical independence among features} \\ &= \frac{p(v_1|c_i) \cdots p(v_m|c_i) \cdot p(c_i)}{p(v_1) \cdots p(v_m)}, \end{aligned}$$

where the elements to the right of the equality can be estimated from a training set, if available, by simply counting the corresponding frequencies. Of course, the statistical independence assumption is rarely fulfilled, but this turns out to be in practice a satisfactory tradeoff between complexity and effectiveness.

**Hidden Markov Models** *Hidden Markov Models* (*HMM*) are a statistical model where a finite number of states are taken into account, and each state is associated to a probability distribution that determines the transition to next states (called the *transition probabilities*) and the corresponding output of symbols. It is a Markov model in that the probability of outputting a symbol depends on the past history of the states that were traversed before emitting that symbol. It is hidden because, given a sequence of output symbols, even if the probability of transitions is known, one cannot say exactly which states have been activated, although the sequence provides a hint about that. In addition to the set of states and the set of symbols in the output alphabet, an HMM is defined by the probability distributions of the transitions/outputs and of the initial states. In a graphical representation, states are nodes, and transitions are edges, each weighted with a corresponding probability and labeled with an output symbol. Obviously, the sum of weights for all outcoming edges from a node must be 1. From a learning viewpoint, the problem is how to infer the structure and/or the weights of the graph representing an HMM starting from instances of output sequences generated by the target model.

**Clustering** *Clustering* is an unsupervised learning task aimed at finding subsets of a given set of objects such that elements in the same subset are very similar to each other, while elements in different subsets are quite different [113]. Often, but not

[1]

$$p(A \wedge B) = p(B|A) \cdot p(A) = p(A|B) \cdot p(B) \quad \Rightarrow \quad p(B|A) = \frac{p(A|B) \cdot p(B)}{p(A)}.$$

mandatorily, the subsets represent a partition of the original set. Clustering is heavily based on the availability of a *distance* function that, applied to the descriptions of two elements, returns the degree of similarity between them. The *hierarchical* approach to clustering requires that, given the similarity between all pairs of elements in the starting set, the groups are found by progressive aggregation of subsets, starting from singleton subsets containing an element each (*agglomerative* techniques), or by progressive split of subsets, starting from the whole initial set (*partitional* techniques), and produces a tree of subsets called a *dendrogram*, in which choosing the desired level of refinement. Noteworthy techniques include *single link* (where the closest subsets, according to their closest elements, are progressively merged) and *complete link* (where the subsets whose farthest elements are closer are merged first). Conversely, in the *k-means* technique an initial random split of the elements in the desired number of subsets is performed, and then the *centroid* of each subset is computed and the elements are re-assigned to the subset of the closest centroid, repeating the procedure until a convergence is found (which might not always happen). A typical problem for clustering is the decision as to how many subsets are to be obtained.

## B.3 Experimental Strategies

Given a learning technique and a dataset of $n$ instances on which it is to be applied, a typical task consists in assessing the performance of the former on the latter, according to some evaluation metric. After learning a model from a set of *training* instances, it is applied to new *test* instances to obtain a performance value. To reach a sufficient degree of confidence in the outcome, several such values are needed, to act as samples for applying a statistical test. Hence, the problem of how to obtain a sample population that ensures reliability of the test outcomes arises. Indeed, a single dataset is usually available for the problem at hand, while the statistical tests may require several dozens samples for being successfully applied. The obvious solution consists in partitioning the dataset many times into different pairs of subsets, a *training set* from which learning a model, and a corresponding *test set* on which evaluating performance. Several techniques are available for this, to be applied depending on the particular experimental conditions.

**$k$-Fold Cross-Validation** *$k$-Fold Cross-validation* consists in partitioning the dataset into $k$ subsets (the *folds*) each containing $n/k$ instances (usually evenly distributed among the various classes to be learned). Each subset becomes in turn a test set, on which evaluating the performance of the model learned on the remaining $k-1$ subsets appended to each other to form the training set. So, $k$ runs of the system on different portions of the original dataset are performed, each yielding a sample to be fed to the statistical test for significance assessment. This strategy can be applied when $n$ is such that the number $n/k$ of instances in each fold is satisfactory even for $k$ large enough to ensure fulfillment of the test applicability requirements.

**Leave-One-Out** *Leave-One-Out* is, in some sense, $k$-fold Cross-Validation taken to its extreme: The dataset is partitioned in as many subsets as the number of instances it contains. Thus, each run of the learning system takes all but one instances as a training set, and applies the learned model on the single example left out. This strategy is typically used for small datasets, where applying $k$-fold Cross-Validation for a sufficiently large $k$ would yield subsets containing very few instances for each class to be learned.

**Random Split** The *Random Split* strategy takes as input the number $k$ of desired runs of the learning technique, and the percentage $t$ of instances to be used for training purposes (a typical value is $t = 70$). Then, the dataset is divided for $k$ times into a training set containing a randomly selected $t\%$ of the instances, and a test set containing the remaining $(100 - t)\%$ ones (again, even distribution of instances from all classes in the training and test set should be ensured in each split). Even for datasets having medium-small size, this strategy allows obtaining a sufficient number $k$ of samples because the available combinations when randomly drawing single examples is much larger than the maximum number of partition elements in $k$-fold Cross-Validation. Additionally, this strategy provides training sets with much more variability in the order of the instances, which makes a significant difference in learning systems whose outcome depends on the order in which instances are considered.

# Glossary

For the sake of comfortable reading, the various concepts used in this book are usually defined and explained directly under the sections where they are exploited. As an exception, a few of them that are needed in different and distant parts of the book are recalled here, so that this can serve as a quick reference to them, avoiding the need for the reader to go through the book searching for the place where they were first introduced.

**Bounding box** The minimum rectangle delimiting a given component of a raster image.

**Byte ordering** The technique used to store in memory the bytes of an information word. Two standards exist: *little endian* (going from the least to the most significant byte) and *big endian* (from the most to the least significant byte).

**Ceiling function** Denoted by $\lceil x \rceil$, returns the smallest integer not less than a real number $x$.

**Chunk** A portion of information, usually having fixed size in a given context.

**Connected component** In a raster image, a set of pixels that fulfill a given property, such that each pixel in the set is adjacent to at least another pixel in the set, and no pixel outside the set fulfills that property and is adjacent to any pixel in the set (if only the horizontal and vertical positions are considered, this is called a *4-connection*; if the diagonal positions are considered as well, this is called an *8-connection*). In a graph, a set of nodes such that any pair of nodes in the set is linked by a sequence of edges, and no node outside the set is linked by an edge to any node in the set.

**Heaviside unit function** A discontinuous function, usually denoted $\mathrm{u}(\cdot)$, that evaluates to 0 for negative arguments and 1 for positive arguments.

**Heterarchy** A taxonomic organization in which an element can have many ancestors. Formally, it can be represented as a Directed Acyclic Graph (DAG), and the resulting organization is that of a forest. The relationship can be many-to-many, i.e., an element can represent the end of arcs that start from many nodes, possibly even belonging to different trees in the forest or placed at different distances from the root. A tree is a particular case that represents the classical hierarchy in which a node may have many children but has one and only one parent, and there exists a single element (the *root*) that is ancestor of all the others and that does not have a parent.

**KL-divergence** Allows comparing a reference probability distribution $p(X)$ to another one $q(X)$ (e.g., an expected distribution with one drawn from empirical data). It is defined as

$$D\big(p(X) \parallel q(X)\big) = \sum_{x \in X} -p(x) \log q(x) - \big(-p(x) \log p(x)\big) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

and is strictly connected to Information Theory, expressing *information gain* and *relative entropy*. Indeed, the *mutual information* between two distributions can be written as

$$I(X;Y) = \sum_{x \in X, y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = D\big(p(X,Y) \parallel p(X)p(Y)\big).$$

**Linear regression** A mathematical technique that, given a set of points in the plane, searches for a straight line in that plane that best ‘fits’ those points according to a given quality function (usually, minimization of the sum of squared distances between each point and the line).

**Run** A sequence of consecutive symbols fulfilling a given property (usually their being equal to each other), embedded in a longer sequence.

**Scanline** A row of pixels in a raster image.

# References

***Although each chapter reports its own bibliographic references, a complete list of cited items, covering the whole landscape of document processing phases and approaches presented in this book, might be interesting, and is reported below.***

1. Data Encryption Standard. Tech. rep FIPS Pub. 46-1, National Bureau of Standards, Washington, DC, USA (1987)
2. Graphics Interchange Format (sm) specification—version 89a. Tech. rep., Compuserve Inc. (1990)
3. TIFF specification—revision 6.0. Tech. rep., Adobe Systems Incorporated (1992)
4. Secure Hash Standard (SHS). Tech. rep. FIPS PUB 180, National Institute of Standards and Technology (1993)
5. Secure Hash Standard (SHS). Tech. rep. FIPS PUB 180-1, National Institute of Standards and Technology (1995)
6. Digital Signature Standard (DSS). Tech. rep. FIPS PUB 186-1, National Institute of Standards and Technology (1998)
7. Document Object Model (DOM) Level 1 Specification—version 1.0. Tech. rep. REC-DOM-Level-1-19981001, W3C (1998)
8. Knowledge for development. Tech. rep., The World Bank (1998/1999)
9. HTML 4.01 specification—W3C recommendation. Tech. rep., W3C (1999)
10. Merriam-Webster's Collegiate Dictionary, 10th edn. Merriam-Webster Inc. (1999)
11. XML Path Language (XPath) 1.0—W3C recommendation. Tech. rep., W3C (1999)
12. XSL Transformations: (XSLT) 1.0—W3C recommendation. Tech. rep., W3C (1999)
13. Document Object Model (DOM) Level 2 Core Specification. Tech. rep. 1.0, W3C (2000)
14. Secure Hash Standard (SHS)—amended 25 February 2004. Tech. rep. FIPS PUB 180-2, National Institute of Standards and Technology (2002)
15. International standard ISO/IEC 10646: Information technology—Universal Multiple-octet coded Character Set (UCS). Tech. rep., ISO/IEC (2003)
16. Portable Network Graphics (PNG) specification—W3C recommendation, 2nd edn. Tech. rep., W3C (2003)
17. Lizardtech djvu reference—version 3, Tech. rep., Lizardtech, A Celartem Company (2005)
18. Extensible Markup Language (XML) 1.1—W3C recommendation, 2nd edn. Tech. rep., W3C (2006)
19. Extensible Stylesheet Language (XSL) 1.1—W3C recommendation. Tech. rep., W3C (2006)
20. Microsoft Office Word 1997–2007 binary file format specification [*.doc]. Tech. rep., Microsoft Corporation (2007)

21. Open Document Format for office applications (OpenDocument) v1.1—OASIS standard. Tech. rep., OASIS (2007)
22. Extensible Markup Language (XML) 1.0—W3C recommendation, 5th edn. Tech. rep., W3C (2008)
23. Secure Hash Standard (SHS). Tech. rep. FIPS PUB 180-3, National Institute of Standards and Technology (2008)
24. Digital Signature Standard (DSS). Tech. rep. FIPS PUB 186-3, National Institute of Standards and Technology (2009)
25. Dublin Core metadata element set version 1.1. Tech. rep. 15836, International Standards Organization (2009)
26. Abdul-Rahman, A.: The PGP trust model. EDI-Forum (1997)
27. Addis, A., Angioni, M., Armano, G., Demontis, R., Tuveri, F., Vargiu, E.: A novel semantic approach to create document collections. In: Proceedings of the IADIS International Conference—Intelligent Systems and Agents (2008)
28. Adobe Systems Incorporated: PDF Reference—Adobe Portable Document Format Version 1.3, 2nd edn. Addison-Wesley, Reading (2000)
29. Allen, J.F.: Natural Language Understanding. Benjamin Cummings, Redwood City (1994)
30. Altamura, O., Esposito, F., Malerba, D.: Transforming paper documents into XML format with WISDOM++. International Journal on Document Analysis and Recognition **4**, 2–17 (2001)
31. Angelici, C.: Documentazione e documento (diritto civile). Enciclopedia Giuridica Treccani **XI** (1989) (in Italian)
32. Baird, H.S.: The skew angle of printed documents. In: Proceedings of the Conference of the Society of Photographic Scientists and Engineers, pp. 14–21 (1987)
33. Baird, H.S.: Background structure in document images. In: Advances in Structural and Syntactic Pattern Recognition, pp. 17–34. World Scientific, Singapore (1992)
34. Baird, H.S.: Document image defect models. In: Baird, H.S., Bunke, H., Yamamoto, K. (eds.) Structured Document Image Analysis, pp. 546–556. Springer, Berlin (1992)
35. Baird, H.S., Jones, S., Fortune, S.: Image segmentation by shape-directed covers. In: Proceedings of the 10th International Conference on Pattern Recognition (ICPR), pp. 820–825 (1990)
36. Begeja, L., Renger, B., Saraclar, M.: A system for searching and browsing spoken communications. In: HLT/NAACL Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval, pp. 1–8 (2004)
37. Bentivogli, L., Forner, P., Magnini, B., Pianta, E.: Revising WordNet Domains hierarchy: Semantics, coverage, and balancing. In: Proceedings of COLING 2004 Workshop on Multilingual Linguistic Resources, pp. 101–108 (2004)
38. Berkhin, P.: Survey of clustering Data Mining techniques. Tech. rep., Accrue Software, San Jose (2002)
39. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, New York (2001)
40. Berry, M.W., Dumais, S.T., O'Brien, G.W.: Using linear algebra for intelligent information retrieval. SIAM Review **37**(4), 573–595 (1995)
41. Berry-Rogghe, G.: The computation of collocations and their relevance to lexical studies. In: Aitken, A.J., Bailey, R.W., Hamilton-Smith, N. (eds.) The Computer and Literary Studies, pp. 103–112. Edinburgh University Press, Edinburgh (1973)
42. Boulgouris, N.V., Kompatsiaris, I., Mezaris, V., Simitopoulos, D., Strintzis, M.G.: Segmentation and content-based watermarking for color image and image region indexing and retrieval. EURASIP Journal on Applied Signal Processing **1**, 418–431 (2002)
43. Breuel, T.M.: Two geometric algorithms for layout analysis. In: Proceedings of the 5th International Workshop on Document Analysis Systems (DAS). Lecture Notes in Computer Science, vol. 2423, pp. 188–199. Springer, Berlin (2002)
44. Briet, S.: Qu'est-ce que la Documentation. EDIT, Paris (1951)
45. Brill, E.: A simple rule-based part of speech tagger. In: HLT '91: Proceedings of the Workshop on Speech and Natural Language, pp. 112–116 (1992)

46. Brill, E.: Some advances in transformation-based part of speech tagging. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI), vol. 1, pp. 722–727 (1994)
47. Brill, E.: Unsupervised learning of disambiguation rules for part of speech tagging. In: Natural Language Processing Using Very Large Corpora Workshop, pp. 1–13. Kluwer, Amsterdam (1995)
48. Buckland, M.K.: What is a 'document'? Journal of the American Society for Information Science **48**(9), 804–809 (1997)
49. Buckland, M.K.: What is a 'digital document'? Document Numérique **2**(2), 221–230 (1998)
50. Burger, W., Burge, M.J.: Digital Image Processing. Texts in Computer Science. Springer, Berlin (2008)
51. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R.: OpenPGP message format. Tech. rep RFC 4880, IETF (2007)
52. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-lite family. Journal of Automated Reasoning **39**(3), 385–429 (2007)
53. Calzolari, N., Lenci, A.: Linguistica computazionale—strumenti e risorse per il trattamento automatico della lingua. Mondo Digitale **2**, 56–69 (2004) (in Italian)
54. Candian, A.: Documentazione e documento (teoria generale). Enciclopedia Giuridica Treccani (1964) (in Italian)
55. Canny, J.F.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **8**(6), 679–698 (1986)
56. Cao, H., Prasad, R., Natarajan, P., MacRostie, E.: Robust page segmentation based on smearing and error correction unifying top-down and bottom-up approaches. In: Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 392–396. IEEE Computer Society, New York (2007)
57. Carnelutti, F.: Studi sulla sottoscrizione. Rivista di Diritto Commerciale, p. 509 ss. (1929) (in Italian)
58. Carnelutti, F.: Documento—teoria moderna. Novissimo Digesto Italiano (1957) (in Italian)
59. International Telegraph and Telephone Consultative Committee (CCITT): Recommendation t.81. Tech. rep., International Telecommunication Union (ITU) (1992)
60. Cesarini, F., Marinai, S., Soda, G., Gori, M.: Structured document segmentation and representation by the modified X–Y tree. In: Proceedings of the 5th International Conference on Document Analysis and Recognition (ICDAR), pp. 563–566. IEEE Computer Society, New York (1999)
61. Cestnik, B.: Estimating probabilities: A crucial task in machine learning. In: Proceedings of the 9th European Conference on Machine Learning (ECAI), pp. 147–149 (1990)
62. Chaudhuri, B.: Digital Document Processing—Major Directions and Recent Advances. Springer, Berlin (2007)
63. Chen, Q.: Evaluation of OCR algorithms for images with different spatial resolution and noise. Ph.D. thesis, University of Ottawa, Canada (2003)
64. Chen, Y., Li, J., Wang, J.Z.: Machine Learning and Statistical Modeling Approaches to Image Retrieval. Kluwer, Amsterdam (2004)
65. Ciardiello, G., Scafuro, G., Degrandi, M., Spada, M., Roccotelli, M.: An experimental system for office document handling and text recognition. In: Proceedings of the 9th International Conference on Pattern Recognition (ICPR), pp. 739–743 (1988)
66. Cocks, C.C.: A note on 'non-secret encryption'. Tech. rep., GCHQ (1973)
67. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, T.: Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) profile. Tech. rep RFC 5280, Internet Engineering Task Force (IETF) (2008)
68. Cowie, J., Wilks, Y.: Information extraction. In: Dale, R., Moisl, H., Somers, H. (eds.) Handbook of Natural Language Processing, pp. 241–260. Marcel Dekker, New York (2000)
69. De Mauro, T.: Grande Dizionario Italiano dell'Uso. UTET, Turin (1999) (in Italian)
70. Deb, S.: Multimedia Systems and Content-Based Image Retrieval. IGI Publishing (2003)

71. Deerwester, S., Dumais, S.T., Landauer, T.K., Furnas, G., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society of Information Science **41**(6), 391–407 (1990)
72. Department for Culture, Media and Sport, Department for Business, Innovation and Skills: Digital Britain—final report. Tech. rep., UK Government (2009)
73. Deutsch, P.: Deflate compressed data format specification 1.3. Tech. rep. RFC1951 (1996)
74. Deutsch, P., Gailly, J.L.: Zlib compressed data format specification 3.3. Tech. rep. RFC1950 (1996)
75. Dewey, M., et al.: Dewey Decimal Classification and Relative Index, 22nd edn. OCLC Online Computer Library Center (2003)
76. Di Mauro, N., Basile, T.M., Ferilli, S.: GRAPE: An expert review assignment component for scientific conference management systems. In: Innovations in Applied Artificial Intelligence. Lecture Notes in Artificial Intelligence, vol. 3533, pp. 789–798. Springer, Berlin (2005)
77. Dietterich, T.G., Lathrop, R.H., Lozano-Perez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence **89**(1–2), 31–71 (1997)
78. Diffie, W.: An overview of public key cryptography. IEEE Communications Society Magazine **16**, 24–32 (1978)
79. Diffie, W.: The first ten years of public-key cryptography. Proceedings of the IEEE **76**(5), 560–577 (1988)
80. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **IT-22**, 644–654 (1976)
81. Diffie, W., Hellman, M.: Privacy and authentication: An introduction to cryptography. Proceedings of the IEEE, **67**, 397–427 (1979)
82. Egenhofer, M.J.: Reasoning about binary topological relations. In: Gunther, O., Schek, H.J. (eds.) 2nd Symposium on Large Spatial Databases. Lecture Notes in Computer Science, vol. 525, pp. 143–160. Springer, Berlin (1991)
83. Egenhofer, M.J., Herring, J.R.: A mathematical framework for the definition of topological relationships. In: Proceedings of the 4th International Symposium on Spatial Data Handling, pp. 803–813 (1990)
84. Egenhofer, M.J., Sharma, J., Mark, D.M.: A critical comparison of the 4-intersection and 9-intersection models for spatial relations: Formal analysis. In: Proceedings of the 11th International Symposium on Computer-Assisted Cartography (Auto-Carto) (1993)
85. Eisenberg, J.: OASIS OpenDocument Essentials—using OASIS OpenDocument XML. Friends of OpenDocument (2005)
86. Ellis, J.H.: The possibility of secure non-secret digital encryption. Tech. rep., GCHQ (1970)
87. Esposito, F., Ferilli, S., Basile, T.M.A., Di Mauro, N.: Machine Learning for digital document processing: From layout analysis to metadata extraction. In: Marinai, S., Fujisawa, H. (eds.) Machine Learning in Document Analysis and Recognition. Studies in Computational Intelligence, vol. 90, pp. 105–138. Springer, Berlin (2008)
88. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T.M., Di Mauro, N.: Incremental multistrategy learning for document processing. Applied Artificial Intelligence: An International Journal **17**(8/9), 859–883 (2003)
89. Fateman, R.J., Tokuyasu, T.: A suite of lisp programs for document image analysis and structuring. Tech. rep., Computer Science Division, EECS Department—University of California at Berkeley (1994)
90. Feistel, H.: Cryptography and computer privacy. Scientific American **128**(5) (1973)
91. Ferilli, S., Basile, T.M.A., Esposito, F.: A histogram-based technique for automatic threshold assessment in a Run Length Smoothing-based algorithm. In: Proceedings of the 9th International Workshop on Document Analysis Systems (DAS). ACM International Conference Proceedings, pp. 349–356 (2010)
92. Ferilli, S., Biba, M., Esposito, F., Basile, T.M.A.: A distance-based technique for non-Manhattan layout analysis. In: Proceedings of the 10th International Conference on Document Analysis Recognition (ICDAR), pp. 231–235 (2009)
93. Frank, A.U.: Qualitative spatial reasoning: Cardinal directions as an example. International Journal of Geographical Information Systems **10**(3), 269–290 (1996)

94. Freitag, D.: Machine learning for information extraction in informal domains. Machine Learning **39**, 169–202 (2000)
95. Gale, W., Church, K., Yarowsky, D.: One sense per discourse. In: Proceedings of the ARPA Workshop on Speech and Natural Language Processing, pp. 233–237 (1992)
96. Garfinkel, S.: PGP: Pretty Good Privacy. O'Reilly (1994)
97. Gatos, B., Pratikakis, I., Ntirogiannis, K.: Segmentation based recovery of arbitrarily warped document images. In: Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), pp. 989–993 (2007)
98. Gattuso, A.: Processo telematico. Mondo Professionisti **I**(13), III–VI (2007) (in Italian)
99. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Prentice Hall, New York (2008)
100. Grishman, R.: Computational Linguistic—An Introduction. Studies in Natural Language Processing. Cambridge University Press, Cambridge (1986)
101. Grishman, R.: Information extraction: Techniques and challenges. In: International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology. Lecture Notes in Computer Science, vol. 1299, pp. 10–27. Springer, Berlin (1997)
102. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2), 199–220 (1993)
103. Gunn, S.R., Nixon, M.S.: A robust snake implementation; a dual active contour. IEEE Transactions on Pattern Analysis and Machine Intelligence **19**(1), 63–68 (1997)
104. Halliday, M.: Categories of the theory of grammar. Word **17**, 241–292 (1961)
105. Hamilton, E.: JPEG file interchange format—version 1.2. Tech. rep. (1992)
106. Hanson, N.R.: Patterns of Discovery—An Inquiry into the Conceptual Foundations of Science. Cambridge University Press, Cambridge (1958)
107. Hough, P.V.: Method and means for recognizing complex patterns. Tech. rep. 3069654, US Patent (1962)
108. Huffman, D.: A method for the construction of minimum-redundancy codes. In: Proceedings of the I.R.E., pp. 1098–1102 (1952)
109. Hunyadi, L.: Keyword extraction: Aims and ways today and tomorrow. In: Proceedings of the Keyword Project: Unlocking Content through Computational Linguistics (2001)
110. Ide, N., Véronis, J.: Introduction to the special issue on word sense disambiguation: The state of the art. Compuational Linguistics **24**(1), 1–40 (1998)
111. Impedovo, S., Ottaviano, L., Occhinegro, S.: Optical character recognition—a survey. International Journal on Pattern Recognition and Artificial Intelligence **5**(1–2), 1–24 (1991)
112. Irti, N.: Sul concetto giuridico di documento. Riv. Trim. Dir. e Proc. Civ. (1969) (in Italian)
113. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM Computing Surveys **31**(3), 164–323 (1999)
114. Kainz, W., Egenhofer, M.J., Greasley, I.: Modeling spatial relations and operations with partially ordered sets. International Journal of Geographical Information Systems **7**(3), 215–229 (1993)
115. Kakas, A.C., Mancarella, P.: On the relation of truth maintenance and abduction. In: Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence (PRICAI), pp. 438–443 (1990)
116. Kaliski, B.: Pkcs #7: Cryptographic message syntax. Tech. rep. RFC 2315, IETF (1998)
117. Karypis, G., Han, E.H.S.: Concept indexing: A fast dimensionality reduction algorithm with applications to document retrieval & categorization. Tech. rep. TR 00-016, University of Minnesota—Department of Computer Science and Engineering (2000)
118. Karypis, G., Han, E.H.S.: Fast supervised dimensionality reduction algorithm with applications to document categorization & retrieval. In: Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM), pp. 12–19 (2000)
119. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: Active contour models. International Journal of Computer Vision **1**(4), 321–331 (1988)
120. Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires **IX**, 5–38 (1883)

121. Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires **IX**, 161–191 (1883)
122. Kise, K., Sato, A., Iwata, M.: Segmentation of page images using the area Voronoi diagram. Computer Vision Image Understanding **70**(3), 370–382 (1998)
123. Krovetz, R.: More than one sense per discourse. In: Proceedings of SENSEVAL Workshop, pp. 1–10 (1998)
124. Lafferty, J., Sleator, D.D., Temperley, D.: Grammatical trigrams: A probabilistic model of link grammar. In: Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language (1992)
125. Lai, X.: On the Design and Security of Block Ciphers. ETH Series in Information Processing, vol. 1. Hartung-Gorre (1992)
126. Lamport, L.: LATEX, A Document Preparation System—User's Guide and Reference Manual, 2nd edn. Addison-Wesley, Reading (1994)
127. Landauer, T.K., Dumais, S.T.: A solution to Plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge. Psychological Review **104**, 111–140 (1997)
128. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Processes **25**, 259–284 (1998)
129. Lesk, M.: Automatic sense disambiguation using machine-readable dictionaries: How to tell a pine cone from an ice cream cone. In: Proceedings of the 5th International Conference on Systems Documentation (SIGDOC), pp. 24–26 (1986)
130. Lindsay, P.H., Norman, D.A.: Human Information Processing: Introduction to Psychology. Academic Press, San Diego (1977)
131. Magnini, B., Cavaglià, G.: Integrating subject field codes into WordNet. In: Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC), pp. 1413–1418 (2000)
132. Manning, C.D., Raghavan, P., Schutze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
133. Manning, C.D., Schutze, H.: Foundations of Statistical Natural Language Processing. MIT Press, New York (1999)
134. Matsuo, Y., Ishizuka, M.: Keyword extraction from a single document using word co-occurrence statistical information. International Journal on Artificial Intelligence Tools **13**(1), 157–169 (2004)
135. McCarthy, J., Minsky, M.L., Rochester, N., Shannon, C.E.: A proposal for the Dartmouth Summer research project on artificial intelligence. Tech. rep., Dartmouth College (1955)
136. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
137. Michalski, R.S.: Inferential theory of learning. developing foundations for multistrategy learning. In: Michalski, R., Tecuci, G. (eds.) Machine Learning. A Multistrategy Approach, vol. IV, pp. 3–61. Morgan Kaufmann, San Mateo (1994)
138. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.J.: Introduction to WordNet: An on-line lexical database. International Journal of Lexicography **3**(4), 235–244 (1990)
139. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
140. Mori, S., Suen, C.Y., Yamamoto, K.: Historical review of OCR research and development. Proceedings of the IEEE **80**(7), 1029–1058 (1992)
141. Nagy, G.: Twenty years of document image analysis in PAMI. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(1), 38–62 (2000)
142. Nagy, G., Kanai, J., Krishnamoorthy, M.: Two complementary techniques for digitized document analysis. In: ACM Conference on Document Processing Systems (1988)
143. Nagy, G., Seth, S., Viswanathan, M.: A prototype document image analysis system for technical journals. Computer **25**(7), 10–22 (1992)
144. Nagy, G., Seth, S.C.: Hierarchical representation of optically scanned documents. In: Proceedings of the 7th International Conference on Pattern Recognition (ICPR), pp. 347–349. IEEE Computer Society, New York (1984)

145. Nienhuys-Cheng, S.H., de Wolf, R. (eds.): Foundations of Inductive Logic Programming. Lecture Notes in Computer Science, vol. 1228. Springer, Berlin (1997)
146. O Kit Hong, F., Bink-Aghai, R.P.: A Web prefetching model based on content analysis (2000)
147. O'Brien, G.W.: Information management tools for updating an SVD-encoded indexing scheme. Tech. rep. CS-94-258, University of Tennessee, Knoxville (1994)
148. O'Gorman, L.: The document spectrum for page layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(11), 1162–1173 (1993)
149. O'Gorman, L., Kasturi, R.: Document Image Analysis. IEEE Computer Society, New York (1995)
150. Oltramari, A., Vetere, G.: Lexicon and ontology interplay in Senso Comune. In: Proceedings of OntoLex 2008 Workshop, 6th International Conference on Language Resources and Evaluation (LREC) (2008)
151. Otsu, N.: A threshold selection method from gray-level histogram. IEEE Transactions on Systems, Man, and Cybernetics **9**(1), 62–66 (1979)
152. Papadias, D., Theodoridis, Y.: Spatial relations minimum bounding rectangles, and spatial data structures. International Journal of Geographical Information Science **11**(2), 111–138 (1997)
153. Papamarkos, N., Tzortzakis, J., Gatos, B.: Determination of run-length smoothing values for document segmentation. In: Proceedings of the International Conference on Electronic Circuits and Systems (ICECS), vol. 2, pp. 684–687 (1996)
154. Pavlidis, T., Zhou, J.: Page segmentation by white streams. In: Proceedings of the 1st International Conference on Document Analysis and Recognition (ICDAR), pp. 945–953 (1991)
155. Pierce, J.R.: Symbols, Signals and Noise—The Nature and Process of Communication. Harper Modern Science Series. Harper & Brothers (1961)
156. Porter, M.: An algorithm for suffix stripping. Program **14**(3), 130–137 (1980)
157. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, Cambridge (2007)
158. Reid, G.: Thinking in PostScript. Addison-Wesley, Reading (1990)
159. Rice, S.V., Jenkins, F.R., Nartker, T.A.: The fourth annual test of OCR accuracy. Tech. rep. 95-03, Information Science Research Institute, University of Nevada, Las Vegas (1995)
160. Rivest, R.: The MD5 Message-Digest algorithm. Tech. rep. RFC 1321, Network Working Group (1992)
161. Rocchio, J.: Relevance feedback in information retrieval. In: Salton, G. (ed.) The SMART Retrieval System, pp. 313–323. Prentice-Hall, New York (1971)
162. Rosenfeld, A.: Picture Processing by Computer. Academic Press, San Diego (1969)
163. Salembier, P., Marques, F.: Region-based representations of image and video: Segmentation tools for multimedia services. IEEE Transactions on Circuits and Systems for Video Technology **9**(8), 1147–1169 (1999)
164. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Communications of the ACM **18**(11), 613–620 (1975)
165. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys **34**(1), 1–47 (2002)
166. Shafait, F., Smith, R.: Table detection in heterogeneous documents. In: Proceedings of the 9th IAPR International Workshop on Document Analysis Systems (DAS). ACM International Conference Proceedings, pp. 65–72 (2010)
167. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press, Champaign (1949)
168. Shih, F., Chen, S.S.: Adaptive document block segmentation and classification. IEEE Transactions on Systems, Man, and Cybernetics—Part B **26**(5), 797–802 (1996)
169. Simon, A., Pret, J.C., Johnson, A.P.: A fast algorithm for bottom-up document layout analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence **19**(3), 273–277 (1997)
170. Singh, S.: The Code Book. Doubleday, New York (1999)
171. Skiena, S.S.: The Algorithm Design Manual, 2nd edn. Springer, Berlin (2008)

172. Sleator, D.D., Temperley, D.: Parsing English text with a link grammar. In: Proceedings of the 3rd International Workshop on Parsing Technologies (1993)
173. Smith, R.: A simple and efficient skew detection algorithm via text row accumulation. In: Proceedings of the 3rd International Conference on Document Analysis and Recognition (ICDAR), pp. 1145–1148. IEEE Computer Society, New York (1995)
174. Smith, R.: An overview of the Tesseract OCR engine. In: Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR), pp. 629–633. IEEE Computer Society, New York (2007)
175. Smith, R.: Hybrid page layout analysis via tab-stop detection. In: Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR), pp. 241–245. IEEE Computer Society, New York (2009)
176. Soderland, S.: Learning information extraction rules for semi-structured and free text. Machine Learning **34**, 233–272 (1999)
177. Sonka, M., Hlavac, V., Boyle, R.: Image Processing, Analysis, and Machine Vision, 3rd (international) edn. Thomson (2008)
178. Sorkin, A.: Lucifer a cryptographic algorithm. Cryptologia **8**(1), 22–24 (1984)
179. Stallings, W.: Cryptography and Network Security. Principles and Practice, 3rd edn. Prentice Hall, New York (2002)
180. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., Weger, B.D.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology. Lecture Notes in Computer Science, vol. 5677, pp. 55–69. Springer, Berlin (2009)
181. Sun, H.M.: Page segmentation for Manhattan and non-Manhattan layout documents via selective CRLA. In: Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR), pp. 116–120. IEEE Computer Society, New York (2005)
182. The Unicode Consortium: The Unicode Standard, Version 5.0, 5th edn. Addison-Wesley, Reading (2006)
183. Tosi, E.: Il codice del Diritto dell'Informatica e di Internet, 6th edn. (2007). I codici vigenti. La Tribuna (in Italian)
184. Uzun, Y.: Keyword Extraction using Naive Bayes (2005)
185. W3C SVG Working Group: Scalable Vector Graphics (SVG) 1.1 specification. Tech. rep., W3C (2003)
186. Wahl, F., Wong, K., Casey, R.: Block segmentation and text extraction in mixed text/image documents. Graphical Models and Image Processing **20**, 375–390 (1982)
187. Wall, K., Danielsson, P.E.: A fast sequential method for polygonal approximation of digital curves. Computer Vision, Graphics, and Image Processing **28**, 220–227 (1984)
188. Wang, D., Srihari, S.N.: Classification of newspaper image blocks using texture analysis. Computer Vision, Graphics, and Image Processing **47**, 327–352 (1989)
189. Welch, T.: A technique for high-performance data compression. IEEE Computer **17**(6), 8–19 (1984)
190. Wong, K.Y., Casey, R., Wahl, F.M.: Document analysis system. IBM Journal of Research and Development **26**, 647–656 (1982)
191. Wood, L.: Programming the Web: The W3C DOM specification. IEEE Internet Computing **3**(1), 48–54 (1999)
192. Woods, W.A.: Conceptual Indexing: A Better Way to Organize Knowledge. SML Technical Report Series. Sun Microsystems (1997)
193. Yarowsky, D.: One sense per collocation. In: Proceeding of ARPA Human Language Technology Workshop, pp. 266–271 (1993)
194. Yarowsky, D.: Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In: Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, pp. 88–95 (1994)
195. Yarowsky, D.: Unsupervised Word Sense Disambiguation rivaling supervised methods. In: Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, pp. 189–196 (1995)

196. Yuille, A.L.: Deformable templates for face recognition. Journal of Cognitive Neuroscience **3**(1), 59–70 (1991)
197. Zimmermann, P.R.: The Official PGP User's Guide. MIT Press, New York (1995)
198. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory **23**(3), 337–343 (1977)
199. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory **24**(5), 530–536 (1978)
200. Zucker, J.D.: Semantic abstraction for concept representation and learning. In: Proceedings of the 4th International Workshop on Multistrategy Learning (MSL), pp. 157–164 (1998)

# Index