Chapter 9

Luis Rabelo
ISE Department
Ohio University
Athens, OH 45701

Albert Jones
National Institute of
Standards and Technology
Gaithersburg, MD 20899

Yuehwern Yih
School of Industrial Eng
Purdue University
W. Lafayette, IN 47907

A Hybrid Approach Using Neural Networks, Simulation, Genetic Algorithms, and Machine Learning for Real-Time Sequencing and Scheduling Problems

Abstract

A hybrid approach for sequencing and scheduling is described which integrates neural networks, real-time simulation, genetic algorithms, and machine learning. This approach has been used to solve both single machine sequencing and multi-machine scheduling problems. Neural networks are used to quickly evaluate and select a small set of candidate sequencing or scheduling rules from some larger set of heuristics. This evaluation is necessary to generate a ranking which specifies how each rule performs against the performance measures. Genetic algorithms are applied to this remaining set of rules to generate a single "best" schedule using simulation to capture the system dynamics. A trace-driven knowledge acquisition technique (symbolic learning) is used to generate rules to describe the knowledge contained in that schedule. The derived rules (in English-like terms) are then added to the original set of heuristics for future use. In this chapter, we describe how this integrated approach works, and provide an example.

9.1 Introduction

Sequencing and scheduling are two of the most important decisions made by any shop floor control system. But, while there has been an enormous research effort expended over the years in these areas, it has had little effect in the marketplace. The reason is simple, the research has led to the development of very few software tools that can solve real problems. The tools that do exist are typically 1) too slow and cannot react to changing shop floor conditions, 2) based on simplistic formulations which ignore important constraints like material handling, 3) based on a single objective function or simplistic trade-offs like goal programming, and 4) difficult to install and integrate into pre-existing commercial shop floor control systems.

In this chapter, we describe a methodology which integrates neural networks, simulation, genetic algorithms, and machine learning techniques. It determines the start and finish times of the jobs assigned to any module in the hierarchical shop floor control architecture proposed in (JONES and SALEH, 1990). Because this hierarchy decomposes the global scheduling problem into multiple levels, this methodology 1) never needs to solve very large problems, and 2) can react to delays on the shop floor in a manner which is not disruptive to the rest of the system. Moreover, by exploiting the parallel processing and modeling capabilities of neural networks, simulation, and genetic algorithms it has the potential to be extremely fast and highly adaptable to customer needs. Finally, the use of a learning technique provides the additional capability to learn what works and what does not work in a variety of situations and utilize that knowledge at a later time. For these reasons, we believe that this technique has the potential to solve real-world sequencing and scheduling problems in real-time.

The chapter is organized as follows. In section 9.2, we describe the generic controller and shop floor hierarchy. In section 9.3, we describe the method for generating start and finish times which is applicable at every level in that hierarchy and the learning technique. In section 9.4, we provide an example.

9.2 Hierarchical Generic Controller

The foundation of this research is the generic controller developed in (DAVIS et al., 1992) and the hierarchical shop floor control system described in (JONES and SALEH, 1990). This hierarchy is based on a decomposition of the global planning and scheduling problems, rather than the traditional partitioning of the physical shop floor equipment. This new approach to building hierarchies led to the fundamental contribution of (DAVIS et al., 1992) — that every controller in this hierarchy performs the exact same four production management functions — assessment, optimization, execution, and monitoring. We now give a brief description of these functions (see Figure 9.1).

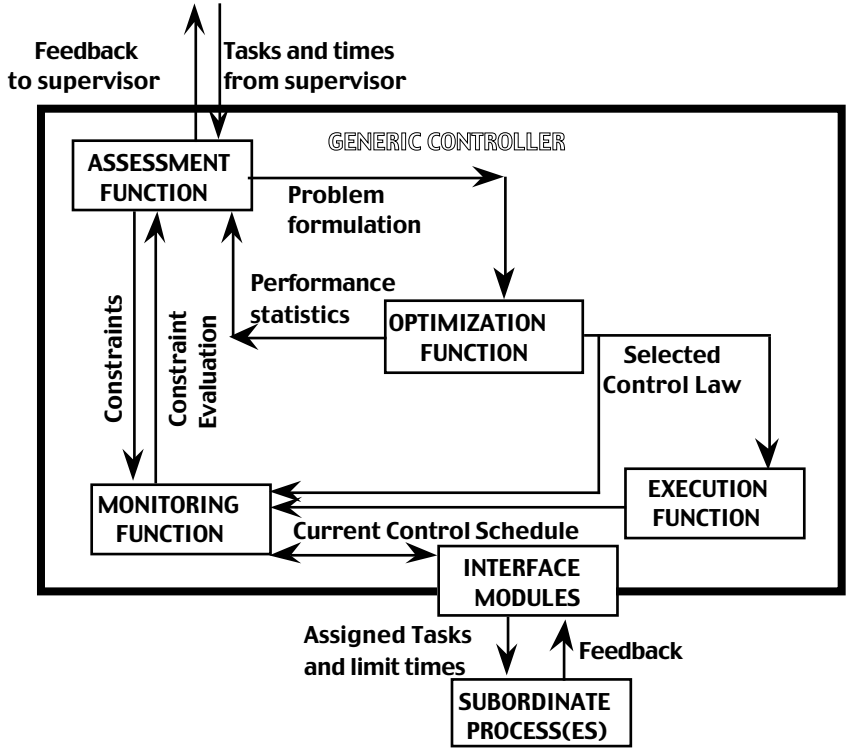

Figure 9.1 Generic controller module.

The Assessment Function formulates the real-time decision-making problems for each control module. The exact nature of those decisions depends on the hierarchical level at which the module resides (JACKSON and JONES, 1987). These decisions can be thought of as optimization problems. This means that the Assessment Function must specify both the constraints and the performance measures. Two types of constraints are allowed: hard and soft. Hard constraints are those that cannot be violated either by the other functions in the same module or by modules in subordinate levels in the hierarchy. These constraints come from three sources: supervisors, process plans, and the physical limitations of the system. A supervisor may impose hard constraints such as due dates, priorities, and maintenance schedules. The process planner may impose hard constraints in the form of precedence relationships, tools, and fixturing requirements. Finally, physical limits such as transfer times and queue sizes also result in hard constraints.

The Assessment Function can also specify soft constraints to further control the evolution and behavior of its subordinates. Typically, minor violations of these constraints will be tolerated, but major violations indicate that the system may be getting into trouble. For example start and finish times for the individual tasks that make up a job can be viewed as soft constraints. As long as the job is on time, delays in the start and finish times of some tasks can be tolerated. However,

as more and more of these tasks are delayed, the on-time completion of the job is jeopardized. Other commonly imposed soft constraints are utilization rates for subordinates and inventory policies. Unlike the hard constraints, these soft constraints may be unknown to subordinates.

As noted above, the Assessment Function also specifies the performance criteria for each optimization problem. There are typically several, possibly conflicting, criteria to be considered simultaneously, which combine the "performance" of subordinates and the "performance" of jobs. Examples of subordinate performance include utilization and throughput. Examples of job performance include lateness, tardiness, and makespan. Priorities can be assigned to particular jobs and weights to particular performance criteria. All of these can be changed to reflect the current state of the system.

The Optimization Function is responsible for solving these decision-making problems posed by the Assessment Function. The solution consists of a) selecting a run-time production plan for each job and b) selecting the start and finish times for each of the tasks in that plan. The production plan identifies the tasks and the resources needed to complete each job. It also includes all precedence relations that exist among those tasks. This run-time plan is selected from the set of feasible process plans passed down by the Assessment Function. Selecting the start and finish times for these tasks may involve the solution of a single machine sequencing problem, a multi-machine scheduling problem, a multi-cell routing problem, or a resource (tools, fixtures, transporters) allocation problem. All selections are made to optimize the current set of performance measures. In addition to making these initial selections, the Optimization Function must deal with violations of the constraints imposed by the Assessment Function. This may involve the selection of new sequences, schedules, or plans.

The Execution Function implements the decisions selected by the Optimization Function. Using the current state of the system, it does a single pass simulation to compute the start and finish times for each task to be assigned to one of its subordinate modules. In addition, when minor deviations from these times are either reported or projected by the subordinates, the Execution Function attempts to restore feasibility using techniques such as perturbation analysis or match-up scheduling (JONES and SALEH, 1990).

Lastly, the Monitoring Function updates the system state using feedback from subordinates, and evaluates proposed subordinate responses against the current set of imposed (from the Assessment Function) and computed (by the Optimization Function) constraints. It determines when violations occur, their severity, and who should deal with them.

9.3 Implementing the Optimization Function

We now describe a methodology for solving the real-time sequencing and scheduling (s/s) problems faced by the Optimization Function. This method consists of a three step refinement process. The first step is to generate a set of candidate s/s rules from a much larger set of heuristics. We have used single-performance, neural networks as discussed in Section 9.3.1. We then evaluate these candidates against all of the performance measures dictated by the

Assessment Function. This ranking is based on a real-time simulation approach as discussed in Section 9.3.2. The last step is to use the top candidates from that ranking as input to a genetic algorithm to determine the "best" sequence or schedule. This is discussed in Section 9.3.3. In Section 9.3.4, we describe a technique for extracting the knowledge contained in that schedule for future use.

9.3.1 Candidate Rule Selection

The first step in this process is to select a small list of candidate rules from a larger list of available rules. For example, we might want to find the best five dispatching rules from the list of all known dispatching rules so that each one maximizes (or minimizes) at least one of the performance measures, with no regard to the others. To carry out this part of the analysis, we have used neural networks. This approach extends earlier efforts by (RABELO, 1990) and (CHRYSSOLOURIS et al., 1991).

Neural networks have shown good promise for solving some classic, textbook job shop scheduling problems. (FOO and TAKEFUJI, 1988) and (ZHOU et al., 1990) have applied stochastic Hopfield networks to solve 4-job 3-machine and 10-job 10-machine job shop scheduling problems, respectively. These approaches tend to be computationally inefficient and frequently generate infeasible solutions. (LO and BAVARIAN, 1991) extended the two-dimensional Hopfield network to 3 dimensions to represent jobs, machines, and time. Another implementation based on stochastic neural networks applied to scheduling can be found in (ARIZONO et al., 1992). However, they have been unable to solve real scheduling problems optimally because of limitations in both hardware and algorithm development.

These implementations have been based on relaxation models (i.e., pre-assembled systems which relax from input to output along a predefined energy contour). The neural networks are defined by energy functions in these approaches. (LO and BAVARIAN, 1991) formulated the objective function which minimizes makespan as

$$E_t = (1/2) \sum_{j=1} \sum_{i=1} \sum_{l=1} (v_{ijl}/C_k) (l + T_{ij} - 1)$$

where C_k is a scaling factor; v_{ijl} is the output of neuron i_{jl}, and T_{ij} is the time required by j^{th} machine to complete the i^{th} job.

However, due to a large number of variables involved in generating a feasible schedule, it has been difficult for these approaches to solve realistic job shop scheduling problems with multiple objectives. It is even difficult to get a good suboptimal solution when attempting to solve problems in real-time.

There are four reasons to select neural networks as candidate rule selectors. First, because of the decomposition that results from the hierarchical control architecture we are using, we never have to solve the global shop floor scheduling problem all at once. Since it is decomposed into several scheduling and sequencing problems (of smaller size and complexity), we don't anticipate the kinds of problems described above. Second, it is no longer necessary to resolve the global problem each time a minor delay occurs or a new job is put into the

system. Local rescheduling and resequencing can be done with little impact on the overall shop floor schedule. Third, as discussed below, each neural network is designed (i.e., they are presented with training sets of representative scheduling instances and they learn to recognize these and other scheduling instances) to optimize a single objective (e.g., minimization of work-in process inventory). Neural networks in our approach are utilized as pattern recognition machines. Neural networks assign a given shop floor status to a specific rule with some degree. Finally, the solution from the neural networks is just the beginning of this methodology, not the end. Therefore, a very fast technique is needed. Neural networks are a proven real-time technique with speed (inherent from their distributed/parallel processing nature), timeliness, responsiveness, and graceful degradation capabilities.

In this research, we will focus our initial efforts on backpropagation neural networks, because they are more developed and much faster than the relaxation models described above (RUMELHART et al., 1988). Backpropagation applies the gradient-descent technique in a feed-forward network to change a collection of weights so that the cost function can be minimized. The cost function, which is only dependent on weights and training patterns, is defined by:

$$C(W) = (1/2) \sum (T_{ip} - O_{ip})^2$$

where the **T** is the target value, **O** is the output of network, **i** is the output nodes, and **p** is the number of training patterns.

After the network propagates the input values to the output layer, the error between the desired output and actual output will be "back-propagated" to the previous layer. In the hidden layers, the error for each node is computed by the weighted sum of errors in the next layer's nodes. In a three-layered network (see Figure 9.2), the next layer means the output layer. If the activation function is sigmoid, the weights are modified according to

$$\Delta W_{ij} = h\, X_j\, (1- X_j\ )(T_j - X_j\ )\, X_i \qquad (9.1)$$

or

$$\Delta W_{ij} = h\, X_j\, (1- X_j\ )\, (S\, d_k W_{jk}\ )\, X_i \qquad (9.2)$$

where W_{ij} is weight from node **i** to node **j**, **h** is the learning rate, X_j is the output of node **j**, T_j is the target value of node **j**, d_k is the error function of node **k**.

If **j** is in the output layer, Relation (9.1) is used. Relation (9.2) is for the nodes in the hidden layers. The weights are updated to reduce the cost function at each step.

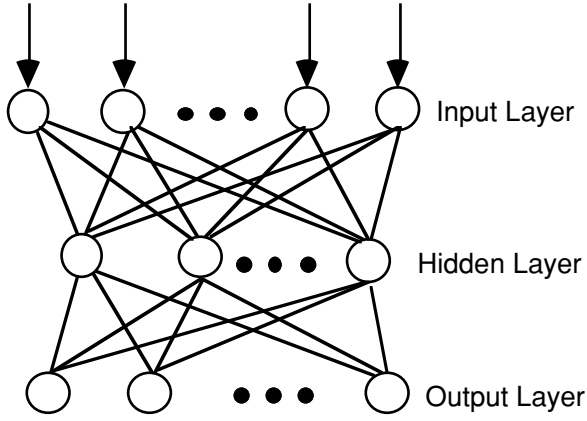

Figure 9.2 An example of a three-layer feed-forward neural network.

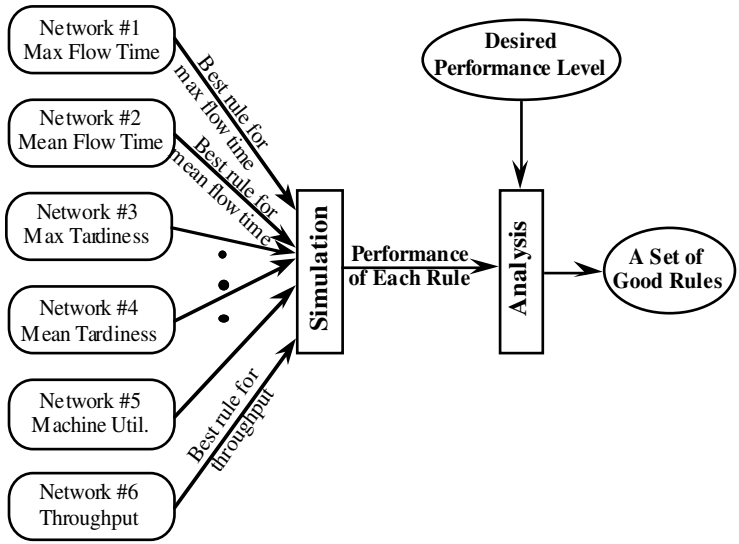

Figure 9.3 Architecture of the intelligent scheduling aid.

As indicated in Figure 9.3 our approach to developing the actual rule selector is to have backpropagation neural network trained to rank the available rules for each individual performance measure of interest (multiple performance evaluation comes in the next section). The weights for each of these networks are selected after a thorough training analysis. To carry out this training, we used two methodologies: 1) off-line training and 2) on-line training.

Off-Line Training

In off-line training, it is needed to generate training data sets for each of these performance measures from simulation studies. Suppose we wanted to train a neural net to minimize the maximum tardiness and we wanted to consider the following dispatching rules: SPT, LPT, FIFO, LIFO, SST, LST, CR, etc. (see Figure 9.4). After simulating each of these rules off-line under a variety of input conditions, we would be able to rank them to determine the best rule for this measure (The example in Section 9.4 gives some of these simulation results.). We would then use these results to train (i.e., choose weights) a neural net.

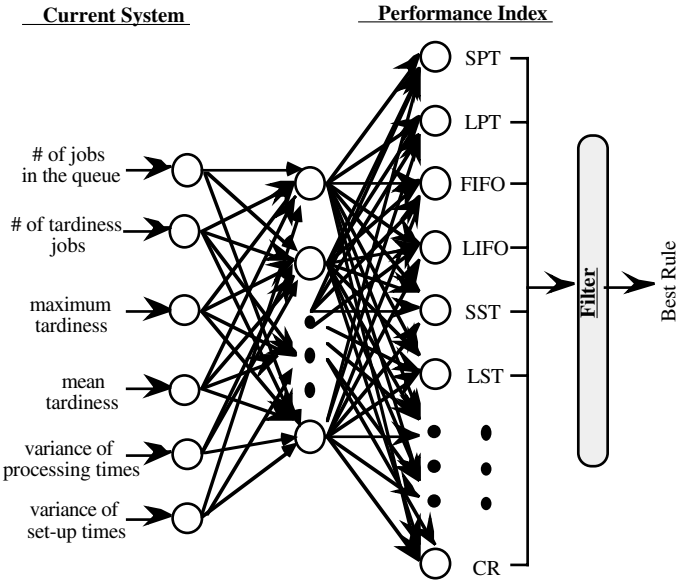

Figure 9.4 Neural network training for maximum tardiness.

On-Line Training

In on-line training, adaptive critics concepts are utilized to train in real-time the neural network structures (BARTO, 1992, WERBOS, 1992). Q-learning (a derivation of adaptive critics) (WATKINS, 1989) is used to predict a scheduling policy to meet the required performance criterion for a given queue status and undefined period of operation and therefore accomplish an effective candidate rule selector. The key idea of Q-learning is to assign values to state (shop floor status)-action (scheduling policy) pairs. Q-learning does not need an explicit model of the dynamic system underlying the decision problem. It directly estimates the optimal Q values (i.e., ranking) for pairs of states and admissible actions. The optimal Q value for state **i** (shop floor status) and action **u** (a scheduling heuristic) is a cost of executing action **u** in state **i**. Any policy selecting actions that are greater with respect to the optimal Q values is an optimal policy. Actions are ranked based on the Q values. On the other hand, ranking through an evaluation function requires more information like immediate costs of state action pairs and state transition probabilities. Instead of state transition probabilities Q-learning requires a random function to generate

successor states. The Q-value of the successful action is updated with learning parameters, although with the other admissible actions, Q values remain the same. Q-learning learns to accurately model the evaluation function. For a given state **x**, the system (e.g., a neural network) chooses the action **a**, where the utility **util(x,a)** is maximal. Q-learning consists of two parts: a utility function and a stochastic action selector. The utility function implemented using a neural network based on backpropagation works as both evaluator and policy maker. It tries to model the system by assigning values to action-state pairs. The neural network has multiple outputs, one for each action (as depicted in Figures 9.3 and 9.4).

We have initiated this training for a wide range of performance measures and dispatching rules for a single machine sequencing and multiple machine scheduling problems. We anticipate using these results for robots, machine tools, material handling devices and inspection devices. They will form the lower level of the two-level scheduling system which we are developing. Preliminary training results are described in (RABELO et al., 1993).

The output from the rule selector phase will be a collection of R matched pairs — {(performance measure, best rule)$_1$, ..., (performance measure, best rule)$_R$}. These pairs form the candidates which are passed on to the next phase for more detailed analysis.

9.3.2 Real-Time Simulation

After these R candidates have been determined, each of the rules must be evaluated to determine the impact that each rule will have on the future evolution of the system as measured from the current state of the system. In other words, we must predict how it does against all of the performance measures simultaneously. To carry out this analysis, we intend to use the technique developed by (DAVIS/JONES 1989) termed real-time Monte Carlo simulation. Since **R** rules must be considered, we plan to run **R** real-time simulations concurrently to avoid unacceptable timing delays in the analysis.

This form of Monte Carlo simulation differs considerably from traditional discrete-event simulation in two ways. First, each simulation trial is initialized to the current system state as updated in real-time by the Monitoring Function. This results in the second important distinction — these types of simulations are neither terminating nor steady state. They are not terminating because the initial conditions may change from one trial to the next. Furthermore, they are not steady state because we are specifically interested in analyzing the transient phenomena associated with the near-term system response. The question is, from a statistical perspective, does this really matter. Early experiments conducted by (DAVIS et al. 1991) indicate that the answer varies. That is, the inclusion or exclusion of new events corresponding to changes in the initial state can bias the statistical estimates of certain, but not all, performance measures.

The outputs from these simulation trials yield the projected schedule of events under each scheduling rule. These schedules are then used to compute the values of the various performance measures and constraints imposed by the Assessment Function. The computed statistics are used to select and develop the rule which

provides the best statistical compromise among the performance criteria. In the next section, we discuss a new approach to this compromise analysis, genetic algorithms.

9.3.3 Genetic Algorithms

No matter how the utility function described above is constructed, only one rule from the candidate list can be selected. This causes an undesirable situation whenever there are negatively correlated performance measures, because no one rule can optimize all objectives simultaneously. Conceptually, one would like to create a new "rule" which 1) combines the best features of the most attractive rules, 2) eliminates the worst features of those rules, and 3) simultaneously achieves satisfactory levels of performance for all objectives. Our approach does not deal with the rules themselves, but rather the actual schedules that result from applying those rules. Consequently, we seek to generate a new schedule from these candidate schedules. To do this, we propose to use a genetic algorithm approach. Presently, we give a brief description of how genetic algorithms (GAs) work. In the next section, we give some results from our preliminary experimentation which demonstrates that this approach can actually generate new and better schedules from existing ones.

9.3.3.1 Genetic Algorithms and Scheduling

GAs have been utilized in job shop scheduling. GAs could be utilized using the following schemes:

(1) GAs with blind recombination operators have been utilized by GOLDBERG and LINGLE (1985), DAVIS (1985), SYSWERDA (1990), and WHITLEY et al. (1989). Their emphasis on relative ordering schema, absolute ordering schema, cycles, and edges in the offspring will arise from differences in such blind recombination operators.

(2) Sequencing problems have been also addressed by the mapping of their constraints to a Boolean satisfiability problem (DE JONG and SPEARS, 1989) using partial payoff schemes. This scheme has produced good results for simple problems. However, this scheme needs more research.

(3) Heuristic genetic algorithms have been applied to job shop scheduling (BAGCHI et al., 1991). In these GAs, problem specific heuristics are incorporated in the recombination operators (such as optimization operators).

Example: Using a simple genetic algorithm for sequencing

This example illustrates the utilization of a simple genetic algorithm based on a blind recombination operator for sequencing problems. The partially mapped crossover (PMX) operator developed by GOLDBERG and LINGLE (1985) will be utilized. Consider a single machine sequencing problem with 7 types of jobs. Each job-type has its own arrival time, due date, and processing time distributions. The set-up time is sequence dependent as shown in Table 9.1. The objective is to determine a sequence that minimizes Maximum Tardiness for the 10-job problem described in Table 9.2.

	Current job-type						
Previous job-type	1	2	3	4	5	6	7
1	0	1	2	2	3	2	2
2	1	0	2	3	4	3	2
3	2	2	0	3	4	3	2
4	1	2	3	0	4	4	2
5	1	2	2	3	0	3	2
6	1	2	2	3	3	0	3
7	1	2	2	2	2	2	0

Table 9.1 Set-up times.

Job #	Job Type	Mean Processing Time	Arrival Time	Due Date
1	6	8	789	890
2	6	8	805	911
3	5	10	809	910
4	1	4	826	886
5	2	6	830	905
6	7	15	832	1009
7	6	8	847	956
8	3	5	848	919
9	1	4	855	919
10	4	3	860	920

Current Time: 863
Previous Job Type executed: 3

Table 9.2 10-Job problem description.

The simple genetic algorithm procedure developed (different possible procedures could be developed, the one demonstrated is only for illustration purposes) for this sequence problem could be described as follows:

1. Randomly generate n legal sequences (n is the population size, e.g., n = 50).

2. Evaluate each sequence using a fitness function (for this problem: Minimization of Maximum Tardiness — the sequences will be ranked according to their Maximum Tardiness — the lower the better)

3. Choose the best sequences (m sequences with the lower values for Maximum Tardiness, m < n, e.g., 25).

4. Reproduction (e.g., duplicate them, stop when you have a population of n). This reproduction could be in function of the fitness value (sequences with the best fitness values will have higher probability to reproduce).

5. Crossover. Select randomly pairs of sequences. (Crossover could be applied to the best sequences. However, the offspring do not replace their parents, but rather a low ranking individual in the population (WHITLEY and STARKWEATHER, 1990). Apply PMX:

The PMX operator produces legal solutions by choosing a swapping interval between two crossover points selected randomly. The offspring will inherit the elements from the interval of one of the parents. Then, it is necessary to detect and fix the illegal situations by mapping and exchanging. For example, consider two sequences (A and B):

Position:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A(Job Numbers):	9	8	4	5	6	7	1	3	2	10
B(Job Numbers):	8	7	1	2	3	10	9	5	4	6

Swapping interval (randomly generated): 4 to 6.

Position:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A:	9	8	4 |	5	6	7 |	1	3	2	10
B:	8	7	1 |	2	3	10 |	9	5	4	6

Exchanging:

Position:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A':	9	8	4 |	2	3	10 |	1	3	2	10
B':	8	7	1 |	5	6	7 |	9	5	4	6

Mapping and Exchanging to create legal sequences:

Position:	1	2	3	4	5	6	7	8	9	10
A'':	9	8	4	2	3	10	1	6	5	7
B'':	8	10	1	5	6	7	9	2	4	3

6. Mutation (with a low probability, e.g., 0.1, exchange two arbitrary jobs' position). (Mutation might not be applied to some schedules.)

Example: we have the following sequence **9 8 4 5 6 7 1 3 2 10**,
applying mutation, the sequence could become **9 8 6 5 4 7 1 3 2 10**,

Jobs 4 and 6 exchanged positions in the sequence.

7. Repeat (2) to (6), until no more improvements are possible.

After 17 iterations (approximately: 17 * 50 = 850 sequences were generated and tested), the genetic algorithm produces the following sequence for this simple problem (this took less than 500 milliseconds in a PC 486 @ 33MHz):

8 5 4 1 2 3 9 10 6 7 with a Maximum Tardiness of **2** (Fitness Function).

This is an optimal sequence. Studies of all possible combinations (**10! = 3628800**) produced Table 9.3 for the same sequencing problem. In addition,

Table 9.4 indicates some of the solutions por the same problem using dispatching rules.

Maximum Tardiness (MT)				
(Range)				
2.<= MT < 6	**6<=MT<10**	**10<=MT<13**	**13<=MT<=75**	
Number of Possible Sequences	68	1344	5568	3621820

Table 9.3 Evaluating 3628800 Sequences (exhaustive search)

Dispatching Rule	**Sequence**	**MT**
SPT Shortest Processing Time	10 4 9 8 5 1 2 7 3 6	23
FIFO First-In/First-Out	1 2 3 4 5 6 7 8 9 10	32
EDD Earliest Due Date	4 1 5 3 2 8 9 10 7 6	10
LIFO Last-In/First-Out	10 9 8 7 6 5 4 3 2 1	65

Table 9.4 Using dispatching rules.

9.3.3.2 Genetic Algorithms for Compromise Analysis

The compromise analysis process carried out by a genetic algorithm can be thought of as a complex hierarchical "generate and test" process. The generator produces building blocks which are combined into schedules. At various points in the procedure, tests are made that help weed out poor building blocks and promote the use of good ones. To reduce and support uncertainty management of the search space, the previous two steps (candidate rules selection and parallel simulation with statistical analysis) provide partial solutions to the problem of compromise analysis. Reducing uncertainty makes the search process more effective, with the complexity of the scheduling problem becoming more manageable in the process (see Figure 9.5).

9.3.4 Inductive Learning Algorithm — TDKA

Now that this new schedule has been generated, we want to extract the knowledge contained in that schedule for future use. To do this, we must derive a "new rule" which can be used to regenerate schedules in the same way that other dispatching rules like SPT (Shortest Processing Time first) are used. This new rule will not, however, be a simple dispatching rule. To do this, we will use a technique developed in (YIH, 1988 and YIH, 1990) called Trace-Driven Knowledge Acquisition (TDKA). TDKA is a method that extracts knowledge from the actual results of decisions rather than statements or explanations of the presumed effects of decisions. There are three steps in the process of trace-driven knowledge acquisition: *data collection*, *data analysis*, and *rule evaluation*.

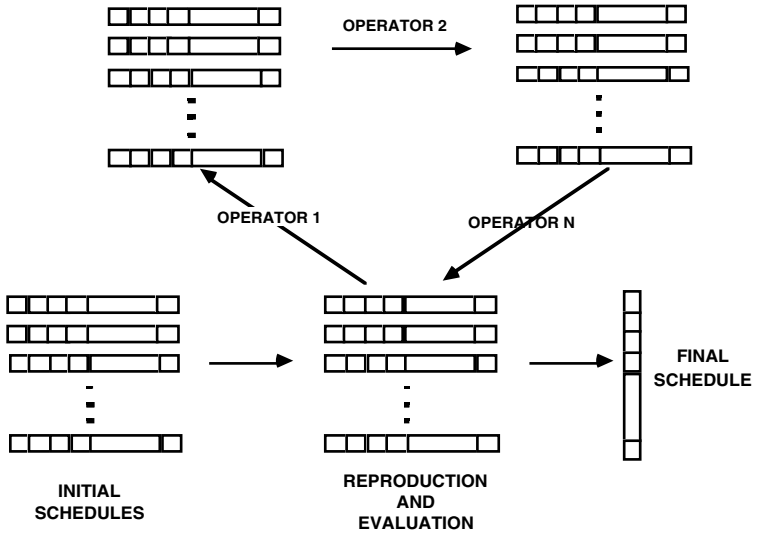

FIGURE 9.5 Genetic algorithm for compromise analysis.

In step one, data is collected through simulation, historical records, human inputs or actual physical experiments. (THESEN et al., 1987 and YIH, 1992) describe their efforts to use human experts. In this application, the data (trace) is simply the schedule generated from the Genetic Algorithm. In step two, the schedule is analyzed to extract a set of production rules (If-Then rules) which could be used to regenerate the same schedule. In step three, simulation is used to compare the generated schedule with the original schedule. The process returns to step two to refine the rules if the comparison is unacceptable.

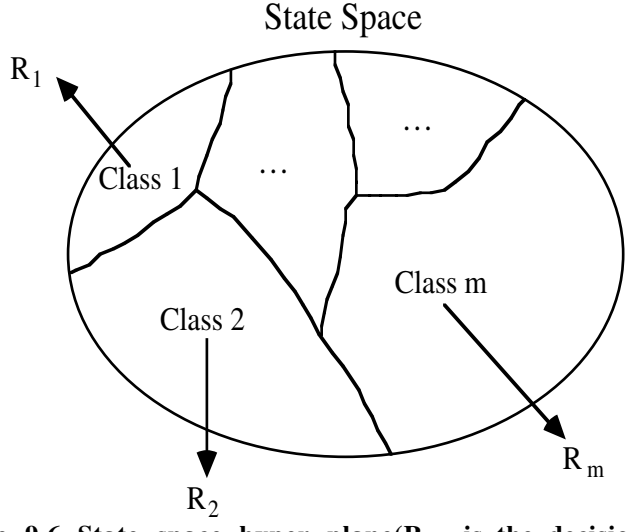

Figure 9.6 State space hyper plane(R_m is the decision rule applied in class m).

The core of the TDKA lies in the step two-data analysis. There are two types of rules involved — decision rules and class assignment rules. If the state space is viewed as a hyper plane, class assignment rules draw the boundaries on the hyper plane to define the areas of classes as shown in Figure 9.6. For each class, a single production rule is used to determine what to do next. These rules are in the form of

"If [state ∈ class i] then apply R_i^*"

In data analysis, the records collected are used to define classes and to select a single rule for each class. The rule formation algorithm can be described by three steps as follows.

I. Initialize

Determine state variables
Determine decision rules
Set initial value for acceptance level (**L**)
Determine initial class C_1
Obtain a trace (a set of records) from simulation

II. Vote

Each record in this class (C_i) votes for all the decision rules that will result in the same decision as in the record
Summarize the vote results in percentage (No. of votes / No. of records)

III. Form rules

The decision rule (R_k^*)with the highest percentage wins.
If the percentage is higher than L, then form the following rule

If [state ∈ C_i] then apply R_k^*

else
Split class C_i into two classes based on selected state variable (V_p*) and threshold (Th_i^*).

Add two class assignment rules.

IF [state ∈ C_i] and [$V_p* < Th_i^*$] THEN [state ∈ C_{i1}]

IF [state ∈ C_i] and [$V_p* \geq Th_i^*$] THEN [state ∈ C_{i2}]

Repeat Steps II and III for classes C_{i1} and C_{i2}.

IV. Stop

This iterative process will stop whenever the rule formation is completed. That is, the state space has been properly divided into several classes and a single production rule is selected for each class. If the performance of the extracted rules is worse than the trace, then it is necessary to return to data analysis and refine the rules by increasing the value of acceptance level (**L**). Otherwise, the process stops.

9.4 An Example

Consider a single machine sequencing problem with 7 types of jobs. Each job-type has its own arrival time, due date, and processing time distributions. The set-up time is sequence dependent as shown in Table 9.1. The objective is to determine a sequence which minimizes the summation of mean flow time and maximum tardiness. Assume that we are to generate a sequence for the 10 jobs in the input queue of the machine as described in Table 9.5.

Job #	Job Type	Mean Processing Time	Arrival Time	Due Date
1	7	5	154	203
2	3	5	154	193
3	4	3	159	194
4	3	5	160	208
5	2	4	166	194
6	1	4	170	202
7	3	5	185	231
8	2	4	186	221
9	7	5	192	243
10	7	5	200	250

Current Time: 200
Previous Job Type executed: 3

Table 9.5 Job description for the example problem.

The candidate rule selector implemented, using backpropagation neural networks (previously trained by off-line training), uses the system status and the performance criteria (mean flow time and maximum tardiness) in order to select a small set of candidates from the 13 rules available (see Table 9.6). Each neural network has six inputs (as shown in Figure 9.4), 12 hidden units in the hidden layer, and 13 outputs (one for each dispatching rule). Each neural network (one for mean flow time and the other for mean tardiness) in parallel ranks all rules. The networks developed in the C programming language take on average less than 10 mS (486 PC compatible @ 33 MHz) to give an answer to the problem. The neural network for mean flow time ranks higher SPST (Shortest Processing and Shortest Set-upTime first) and SST (Set-up Time first). On the other hand, the neural network for maximum tardiness ranks higher EDD (Earliest Due Date first) and mSLACK (Smallest Slack first).

A genetic algorithm is utilized having as initial populations the selected schedules and some randomly generated schedules. The fitting function is a combination of all performance measures with coefficients reflecting the "importance" of each one according to the imposed criteria. The crossover mechanism utilized is based on "order Crossover" as developed by Syswerda (1990). The genetic algorithm takes on average eleven iterations with an average time of less than 700 mS on a 486 PC @ 33 MHz. The new schedule compromises both measures with an acceptable degree of success (Mean Flow Time = 73.5 and Maximum Tardiness = 6). In order to verify the answers, a program that generates all possible solutions to the scheduling problem is

generated (10!), taking on average 2 hours of cpu time of the 486 PC @ 33 MHz. The "new" schedule generated by the genetic algorithm is superior (based on the combined performance criteria) to the initial schedules selected by the neural networks — (see Table 9.6). The output, shown at the bottom of Table 9.6, is (**2 3 6 5 1 4 7 8 10 9**). This sequence performs better than EDD for maximum tardiness while maintaining good performance in mean flow time (better than the third ranked rule-SPT). However, it is possible to identify some of the relative positions of the dispatching heuristics in the final schedule. This schedule is selected to be applied to the manufacturing system.

Dispatching Rule	Mean Flow Time	Rank	Maximum Tardiness	Rank	Job Sequence
SPT	62.4	3	46	7	3 5 6 8 1 2 4 7 9 10
LPT	63.4	5	67	10	1 2 4 7 9 10 5 6 8 3
FIFO	66.0	11	42	5	1 2 3 4 5 6 7 8 9 10
LIFO	64.4	7	73	13	10 9 8 7 6 5 4 3 1 2
SST	59.2	2	61	8	2 4 7 1 9 10 6 5 8 3
LST	66.2	12	40	4	3 1 2 5 4 6 7 8 9 10
SPST	58.5	1	42	6	2 4 7 3 6 5 8 1 9 10
LPST	67.6	13	63	9	1 2 9 4 10 7 3 5 6 8
EDD	63.2	4	31	1	2 3 5 6 1 4 8 7 9 10
LDD	64.8	8	72	11	10 9 7 8 4 1 6 3 5 2
mSLACK	63.4	6	31	2	2 3 5 1 6 4 8 7 9 10
MSLACK	64.8	9	72	12	10 9 7 8 4 1 6 3 5 2
CR	65.7	10	34	3	3 5 2 6 1 4 8 7 9 10
Genetic Algorithm*	61.7	-	30	-	2 3 6 5 1 4 7 8 10 9

Note:

SPT - Shortest Processing Time
LPT - Longest Processing Time
FIFO - First In First Out
LIFO - Last In First Out
SST - Shortest Set-up Time
LST - Longest Set-up Time
SPST - Shortest Pro and Set-up Time
LPST - Longest Proc and Set-up Time
EDD - Earliest Due Date
LDD - Latest Due Date
mSLACK - Smallest slack
MSLACK - Largest slack
CR - Critical Ratio (slack/processing time)

Table 9.6 Results from commonly used heuristics and genetic algorithm.

We now show how to use TDKA to generate a new sequencing rule. Using the GA sequence (**2 3 6 5 1 4 7 8 10 9**), we first obtain the trace from simulation in the following format.

(Q, NT, MT, AT, VP, VS) ---> Action where
Q: number of jobs in the queue
NT: number of tardy jobs
MT: maximum tardiness

AT: mean tardiness
VP: variance of processing times
VS: variance of set-up times
Action: job identification number to be selected next

For instance, the trace from simulation is:

(10, 4, 10, 7.25, 0.50, 1.17) --> 2
(9, 4, 14, 9.75, 0.53, 1.00) --> 3
(8, 4, 17, 11.3, 0.27, 0.13) --> 6
(7, 3, 14, 13.0, 0.24, 0.24) --> 5
(6, 2, 16, 16.0, 0.17, 0.67) --> 1
(5, 2, 16, 9.5, 0.20, 1.20) --> 4
(4, 1, 8, 8.0, 0.25, 1.00) --> 7
(3, 1, 13, 13.0, 0.33, 0.00) --> 8
(2, 0, 0, 0.0, 0.00, 0.00) -->10

Each record in the trace will vote for the rules that could yield the same decision as in the sequence. The summary of votes is listed in the following table.

We start with one class, called Class 1. If we are satisfied with the accuracy of 67%, that is 67% is higher than the acceptance level (L), we may arbitrarily choose SST, SPST, EDD or mSLACK and form the following rule.

"If [state ∈ Class 1] then apply SST" (9.1)

If we would like to obtain higher accuracy, the variable MT (maximum tardiness) can be used to split Class 1 into two subclasses, Class 11 and Class 12. Class 11 includes the states with MT ≥ 10, and Class 12 has the remainder. The following class assignment rules will be formed.

Rule	Votes	%
SPT	5	55%
LPT	5	55%
FIFO	5	55%
LIFO	1	11%
SST	6	67%
LST	5	55%
SPST	6	67%
LPST	3	33%
EDD	6	67%
LDD	1	11%
mSLACK	6	67%
MSLACK	1	11%
CR	5	55%

Number of records = 9

Table 9.7 Summary of the record votes (Class 1).

"If [MT ≥ 10] then state ∈ Class 11" (9.2)

"If [MT < 10] then state ∈ Class 12" (9.3)

After splitting into two classes, the voting process repeats within each class. The results are listed in Tables 9.8 and 9.9 in Appendix 1.

The following rule for Class 12 could be formed with 100% accuracy with LPT, SST, or SPST.

"If [state ∈ Class 12] then apply SPST" (9.4)

In Class 11, if we are satisfied with the accuracy of 86%, we may form the following rule with EDD or mSLACK.

"If [state ∈ Class 11] then apply EDD" (9.5)

However, if we would like to achieve higher accuracy, MT is used again to split Class 11 into Class 111 and Class 112 with threshold of 17. The following rules are formed.

"If [state ∈ Class 11] and [MT≥17] then state ∈ Class 111"(9.6)

"If [state ∈ Class 11] and [MT<17] then state ∈ Class 112"(9.7)

After splitting, the voting results are summarized in Tables 9.10 and 9.11 in the Appendix included at the end of this chapter.

Two rules may be formed with 100% accuracy as follows.

"If [state ∈ Class 111] then apply SPT" (9.8)

"If [state ∈ Class 112] then apply EDD" (9.9)

Finally, rules (9.2), (9.3), (9.4), (9.6), (9.7), (9.8), and (9.9) will be included in the rule base and this set of rules is able to generate sequences based on the strategy embedded in the sequence derived from the GA.

9.5 Remarks

In this chapter, we have described a methodology to solve sequencing and scheduling problems which integrate neural networks, simulation, genetic algorithms, and machine learning techniques. We also described a small example which demonstrates the methodology. By exploiting the parallel processing and modeling capabilities of the neural nets, simulation, and genetic algorithms it has the potential to be extremely fast and highly adaptable to customer needs. Finally, we described a learning technique which extracts the knowledge from the derived schedules and creates new rules. This provides the additional capability to

learn what works and what does not work in a variety of situations and utilize that knowledge at a later time. For these reasons, we believe that this technique has the potential to solve real-world sequencing and scheduling problems in real-time.

References

ARIZONO I., YAMAMOTO A., and OHTA, H., "Scheduling for Minimizing Total Actual Flow Time by Neural Networks," *International Journal of Production Research*, 1992, Vol. 30, No. 3, pp. 503-511.

BAGCHI, S., UCKUN, S., MIYABE, Y., and KAWAMURA, K., "Exploring Problem-Specific Recombination Operators for Job Shop Scheduling," *Proceedings of the Fourth International Conference on Genetic Algorithms and Their Applications*, pp. 10-17, 1991.

BARTO, A., "*Reinforcement Learning and Adaptive Critic Methods,*" Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive approaches. D.A. White and D.A. Sofge (Ed.) Van Nostrand Reinhold Publication, 1992.

CHRYSSOLOURIS, G., LEE, M., and DOMROESE, M., "The Use of Neural Networks in Determining Operational Policies for Manufacturing Systems," *Journal of Manufacturing Systems*, 10, pp. 166-175, 1991.

DAVIS L., "Job Shop Scheduling with Genetic Algorithms," *Proceedings on an International Conference on Genetic Algorithms and Their Applications*, Carnegie-Mellon University, pp. 136-140, 1985.

DAVIS W. and JONES A., "Issues in real-time simulation for flexible manufacturing systems", *Proceedings of the European Simulation Multiconference*, Rome, Italy, June 7-9, 1989.

DAVIS, W., WANG, H., and HSIEH, C., "Experimental Studies in Real-time, Monte Carlo Simulation", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 4, pp. 802-814, 1991.

DAVIS W., JONES A., and SALEH A., "A Generic Architecture for Intelligent Control Systems", *Computer Integrated Manufacturing Systems*, Vol. 5, No. 2, pp. 105-113, 1992.

DE JONG, K. and SPEARS, W., "Using Genetic Algorithms to solve NP-Complete Problems," *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 124-132, 1989.

JACKSON, R. and JONES, A., "An Architecture for Decision Making in the Factory of the Future, *INTERFACES*, Vol. 17, NO. 6, pp. 15-28, 1987.

JONES, A. and SALEH, A., "A Multi-level/Multi-layer Architecture for Intelligent Shop Floor Control," *International Journal of Computer Integrated Manufacturing Special Issue on Intelligent Control*, 3, 1, pp. 60-70, 1990.

FOO Y. and TAKEFUJI Y., "Stochastic Neural Networks for solving job shop Scheduling", *Proceedings of the IEEE international Conference on Neural Networks*, published by IEEE TAB, 1988, pp. II275-II290.

GOLDBERG D., *Genetic Algorithms in Machine Learning*, Addison-Wesley, Menlo Park, California, 1988.

GOLDBERG, D., and LINGLE, R., "Alleles, loci, and the Traveling Salesman Problem," *Proceedings of the of the International Conference on Genetic Algorithms and Their Applications*, 1985.

LO, Z. and BAVARIAN, B., "Scheduling with Neural Networks for Flexible Manufacturing Systems," *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, Sacramento, California, pp. 818-823, 1991.

RABELO, L., "A hybrid artificial neural network and expert system approach to flexible manufacturing system scheduling", PhD Thesis, University of Missouri-Rolla, 1990.

RABELO, L, YIH, Y., JONES, A., and WITZGALL, G., "Intelligent FMS Scheduling using Modular Neural Netwroks", *Proceedings of ICNN'93*, pp. 1224-1229, 1993.

RUMELHART, D., McCLELLAND, J., and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations, Cambridge, MA: MIT Press/Bradford Books, 1988.

SYSWERDA, G., "*Scheduling Optimization using Generic Algorithms,*" Handbook of Genetic Algorithms, pp. 332-349, 1990.

WATKINS, C., Learning From Delayed Rewards, PhD Thesis, Cambridge University, Cambridge, England, 1989.

WERBOS, P., "Approximate Dynamic Programming for Real Time Control Neural Modelling," *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches.* White and Sofge (eds.) Van Nostrand Reinhold Publication, pp. 493-525, 1992.

WHITLEY, D. and STARKWEATHER, T., "*GENITOR II: A Distributed Genetic Algorithm,"* Journal of Experimental and Theoretical Artificial Intelligence, Vol. 2, pp. 189-214, 1990.

WHITLEY D., STARKWEATHER T., and FUQUAY D., "Scheduling Problems and the Traveling Salesman: the genetic edge recombination operator," *Proceedings of the Third International Conference on Genetic Algorithms,* pp. 133-140, 1989.

YIH, Y., Trace Driven Knowledge Acquisition for Expert Scheduling System, Ph.D. dissertation, University of Wisconsin-Madison, December, 1988.

YIH, Y., "Trace-Driven Knowledge Acquisition (TDKA) for Rule-Based Real-Time Scheduling Systems," *Journal of Intelligent Manufacturing*, 1, 4, pp. 217-230, 1990.

YIH, Y., "Learning Real-Time Scheduling Rules from Optimal Policy of Semi-Markov Decision Processes," *International Journal of Computer Integrated Manufacturing*, Vol. 5, No. 3, pp. 171-181, 1992.

ZHOU D., CHERKASSKY V., BALDWIN T., and HONG D., "Scaling Neural Network for Job Shop Scheduling," *Proceedings of the International Conference on Neural Networks*, Vol. 3, pp. 889-894, 1990.

Rule	Votes	%
SPT	4	57%
LPT	3	43%
FIFO	4	57%
LIFO	0	0%
SST	4	57%
LST	4	57%
SPST	4	57%
LPST	2	29%
EDD	6	86%
LDD	0	0%
mSLACK	6	86%
MSLACK	0	0%
CR	5	71%

Number of records = 7

Table 9.8 Summary of the record votes (Class 11).

Rule	Votes	%
SPT	1	50%
LPT	2	100%
FIFO	1	50%
LIFO	1	50%
SST	2	100%
LST	1	50%
SPST	2	100%
LPST	1	50%
EDD	0	0%
LDD	1	50%
mSLACK	0	0%
MSLACK	1	50%
CR	0	0%

Number of records = 2

Table 9.9 Summary of the record votes (Class 12).

Rule Candidate	# of Votes	Percentage
SPT	1	100%
LPT	0	0%
FIFO	0	0%
LIFO	0	0%
SST	1	100%
LST	0	0%
SPST	1	100%
LPST	0	0%
EDD	0	0%
LDD	0	0%
mSLACK	0	0%
MSLACK	0	0%
CR	0	0%

Number of records = 1

Table 9.10 Summary of the record votes (Class 111).

Rule Candidate	# of Votes	Percentage
SPT	4	57%
LPT	3	43%
FIFO	4	57%
LIFO	0	0%
SST	4	57%
LST	4	57%
SPST	4	57%
LPST	2	29%
EDD	6	86%
LDD	0	0%
mSLACK	6	86%
MSLACK	0	0%
CR	5	71%

Number of records = 6

Table 9.11 Summary of the record votes (Class 112).