Simulations among Multidimensional Turing Machines

Michael Conrad Loui

August 1980

This report was prepared with the support of the National Science Foundation Grant No. MCS-77-19754 and the Fannie and John Hertz Foundation.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

Simulations among Multidimensional Turing Machines

by
Michael Conrad Loui

Submitted to the Department of Electrical Engineering and Computer Science
on August 12, 1980
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

ABSTRACT

This thesis presents three independent papers: nearly optimal on-line simulations among multidimensional Turing machines, a space bound for one-tape multidimensional Turing machines, and new proofs in the pebble game.

For all $d \geq 1$, all $e > d$, and all $\epsilon > 0$, every deterministic multihead e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line by a deterministic multihead d-dimensional Turing machine in time $O(T(n)^{1 + 1/d - 1/e + \epsilon})$. This simulation almost achieves the known lower bound $\Omega(T(n)^{1 + 1/d - 1/e})$ on the time required.

Every nondeterministic d-dimensional Turing machine with one worktape head of time complexity $T(n)$ can be simulated by a deterministic Turing machine of space complexity $(T(n) \log T(n))^{d/(d+1)}$. The proof includes a generalization of crossing sequences.

An overlap argument is used to derive two new results in the pebble game. Every directed acyclic graph G with n vertices and bounded indegree can be pebbled with $O(n/\log n)$ pebbles. Furthermore, if $S \geq O(n/\log n)$, then G can be pebbled with S pebbles in time $S2^{2^{O(n/S)}}$.

Key Words: automata, computational complexity, simulation, Turing machine, multidimensional Turing machine, nondeterminism, time complexity, tape complexity, pebble game, time-space tradeoff, overlap.

THESIS SUPERVISOR: Albert R. Meyer
TITLE: Professor of Computer Science and Engineering

ACKNOWLEDGMENTS

For the last three years Albert Meyer has supervised my doctoral studies with patience and wisdom. Under his tutelage I have honed my research skills. I have benefited immeasurably from his advice on lecturing and technical writing. His exemplary fastidiousness bespeaks his high standards of excellence.

Harold Abelson and Leonard Adleman, the readers on my thesis committee, suffered through early drafts of this thesis. I appreciate their perspicacious recommendations. In addition, Professor Adleman suggested the generalization of crossing sequences used in Chapter 2.

I value the encouragement of Nicholas Pippenger and Joel Seiferas, who have taken particular interest in my work and have generously shared their time with me. Dr. Pippenger noticed the extension of the proof of Theorem 3.1 to the proof of Theorem 3.2. Moreover, Professor Seiferas and Robert Melville provided a copy of a paper by Grigoriev [4] that is related to Chapter 1.

My colleagues in the Theory of Computation Group have contributed to an intellectually exciting environment. Neil Immerman and Andrea LaPaugh suggested expository improvements to parts of this thesis. Daniel Weise devised the pebbling strategy in the proof of Lemma 3.1, and Christos Papadimitriou provided the proof of Lemma 4.1. I also thank Alan Baratz, Errol Lloyd, Jeffrey Jaffe, Gary Miller, Ron Pinter, Vaughan Pratt, Ronald Rivest, Adi Shamir, Alan Sherman, Michael Sipser, and Karl Winklman for enlightening discussions.

I am fortunate that Ronald Book, James Munkres, and Alan Willsky guided my early professional development. I hope to emulate the clarity of their writing and the effectiveness of their teaching.

My family and friends have been kind and encouraging throughout my graduate study.

I am grateful for the fellowship provided by the Fannie and John Hertz Foundation for my five years at M.I.T. My research has also been supported by the National Science Foundation under Grant No. MCS-77-19754.

Table of Contents

PREFACE

The four chapters of this thesis were written independently and may be read seapartely. Each has its own introduction, terminology, and notations, but all references have been collected at the end of the thesis.

Chapter 1 presents an on-line simulation of a deterministic multihead e-dimensional Turing machine of time complexity $T(n)$ by a deterministic multihead d-dimensional machine of time complexity $O(T(n)^{1 + 1/d - 1/e + \epsilon})$ for all $\epsilon > 0$. In Theorem 1.2 the ϵ in the exponent is replaced by $o(1)$. This simulation nearly achieves the known lower bound $\Omega(T(n)^{1 + 1/d - 1/e})$ on the time required.

Continuing the study of multidimensional machines, Chapter 2 presents an off-line simulation of a nondeterministic d-dimensional machine with one worktape head that runs in time $T(n)$ by a deterministic machine in space $(T(n) \log T(n))^{d/(d+1)}$. An anonymous referee noticed the simulation by an alternating Turing machine in time $O((T(n) \log T(n))^{d/(d+1)})$ (Theorem 2.3). This chapter has been accepted for publication in *Theoretical Computer Science*. An earlier version appeared as Technical Memorandum TM-145 of the M.I.T. Laboratory for Computer Science [14].

Chapter 3 uses an overlap argument to derive new proofs in the pebble game. We develop a strategy that uses $O(n/\log n)$ pebbles to pebble every directed acyclic graph with n vertices and bounded indegree. A variation of this strategy uses S pebbles to pebble the graph in at most $2^{2^{O(n/S)}}$ steps. This note on the pebble game will appear in *Information Processing Letters*.

Chapter 4 recommends further research on automata with nonsequential storage structures. It includes a novel geometric argument that suggests a time-space tradeoff for simulating a multidimensional Turing machine by a tree machine.

Chapter 1. Simulations among Multidimensional Turing Machines

1.1. Background

Introduced by Hartmanis and Stearns [5], multidimensional Turing machines are natural generalizations of conventional Turing machines. Hennie and Grigoriev [3, 6] established a lower bound of $\Omega(T(n)^{1 + 1/d - 1/e})$ on the time required by a multihead d-dimensional Turing machine to simulate an e-dimensional machine of time complexity $T(n)$ on-line. We present a simulation that nearly achieves this bound.

Theorem 1.1. For all $d \geq 1$, all $e > d$, and all $\varepsilon > 0$, every multihead e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line by a multihead d-dimensional Turing machine in time $O(T(n)^{1 + 1/d - 1/e + \varepsilon})$.

For the case $d = 1$, Pippenger and Fischer [22] devised an optimal simulation that runs in time $O(T(n)^{2 - 1/e})$ on-line. Grigoriev [3] described an on-line simulation in time $O(T(n)^{1 + 1/d})$ when $e = d + 1$; even in this special case, Theorem 1.1 provides a better upper bound. Also, Grigoriev proved that every storage modification machine of time complexity $T(n)$ can be simulated on-line by a d-dimensional machine in time $O(T(n)^{1 + 1/(d+1)})$; since every multidimensional Turing machine can be simulated in real time by a storage modification machine [27], every e-dimensional machine can be simulated on-line by a d-dimensional machine in time $O(T(n)^{1 + 1/(d+1)})$. The time required by our simulation is smaller, however.

Grigoriev [4] demonstrated that every nondeterministic e-dimensional machine can be simulated off-line by a nondeterministic d-dimensional machine in time $O(T(n)^{1 + 1/d - 1/e + \varepsilon})$ for every $\varepsilon > 0$. We consider only deterministic machines. Our simulation can be modified to yield this result about nondeterministic machines.

Paul, Seiferas, and Simon [19] studied simulations among multidimensional machines with limited numbers of worktape heads. They established nonlinear lower bounds on the time required to simulate multidimensional machines on-line by machines with fewer worktape heads. Furthermore, they presented simulations of multitape multidimensional machines by machines with just two worktapes having one head each. In contrast, we present a simulation by a machine with more worktape heads.

1.2. Simulation

Let us review definitions for multidimensional Turing machines. To each cell of a d-dimensional worktape assign in the usual way a d-tuple of integers called the *coordinates* of the cell. The coordinates of adjacent cells differ in just one component by ± 1. The *origin* is the cell whose coordinates are all zero. A *d-dimensional Turing machine* has a finite-state control, a read-only input tape, a write-only output tape, and a finite number of d-dimensional worktapes, each of which has a finite number of heads. At each step the machine reads the symbols in the cells on which the input and worktape heads are positioned, writes symbols on these worktape cells and possibly on the output tape too, and shifts each worktape head in one of $2d + 1$ possible directions – either to one of $2d$ adjacent cells or to the same cell. Initially, all worktape cells hold blanks, and every worktape head is positioned on the origin of its tape. Leong and Seiferas [12] proved that every d-dimensional Turing machine can be simulated in real time by a d-dimensional machine having just one head on each of its worktapes.

Fix integers $d \geq 1$ and $e > d$, a positive real number ϵ, and a finite alphabet Δ. To establish Theorem 1.1, it suffices to exhibit an on-line simulation of a particular e-dimensional machine **E** with worktape alphabet Δ by a d-dimensional machine **D** in time $O(n^{1 + 1/d - 1/e + \epsilon})$. Machine **E** has one head on one worktape and operates in real time as follows. At each step it reads another input symbol, called a *command*, that has the form $\langle b, \delta \rangle$, where $b \in \Delta$, and δ is one of the $2e + 1$ directions in which the worktape head can shift. Suppose **E** is in a configuration in which the cell y scanned by the worktape head contains b'. When **E** then reads the input symbol $\langle b, \delta \rangle$, it writes b on y, writes b' on its output tape, and shifts the worktape head in direction δ. Call symbols of Δ *responses*. Let Σ be the set of commands for **E**. Machine **E** defines a function from Σ^* to Δ^* that maps a string of commands into a string of responses of the same length. Machine **D** *simulates* **E** *on-line* in time $T(n)$ if it computes this function in time $T(n)$ on inputs of length n, and for every input, for every j, machine **D** produces the jth response before reading the $(j + 1)$st command.

On a given input string of commands, when **E** has processed just the first τ commands, we say that **E** is *at time* τ. When **D** has processed only the first τ commands (and produced the first τ output responses), **D** is *at simulated time* τ.

Consider a string of n commands. For simplicity we describe a simulation in which n is available off-line. The simulation can be converted routinely to an on-line simulation with time loss of only a constant factor [19]. (For $n' = 1, 2, 4, 8, \ldots$, machine D repeats the simulation *ab initio* using n' for the length of the input until n' is at least as large as n.)

On a d-dimensional worktape, a *box* is a set of cells that form a d-dimensional cube. The *volume* of a box is the number of cells in it. The *base cell* of a box is the cell whose coordinates are the smallest. Cell y is *at distance* s from cell z if the shortest rectilinear path from y to z has length s; equivalently, a worktape head requires exactly s steps to move from y to z. Cell y is *well within* box C of side s if it is at distance strictly greater than $s/3$ from every cell outside C. The *relative position* of a cell y with respect to a box C is the list of coordinates of y when the base cell of C is taken as the origin; if y is at distance s from the base cell of C, then its relative position can be specified by a binary string of length proportional to $\log s$. Write $x(h, C)$ for the cell at relative position h with respect to box C. The *relative position* of a box in C is the relative position of its base cell with respect to C.

Set

$$c_0 = 8^d 5^e,$$

$$L = de(\log n)/\log c_0,$$

$$c = c_0^{1/de},$$

$$s_L = n,$$

$$s_{L-1} = n^{1/e}.$$

We may assume that c is an integer and that ϵ is sufficiently small so that

$$de \leq 1/d - 1/e + \epsilon \tag{1.1}$$

and

$$c^{d-1} > 5^e. \tag{1.2}$$

Note that $c^L = n^{1/e}$ and $c_0^{L/d} = n^e$. For each $i < L - 1$, set

$$s_i = s_{i+1}/c = c^{i+1};$$

clearly, $s_0 = c = O(1)$.

For each $i = 0, 1, \ldots, L$, the *blocks at level i* are precisely the boxes B of side $3s_i$ on the worktape of E for which every component of the coordinates of the base cell of B is an integral multiple of s_i. Fix a block B_L at level L such that the origin is well within B_L. On every input sequence of n commands the worktape head of E remains within B_L. This covering of the worktape of E enjoys two important properties:

(i) for every cell y, there are 5^e blocks at level i that contain a cell at distance s_i or less from y;

(ii) for every cell y in a block B at level i, if y is at distance strictly greater than s_{i-1} from every cell outside B, then there is a block $B' \subseteq B$ at level $i-1$ such that y is well within B'; the relative position of B' in B is easily calculated from the position of y.

Reischuk [23] employs a similar covering. The blocks at level $i-1$ that are contained in a block B at level i are the *subblocks* of B. Every block at level i has at most $(3s_i/s_{i-1})^e$ subblocks.

Let π be the function defined by

$$\pi(x) = 2^{\lceil \log x \rceil};$$

if x is not a power of 2, then π maps x to the next larger power of 2. Let $m_L{}^* = n$, and for $i < L$ let

$$m_i{}^* = (3s_i)^e,$$

the volume of a block at level i. For each i set

$$\gamma_i = (3c)^e c_0{}^i,$$
$$u_i = \pi((\gamma_i m_i{}^*)^{1/d}),$$
$$t_i = \pi((\gamma_i s_i)^{1/d}).$$

A routine calculation shows that $u_L = O(n^{1/d + \epsilon})$, and for $i < L$,

$$u_i \geq (c_0\, s_i{}^e / s_{i-1}{}^e)^{1/d}\, u_{i-1}/2. \tag{1.3}$$

In D a *page at level 0* is a box of side t_0; for $i > 0$, a *page at level i* is a box whose side is a power of 2 that has a *mass store*, a *memory map*, a *free storage list*, and a *nonblank cell counter*. If P is a page at level i and P' is a page at level $i-1$ and $P' \subseteq P$, then P' is *subpage* of P. We describe how the contents of a page P at level i *represent* the contents of a block B at level i.

If $i = 0$, then P represents the contents of B literally: for each of the $(3s_0)^e$ cells y of B there is a representative cell z in P whose relative position in P is determined by the relative position of y in B, and z holds the same symbol as y. The details of this representation are unimportant, provided that relative position of z in P can be computed from the relative position of y in B in constant time.

If $i > 0$, then for every nonblank subblock B' of B, there is a subpage of P whose contents represent the contents of B' recursively. All subpages at level $i-1$ are pairwise disjoint. Let P have side p. The mass store of P is a box of side $p/2$ in P that contains these subpages. The *address* of a box in the mass store is its relative position with respect to the mass store.

The memory map of P is a box of side $p/(4 \log s_i)$ in P. Its contents maintain the addresses of subpages of P whose contents represent the contents of subblocks of B. The relative positions of subblocks in B are specified by binary strings of length $O(\log s_i)$. Each binary string corresponds in a natural way to a leaf of a binary tree of height $O(\log s_i)$. The number of leaves of this tree is the number of subblocks of B whose contents are represented by the contents of subpages of P. We embed this binary tree in the memory map in depth-first order. For every internal node v of the binary tree, there is a *pointer box* in the memory map that contains the relative positions of the pointer boxes for the one or two direct descendants of v. If B' is a nonblank subblock of B whose relative position in B corresponds to leaf node λ of the binary tree, then the pointer box for λ contains the address of the subpage P' of P whose contents represent the contents of B'. We say that P' is *assigned* to B' and that the memory map [illegible] an address with a relative position in B. A subpage is *active* if it is assigned to some subblock.

The free storage list is a list of addresses that specify a box of side $p/4$ in P. For $q = 1, 2, 4, \ldots, p/4, p/2$, the free storage list has addresses of at most $2^d - 1$ blank boxes of side q in the mass store of P. The *free volume* of P is the total volume of the blocks whose addresses are on the free storage list.

The nonblank cell counter specifies an integer between 0 and m_i^* (inclusive). This value of the nonblank cell counter is at least as large as the number of nonblank cells of B.

Page P *represents block* B *at time* τ if its contents represent the contents of B in the configuration of E at time τ.

We present an informal overview of the simulation before giving the details. Suppose in a configuration of E the worktape head is well within block B at level i, and the contents of page P at level i represent the contents of B. Using P, procedure *SIMULATE* processes the next s_i commands in s_i/s_{i-1} phases of s_{i-1} commands each. At each phase procedure *PAGEADDRESS* uses the memory map of P to locate the subpage P' of P assigned to the subblock B' of B such that the worktape head of E is well within B'. Next, *SIMULATE* uses P' to process s_{i-1} commands recursively. Then for each subblock C' of B that meets B', procedure *UPDATE* recursively updates the contents of the subpage of P assigned to C' in order to represent the new contents of C'.

Procedure $a \leftarrow$ ***ALLOCATE***(i, r, K');

Hypothesis: The worktape heads of D are on a page P of side p at level i.

Parameters: K' is a binary string of length $O(\log s_i)$; r is a power of 2.

Value returned: The address a of a blank box of side r in the mass store of P, if the procedure call does not fail.

Effects: The contents of the memory map are altered to associate a with k', and the free volume of P is reduced by r^d. (During this call to *ALLOCATE*, the free storage list may temporarily hold addresses of 2^d blank boxes of the same side.)

Method: A buddy system is used [10].

Step 1. If the free storage list has an address of a box of side r, then skip to Step 2. Let q^* be the smallest power of 2 greater than r for which the free storage list does have an address of a box of side q^*; if no such q^* exists, then terminate with failure. For $q = q^*, q^*/2, \ldots, 4r, 2r$ in order, select an address a_q of a box C_q of side q, delete a_q from the list, and add to the list the addresses of the 2^d pairwise disjoint boxes of side $q/2$ whose union is C_q. The free storage list now has addresses of $2^d - 1$ blank boxes of sides $2r, 4r, \ldots, q^*/2$ and of 2^d blank boxes of side r.

Step 2. Let a be the address of a box of side r on the free storage list, and delete this address from the list. In the memory map of P, set up at most $O(\log s_i)$ pointer boxes of volume $O(\log u_i)$ for the binary tree (described above) to associate address a with k'. If the pointer boxes for the binary tree no longer fit in a box of side $p/(4 \log s_i)$, then terminate with failure.

Procedure $a \leftarrow$ *PAGEADDRESS* (i, k'):

Hypothesis: The worktape heads of **D** are on a page P at level i.

Parameters: k' is a binary string of length $O(\log s_i)$ that specifies the relative position of a subblock B' of a block at level i.

Value returned: The address a of subpage P' in P assigned to B' such that the side of P' is $\min\{\pi((\gamma_{i-1}(m' + s_{i-1}))^{1/d}), u_{i-1}\}$, where m' is the value of the nonblank cell counter of P'. If a call to *ALLOCATE* fails, then this call to *PAGEADDRESS* fails.

Effects: This procedure may alter the contents of the memory map and the free storage list by a call to *ALLOCATE* and may set up a new page in the mass store.

Method: Using k' and the memory map of P, retrieve the address of the subpage P' of P assigned to B': visit the $O(\log s_i)$ pointer boxes for the nodes on the path in the binary tree (described above) from the root to the leaf that corresponds to k' to obtain the address associated with k'.

If no address is associated with k', then call *ALLOCATE* to obtain a blank box of side t_{i-1} in the mass store. Initialize this box so that it becomes a subpage P' whose contents represent a block whose cells all hold blanks: the free storage list of P' contains just the address of the blank box of side $t_{i-1}/2$ in its mass store (namely, the mass store itself); the value of the nonblank cell counter of P' is 0. Return the address of P' in P as the value of a.

Let p' be the side of P' and m' be the value of its nonblank cell counter; by definition, p' is a power of 2. If $p' < \min\{\pi((\gamma_{i-1}(m' + s_{i-1}))^{1/d}), u_{i-1}\}$, then call *ALLOCATE* to obtain a new box of side $p'' = \min\{\pi((\gamma_{i-1}(m' + s_{i-1}))^{1/d}), u_{i-1}\} \geq 2p'$ in the mass store of P assigned to B'. Copy the contents of P' into this box to produce a page P'' such that if the contents of P' represent the contents of B', then the contents of P'' also represent the contents of B'; in particular, to ensure that the addresses in the memory map remain valid, copy the contents of the mass store of P', which has side $p'/2$, into the box of side $p'/2$ whose base cell is the same as the base cell of the mass store of P''. Augment the free storage list of P'' with addresses of $2^d - 1$ blank boxes of side $p''/4$ in its mass store. In this case return the address of P'' in P as the value of a.

Let h be the relative position of a cell with respect to a box C on the worktape of E and σ be a sequence of commands. Procedure *SHIFT* on input (h, σ) produces the relative position of the head of E with respect to C that results from starting on cell $x(h, C)$ and performing the shifts indicated by σ. *SHIFT* operates in time proportional to the sum of the lengths of its inputs: using e unary counters, one for each dimension to maintain the displacement of the head from $x(h, C)$, change one of these counters by ± 1 for each of the shifts in σ; finally, with e additions or subtractions, calculate the new relative position.

Procedure *UPDATE* (i, h, σ):

Hypotheses:

(i) At the beginning and end of this call to *UPDATE*, the worktape heads of D are on the base cell of a page Q at level i.

(ii) Let m be the value of the nonblank cell counter of Q at the beginning of this call; Q has side $\min\{\pi((\gamma_i(m + s_i))^{1/d}), u_i\}$.

Parameters: h is a binary string of length $O(\log s_i)$ that specifies the relative position of a cell with respect to a block C at level i. σ is a sequence of s_i commands.

Effects: If at the beginning of this procedure call Q represents C at time τ, the worktape head of E is on $x(h, C)$ at time τ, and σ is the sequence of commands at times $\tau + 1, \ldots, \tau + s_i$, then at the end of the call, Q represents C at time $\tau + s_i$. The value of nonblank cell counter of Q is set to $m' = \min\{m + s_i, m_i^*\}$; the side of Q is $\pi((\gamma_i m')^{1/d})$. If any procedure that *UPDATE* calls fails, then this call to *UPDATE* fails.

Method: If $i = 0$, then use σ to determine the new contents of every cell y in C that is visited by the worktape head of **E** when it starts from $x(h, C)$ and shifts according to σ; copy this new symbol into the representative of y in Q.

Otherwise, if $i > 0$, then set $k \leftarrow h$; add s_i to the value of the nonblank cell counter, unless it already equals m_i^*; partition σ into s_i/s_{i-1} consecutive subsequences σ' of length s_{i-1}; and perform Step 1 through Step 3 for each σ', in order.

Step 1. For each of the at most 5^e subblocks C' of C that contains a cell within distance s_{i-1} of $x(k, C)$, perform Steps 1.1 and 1.2.

Step 1.1. Call *PAGEADDRESS* to determine the address of the subpage Q' of Q assigned to C'. Let h' be the relative position of $x(k, C)$ with ecspect to C'.

Step 1.2. Move the heads of **D** to the base cell of Q' and call $UPDATE(i-1, h', \sigma')$.

Step 2. Set $k \leftarrow SHIFT(k, \sigma')$.

Step 3. Return the heads of **D** to the base cell of Q.

Correctness: To check that *UPDATE* operates properly, show by induction that for each j, at the jth execution of Step 3, σ' is the sequence of commands at times $\tau + (j-1)s_{i-1} + 1, \ldots, \tau + j\,s_{i-1}$, the worktape head of **E** is on $x(k, C)$ at time $\tau + j\,s_{i-1}$, and Q represents C at time $\tau + j\,s_{i-1}$.

Procedure $\sigma \leftarrow SIMULATE(i, h)$:

Hypotheses:

(i) At the beginning and end of this call to *SIMULATE*, all heads of **D** are on the base cell of a page P at level i.

(ii) At the beginning of this call let **D** be at simulated time τ and let m be the value of the nonblank cell counter of P; page P has side $\min\{\pi((\gamma_i(m + s_i))^{1/d}), u_i\}$.

(iii) At the beginning of this call, P represents block B at level i at time τ. At time τ the worktape head of **E** is on $x(h, B)$, which is well within B. (Consequently, the worktape head of **E** is in B at times $\tau + 1, \ldots, \tau + s_i$.)

Parameters: h is a binary string of length $O(\log s_i)$ that specifies the relative position of a cell in B.

Value returned: σ is the sequence of commands at times $\tau + 1, \ldots, \tau + s_i$. If any procedure that *SIMULATE* calls fails, then this call to *SIMULATE* fails.

Effects: During this procedure call, the next s_i commands are read, the corresponding responses are produced, and the contents of P are [illegible]. The value of the nonblank cell counter of P is set to $m' = \min\{m + s_i, m_i^*\}$: the side of P is $w((\gamma_i m')^{1/d})$. At the end of this call, P represents B at time $\tau + s_i$.

Method: If $i = 0$, then simulate directly in P for $s_0 = c$ steps, reading s_0 commands and producing the appropriate responses. Let σ be the sequence of s_0 commands read.

Otherwise, if $i > 0$, then set $k \leftarrow k$; add s_i to the value of the nonblank cell counter, unless it already equals m_i^*; and perform Step 1 through Step 7 exactly s_i/s_{i-1} times [illegible] Step 8.

Step 1. Use k to calculate the relative position of the subblock B' of B such that $x(k, B)$ is well within B'; property (ii) of the definition of the blocks [illegible]. Calculate the relative position k' of $x(k, B)$ with respect to B'.

Step 2. Call *PAGEADDRESS* to retrieve the address of the subpage P' of P assigned to B'.

Step 3. Move the worktape heads of D to the base cell of P' and call *SIMULATE*$(i - 1, k')$ to process the next s_{i-1} commands. Let σ' be the sequence of commands [illegible] by this call.

Step 4. For each of the at most $3^d + 1$ subblocks C of B [illegible] C contains a cell at distance s_{i-1} or less from $x(k, B)$, perform Steps 4.1 and 4.2.

Step 4.1. Call *PAGEADDRESS* to determine the address of the subpage Q' of P assigned to C'. Let k' be the relative position of $x(k, B)$ with respect to C'.

Step 4.2. Move the heads of D to the base cell of Q' and call *UPDATE*$(i - 1, k', \sigma')$.

Step 5. Use σ' to update k: set $k \leftarrow$ *SHIFT*(k, σ').

Step 6. Append σ' to σ.

Step 7. Return the heads of D to the base cell of P.

Correctness: To verify that *SIMULATE* operates properly, establish by induction that for each j, at the jth execution of Step 7, P represents B at time $\tau + js_{i-1}$ and the worktape head of E is on $x(k, B)$ at time $\tau + js_{i-1}$.

To simulate **E** on an input string of n commands, move the worktape heads of **D** to the base cell of the page P_L of side u_L that represents B_L at time 0, and call *SIMULATE* with parameters (L, h_0), where h_0 is the relative position of the origin with respect to B_L.

1.3. Analysis of the Simulation

We prove that every call to *SIMULATE* completes successfully; the proof for *UPDATE* is the same.

Lemma 1.1. Let r be a power of 2. Suppose the free volume of page P is at least r^d in a configuration of **D** at the beginning of a call to *ALLOCATE* on P. Then this call can produce the address of a blank box of side r in the mass store of P.

Proof. Let $q_1 \le q_2 \le ... \le q_m$ be the sides of boxes whose addresses are on the free storage list of P. Since the free storage list has at most $2^d - 1$ boxes of each distinct side, the free volume v of P satisfies

$$v = q_m{}^d + ... + q_1{}^d \le (2^d - 1)q_m{}^d + (2^d - 1)(q_m/2)^d + ... + (2^d - 1)(1)^d < (2q_m)^d.$$

If $r^d \le v$, then $r^d < (2q_m)^d$, hence since r is a power of 2, $r \le q_m$. Consequently, *ALLOCATE* can find a blank box of side r in the mass store of P. ∎

Let **D** be at simulated time τ at the beginning of a call to *SIMULATE* on a page P at level $i > 0$ that represents block B at time τ. Let m be the value of the nonblank cell counter of P in this configuration and $m' = \min\{m_i^*, m + s_i\}$. The side of P is $\pi((\gamma_i m')^{1/d})$.

Lemma 1.2. Throughout this call to *SIMULATE*

(i) P has at most $5^e m'/s_{i-1}$ active subpages, and

(ii) the total of the nonblank cell counters of the active subpages of P never exceeds $5^e m'$.

Proof. First, suppose $m' = m_i^*$. Since B has at most $(3s_i/s_{i-1})^e$ subblocks, P has at most $(3s_i/s_{i-1})^e \le m_i^*/s_{i-1} \le 5^e m'/s_{i-1}$ active subpages, and the sum of their nonblank cell counters is at most

$$(3s_i/s_{i-1})^e\, m_{i-1}{}^* = 3^e m_i^* \le 5^e m'.$$

Now suppose $m' = m + s_i$. By induction on τ, at simulated time τ, P has at most $5^e m/s_{i-1}$ active subpages, and the total of their nonblank cell counters is at most $5^e m$. At each of s_i/s_{i-1} iterations during the execution of *SIMULATE*, at most 5^e new active subpages are created, and s_{i-1} is added to the nonblank cell counters of at most 5^c subpages. Thus, the number of active subpages of P is always at most

$$5^c m/s_{i-1} + 5^c(s_i/s_{i-1}) = 5^c m'/s_{i-1}.$$

The total of the nonblank cell counters of the active subpages of P is at most

$$5^c m + 5^c(s_i/s_{i-1})s_{i-1} = 5^c m'. \blacksquare$$

We show that during this call to *SIMULATE*, every call to *PAGEADDRESS* in Step 2 or Step 4.1 completes successfully. First, we verify that P has enough space for the free storage list and the memory map. Since the side of P is $\pi((\gamma_i m')^{1/d}) \leq \pi((\gamma_i m_i^*)^{1/d}) = u_i$, the relative position of a box in P can be specified by a string of $O(\log u_i)$ symbols. (By choosing the size of the worktape alphabet of **D**, we can adjust the constant of proportionality to ensure that assertions (1.4) and (1.5) below are true.) The free storage list of P comprises $O(\log u_i)$ addresses of length $O(\log u_i)$; thus, the free storage list occupies a box of side

$$O((\log u_i)^{2/d}) \leq O(t_i^{1/d}) \leq \pi((\gamma_i m')^{1/d})/4 \tag{1.4}$$

because $m' \geq t_i$. In the memory map of P there are $O(\log s_i)$ pointer boxes of fixed volume $O(\log u_i)$ for each of the $O(m'/s_{i-1})$ active subpages of P. These pointer boxes fit in a box of volume

$$O((m'/s_{i-1})(\log s_i)(\log u_i)) \leq (\pi((\gamma_i m')^{1/d})/(4 \log s_i))^d, \tag{1.5}$$

the volume of the memory map of P. When *PAGEADDRESS* calls *ALLOCATE*, this call cannot fail for lack of space for the memory map.

Consider a configuration of **D** just before a call to *PAGEADDRESS* on P in Step 2 or Step 4.1 between simulated time τ and simulated time $\tau + s_i$. In this configuration let $P_1, P_2, \ldots$ be the active subpages of P and let $m_1', m_2', \ldots$ be the values of their nonblank cell counters; let P_j represent subblock B_j in B. The side of P_j is $\pi((\gamma_{i-1} m_j')^{1/d})$. The mass store of P holds the contents of smaller subpages that were assigned to B_j in previous configurations. Because the sides of these smaller subpages are powers of 2, their total volume is at most the volume of P_j, namely, $(\pi((\gamma_{i-1} m_j')^{1/d}))^d$. Consequently, the volume of used boxes in the mass store of P in this configuration is bounded by

$$\Sigma_j \, 2(\pi((\gamma_{i-1} m_j')^{1/d}))^d.$$

Suppose *PAGEADDRESS* decides to assign to B_1 a new page of side $\pi((\gamma_{i-1} m_1'')^{1/d})$, where $m_1'' = m_1' + s_{i-1}$; according to the definition of *PAGEADDRESS*,

$$\pi((\gamma_{i-1} m_1'')^{1/d}) \geq 2\, \pi((\gamma_{i-1} m_1')^{1/d}). \tag{1.6}$$

Lemma 1.2(ii) implies that

$$m_1'' + \Sigma_{j>1} m_j' \leq 5^c m'. \tag{1.7}$$

Because the mass store has side $\pi((\gamma_i m')^{1/d})/2$, the free volume of P in this configuration is at least

$$
\begin{aligned}
&(\pi((\gamma_i m')^{1/d})/2)^d - \Sigma_j\, 2(\pi((\gamma_{i-1}\, m_j')^{1/d}))^d \\
&\qquad \geq 4^d 5^e\, \gamma_{i-1}\, m' - 2(\pi((\gamma_{i-1}\, m_1'')^{1/d})/2)^d - 2\, \Sigma_{j>1}\, (2^d\, \gamma_{i-1}\, m_j') \quad \text{by (1.6)} \\
&\qquad \geq 4^d\, \gamma_{i-1}\, (5^e m') - 4^d\, \gamma_{i-1}\, (\Sigma_{j>1}\, m_j') - 2\gamma_{i-1}\, m_1'' \\
&\qquad \geq 4^d\, \gamma_{i-1}\, m_1'' - 2\, \gamma_{i-1}\, m_1'' \quad \text{by (1.7)} \\
&\qquad \geq (\pi((\gamma_{i-1}\, m_1'')^{1/d}))^d.
\end{aligned}
$$

Lemma 1.1 guarantees that *ALLOCATE* can find a blank box of side $\pi((\gamma_{i-1}\, m_1'')^{1/d})$ in the mass store of P. Therefore, this call to *PAGEADDRESS* completes successfully.

By induction on i, neither the recursive calls to *SIMULATE* nor the calls to *UPDATE* fail. Ergo, the call to *SIMULATE* at simulated time τ completes successfully.

In the memory map of a page at level i of side $p \leq u_i$, to move a head from one pointer box to another takes time proportional to $p/(4 \log s_i)$, the side of the memory map. Thus, to determine the address of the subpage assigned to a subblock or to associate an address with the relative position of a subblock takes time

$$(O(\log u_i) + O(p/(4 \log s_i)))O(\log s_i) = O(u_i)$$

because $O(\log s_i)$ pointer boxes, each of volume $O(\log u_i)$, are accessed.

Let $T_A(i)$ be the time used by *ALLOCATE* on a page at level i. Since time $O(u_i)$ is consumed in moving the heads around the page and in the memory map and time $O((\log u_i)^2)$ in handling the addresses in the free storage list,

$$T_A(i) = O(u_i) + O((\log u_i)^2) = O(u_i).$$

Let $T_P(i)$ be the time used by *PAGEADDRESS* on a page at level i, ***excluding the copying of subpages***. This procedure retrieves an address from the memory map (time $O(u_i)$), performs some arithmetic calculations (time $O(\log u_i)$), moves heads around the mass store (time $O(u_i)$), and calls *ALLOCATE*:

$$T_P(i) = O(\log u_i) + O(u_i) + T_A(i) = O(u_i).$$

Let $T_U(i)$ be the time used by *UPDATE* on pages Q at level $i < L$, excluding the copying of subpages in calls to *PAGEADDRESS*. Evidently, $T_U(0) = O(1)$. For $i > 0$ we assess the time taken by each Step.

Step 1.1: $O(\log s_i) + T_P(i)$ for each iteration of Step 1 [illegible] for the call to *PAGEADDRESS.*

Step 1.2: $O(u_i) + T_S(i-1)$ for each [illegible] to move the heads across Q, whose side is at most u_i, and $T_S(i-1)$ for the recursive call.

Step 2: $O(\log s_i) + O(s_{i+1})$ for the call to [illegible]

Step 3: $O(u_i)$ for each iteration.

Thus, by induction on i, (1.2), and (1.3), there exist constants k_1 and k_2 such that

$$T_S(i) \le (s/s_{i+1}) [\text{illegible}]$$

$$\le c(k_1 u_i + 5^d k_2 u_{i+1})$$

$$\le (k_1 c + 5^d k_2 c\, [\text{illegible}]^{1/d}) u_i$$

$$\le (k_1 c + 5^d k_2 c^{1-\alpha/d}) u_i$$

$$\le k_2 u_i$$

Let *SIMULATE* run within time $T_S(i)$ on every page at level i, excluding the time for the copying of subpages in *PAGEADDRESS.* Clearly, $T_S(0) = O(1)$. For $i > 0$ we reckon the time used by each Step.

Step 1: $O(\log s_i)$ for each iteration.

Step 2: $T_A(i)$ for each iteration.

Step 3: $O(u_i) + T_S(i-1)$ for each iteration — $O(u_i)$ to move the heads and $T_S(i-1)$ for the recursive call.

Step 4.1: $O(\log s_i) + T_P(i)$ for each iteration of Step 4.

Step 4.2: $O(u_i) + T_S(i-1)$ for each iteration of Step 4.

Step 5: $O(\log s_i) + O(s_{i+1})$ for each iteration.

Step 6: $O(s_{i+1})$ for each iteration.

Step 7: $O(u_i)$ for each iteration.

Then

$$T_S(i) \le (s/s_{i+1}) (O(\log s_i) + O(s_{i+1}) + O(u_i) + O(T_A(i)) + O(T_P(i-1)) + T_S(i-1))$$

$$\le (s/s_{i+1}) (O(u_i) + T_S(i-1))$$

If $i < L$, then by induction on i and (1.3), there exist constants k_3 and k_4 such that

$$T_S(i) \le c(k_3 u_i + k_4 u_{i+1})$$

$$\le (k_3 c + k_4 c\,(s_{i+1}{}^d / c_0\, s_i^d)^{1/d}) u_i$$

$$\le (k_3 c + k_4 c^{1-\alpha/d}) u_i$$

$$\le k_4 u_i$$

We prove that during the simulation of **E** by **D**, the total time taken by copying the contents of pages in calls to procedure *PAGEADDRESS* is $O(n^{1+de})$. Consider a configuration of **D** at simulated time τ at the end of a call to *SIMULATE* on a page P at level i of volume ν. Let $Q_1, Q_2, \ldots, Q_f = P$ be the sequence of pages such that for each $j > 1$, prior to simulated time τ, *PAGEADDRESS* called *ALLOCATE* to obtain Q_j and copied the contents of Q_{j-1} into Q_j. Call the sum over j of the time for copying Q_{j-1} into Q_j the *ancestral copying time* for P. The sides of these Q_j are increasing powers of 2. Consequently, since the time to copy the contents of Q_{j-1} into Q_j is bounded above by the volume of Q_j, the ancestral copying time for P is at most twice the volume of P, namely, 2ν. In this configuration of **D** let $P_1, P_2, \ldots$ be the active subpages of P and $\nu_1, \nu_2, \ldots$ be their volumes. Since these pairwise disjoint subpages lie in the mass store of P, whose volume is $\nu/2^d$,

$$\Sigma_j \; \nu_j \leq \nu/2^d.$$

Call a page P' in P at any level *active* if either P' is an active subpage of P (at level $i-1$) or P' is active in an active subpage of P. Suppose inductively that there is a constant $k_5 \geq 4$ such that for each j, the sum of the ancestral copying times for all active pages in P_j over all levels is at most $k_5\nu_j$. Then the sum of the ancestral copying times for all active pages in P over all levels is at most

$$2\nu + \Sigma_j \; k_5\nu_j \leq 2\nu + k_5\nu/2^d \leq k_5\nu.$$

In particular, when $P = P_L$, the page at level L assigned to B_L, this total copying time is at most $k_5 {u_L}^d = O(\gamma_L \, n) = O(n^{1+de})$.

Therefore, by (1.1), the simulation uses time

$$\begin{aligned} T_S(L) + O(n^{1+de}) &\leq (s_L/s_{L-1})(O(u_L) + T_S(L-1)) + O(n^{1+de}) \\ &\leq n^{1-1/e}(O(n^{1/d+\epsilon}) + O(u_{L-1})) + O(n^{1+de}) \\ &\leq O(n^{1+1/d-1/e+\epsilon}). \end{aligned} \tag{1.8}$$

Theorem 1.2. For all $d \geq 1$ and all $e > d$, every multihead e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line by a multihead d-dimensional Turing machine in time $O(T(n)^{1+1/d-1/e+O((\log T(n))^{-1/2})})$.

Proof. The constant of proportionality in (1.8) can be bounded by $k_6 {k_7}^{1/\epsilon}$, where k_6 and k_7 depend only on d and e. Choose ϵ as a function of n to minimize

$$k_6 {k_7}^{1/\epsilon} n^{1+1/d-1/e+\epsilon};$$

$\epsilon = O((\log n)^{-1/2})$. Ergo, every e-dimensional Turing machine of time complexity $T(n)$ can be simulated on-line by a d-dimensional machine in time $O(T(n)^{1+1/d-1/e+O((\log T(n))^{-1/2})})$. ∎

Chapter 2. A Space Bound for One-Tape Multidimensional Turing Machines

2.1. Introduction

It is generally believed that the computational resources time and space can be exchanged for each other. For instance, a program that saves space (storage) by compressing data spends extra time encoding the data and decoding the stored representation. Some data structures use minimum space, but require long access times; others reduce access times by occupying large amounts of memory. Quantitative tradeoffs have been established between time and space for multitape Turing machines [7] and for straight-line programs [21, 26, 29].

Recently, Paul and Reischuk [18, 23] proved that the tradeoff of Hopcroft, Paul, and Valiant [7] is not an artifact of the linearity of the Turing machine tapes: every deterministic multitape multidimensional Turing machine of time complexity $T(n)$ can be simulated by a deterministic Turing machine of space complexity $T(n)\, c^{\log^* T(n)}/\log T(n)$ for some constant c. We derive a space bound for a restricted class of multidimensional Turing machines: for every nondeterministic d-dimensional machine M with one worktape head that runs in time $T(n)$, there is a deterministic Turing machine M' such that M' accepts the same language as M in space $(T(n) \log T(n))^{d/(d+1)}$, provided that $T(n)$ is constructible in space $(T(n) \log T(n))^{d/(d+1)}$.

Section 2.2 introduces definitions, including a generalization of crossing sequences. Section 2.3 describes a deterministic simulation of a nondeterministic d-dimensional machine M with just one worktape head, and Section 2.4 proves that this simulation uses space $(T(n) \log T(n))^{d/(d+1)}$ when M runs in time $T(n)$. (All logarithms are taken to base 2.) The simulation and proof generalize Paterson's [15] for the case $d = 1$.

2.2. Definitions

Fix a finite alphabet Σ and a positive integer d. A *worktape* over Σ is a set of *cells*, each of which can contain a symbol in Σ. A worktape is *d-dimensional* if its cells are in bijective correspondence with Z^d, the set of d-tuples of integers. For every x in Z^d there is a unique worktape cell $C(\mathrm{x})$ at *location* x. Location $(x_1, ..., x_d)$ is *adjacent* to locations $(x_1 \pm 1, x_2, ..., x_d)$, $(x_1, x_2 \pm 1, ..., x_d)$, ..., and $(x_1, x_2, ..., x_d \pm 1)$. In Z^d let $\mathrm{e}_0 := (0, 0, ..., 0)$. A *box* B is a subset of Z^d comprising the d-tuples

$$[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$$

for some integers $a_1, b_1, \ldots, a_d, b_d$. The *boundary* of B is the subset of locations $(x_1, \ldots, x_d)$ such that for some i either $x_i = a_i$ or $x_i = b_i$. The *volume* of B, denoted $|B|$, is the number of locations that it comprises. A *content function* on a box B is a map from B to Σ; such a function specifies the contents of cells whose locations are in B.

A *d-dimensional Turing machine* (with alphabet Σ) has a d-dimensional worktape on which the worktape head can move one cell along any of the d orthogonal dimensions in either positive or negative direction at each step; if the head reads cell $C(\mathbf{x})$ at step s, then at step $s+1$ it reads a cell at a location adjacent to $\mathbf{x}$. In each cell the worktape head can write a symbol from Σ. The input to the machine is presented on a two-way read-only input tape. Initially, at step 0, the worktape is completely blank, the input head is positioned on the leftmost symbol of the input word, and the worktape head reads cell $C(\mathbf{e}_0)$.

Let M be a nondeterministic d-dimensional Turing machine (with one worktape head) that runs in time $T(n)$ on inputs of length n: for every word of length n that M accepts there is an accepting computation of at most $T(n)$ steps. Assume that M reads all of its input — $T(n) \geq n$ — and that $T(n)$ is constructible in space $(T(n) \log T(n))^{d/(d+1)}$. The worktape head remains on cells whose locations are in the box

$$B_0(n) := [-T(n), T(n)] \times [-T(n), T(n)] \times \ldots \times [-T(n), T(n)].$$

We may assume without loss of generality that to accept an input word, M halts with its worktape entirely blank, its worktape head positioned on $C(\mathbf{e}_0)$, and its input head on the leftmost symbol of the input word. (If necessary, M can be modified to erase its worktape by depth-first search on the cells that it has visited; the modified machine runs in time $O(T(n))$.) For the remainder of this chapter we consider the computation(s) of M on a fixed input word of length n.

A *partial configuration* π on a box B consists of

- a content function σ_π on B,
- a state,
- a step number,
- a position on the input tape, and
- a worktape cell location $\mathbf{x}_\pi$ such that either $\mathbf{x}_\pi \in B$ or $\mathbf{x}_\pi = \perp$ (unspecified).

Let ι_0 be the partial configuration on $B_0(n)$ that specifies the initial configuration on M at step 0. For

$t \geq 1$, let α_t be the partial configuration on [illegible] that specifies the (unique) accepting configuration of M.

Suppose B and B' are boxes such that $B' \subseteq B$. Let α be a partial configuration on B and ρ be a partial configuration on B' [illegible]

(i) α_ρ is the restriction of α_w to B'; and

(ii) if $x_w \in B'$, then [illegible]

Then ρ is the restriction of α to B' [illegible]

For partial configurations α and ρ on box B [illegible] if the step number of ρ is larger than that of α by exactly 1 and one of the following holds:

(i) [illegible] when M is started in the state [illegible] and its input head at the input position of w, it may write $\alpha_\rho(x_w)$ on $C(x_w)$, change its state to the state of ρ, and move its worktape head to [illegible] of ρ. (Thus, α_w and α_ρ differ only on [illegible]

(ii) x_w is on the boundary of B [illegible] by a single transition of M in which the [illegible]

(iii) $x_w = \bot$ and x_ρ is on the boundary of B and $\alpha_w = \alpha_\rho$

(iv) $x_w = \bot$ [illegible]

Call (ii) an *exit transition* and (iii) an *entry transition*.

Crossing sequences. [illegible] dimensional Turing machines. [illegible] of M, a *crossing event* from box B_1 to box B_2 occurs at step s [illegible] cell $C(x_1)$ at location x_1 in B_1 to a cell $C(x_2)$ at adjacent location x_2 in B_2 at the end of step s. The *crossing record* for a crossing event specifies s, x_1, x_2, the state of the machine, [illegible] at the end of step s. Call s the *step number* of the crossing record. This record [illegible] B_1 into B_2. Let R be a set of crossing records. The *earliest* record of R has the smallest step number; the *latest* record has the largest step number. The *restriction* of R to a box B, written $R \setminus B$, comprises precisely the records of R for which $x_1 \in B$ or $x_2 \in B$; the restriction of R to [illegible] $[s_1, s_2]$, written $R \setminus [s_1, s_2]$, is the subset of records for which $s \in [s_1, s_2]$.

Let α and ρ be partial configurations [illegible] sequence of partial configurations $\pi = \pi_0, \pi_1, ..., \pi_k = \rho$ on B such that $\pi_i \vdash \pi_{i+1}$ for each i. A set of crossing records that enter or exit B can specify the entry and exit transitions of a partial computation. Let R be a set of crossing records that enter or exit B. The triple (π, ρ, R) is *compatible* if there is a partial computation from π to ρ for which R specifies the entry and exit transitions. Define the predicate **Comp** (π, ρ, R) to be true if and only if (π, ρ, R) is compatible.

Call (π, ρ, R) *consistent* if either (i) or (ii) holds:

(i) $R = \emptyset$ and either $x_\pi = \perp = x_\rho$ or both $x_\pi \in B$ and $x_\rho \in B$;

(ii) (a) $R \neq \emptyset$;

(b) the records in R, strictly ordered by step number, alternate between records that enter B and records that exit B;

(c) if $x_\pi \in B$, then the earliest record in R exits B; if $x_\pi = \perp$, then the earliest record in R enters B; and

(d) if $x_\rho \in B$, then the latest record in R enters B; if $x_\rho = \perp$, then the latest record in R exits B.

When (π, ρ, R) is compatible, (π, ρ, R) is necessarily consistent.

Define a predicate for a box B, a positive integer t, and a set of crossing records R:

$$\textbf{Blank-Comp}\,(B, t, R) := \textbf{Comp}\,(\iota_0, \alpha_t \backslash B, R).$$

Machine M accepts the input word if and only if **Blank-Comp** $(B_0(n), t, \emptyset)$ is true for some t.

2.3. Simulation

To determine whether M accepts its input word, deterministic Turing machine M' checks whether **Blank-Comp** $(B_0(n), t, \emptyset)$ is true by repeatedly partitioning the box $B_0(n)$ and the step interval $[1, t]$. Using a balanced divide-and-conquer method, M' introduces either a set of crossing records or a partial configuration to ascertain recursively whether a partial computation on a box exists. The consistency condition ensures that partial computations on two boxes can be combined.

Lemma 2.1, which is straightforward to prove, guarantees that for each box, there is some partition into two boxes that induces a small number of crossing events. To simplify our arguments, we neglect to distinguish among z, $\lfloor z \rfloor$, and $\lceil z \rceil$ for real numbers z; one can justify this simplification routinely.

Lemma 2.1. Let B be a box with volume $v = |B|$. Let $s_2 \geq s_1$ be steps during a computation of M and $s = s_2 - s_1$. There is a partition of B into two boxes B_1 and B_2 such that

(i) the number of crossing events between B_1 and B_2 during $[s_1+1, s_2]$ is at most $3s/v^{1/d}$, and

(ii) B_1 and B_2 have volumes between $v/3$ and $2v/3$.

We describe the simulating machine M' informally. It is not difficult to verify that M' correctly simulates M.

When a procedure is invoked, it is constrained to operate within an amount of space determined by the calling procedure. If this amount of space is insufficient, then the invoked procedure reports a failure to the caller. Both procedures *BLANK-COMP* and *COMP* run strategies in "*parallel space*" with space bounds. For $S = 1, 2, 3, \ldots$, they give each strategy S cells to execute. If one strategy completes successfully (without failure of one of its procedure calls), then the value that it computes is the value returned.

MAIN PROGRAM FOR M'

For $t = 1, \ldots, T(n)$ calculate *BLANK-COMP* $(B_0(n), t, \emptyset)$ with space bound $(T(n) \log T(n))^{d/(d+1)}$. If *BLANK-COMP* $(B_0(n), t, \emptyset)$ completes successfully and is true for some t, then accept the input word. Otherwise, reject the input word.

Procedure *BLANK-COMP* (B, t, R):

Inputs: Box B, positive integer t, set of crossing records R that enter or exit B.

Output: The value of **Blank-Comp** (B, t, R).

Assumption: There is a space bound for this procedure call.

Method: Let $v = |B|$. Run the following two strategies in parallel space with space bounds. If this invocation of *BLANK-COMP* runs out of space, then report a failure.

Strategy B1: Return the value of *COMP* $(\iota_0, \alpha_t \backslash B, R)$.

Strategy B2: If $(\iota_0, \alpha_t \backslash B, R)$ is not consistent, then return *false*. Iterating through all partitions of B into two boxes B_1, B_2 with volumes between $v/3$ and $2v/3$ and through all sets R' of at most $3t/v^{1/d}$ crossing records for crossing events between B_1 and B_2, search for B_1, B_2, and R' for which both *BLANK-COMP* $(B_1, t, (R \cup R') \backslash B_1)$ and *BLANK-COMP* $(B_2, t, (R \cup R') \backslash B_2)$ are true. If suitable B_1, B_2, and R' are found, then return *true*; otherwise, return *false*.

Procedure $COMP(\pi_1, \pi_2, R)$:

Inputs: Partial configurations π_1 and π_2 on the same box B, set of crossing records R that enter or exit B.

Output: The value of **Comp** (π_1, π_2, R).

Assumption: There is a space bound on this procedure call.

Method: Let $v = |B|$ and $r = |R|$; let s_1 be the step number of π_1 and s_2 be the step number of π_2 and $s = s_1 - s_2$. Verify that (π_1, π_2, R) is consistent; if it is not, then return *false*. If $v = 1$, then return *true* if (π_1, π_2, R) is compatible on the one cell whose location is in B, *false* if not. If $s = 1$, then return *true* if $\pi_1 \vdash \pi_2$ and, if this is an entry or exit transition, R specifies the transition; otherwise, return *false*. For larger v and s, run the following three strategies in parallel space with space bounds. If this invocation of $COMP$ runs out of space, then report a failure.

Strategy C1: Reduce r. Determine a step s' at which $|R\backslash[s_1+1, s']| = r/2$. Enumerating all partial configurations π' on B, search for π' with step number s' such that both $COMP(\pi_1, \pi', R\backslash[s_1+1, s'])$ and $COMP(\pi', \pi_2, R\backslash[s'+1, s_2])$ are true. Return *true* if an appropriate π' is found, *false* if not.

Strategy C2: Reduce s. Set $s' = (s_1 + s_2)/2$. As in Strategy C1, search for π' with step number s' such that both $COMP(\pi_1, \pi', R\backslash[s_1+1, s'])$ and $COMP(\pi', \pi_2, R\backslash[s'+1, s_2])$ are true.

Strategy C3: Reduce v. Enumerating all partitions of B into two boxes B_1, B_2 with volumes between $v/3$ and $2v/3$ and through all sets R' of at most $3s/v^{1/d}$ crossing records for crossing events between B_1 and B_2, search for B_1, B_2, and R' for which both $COMP(\pi_1\backslash B_1, \pi_2\backslash B_1, (R \cup R')\backslash B_1)$ and $COMP(\pi_1\backslash B_2, \pi_2\backslash B_2, (R \cup R')\backslash B_2)$ are true. If suitable B_1, B_2, and R' are found, then return *true*; otherwise, return *false*.

2.4. Analysis of the Simulation

We show that M' uses space $O((T(n) \log T(n))^{d/(d+1)})$. The amount of space used by procedures $COMP$ and $BLANK\text{-}COMP$ is dominated by the storage required for the input parameters.

Since every location of the d-dimensional worktape can be specified by a list of d integers written in binary, each box B in $B_0(n)$ can be specified in space $O(\log T(n))$. A content function on a box of volume v requires space proportional to v to store. Thus, each partial configuration can be stored in space $O(v + \log T(n))$. Since each crossing record can be stored in space $O(\log T(n))$, a set of r crossing records can be stored in space $O(r \log T(n))$.

Let $S_C(v, r, s)$ denote the space required by *COMP* to run successfully on all inputs (π_1, π_2, R) such that π_1 and π_2 are partial configurations on a box B with step numbers s_1 and s_2 for which $v = |B|$, $r = |R|$, and $s = s_1 - s_2$. The definition of *COMP* implies

$$S_C(v, r, s) \leq k_1(v + (r + 1) \log T) + \min \{S_C(v, r/2, s), S_C(v, r, s/2), S_C(2v/3, r + 3s/v^{1/d}, s)\}$$

for a constant k_1. Similarly, let $S_B(v, r)$ denote the maximum space required by *BLANK-COMP* on inputs (B, t, R) for which $v = |B|$ and $r = |R|$. The definition of *BLANK-COMP* implies

$$S_B(v, r) \leq k_2(r + 1) \log T + \min \{S_C(v, r, T), S_B(2v/3, r + 3T/v^{1/d})\}$$

for a constant k_2.

Fix $\delta := d/(d + 1)$ and

$$k_3 := 2 \times 18^{\delta}. \tag{2.1}$$

Choose constants k_4, k_5, k_6, and k_7 such that

$$k_4 \geq 12k_1, \tag{2.2}$$

$$k_4 \geq k_1 k_3 + k_4/2^{\delta}, \tag{2.3}$$

$$k_5 \geq (2/3)^{1/d}(k_5 + 3), \tag{2.4}$$

$$k_6 \leq (3/2)^{1/d} k_6 - k_2 k_5, \tag{2.5}$$

$$k_7 \geq 4k_4 + (k_2 + k_4)k_5 + k_6. \tag{2.6}$$

Lemma 2.2. $S_C(v, r, s) \leq k_4(v + (r + 1 + \log s) \log T + (s \log T)^{\delta})$.

Proof. By induction on (v, r, s), in lexicographic order. If $v = 1$ or $s = 1$, then *COMP* uses only the space occupied by the inputs, $k_1(v + (r + 1) \log T)$ space. Otherwise, there are four cases.

Case 1: $v \leq (r + 1) \log T$ and $r \geq 1$. Then

$$\begin{aligned} S_C(v, r, s) &\leq k_1(v + (r + 1) \log T) + S_C(v, r/2, s) \\ &\leq (2k_1(r + 1) + k_4 r/2) \log T + k_4(v + (1 + \log s) \log T + (s \log T)^{\delta}) \\ &\leq k_4 r \log T + k_4(v + (1 + \log s) \log T + (s \log T)^{\delta}) \end{aligned}$$

because k_4 satisfies (2.2) and $r \geq 1$.

Case 2: $v \leq \log T$ and $r = 0$. By (2.2) again,

$$
\begin{aligned}
S_C(v, 0, s) &\leq k_1(v + \log T) + S_C(v, 0, s/2) \\
&\leq 2k_1 \log T + k_4(1 + \log (s/2)) \log T + k_4(v + (s \log T)^{\delta}) \\
&\leq k_4 \log T + k_4(\log s) \log T + k_4(v + (s \log T)^{\delta}).
\end{aligned}
$$

Case 3: $v + (r + 1) \log T \leq k_3(s \log T)^{\delta}$. Use (2.3) to establish that

$$
\begin{aligned}
S_C(v, r, s) &\leq k_1(v + (r + 1) \log T) + S_C(v, r, s/2) \\
&\leq (k_1 k_3 + k_4/2^{\delta})(s \log T)^{\delta} + k_4(v + (r + 1 + \log (s/2)) \log T) \\
&\leq k_4(s \log T)^{\delta} + k_4(v + (r + 1 + \log s) \log T).
\end{aligned}
$$

Case 4: $v \geq (r + 1) \log T$ and $v + (r + 1) \log T \geq k_3(s \log T)^{\delta}$. In this case,

$$k_3(s \log T)^{\delta} \leq 2v,$$

hence since $\delta = d/(d + 1)$,

$$(s \log T)/v^{1/d} \leq (2/k_3)^{1/\delta} v. \tag{2.7}$$

Therefore,

$$
\begin{aligned}
S_C(v, r, s) &\leq k_1(v + (r + 1) \log T) + S_C(2v/3, r + 3s/v^{1/d}, s) \\
&\leq (2k_1 + 2k_4/3)v + k_4((r + 1 + 3s/v^{1/d} + \log s) \log T + (s \log T)^{\delta}) \\
&\leq (2k_1 + 2k_4/3 + 3k_4(2/k_3)^{1/\delta})v + k_4((r + 1 + \log s) \log T + (s \log T)^{\delta}) \\
&\leq k_4 v + k_4((r + 1 + \log s) \log T + (s \log T)^{\delta})
\end{aligned}
$$

by (2.7), (2.1), and (2.2). ∎

Lemma 2.3. If $(T \log T)^{\delta} \leq v$ and $r + 1 \leq k_5 T/v^{1/d}$, then

$$S_B(v, r) \leq k_7(T \log T)^{\delta} + k_4(\log T)^2 - k_6(T \log T)/v^{1/d}.$$

Proof. By induction on v. There are two cases.

Case 1: $v \leq 3(T \log T)^{\delta}$. According to the hypotheses,

$$(r + 1) \log T \leq k_5(T \log T)/(T \log T)^{\delta/d} = k_5(T \log T)^{\delta}. \tag{2.8}$$

Lemma 2.2, (2.8), and (2.6) imply

$$
\begin{aligned}
S_B(v, r) &\leq k_2(r + 1) \log T + S_C(v, r, T) \\
&\leq k_4 v + k_4(r + 1 + \log T) \log T + (k_2 k_5 + k_4)(T \log T)^{\delta} \\
&\leq (3k_4 + (k_2 + k_4)k_5 + k_4)(T \log T)^{\delta} + k_4(\log T)^2 \\
&\leq k_7(T \log T)^{\delta} - k_6(T \log T)/v^{1/d} + k_4(\log T)^2.
\end{aligned}
$$

Case 2: $v > 3(T \log T)^{\delta}$. By definition, hypothesis, and (2.4),

$$r + 1 + 3T/v^{1/d} \le (k_5 + 3)T/v^{1/d} = (2/3)^{1/d}(k_5 + 3)T/(2v/3)^{1/d} \le k_5 T/(2v/3)^{1/d}.$$

Thus, by induction and (2.5),

$$\begin{aligned} S_B(v, r) &\le k_2(r + 1)\log T + S_B(2v/3,\, r + 3T/v^{1/d}) \\ &\le k_7(T \log T)^{\delta} + k_4(\log T)^2 + (k_2 k_5 - (3/2)^{1/d} k_6)(T \log T)/v^{1/d} \\ &\le k_7(T \log T)^{\delta} + k_4(\log T)^2 - k_6(T \log T)/v^{1/d}. \quad \blacksquare \end{aligned}$$

Theorem 2.1. For all $T(n) \ge n$ that can be constructed in space $(T(n) \log T(n))^{d/(d+1)}$, every nondeterministic d-dimensional machine M with one worktape head that runs in time $T(n)$ can be simulated by a deterministic Turing machine in space $O((T(n) \log T(n))^{d/(d+1)})$.

Proof. Section 2.3 presents a simulation of M by a deterministic machine M'. The main program for M' calls *BLANK-COMP* with actual parameters [illegible]. Since $|B_0(n)| \ge T(n)^d \ge (T(n) \log T(n))^{\delta}$, Lemma 2.3 implies that M' uses space $S_B(|B_0(n)|, 0) = O((T(n) \log T(n))^{\delta})$. Apply constant-factor tape reduction to decrease the space to [illegible]

2.5 Remarks

The constructibility hypothesis in Theorem 2.1 seems essential because a nondeterministic machine M may reject its input word by failing to halt. The simulator should have a bound on the number of steps of M to simulate before deciding that M rejects its input. [illegible] however, then the [illegible]

Theorem 2.2. For all $T(n) \ge n$, every deterministic d-dimensional machine with one worktape head that runs in time $T(n)$ can be simulated by a deterministic Turing machine in space $(T(n) \log T(n))^{d/(d+1)}$.

Proof. Run the simulation in Section 2.3 for $T = 1, 2, 3, \ldots$, searching for both an accepting configuration and a rejecting configuration. $\blacksquare$

Like Paul, Prauss, and Reischuk [17], we can devise a simulation by a time-bounded alternating Turing machine.

Theorem 2.3. For all $T(n)$, every nondeterministic d-dimensional machine with one worktape head that runs in time $T(n)$ can be simulated by an alternating Turing machine in time $O(\max\{n, (T(n)\log T(n))^{d/(d+1)}\})$.

Proof. (Sketch) In the simulation in Section 2.3 make the following routine modifications.

(i) Guess $T(n)$ nondeterministically.

(ii) Choose strategies existentially without imposing a bound on space.

(iii) Replace iterations through partitions of B and through enumerations of R' and π' by existential choices.

(iv) When a strategy makes two procedure calls, choose both universally.

The time analysis of this modified simulation is identical to the space analysis of Section 2.4. ∎

Chapter 3. A Note on the Pebble Game

A combinatorial "pebble" game on graphs has been used to establish tradeoffs between time and space required for arithmetic expression evaluation [21] and for Turing machine simulation [7]. One places pebbles on the vertices of a directed acyclic graph G in steps according to the following rules:

(i) A *step* is either a placement of a pebble on an empty vertex or a removal of a pebble from a vertex.

(ii) A pebble may be placed on a vertex only if there are pebbles on all immediate predecessors of the vertex. (Thus, a vertex with no predecessors can always be pebbled.)

(iii) A pebble may always be removed from a vertex.

A *pebbling strategy* is a sequence of steps in the pebble game. The goal is to find a pebbling strategy that places a pebble on every vertex of G at least once when the supply of pebbles is limited. This pebble game has been studied extensively; Lengauer and Tarjan [11] provide an exhaustive list of references.

This note develops an explicit strategy that uses $O(n/\log n)$ pebbles to pebble every directed acyclic graph G with n vertices and bounded indegree. Furthermore, for every $S \geq O(n/\log n)$, a variation of this strategy uses S pebbles to pebble G in at most $S\,2^{2^{O(n/S)}}$ steps. The proofs of these upper bounds, which employ an overlap argument [16], seem more natural than the original proofs [7, 11, 20].

Fix a directed acyclic graph $G = (V, E)$ with vertices V and edges E. Let $n = |V|$ and d be the maximum indegree of the vertices. For subsets W_1, W_2 of V let $E(W_1, W_2)$ be the set of edges from W_1 to W_2:

$$E(W_1, W_2) = \{(x, y): (x, y) \in E,\ x \in W_1, \text{ and } y \in W_2\}.$$

Let $W \subseteq V$. A sequence $(W_1, \ldots, W_m)$ of subsets of W is a *layered partition* of W if $\{W_1, \ldots, W_m\}$ is a partition of W and $E(W_j, W_i) = \emptyset$ for all i and j such that $i < j$. Let $\omega(W)$ denote the *internal overlap* of W:

$$\omega(W) = \max\{|E(W_1, W_2)|: (W_1, W_2) \text{ is a layered partition of } W\}.$$

Lemma 3.1. If $\omega(V) = r$, then G can be pebbled with $r + 1$ pebbles in $2n$ steps.

Proof. Our pebbling strategy comprises n Stages. Put $W_0 = \emptyset$. For $j = 1, ..., n$, assume inductively that W_{j-1} is the set of vertices that have been pebbled prior to Stage j. At Stage j place a pebble on a vertex x in $V \setminus W_{j-1}$, provided that all its immediate predecessors hold pebbles, and put $W_j = W_{j-1} \cup \{x\}$; then remove pebbles from all vertices y for which all immediate successors of y are in W_j. At the end of each Stage, a pebble remains on a vertex z if and only if some immediate successor of z has not been pebbled.

The rules of the pebble game guarantee that every $(W_j, V \setminus W_j)$ is a layered partition of V. By hypothesis, every $|E(W_j, V \setminus W_j)| \leq r$. Therefore, at the end of each Stage there are at most r pebbles on the graph, and the strategy uses at most $r + 1$ pebbles. The strategy has $2n$ steps because for every x, a pebble is placed on x and removed from x just once. ∎

Lemma 3.2. Let $(W_1, ..., W_m)$ be a layered partition of V. There is a strategy that pebbles G with at most

$$\Sigma_i\ (\omega(W_i) + d + 1)$$

pebbles.

Proof. By induction on m. For $m = 1$, Lemma 3.1 asserts *a fortiori* that $\omega(W_1)$ pebbles suffice. Assume that the subgraph of G induced by $V \setminus W_m = W_1 \cup ... \cup W_{m-1}$ can be pebbled with

$$\sum_{i=1}^{m-1} (\omega(W_i) + d + 1)$$

pebbles via strategy $\mathbf{S}_{m-1}$. We describe informally how to pebble vertices in W_m using $\omega(W_m) + d + 1$ more pebbles.

Let P_m be a set of $\omega(W_m) + 1$ pebbles and Q_m be a set of d pebbles. The pebbles in P_m are placed only on vertices in W_m. The pebbles in Q_m are placed only on vertices in $V \setminus W_m$.

As in the proof of Lemma 3.1, our strategy comprises $|W_m|$ Stages and uses at most $\omega(W_m) + 1$ pebbles on W_m. At each Stage select a vertex x in W_m that has not yet been pebbled but all immediate predecessors of x in W_m hold pebbles. Use strategy $\mathbf{S}_{m-1}$ to place pebbles from Q_m on the immediate predecessors of x in $V \setminus W_m$. These Q_m-pebbles remain on the immediate predecessors of x until x is pebbled. (By hypothesis, no vertex in W_m is an immediate predecessor of a vertex in $V \setminus W_m$; thus, strategy $\mathbf{S}_{m-1}$ may always be employed.) Place a pebble from P_m on x.

Remove all Q_m-pebbles from the graph, returning them to Q_m for later use. Also, remove pebbles from all vertices y in W_m for which all immediate successors of y in W_m have been pebbled, and return these pebbles to P_m. ∎

Lemma 3.3. For every r, there is a layered partition $(W_1, ..., W_m)$ of V such that $m \leq 2^{\lceil dn/r \rceil}$ and $\Sigma_i\, \omega(W_i) \leq r$.

Proof. Assume, to the contrary, that for every layered partition $(W_1, ..., W_m)$ of V, if $m \leq 2^{\lceil dn/r \rceil}$, then $\Sigma_i\, \omega(W_i) > r$. Let $i_0 = \lceil dn/r \rceil$ and $V_{0,0} = V$. For $i = 0, ..., i_0 - 1$, inductively suppose sets $V_{i,j}$ for $j = 0, ..., 2^i - 1$ have been defined. For each j find a layered partition $(V_{i+1,2j}, V_{i+1,2j+1})$ of $V_{i,j}$ such that $|E(V_{i+1,2j}, V_{i+1,2j+1})| = \omega(V_{i,j})$. By assumption, for every i,

$$\Sigma_j\, \omega(V_{i,j}) > r,$$

and consequently,

$$\Sigma_{i<i_0} \Sigma_j\, \omega(V_{i,j}) > i_0 r \geq dn.$$

By definition of the sets $V_{i,j}$, the sets of edges $E(V_{i+1,2j}, V_{i+1,2j+1})$ are pairwise disjoint. Since G has at most dn edges,

$$\Sigma_{i<i_0} \Sigma_j\, \omega(V_{i,j}) = \Sigma_{i<i_0} \Sigma_j\, |E(V_{i+1,2j}, V_{i+1,2j+1})| \leq dn.$$

Contradiction. ∎

Theorem 3.1. (Hopcroft, Paul, and Valiant [7]) Every directed acyclic graph with n vertices and bounded indegree can be pebbled with $O(n/\log n)$ pebbles.

Proof. Let $G = (V, E)$ be a directed acyclic graph with n vertices and indegree d. Let $S(n)$ satisfy

$$S(n) = O(n/\log n),$$

$$\log_2 (S(n)/(2d + 2)) \geq \lceil 2dn/S(n) \rceil.$$

According to Lemma 3.3, there is a layered partition $(W_1, ..., W_m)$ of V such that $m \leq 2^{\lceil 2dn/S(n) \rceil}$ and $\Sigma_i\, \omega(W_i) \leq S(n)/2$. For this partition Lemma 3.2 asserts that some strategy pebbles G with

$$\Sigma_i\, (\omega(W_i) + d + 1) \leq S(n)/2 + m(d + 1) \leq S(n)$$

pebbles. ∎

Theorem 3.2. (Lengauer and Tarjan [11]) For every n, d, and S and every directed acyclic graph G with n vertices and indegree d, if $(3d + 4)n/\log_2 n \leq S \leq n$, then there is a strategy that uses S pebbles to pebble G in at most $S\,2^{2^{O(n/S)}}$ steps.

Proof. Let $G = (V, E)$. Set $\alpha = 2d/(3d + 2)$ and $\beta = (d + 2)/(3d + 4)$ and $\gamma = 1 - \alpha - \beta$. Evidently, $\log_2 \log_2 n \leq (\log_2 n)/2$ because $(3d + 4)n/\log_2 n \leq n$ implies $n \geq 16$. It follows that

$$\log_2 (\beta S/(d + 2)) \geq (\log_2 n)/2 \geq (3d + 2)n/2S + n/S \geq dn/\alpha S + 1,$$

hence $(d + 2)2^{\lceil dn/\alpha S \rceil} \leq \beta S$.

Apply Lemma 3.3 with $r = \alpha S$ to obtain a layered partition $(W_1, ..., W_m)$ of V with $m \leq 2^{\lceil dn/\alpha S \rceil}$ such that $\Sigma_i\, \omega(W_i) \leq \alpha S$. For $i = 1, ..., m$, let P_i be a set of $\omega(W_i) + 1$ pebbles. Distribute the remaining $\beta S - m + \gamma S$ pebbles among sets $Q_1, ..., Q_m$ such that each $|Q_i| \geq \lfloor \gamma S\, |W_i|/n \rfloor + d + 1$.

We define the pebbling strategy inductively. Let $T(k)$ be the number of steps used by this strategy on the subgraph induced by $W_1 \cup ... \cup W_k$. For $k = 1$, the strategy in the proof of Lemma 3.1 uses $\omega(W_1) + 1$ pebbles, and $T(1) = 2\,|W_1| \leq 2n$.

Suppose that strategy $\mathbf{S}_{m-1}$ with $T(m - 1)$ steps uses the pebbles in $P_1 \cup ... \cup P_{m-1}$ and $Q_1 \cup ... \cup Q_{m-1}$ to pebble the subgraph of G induced by $V \setminus W_m$. To pebble vertices in W_m, carry out the strategy in the proof of Lemma 3.2 with the following modification. Whenever the immediate predecessors in $V \setminus W_m$ of a vertex in W_m must be pebbled, use strategy $\mathbf{S}_{m-1}$ to place pebbles from Q_m on the immediate predecessors in $V \setminus W_m$ of the $\lfloor |Q_m|/d \rfloor$ vertices in W_m that are pebbled next. Strategy $\mathbf{S}_{m-1}$ is invoked at most $\lceil |W_m|/\lfloor |Q_m|/d \rfloor \rceil \leq \lceil dn/\gamma S \rceil$ times. Therefore,

$$\begin{aligned}
T(m) &\leq \lceil |W_m|/\lfloor |Q_m|/d \rfloor \rceil\, T(m - 1) + 2\,|W_m| \\
&\leq (1 + dn/\gamma S)\, T(m - 1) + 2n \\
&\leq 2n\,[1 + (1 + dn/\gamma S) + ... + (1 + dn/\gamma S)^{m-1}] \\
&\leq 2n\,(1 + dn/\gamma S)^m/(dn/\gamma S) \\
&\leq (2\gamma/d)S \exp_2 \exp_2 (\lceil dn/\alpha S \rceil + \log_2 \log_2 (1 + dn/\gamma S)) \\
&= S\,2^{2^{O(n/S)}}.
\end{aligned}$$

(In general, $\exp_2 u = 2^u$.) ∎

Chapter 4: Embeddings and Simulations among Data Structures

4.1. Introduction

It is important to understand how efficiently one data structure or machine simulates another. Frequently, a programmer uses a data structure that must be simulated by the natural data structures provided by the programming language. More generally, for conceptual clarity, designers of computers and programs devise levels of "virtual machines." The constructs of abstract machines are defined in terms of operations of lower-level machines. For instance, an ALGOL statement is translated into machine code for a PDP-10 computer, and each PDP-10 machine instruction is executed by running several microcode instructions.

Several researchers have investigated embeddings of one data structure in another. These embeddings are defined in terms of the underlying graphs.

Let $G = (V, E)$ be an undirected graph with vertices V and edges E: an edge is a set of two distinct vertices. Vertex x is a neighbor of y if $\{x, y\}$ is an edge in E. For x, y in V and $r \geq 0$, a path of length r from x to y is a sequence of $r + 1$ vertices $v_0, v_1, \ldots, v_r$ such that $x = v_0$, $y = v_r$, and for every i, $\{v_i, v_{i+1}\}$ is an edge in E. If $r \geq 2$, then the vertices $v_1, \ldots, v_{r-1}$ are the interior vertices of the path. Let $d_G(x, y)$ be the length of the shortest path from x to y. If there is no path from x to y, then $d_G(x, y) = \infty$. Call $d_G(x, y)$ the distance from x to y.

The grid graph Γ_m has vertex set $\Phi_m = \{(i, j): 1 \leq i \leq m, 1 \leq j \leq m\}$ and edges $\{(i, j), (i, j + 1)\}$ for $1 \leq i \leq m$, $1 \leq j < m$ and $\{(i, j), (i + 1, j)\}$ for $1 \leq i < m$, $1 \leq j \leq m$.

A *simple embedding* of a *guest graph* $G = (V, E)$ into a *host graph* $H = (V^*, E^*)$ is an injection $\phi: V \rightarrow V^*$. Each vertex x in G has a unique *representative* $\phi(x)$ in H. The *cost* of the embedding ϕ is the minimum T such that

$$d_H(\phi(x), \phi(y)) \leq T\, d_G(x, y)$$

for all x, y in V. This cost measures the time required for accessing an item in a guest data structure that is simulated by a host data structure. DeMillo, Eisenstat, and Lipton [1] proved that the cost of every simple embedding of Γ_m into a binary tree is at least $\log_2 m - 3/2$. Rosenberg and Snyder [24, 25] analyzed a more elaborate cost measure that incorporates "weights" on edges in E^* and "usage patterns" on G. We would like to extend these results on static embeddings among data structures to derive bounds on the time complexity of dynamic simulations among automata with different storage structures.

A *tree machine* has a finite-state control and several heads on a worktape having the structure of a complete infinite binary tree. Section 4.2 presents an argument that suggests that to simulate a two-dimensional Turing machine by a tree machine on-line requires time $\Omega(n(\log n)^{1/2})$ if the simulator uses space $O(n)$. Proofs omitted from Section 4.2 appear in Section 4.3.

Section 4.4 outlines further research problems on comparing automata with different storage structures.

4.2. Static Embeddings Versus Dynamic Simulations

Let ψ be a simple embedding of Γ_m into a binary tree B. Call a pair of vertices (x, y) a *separated pair* (for ψ) if

$$d_B(\psi(x), \psi(y)) \geq \log_2 m - (\log_2 \log_2 m)/2 - 3.$$

A path $(v_0, ..., v_r)$ *includes a consecutive pair of vertices* (x, y) if $x = v_i$ and $y = v_{i+1}$ for some i. DeMillo, Eisenstat, and Lipton [1] proved that separated pairs for ψ exist. Proposition 4.1, which is proved in Section 4.3, asserts that some path in Γ_m includes many consecutive separated pairs.

Proposition 4.1. For every binary tree B, every even $m \geq 32$, and every simple embedding ψ of Γ_m into B, there is a path in Γ_m of length at most $7m$ that includes at least

$$m/32(\log_2 m)^{1/2}$$

distinct consecutive separated pairs.

We employ Proposition 4.1 to argue informally that every tree machine that simulates a two-dimensional Turing machine on-line in space $O(n)$ for inputs of length n may require time $\Omega(n\,(\log n)^{1/2})$ in the worst case. This argument has not been developed into a rigorous proof yet, however.

Consider a two-dimensional Turing machine **M** with one worktape head whose input alphabet consists of the eight pairs $\langle b, \delta\rangle$ where $b \in \{0, 1\}$ and δ is one of the four directions that the worktape head can move at each step. Machine **M** operates in real time – it processes one input symbol at every step. Suppose **M** is in a configuration in which the cell C scanned by the worktape head contains b'; on input $\langle b, \delta\rangle$, **M** writes b on C, writes b' on the output tape, and moves the worktape head in direction δ.

Let $\mathbf{M}'$ be a tree machine that simulates $\mathbf{M}$ on-line in space $O(n)$. Assume that for every worktape cell C of $\mathbf{M}$, machine $\mathbf{M}'$ assigns one of its tree tape cells to hold the contents of C. Thus, $\mathbf{M}'$ determines a simple embedding of every Γ_m into its binary tree structure.

Construct an input word w of length n as follows. The first $n/2$ symbols of w induce $\mathbf{M}$ to fill a square of side $m = (n/2)^{1/2}$ on its worktape with 0's and 1's. The last $n/2$ symbols drive the head of $\mathbf{M}$ on $n/18m$ paths of length $9m$ that each includes at least $m/32(\log_2 m)^{1/2}$ distinct consecutive separated pairs of cells; each of these paths begins with a path of length $2m$ that drives the worktape head of $\mathbf{M}$ to the first cell of the path of length $7m$ whose existence is guaranteed by Proposition 4.1. Because $\mathbf{M}'$ has a finite-state control, it can remember the contents of only a finite number of separated pairs internally. Consequently, it is plausible that for each new separated pair (x, y) that $\mathbf{M}$ encounters, $\mathbf{M}'$ spends time $\Omega(\log m)$ moving a worktape head from the representative of x to the representative of y. Hence on input w, $\mathbf{M}'$ may require time

$$\Omega[(n/18m)(m/32(\log_2 m)^{1/2})(\log m)] = \Omega(n\,(\log m)^{1/2}) = \Omega(n\,(\log n)^{1/2}).$$

A tree machine $\mathbf{M}''$ that uses superlinear space might simulate $\mathbf{M}$ faster. Reischuk [23] devised an on-line simulation that operates in time $O(n\, c^{\log^* n})$ for a constant c, but uses space $O(n\, c^{\log^* n})$. Each cell that $\mathbf{M}$ uses has $O(c^{\log^* n})$ representatives in the worktape of $\mathbf{M}''$.

Lipton, Eisenstat, and DeMillo [13] introduced a formulation of data structure embedding that permits multiple representatives of vertices of the guest graph. Let $G = (V, E)$ and $H = (V^*, E^*)$ be graphs. An *embedding* of G into H is a map $\varphi: V^* \to V \cup \{\Lambda\}$, where $\Lambda \notin V$, such that $|\varphi^{-1}(x)| \geq 1$ for every x in V. If $\varphi(x^*) = x$, then x^* is a *representative* of x. The *space cost* of φ is $\max\{|\varphi^{-1}(x)|: x \in V\}$. The *strong time cost* $T_s(\varphi)$ of φ is the smallest T such that

> for every x^* in V^* such that $\varphi(x^*) \neq \Lambda$ and every y in V such that $d_G(\varphi(x^*), y) < \infty$, there exists y^* in $\varphi^{-1}(y)$ and $d_H(x^*, y^*) \leq T\, d_G(\varphi(x^*), y)$.

The *weak time cost* $T_w(\varphi)$ of φ is the smallest T such that

> for every x and y in V such that $d_G(x, y) < \infty$, there exist x^* in $\varphi^{-1}(x)$ and y^* in $\varphi^{-1}(y)$ such that $d_H(x^*, y^*) \leq T\, d_G(x, y)$.

In general, $T_s(\varphi) \geq T_w(\varphi)$. If φ has space cost 1, then $T_s(\varphi) = T_w(\varphi)$.

Proposition 4.2. (Lipton, Eisenstat, and DeMillo [2].) If φ is an embedding of Γ_m into a binary tree with space cost S, then

$$T_s(\varphi) + \log_2 \log_2 S \geq \log_2 m - c' \log_2 \log_2 m$$

for a positive constant c' independent of m.

Reischuk's embedding of Γ_m into a binary tree [23] has space cost $c^{\log^* m}$, hence unbounded strong time cost by Proposition 4.2, but bounded weak time cost. His simulation runs quickly because the weak time cost of the embedding is constant. Therefore, Reischuk's simulation suggests that the strong time cost measure may be inappropriate for establishing a lower bound on the time required by a tree machine to simulate a multidimensional Turing machine on-line when the simulator is not confined to $O(n)$ space.

4.3. Proof of Proposition 4.1

The graph $G = (V, E)$ is *connected* if for every x, y in V there is a path from x to y. Let $U \subseteq V$. The *boundary* of U, denoted ∂U, is the set of vertices in U that have a neighbor in $V \setminus U$. Write $G(U)$ for the subgraph of G induced U. A *connected component* of G is a subgraph $G(W)$ induced by a set of vertices W such that $G(W)$ is connected and $d_G(x, z) = \infty$ for all x in W and z in $V \setminus W$. The *size* of a component is the number of vertices that it has.

If P_1 is a path from x to y and P_2 is a path from y to z, then the *concatenation* of P_1 and P_2, written $P_1 \bullet P_2$, is the path from x to z obtained from P_1 by replacing the last vertex y by P_2. The concatenation operator $\bullet$ is associative.

Lemma 4.1. For every set U of u vertices in Γ_m, there is a path of length at most $2m\,(u^{1/2} + 1)$ in Γ_m that includes all the vertices in U.

Proof. (Steiglitz and Papadimitriou [28].) Set $s = \lceil m/u^{1/2} \rceil$, $h^* = \lceil m/s \rceil - 1$, and for $h = 0, \ldots, h^*$,

$$U_h = \{(i, j): (i, j) \in U \text{ and } hs + 1 \leq j \leq (h + 1)s\};$$

$\{U_0, \ldots, U_{h^*}\}$ is a partition of U. Construct the path P as follows. First visit the vertices in U_0 in lexicographic order, then the vertices in U_1 in reverse lexicographic order, then the vertices in U_2 in lexicographic order, then U_3 in reverse lexicographic order, and so on. (In the usual lexicographic ordering of pairs of integers, (i, j) precedes (i', j') if either $i < i'$ or $i = i'$ and $j < j'$.) Index U according to the order visited by P: $U = \{(i_1, j_1), (i_2, j_2), ..., (i_u, j_u)\}$. Set $\Delta i_k = |i_{k+1} - i_k|$ and $\Delta j_k = |j_{k+1} - j_k|$. If (i_k, j_k) and (i_{k+1}, j_{k+1}) are in the same U_h, then $\Delta j_k \leq s - 1$; if $(i_k, j_k) \in U_h$ but $(i_{k+1}, j_{k+1}) \in U_{h+1}$, then $\Delta j_k \leq 2s - 1 = (s - 1) + s$. Therefore,

$$\Sigma_k \; \Delta j_k \leq (u - 1)(s - 1) + h^* s \leq u(s - 1) + m.$$

One verifies routinely that

$$\Sigma_k \; \Delta i_k \leq (h^* + 1)m \leq (1 + m/s)m = m + m^2/s.$$

Ergo, P has length

$$\begin{aligned}\Sigma_k \; (\Delta i_k + \Delta j_k) &\leq 2m + u(s - 1) + m^2/s \\ &\leq 2m + u(\lceil m/u^{1/2} \rceil - 1) + m^2/\lceil m/u^{1/2} \rceil \\ &\leq 2m + m\,u^{1/2} + m\,u^{1/2} \\ &\leq 2m(u^{1/2} + 1). \; ∎\end{aligned}$$

Lemma 4.2. Let σ be a sequence of s symbols over $\{\alpha, \beta\}$ and let b be the number of β symbols in σ. For every $r \leq b/2$ there is a consecutive subsequence of σ of $\lceil sr/(b - r) \rceil$ symbols that contains at least r symbols β.

Proof. Set $t = \lceil sr/(b - r) \rceil$; note that $t \geq sr/(b - r)$ implies $tb/(s + t) \geq r$. Form a sequence σ' of $t \lceil s/t \rceil$ symbols by appending $t \lceil s/t \rceil - s$ symbols α to the end of σ. Partition σ' into $\lceil s/t \rceil$ consecutive subsequences of t symbols each. One of these consecutive subsequences σ'' must have at least $b/\lceil s/t \rceil \geq b/(s/t + 1) = tb/(s + t) \geq r$ symbols β. If σ'' is not the final subsequence of σ', then it is a subsequence of σ; otherwise, if σ'' is the final subsequence of σ', then the last t symbols of σ form a consecutive subsequence of σ with at least r symbols β. ∎

Lemma 4.3. Let $m \geq 32$ and U be a nonempty set of vertices in Γ_m such that $|U| \leq m^2/2$ and $\Gamma_m(U)$ is connected. For every $r \leq (|U|/8)^{1/2}$, there is a path P of length at most $9r - 1$ that includes at least r distinct vertices of ∂U.

Proof. Call (i, j) in Φ_m a *boundary* vertex if $(i, j) \in \partial U$; call other vertices of Φ_m *nonboundary* vertices. We shall find a path with at most $9r$ vertices that contains at least r distinct boundary vertices. Set

$$i^* = \max\{i: (i, j) \in U \text{ for some } j\},$$
$$i_* = \min\{i: (i, j) \in U \text{ for some } j\},$$
$$j^* = \max\{j: (i, j) \in U \text{ for some } i\},$$
$$j_* = \min\{j: (i, j) \in U \text{ for some } i\}.$$

Case 1: Either $i_* > 0$ or $i^* < m$ and either $j_* > 0$ or $j^* < m$. Without loss of generality, assume $i^* - i_* \geq j^* - j_*$. Becasue

$$|U| \leq (i^* - i_* + 1)(j^* - j_* + 1),$$

it follows that

$$i^* - i_* + 1 \geq |U|^{1/2}.$$

We construct a path Q such that for every $r \leq (|U|/8)^{1/2} \leq (i^* - i_* + 1)/2$, there is a consecutive subsequence of Q with at most $4r$ vertices that contains at least r distinct boundary vertices.

Assume $j^* < m$; the case $j_* > 0$ is similar. Since $\Gamma_m(U)$ is connected, for every i such that $i_* \leq i \leq i^*$ there is some j for which $(i, j) \in U$. For $i = i_*, i_* + 1, ..., i^*$, set

$$J(i) = \max\{j: (i, j) \in U\}.$$

By assumption, every $J(i) \leq j^* < m$; thus, every $(i, J(i))$ is a boundary vertex.

For $i = i_*, i_* + 1, ..., i^* - 1$, construct a path $Q(i)$ from $(i, J(i))$ to $(i + 1, J(i + 1))$ as follows. If $J(i) > J(i + 1)$, then let $Q(i)$ be the path

$$(i, J(i)), (i, J(i) - 1), ..., (i, J(i + 1) + 1), (i, J(i + 1)), (i + 1, J(i + 1));$$

all vertices on this path except possibly $(i, J(i + 1))$ are boundary vertices. If $J(i) \leq J(i + 1)$, then let $Q(i)$ be the path

$$(i, J(i)), (i + 1, J(i)), (i + 1, J(i) + 1), ..., (i + 1, J(i + 1) - 1), (i + 1, J(i + 1));$$

all vertices on this path except possibly $(i + 1, J(i))$ are boundary vertices.

Set $Q = Q(i_*) \cdot Q(i_* + 1) \cdot ... \cdot Q(i^* - 1)$. Path Q contains the $i^* - i_* + 1$ boundary vertices $(i, J(i))$, which each occur exactly once in Q. Let b' be the number of other boundary vertices in Q; each of these occurs at most twice. There are at most $i^* - i_*$ occurrences of nonboundary vertices -- one in each $Q(i)$. Path Q has at most $2(i^* - i_*) + 2b' + 1$ vertices, including repetitions; it has at least $i^* - i_* + 1 + b'$ distinct boundary vertices. Apply Lemma 4.2 with $s = 2(i^* - i_*) + 2b' + 1$ and $b = i^* - i_* + 1 + b'$ to obtain a path with $\lceil (2(i^* - i_*) + 2b' + 1)r/(i^* - i_* + 1 + b' - r) \rceil \leq \lceil (2(i^* - i_*) + 2b' + 1)/((i^* - i_* + 1)/2 + b') \rceil r \leq 4r$ vertices that contains at least r distinct boundary vertices.

Case 2: $i_* = 0$ and $i^* = m$. For $j = 1, \ldots, m$, set

$$U_j = U \cap \{(1, j), \ldots, (m, j)\};$$

U_j is the jth *column* of U. There is at least one column U_k such that $|U_k| \leq \lceil m/2 \rceil$; otherwise, all m columns of U would have at least $\lceil m/2 \rceil + 1$ vertices, and $|U| \geq m(\lceil m/2 \rceil + 1) > m^2/2$, contrary to hypothesis.

Let $|U_k| = u$, and let $i_1, \ldots, i_u$ be the i in increasing order for which $(i, k) \in U_k$. Define

$$J(i_t) = k \text{ for } t = 1, \ldots, u.$$

We define $J(i)$ for other i as follows. Set $i_0 = 0$ and $i_{u+1} = m$. Since $\Gamma_m(U)$ is connected, for every $0 \leq t \leq u$ either

(a) for every i such that $i_t < i < i_{t+1}$ there exists $j < k$ such that $(i, j) \in U$; or

(b) for every i such that $i_t < i < i_{t+1}$ there exists $j > k$ such that $(i, j) \in U$.

If condition (a) holds, then call $[i_t, i_{t+1}]$ an interval of *type (a)*, and for $i_t < i < i_{t+1}$, set

$$J(i) = \max\{j : j < k \text{ and } (i, j) \in U\}.$$

If condition (b) holds, then call $[i_t, i_{t+1}]$ an interval of *type (b)*, and for $i_t < i < i_{t+1}$, set

$$J(i) = \min\{j : j > k \text{ and } (i, j) \in U\}.$$

By definition, unless $i = i_t$ for some t, every $(i, J(i)) \in \partial U$. Thus, at least $m - u \geq \lfloor m/2 \rfloor$ vertices of the form $(i, J(i))$ are boundary vertices.

For $i = 1, \ldots, m - 1$, we define a path $Q(i)$ from $(i, J(i))$ to $(i + 1, J(i + 1))$ such that at most one interior vertex of $Q(i)$ is a nonboundary vertex. Suppose $[i, i+1]$ lies in an interval $[i_t, i_{t+1}]$ of type (a); the definition of $Q(i)$ for an interval of type (b) is similar. If $J(i) > J(i + 1)$, then let $Q(i)$ be the path

$$(i, J(i)), (i, J(i) - 1), \ldots, (i, J(i + 1) + 1), (i, J(i + 1)), (i + 1, J(i + 1));$$

all interior vertices on this path except possibly $(i, J(i + 1))$ are boundary vertices. If $J(i) \leq J(i + 1)$, then let $Q(i)$ be the path

$$(i, J(i)), (i + 1, J(i)), (i + 1, J(i) + 1), \ldots, (i + 1, J(i + 1) - 1), (i + 1, J(i + 1));$$

all interior vertices on this path except possibly $(i + 1, J(i))$ are boundary vertices.

Set $Q = Q(1) \cdot Q(2) \cdot \ldots \cdot Q(m - 1)$. This path contains at least $\lfloor m/2 \rfloor$ boundary vertices $(i, J(i))$ for $J(i) \neq k$. Let Q have b' other boundary vertices; each of these occurs at most twice in Q. Path Q has m vertices $(i, J(i))$ that each occur once; it has $m - 1$ occurrences of nonboundary interior vertices among the $Q(i)$. Therefore, Q has at most $m + 2b' + m - 1 = 2m + 2b' - 1$ vertices, including repetitions; it has at least $\lfloor m/2 \rfloor + b'$ distinct boundary vertices. By hypothesis, $r \leq (|U|/8)^{1/2} \leq$

$m/4 \leq \lfloor m/2 \rfloor/2 + 1/2$. Apply Lemma 4.2 to Q with $s = 2m + 2b' - 1$ and $b = \lfloor m/2 \rfloor + b'$ to find a path with

$$\lceil (2m + 2b' - 1)r/(\lfloor m/2 \rfloor + b' - r) \rceil \leq \lceil (2m + 2b' - 1)/(\lfloor m/2 \rfloor/2 + b' - 1/2) \rceil r$$
$$\leq \lceil (2m + 2b' - 1)/(m/4 - 1 + b') \rceil r$$
$$\leq 9r \text{ (because } m \geq 32)$$

vertices that contains at least r distinct boundary vertices.

Case 3: $j_* = 0$ and $j^* = m$. Similar to Case 2. ∎

Lemma 4.4. Let U^* be the vertices of a subtree of a binary tree. For every subset W^* of r vertices in U^*, at least $r/2$ vertices of W^* are at distance at least $\log r$ from vertices not in U^*.

Proof. Let $D = \log r - 1$. The maximum number of vertices of W^* that can be at distance at most D from a vertex not in U^* is $2^D - 1 = r/2 - 1$. At least $r - (r/2 - 1) \geq r/2$ vertices of W^* must be at distance at least $D + 1$ from vertices not in U^*. ∎

Proof of Proposition 4.1. Let U^* be the vertices of a subtree of B such that $m^2/4 \leq |\psi^{-1}(U^*)| \leq m^2/2$; such a subtree exists because m is even. Set $U = \psi^{-1}(U^*)$. Let the subgraph $\Gamma_m(U)$ induced by U have c connected components, and let $u_1, u_2, ..., u_c$ be the sizes of these components in decreasing order.

Set $M = m^2$ and $k_0 = \log M - \log\log M - 2$. (All logarithms are taken to base 2.) For $k = 0, 1, ..., k_0$ set

$$t_k = M/4(2^k \log M) = m^2/8(2^k \log m).$$

By definition, $t_{k_0} = 1$.

We claim that for some k there are at least 2^k connected components of $\Gamma_m(U)$ of size at least t_k. Suppose, to the contrary, that for every k there are at most $2^k - 1$ components of $\Gamma_m(U)$ of size at least t_k. Then $u_1 < t_0$. Since there is at most 1 component of size at least t_1 and $u_1 \geq u_2 \geq u_3$, we infer that $u_2 < t_1$ and $u_3 < t_1$. In general, for all k and all $0 \leq j \leq 2^k - 1$,

$$u_{2^k + j} < t_k$$

Consequently,

$$|U| = \Sigma_i\, u_i < \sum_{k=0}^{k_0} 2^k t_k = (k_0 + 1)(M/4 \log M) \leq M/4.$$

But $|U| \geq M/4$. Contradiction.

Let $U_1, \ldots, U_{2^k}$ be the sets of vertices of the 2^k largest components of $\Gamma_m(U)$ such that $|U_i| \geq t_k$ for each i. For each i, $|U_i| \leq |U| \leq m^2/2$. Apply Lemma 4.3 with $r = (t_k/8)^{1/2}$ to obtain a path P_i of length at most $9(t_k/8)^{1/2} - 1$ that contains a set W_i of at least $(t_k/8)^{1/2}$ vertices of ∂U_i. Put

$$W = W_1 \cup \ldots \cup W_{2^k}.$$

By definition, $|W| \geq 2^k(t_k/8)^{1/2} = 2^{k/2}m/8(\log m)^{1/2}$, and every vertex in W has a neighbor in $\Phi_m \setminus U$.

By Lemma 4.4, since $\psi(W) \subseteq \psi(U) = U^*$, at least half of the vertices in $\psi(W)$ are at distance at least

$$\log |W| \geq \log m - (\log \log m)/2 - 3$$

from all vertices in $\psi(\Phi_m \setminus U)$. Let W' be a set of $|W|/2$ vertices x in W such that (x, y) is a separated pair for every y in $\Phi_m \setminus U$.

Let y_i be the first vertex in P_i and z_i be the last. Let P_i' be a path from z_i to y_i whose length is at most the length of P_i. Let $Q_i = P_i \cdot P_i'$; path Q_i from y_i to y_i has length at most $18(t_k/8)^{1/2} - 2$. Invoke Lemma 4.1 to obtain a path R of length at most $2m(2^{k/2}) + 2m$ that visits every y_i. Construct a path R' from R by substituting Q_i for an occurrence of y_i in R for each i. Path R' has length at most

$$\begin{aligned} 2m(2^{k/2} + 1) + 2^k(18(t_k/8)^{1/2} - 2) &\leq 4m2^{k/2} + 2^{k/2}18m/8(\log m)^{1/2} - 2 \\ &\leq 4m2^{k/2} + 2^{k/2}18m/16 - 2 \text{ (because } m \geq 16) \\ &\leq 6m2^{k/2} - 2 \end{aligned}$$

and includes the vertices in W'. Apply Lemma 4.2 with $s = 6m2^{k/2}$, $b = |W'| \geq 2^{k/2}m/16(\log m)^{1/2}$ and $r = m/32(\log m)^{1/2}$ to obtain a subsequence S of R' of length at most $6m$ with a subset W'' of $m/32(\log m)^{1/2}$ vertices in W'. Construct path S' from S by replacing an occurrence in S of each vertex x in W'' by the sequence (x, y, x) for some neighbor y of x such that $y \in \Phi_m \setminus U$; by definition of W'', each (x, y) is a separated pair. Path S' has length at most $6m + 2|W''| \leq 7m$ and includes at least $|W''| = m/32(\log m)^{1/2}$ distinct separated pairs. ∎

4.4. Open Problems

A comparison of multidimensional Turing machines and machines with other storage structures describes quantitatively how the structures of the machines affect their efficiency. When studying these machines, we may attempt to generalize theorems about conventional one-dimensional machines. But we should not be interested in generalization for its own sake. Rather, we should determine what properties of conventional Turing machines are not artifacts of the linearity of the machine's tapes to demonstrate that phenomena such as the time-space tradeoff [7] occur ubiquitously in computations.

The following problems remain open.

1. Can a d-dimensional Turing machine simulate an e-dimensional Turing machine of time complexity $T(n)$ in time $O(T(n)^{1 + 1/d - 1/e})$ on-line? Or can the lower bound $\Omega(T(n)^{1 + 1/d - 1/e})$ be increased?

2. Can Reischuk's simulation of a multidimensional machine by a tree machine [23] be improved? If the space used by the on-line simulator is restricted to $O(n)$ when the d-dimensional machine runs for n steps, must the simulator use $\Omega(n (\log n)^{1 - 1/d})$ time?

3. Can a d-dimensional machine simulate a tree machine of time complexity $T(n)$ in time $O(T(n)^{1 + 1/d}/\log T(n))$ on-line?

4. Do similar time bounds hold for simulations among nondeterministic machines? Can a nondeterministic Turing machine of time complexity $T(n)$ be simulated by a nondeterministic machine in space $T(n)/\log T(n)$?

References

[1] R.A. DeMillo, S.C. Eisenstat, and R.J. Lipton. Preserving average proximity in arrays. *Comm. ACM* 21 (1978) 228-231.

[2] R.A. DeMillo, S.C. Eisenstat, and R.J. Lipton. Space-time trade-offs in structured programming: An improved combinatorial embedding theorem. *J. ACM* 27 (1980) 123-127.

[3] D. Ju. Grigor'ev. Imbedding theorems for Turing machines of different dimensions and Kolmogorov's algorithms. *Soviet Math. Dokl.* 18 (1977) 588-592.

[4] D. Yu. Grigoriev. Time complexity of multidimensional Turing machines. Translated by A. Shvartsman, Cornell Univ., 1980.

[5] J. Hartmanis and R.E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.* 117 (1965) 285-306.

[6] F.C. Hennie. On-line Turing machine computations. *IEEE Trans. Elec. Comp.* EC-15 (1966) 35-44.

[7] J. Hopcroft, W. Paul, and L. Valiant. On time versus space. *J. ACM* 24 (1977) 332-337.

[8] J.E. Hopcroft and J.D. Ullman. Relations between time and tape complexities. *J. ACM* 15 (1968) 414-427.

[9] J.E. Hopcroft and J.D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.

[10] D.E. Knuth. *The Art of Computer Programming*, vol. 1: *Fundamental Algorithms.* Addison-Wesley, 1968.

[11] T. Lengauer and R.E. Tarjan. Upper and lower bounds on time-space tradeoffs. *Proc. 11th Ann. ACM Symp. on Theory of Computing*, 1979, pp. 262-277.

[12] B. Leong and J. Seiferas. New real-time simulations of multihead tape units. *Proc. 9th Ann. ACM Symp. on Theory of Computing*, 1977, pp. 239-248.

[13] R.J. Lipton, S.C. Eisenstat, and R.A. DeMillo. Space and time hierarchies for classes of control structures and data structures. *J. ACM* 23 (1976) 720-732.

[14] M.C. Loui. A space bound for one-tape multidimensional Turing machines. Tech. Memo. TM-145, Lab. for Computer Science, Mass. Inst. Tech., Nov. 1979.

[15] M.S. Paterson. Tape bounds for time-bounded Turing machines. *J. Comp. Sys. Sci.* 6 (1972) 116-124.

[16] M.S. Paterson, M.J. Fischer, and A.R. Meyer. An improved overlap argument for on-line multiplication. *Complexity of Computation*, SIAM-AMS Proc. vol. 7, ed. R. Karp, Amer. Math. Soc., 1974, pp. 97-111.

[17] W.J. Paul, E.J. Prauss, and R. Reischuk. On alternation. *Proc. 19th Ann. Symp. on Foundations of Computer Science*, 1978, pp. 113-122.

[18] W.J. Paul and R. Reischuk. On time versus space II. *Proc. 20th Ann. Symp. on Foundations of Computer Science*, 1979, pp. 298-206.

[19] W.J. Paul, J.I. Seiferas, and J. Simon. An information-theoretic approach to time bounds for on-line computation. *Proc. 12th Ann. ACM Symp. on Theory of Computing*, 1980, pp. 357-367.

[20] W.J. Paul, R.E. Tarjan, and J.R. Celoni. Space bounds for a game on graphs. *Math. Systems Theory* **10** (1977) 239-251.

[21] N. Pippenger. A time-space trade-off. *J. ACM* **25** (1978) 509-515.

[22] N. Pippenger and M.J. Fischer. Relations among complexity measures. *J. ACM* **26** (1979) 361-381.

[23] R. Reischuk. A fast implementation of a multidimensional storage into a tree storage. *Proc. 7th Intern. Colloq. on Automata, Languages, and Programming*, Springer-Verlag, 1980, pp. 531-542.

[24] A.L. Rosenberg. Data encodings and their costs. *Acta Informatica* **9** (1978) 273-292.

[25] A.L. Rosenberg and L. Snyder. Bounds on the costs of data encodings. *Math. Systems Theory* **12** (1978) 9-39.

[26] J.E. Savage and S. Swamy. Space-time trade-offs on the FFT algorithm. *IEEE Trans. Info. Theory* **IT-24** (1978) 563-568.

[27] A. Schönhage. Real-time simulation of multidimensional Turing machines by storage modification machines. Tech. Memo. 37, Project MAC, Mass. Inst. Tech., Dec. 1973.

[28] K. Steiglitz and C.H. Papadimitriou. *The Computational Complexity of Combinatorial Optimization Problems*. Prentice-Hall, to appear 1981.

[29] M. Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *Proc. 10th Ann. ACM Symp. on Theory of Computing*, 1978, pp. 196-204.

Biographical Note

Michael Conrad Loui was born June 1, 1955 in Philadelphia, Pennsylvania. He attended Punahou School in Honolulu, Hawaii and graduated June 1972 with awards in forensics, German, mathematics, and science.

At Yale University, which he entered in September 1972, Mr. Loui earned the B.S. degree in May 1975, summa cum laude, with distinction in Mathematics and Computer Science. He was elected to Phi Beta Kappa, Sigma Xi, and Tau Beta Pi. Also, he received a German prize and the Yale Science and Engineering Association High Scholarship Award. His Senior Essay took second place in the 1975-1976 ACM Forsythe Student Paper Competition. In 1975 he edited a collection of critiques of engineering and computer science courses at Yale.

In the summer of 1974 he performed research in adaptive control in the Department of Engineering and Applied Science at Yale. During the summers of 1973 and 1975 he was a data processing programmer at Industry Data Services in Honolulu.

Mr. Loui matriculated at M.I.T. in September 1975 and completed the S.M. degree in Electrical Engineering and Computer Science in February 1977. He served as an officer of the M.I.T. Graduate Student Council. Throughout his graduate study he has been supported by a fellowship from the Fannie and John Hertz Foundation.

His professional interests include the theory of computation (algebraic and combinatorial algorithms, automata, and computational complexity), software engineering, and other areas of modern applied mathematics, especially stochastic processes, optimization, and systems science. In addition to the papers in this thesis he has published the following:

Space-bounded simulation of multitape Turing machines, with L.M. Adleman. Tech. Memo. TM-148, Lab. for Comp. Sci., M.I.T., Jan. 1980. To appear in *Mathematical Systems Theory*.

The space complexity of two pebble games on trees. Tech. Memo. TM-133, Lab. for Comp. Sci., M.I.T., May 1979.

Minimum register allocation is complete in polynomial space. Tech. Memo. TM-128, Lab. for Comp. Sci., M.I.T., Mar. 1979.

Efficient multiplication in semisimple algebras, with A.S. Willsky. *Proc. 1978 Conf. on Information Sciences and Systems*, Johns Hopkins Univ., pp. 61-65.

Efficient multiplication in semisimple algebras. S.M. Thesis, M.I.T., Feb. 1977. Also Tech. Rep. R-700, Electronic Systems Lab., M.I.T., Nov. 1976.

Weighted derivation trees. *Communications of the ACM* **19** (1976) 509-513.

Comparison of learning automata operating in nonstationary environments, with K.S. Narendra. Becton Ctr. Tech. Rep. CT-65, Yale Univ., May 1975.

Mr. Loui regularly serves as a referee for scientific journals. Beginning September 1980 he will be a research associate in computer science at M.I.T.

Active in the performing arts, Mr. Loui plays the piano and writes music. At Punahou he composed and arranged several songs for an original musical comedy. His "Waltz-Fantasie" was performed at Yale in 1974. At M.I.T. he played the Boatswain in *H.M.S. Pinafore* and was stage manager for two one-act plays.

He enjoys running, sailing, squash, and ballroom dancing. In his spare time he cultivates interests in music, philosophy, and American history.