[illegible]

# COMBINING GENETIC ALGORITHMS AND ARTIFICIAL NEURAL NETWORKS TO SELECT HETEROGENEOUS DISPATCHING RULES FOR A JOB SHOP SYSTEM

A Thesis Presented to

The Faculty of the

Fritz J. and Dolores H. Russ
College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

Thesis
M
1996
WIL

by

Daniel B. Wilson

November, 1996

[illegible]

OHIO UNIVERSITY
LIBRARY

TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# 1. INTRODUCTION

## 1.1 Scheduling Importance

The ability to produce goods to meet demand is what allows manufacturers to stay in business. If they cannot produce the required goods within a specified amount of time the purchasers for those goods will not buy the goods from that manufacturer. From the manufacturing viewpoint however, the goal is to produce the required goods with as little cost incurred as possible. This leads the manufacturer to purchase as few resources as possible to produce the required goods, and thus causes the usage of the resources to become a very important issue. If the resources are used inefficiently, either more resources will be required to produce the necessary goods, or the goods will not be produced on time. For these reasons, several methods of scheduling products through production facilities have been tried.

Some of the methods aimed at producing efficient schedules try to produce optimal schedules, while others try to produce good schedules quickly. By creating efficient schedules, a system's capacity can be increased and the time needed to produce goods can be decreased. Since this increase in efficiency costs nothing in terms of machinery or labor (other than the costs of finding the scheduling improvements), this is an area of concentration which can greatly reduce manufacturing costs without the amount of capital investment required to procure more production resources.

## 1.2 Scheduling in Academia

The methods used by academia to produce schedules are often very different from those used in the manufacturing environment. In academia, mathematical approaches such as linear and nonlinear programming are often used to find "optimal" schedules. The problems with mathematical approaches are that they require a total understanding of the entire facility and its practices, and a great deal of time is required to formulate the problems. Another approach used in academia is to either enumerate all possible solutions for scheduling individual parts, or run a search of the possibilities for individual routing of each part. While these methods may find optimal or nearly optimal schedules, they too require immense investments of time and energy into the problem, often far too much time to allow the schedules to be modified for different sets of inputs.

Operations Research (OR) is used extensively in the academic field, but is seldom implemented in manufacturing. According to Ackoff (1979) "OR came to be identified with the use of mathematical models and algorithms rather than the ability to formulate management problems, solve them, and implement and maintain their solutions in turbulent environments.". It is worth noting that OR is *not* just limited to mathematical models and algorithms, but it has come to be identified with these things. In many "OR" departments a much wider range of techniques is seen to fall under the umbrella of what is considered operations research. These other techniques include genetic algorithms, tabu search, neural networks, and many other "non-mathematical" techniques (Technically these methods all depend on mathematics to some degree, but they are not necessarily based upon an explicit mathematical representation of the system being analyzed.), as well as the application of heuristics to certain types of problems.

Another reason for the lack of application of OR in industry is that a great deal of simplification is needed to allow proper formulation of the problems. O'Grady and Menon (1986) state that "The application of these approaches is dependant on sustaining the simplifying assumptions which are invoked either to reduce problem complexity to manageable proportions or to restructure the problem so as to make it compatible with the format of a general approach, e.g., queuing theory, simulation or mathematical programming." In many cases the magnitude of simplification required to reduce a problem to some manageable form makes the results obtained through the use of such a simplistic model useless. Again this is not to say that OR techniques cannot be used, but rather that the types of mathematical approaches commonly associated with OR are not necessarily the best methods for solving these problems.

It is also important to note that out of Operations Research has come the dispatching rules which are most commonly used in industry. These dispatching rules, often called heuristics, do not necessarily produce optimal results, but in many cases the use of these rules allows quick "workable" solutions to be found. It is the selection of these rules that is the focus of this thesis.

## 1.3 Scheduling in Industry

In the manufacturing environment, the emphasis is not so much on optimality as it is on speed and effectiveness. In this setting the "optimum solution" may be made sub-optimal by something as simple as a machine stopping or a worker showing up late for work. This along with the time and effort required to find such schedules causes the emphasis in manufacturing to shift from optimal to just good-enough. Thus schedules in manufacturing depend much more heavily on quick and simple approaches to scheduling, such as using the scheduling rules that come with commercial simulation-based scheduling packages, and much less on complex mathematical or analytical approaches.

Discrete-event, simulation-based scheduling packages are widely used in manufacturing plants around the world. These packages include a large number of "canned" dispatching rules which will produce schedules for both simple and complex production environments. Users can extend these rules by adding their own plant-specific rules. It is important to note that these rules are guaranteed to produce only feasible schedules. The notion of optimality (or near optimality) with respect to one or more performance measures is typically not considered. In fact, the knowledge about which rule(s) to use to achieve some desired performance measure(s) must be supplied by the user or derived from extensive experimentation with the packages.

In many cases the scheduling rules chosen for implementation are not selected based on any real data, but merely on personal biases, previous practice, and in some cases rules are used because they are the defaults used by the simulation packages. These methods of selection are not conducive to the creation of good schedules. Stecke and Solberg (1981) reported that for one facility, good selections for scheduling rules, along with other changes, led to an improvement of 25% over the previous rules which had been selected intuitively. This type of dramatic improvement is possible in many situations, but there must be some method available to provide feedback as to which rules will yield this type of improvement.

Another difficulty encountered with many simulation packages is that performance measures are often not included in them. Many packages include statistical analysis and graphical outputs, but the types of performance measures needed are not readily available. Since the scheduling in a particular facility often is based on a particular measure, this shortcoming is a very serious one. Any approach which attempts to address these difficulties must provide a method for dealing with the differing problems encountered with the use of commercial software packages.

## 1.4 General Hypothesis

The purpose of this thesis is to design a method through which neural networks may be used to predict what dispatching rules should be used on a given system for a set of orders into the system. This will be done by combining the techniques used for scheduling in academia with the techniques used in manufacturing in order to provide a quick and effective way to produce near-optimal choices of scheduling rules. This will allow an industry to increase its efficiency, and thus produce more goods with fewer resources, without the time-consuming problems involved in many scheduling algorithms.

The specific problem approached in this thesis is the heterogeneous dispatching rule problem, in a finite horizon job shop scheduling system, with empty start and finish conditions. This limits the problem to very specific bounds, but once this research is complete, more work can be done to expand the results to other types of problems. The problem is called a heterogeneous dispatching rule problem because it does not limit the choice of dispatching rules to a single rule for use in all parts of the system. In many manufacturing facilities, one dispatching rule is used on every machine, and for this problem there is little difficulty in determining which rule to use because there are only a very limited number of choices to be made. In the heterogeneous dispatching rule problem, however, a different dispatching rule may be used for each machine, making the problem of solving which dispatching rules to use much more complex and difficult to solve. The benefit of using the heterogeneous approach is that a better solution can often be found, allowing the system to operate much more efficiently. The problem is considered a finite horizon problem due to the fact that a solution is found only for the orders given at time zero, and no past or future orders will be considered. This assumption is consistent with a job shop in which orders are received in the morning and no new orders are obtained until all previous orders are completed. This assumption was also made to simplify the problem, and allow the neural networks to learn easily. Once this thesis is completed, further work may be done into the "infinite horizon" or "rolling-horizon" problems in which new orders are considered as they are received.

In this thesis a new approach is used in which first rule selections are generated using a genetic algorithm (GA), then a rule-set replacement technique is performed to reduce the noise present in the GA results, and finally these generated rules are used to train a neural network to create near optimal schedules. The GA is an optimal search technique which generates candidate rule selections. Candidate rules are evaluated using a commercial, simulation-based scheduling package. The evaluation will predict how well the system will perform if the resulting rules are implemented. This "generate-evaluate" loop continues until the optimal rules for a given order are found, or some threshold limit has been reached. By optimality what is meant is that no further improvement in the performance measure(s) can be found. Threshold limit is a search time (such as one second) or a maximum number of candidates (such as 100 sets of rules) to evaluate. Once this part of the process is completed for a number of different orders or inputs into the system, the orders and the resultant rule selections are modified according to the rule-set replacement technique. This technique reads the rules selected by the GA for one order, and tries these rules on other orders to see if the same rules can be used successfully in the system for other orders. This allows the neural networks to learn from the data better by reducing the random variation in the data. Once this step is completed, each order will have four rules associated with it, one for each machine, which produce good results when used in the simulation for that order. These order-rule pairs will then be used to train the neural network with information from the orders as the inputs into the network, and the four rules as outputs from the network. Once the network has been trained, it will provide a set of rules to be used for a given order or set of inputs, that will cause the performance measure(s) to be optimized (or nearly optimized). A diagram of this procedure is shown in Figure 1.1.

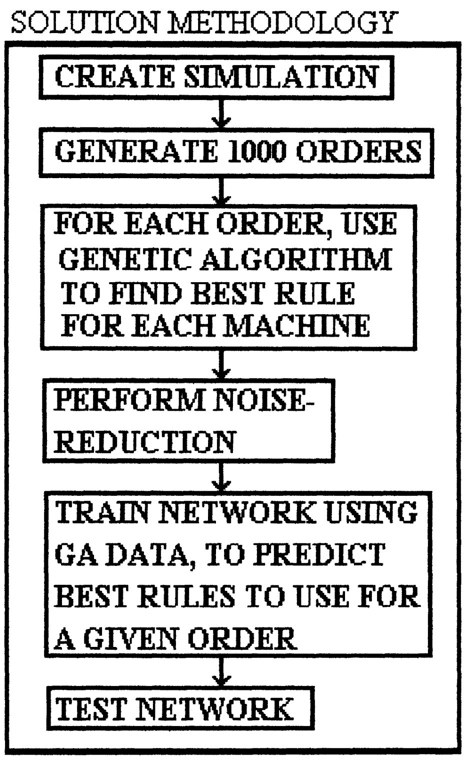

SOLUTION METHODOLOGY
CREATE SIMULATION
GENERATE 1000 ORDERS
FOR EACH ORDER, USE GENETIC ALGORITHM TO FIND BEST RULE FOR EACH MACHINE
PERFORM NOISE-REDUCTION
TRAIN NETWORK USING GA DATA, TO PREDICT BEST RULES TO USE FOR A GIVEN ORDER
TEST NETWORK

Figure 1.1

The problems encountered with traditional academic techniques are dealt with in this methodology. No mathematical models of a system are needed to utilize this method due to the fact that it uses only a simulation to obtain knowledge about the facility modeled. The problem will not need to be simplified beyond what is needed to create a simulation model of the system environment. The selection of the performance measure to be used is arbitrary, and while this project will be completed using make-span, this performance measure may be easily changed in the genetic algorithm's objective function, allowing the method to be executed using any performance measure desired.

The assumptions made in the development of this approach are that: the orders into the system are created randomly, there is no work-in-process in the system at the start of the simulation, and the performance can be measured using make-span. The assumption that orders are created randomly allows the research to avoid data bias to the results of the system. In an actual production facility the orders would not be created randomly, but the procedure for creating the orders can easily be changed, or a set of past orders could be used to train the network. In either case, creating the orders randomly represents a more difficult learning task than either of the other methods, so this assumption is considered valid. The assumption of no work-in-process was made to reduce the amount of programming difficulty; however, the methods used to train the neural network are immune to these problems. Thus, if the system loading were done as part of the order creation step, then the approach should still perform well; however, this assumption will require further testing before it can be fully accepted. As was previously stated, the use of make-span is arbitrary, and as long as the system to be solved is of the same type, this choice should not be important. Since most commercial scheduling/ simulation packages do not contain performance measures, the choice of which one to use is purely a programming issue. Make-span was used in this project due to the simplicity of programming this performance measure but due to the fact that only one module obtains the performance measure, this module could easily be modified to obtain any performance measure desired.

As was previously mentioned, the solution methodology used for this problem included the training of a neural network to produce a good set of dispatching rules to be used, one for each machine, based on information from the orders into the system. A genetic algorithm was developed to run a simulation in order to create the training data to be used to train the neural network.

The specific system analyzed contains four machines, eight process plans, and orders into the system of 20 parts. While this system is small, the assumption was made that the size of the system was irrelevant and that by increasing the number of orders into the system the approach used would still perform adequately.

## 1.5 Affected Fields: Case Studies

The types of facilities and problems which will be the most affected by this research are those in which there is a wide variation of orders coming into a system. One such facility is the AMP plant which is beginning production of SEC III and SEC 50 part lines in the same facility. These parts are connectors which contain different types and quantities of pins. This facility receives orders which vary significantly on a daily basis. Currently this plant uses AutoSched for its scheduling operations but they are having trouble determining which scheduling rules to use to make the facility perform at its highest efficiency. AutoSched allows the orders to be read into the order file as soon as they are received. After the orders have been entered, AutoSched uses the scheduling rules which were set when the simulation was built to create a schedule for producing the parts ordered for that day.

Since the Sec III and Sec 50 parts use different processes and machines for some of the processes, yet use the same machines for other processes, there is a significant problem with interactions among orders which contain large quantities of both parts. For this and other reasons, the same set of scheduling rules may not provide satisfactory results with two different sets of orders. By training a neural network with order-rule sets, the variations between orders could be recognized and correction made to accommodate the changes as soon as the order was received in the morning. This process could be automated so that before the simulation was run and a schedule produced, the rules in the station file would be changed by the neural network. This would allow the schedules to be produced on a daily basis without the need for extensive work on the part of the person in charge of creating the schedule.

This procedure will provide a method by which many different types of problems may be solved in academia. Many procedures utilize either genetic algorithms or neural networks, very little has been done in combining the two in order to solve a problem. This research will provide a step by step method by which problems which cannot otherwise be solved quickly or accurately, may be solved in real-time. While the specific target of this research is in scheduling rule selection, the wider task of schedule creation without scheduling rules may also utilize this procedure, along with problems outside of the area of scheduling.

## 1.6 Thesis Structure

This thesis is distributed as follows. The first chapter contains the introduction and problem statement. The second chapter contains an overview of the literature that relates to this work. Chapter three contains an explanation of Genetic Algorithms, Neural Networks, and the Rule-Set Replacement technique to be used in the project.

Chapter four describes the simulation package used, the programs developed, and the techniques used for the integration of these programs. Chapter five contains findings. Chapter six contains the conclusions and recommendations.

| | |
|---|---|
| CHAPTER 1 | INTRODUCTION |
| CHAPTER 2 | LITERATURE REVIEW |
| CHAPTER 3 | GENETIC ALGORITHMS, NEURAL NETWORKS, RULE SET REPLACEMENT |
| CHAPTER 4 | PROGRAMMING AND IMPLEMENTATION |
| CHAPTER 5 | RESULTS |
| CHAPTER 6 | CONCLUSIONS/RECOMMENDATIONS |

# 2. LITERATURE REVIEW

## 2.1 Dispatching Rules

Optimization in the production setting is often either too difficult and/or to time consuming to obtain on a daily basis. This is because the difference in cost between a "good" solution and the "optimal" solution may be so small that the cost of attaining the optimal solution is greater than the savings in time or money achieved when the optimal solution is implemented. Thus in many situations what is needed is some rule of thumb or *heuristic* to provide a "good" workable solution with very low cost in time or money. Some of the most common heuristics are FIFO (First In First Out), LIFO (Last In First Out) and SPT (Shortest Processing Time). FIFO is often called the waiting line example because when standing in line, at a supermarket for example, the first person to stand in line is the first person to check out. LIFO is more like parts in a bin where the last part put into the bin is on top and so is the first one to be used. SPT looks at the parts in the queue and takes the one with the shortest processing time and does it first. These heuristic rules are very easy to conceptualize and so are often used by managers and others in charge of scheduling in production facilities. Unfortunately these rules, and others with conceptual appeal are often not the best ones to choose. In fact there is not any way to determine which scheduling rules are the best for a given situation, without extensive testing. The following papers focus on some of the most common heuristics and classify them on the basis of their effectiveness.

Conway (1965) was one of the first to analyze a large number of scheduling heuristics and evaluate their performance. He tested the heuristics on a system containing nine machine groups, each with one machine. He measured the performance of the different rules with various measures of WIP inventory and job lateness. For WIP Conway analyzed 16 rules and determined that the shortest imminent operation time (SIO) rule outperformed the other rules tested. And while SIO also performed well with average job lateness, the smallest ratio of slack time to the number of remaining operations (SLACK/RO) was the best due date based rule for due date related performance measures.

Hershauer and Ebert (1975) also tested multiple heuristic rules. They tested three due date rules against three processing time-based rules and seven combined rules. They determined that the SIO rule minimized mean flow time, but that all of the due date based rules performed better than SIO with respect to cost-per-order, and of the due date based rules SLACK/RO performed the best.

McCartney and Hinds (1981) examined the priority rules FIFO, SIO, and SLACK/RO. They tested these rules in an FMS system in which the rules were used to assign pieces to seven machines through a transportation system. For average tardiness, the study demonstrated that SLACK/RO performed the best when the due-dates for the pieces were loose, but when the due-dates were tight the testing concluded that SIO performed the best.

Stecke and Solberg (1981) reported on an actual FMS system in which an experiment was performed. The system examined consisted of a transportation system connecting nine machines, an inspection station, and a control queuing area. The performance measure utilized was the number of completed parts. The experiment tested sixteen scheduling rules on this system and it was determined that the best rule for the system was the SDT (smallest ratio of imminent operation time to the total processing time) priority rule.

Blackstone *et al.* (1982) performed a survey of heuristic rules, and reported that SIO was the best priority rule when one of the following conditions was met. The shop had no control over due-dates. The shop had control over due dates and the due-dates were tight. The shop could control the due-dates, and the due-dates were loose, but there was a high degree of congestion in the shop.

Dar-El and Wysk (1982)used a job shop and two performance measures to test six heuristic rules. The performance measures used were mean tardiness, and root mean square (RMS) tardiness. When overall ranking was considered, SIO and WINQ (work in next queue) were superior to the other four rules.

Elmaraghy (1982) tested four heuristic rules (SIO, FIFO, RANDOM, FRO) using a system consisting of five machines, one load/unload station, and two material handling devices. SIO yielded the highest production rate in terms of the total number of parts produced, and also in terms of the total processing time at the stations. SIO was also better at reducing the average flow time than the other rules tested.

Ballakur and Steudel (1984) performed a review of job shop control systems. When they compared several heuristic rules they found the following. The SIO rule produced the best results when the due-dates were loose and machine utilization was moderate. Most researchers found SLACK/RO to be consistently better than other due date based rules. Combined rules such as SLACK and SIO seem to have the most potential for further research.

Kim (1990) examined eight heuristic rules in a job shop environment with multiple identical jobs and alternative routings. These specific circumstances caused the problem to impose precedence relationships between the operations, and caused MDD (modified due date) to be the most effective of the rules studied.

Montazeri and Wassenhove (1990) tested fourteen heuristic rules in an FMS environment and found that the rule selected as the best greatly depended on the performance criteria utilized in the examination. When the criteria used was the average part waiting time, the SMT (smallest value obtained by the product of the imminent operation time and the total processing time), a rule not often tested, produced the minimum value of the rules tested, but it also produced a high value for the variance of part waiting. SPT had the overall lowest average buffer and shuttle utilization times. In general they found that the SPT based rules minimized average waiting times and the LPT (longest total processing time) based rule maximized machine utilization.

## 2.2 Learning and Scheduling

Many approaches to scheduling have been tried. Some involve performing a search of possible solutions using genetic algorithms, others use neural networks, and some use combinations of these and other techniques. The following literature contains the work which is the closest to the research done for this paper.

Minagawa and Yukinori (1992) used genetic algorithms combined with other techniques, to solve a flow-shop scheduling problem. In the experiment they modeled a system which had multiple resources at each stage of production and the purpose of the genetic algorithm was to determine the best choice of resources at each point along production.

Caskey (1993) attempted to apply genetic algorithms and neural networks to manufacturing scheduling. He used a genetic algorithm to search for the best scheduling rules to use based on three factors: factory flow type, due date tightness, and machine utilization levels. The results from the genetic algorithm were then used to train the neural network. Due to the fact that the simulator used required approximately five minutes per simulation, only 150 runs were performed with the genetic algorithm. This small number of runs, along with the fact that binary strings were used in the GA, created several significant problems. First, the speed of the simulator prohibited him from performing any design of experiments of the genetic algorithm. Second, the genetic algorithm used a binary string for the chromosome which not only complicated the programming of the GA, but also significantly inhibited the ability of his neural network to learn from the resulting data. Third, the 150 runs did not provide a large enough data sample to properly train the neural network. And fourth, different types of inputs were not tried when training the neural network, so that the importance of different types of inputs was not observed. In his conclusion he suggested that as many as 500 to 1000 training samples may be required to sufficiently train the neural network.

Rabelo, Yih, Jones, and Tsai (1993) used a system which integrated neural networks, parallel monte-carlo simulation, genetic algorithms, and machine learning to solve a flexible manufacturing system scheduling problem. In this case the output from the Monte Carlo simulation was used to determine the best results, and then the genetic algorithm was used to combine the two best solutions into a single best choice, at which time another iteration was performed by the system.

Chiu (1994) proposed a learning based methodology that was successfully used for dynamic scheduling in a distributed manufacturing system. He trained neural networks (in a decision tree format) to schedule jobs through machines on a one-at-a-time basis. Genetic algorithms were used in this technique to generate samples which were used to train the neural networks. Significant improvements over static schedules were achieved by implementing this technique at the machine level, allowing the entire process to proceed without a "master schedule" to follow.

Sittisathanchai (1994) used GA's to optimize a job shop scheduling problem. When implementing the problem he had to come up with a technique that would allow the genetic algorithm to be unconstrained because a genetic algorithm works based on "probabilistic transition rules, not deterministic." (Goldberg, 1989). The techniques used to overcome this limitation were order-based crossover and order-based mutation; this allowed the functions of the genetic algorithm to continue while not limiting its ability to reproduce. In his research, Sittisathanchai used a chromosome in which the genes represented jobs to be done, so that the entire chromosome represented a single schedule. He tested the genetic algorithm scheme against several heuristic algorithms and found the genetic algorithms to perform better on small, medium, and large problems, and he even used a problem submitted from an actual factory in order to verify his results.

## 2.3 Summary and Justification

While work has been done in the areas of schedule creation using GA's and Neural Networks, simulation has not been used to the extent possible, and nobody has attempted to model experimentation in their procedures. One reason that simulation has not been used to the extent it will be in this project is that the time to simulate has previously been far too long. The work which most closely resembles this was done by Caskey in 1993. The simulation used by Caskey required approximately five minutes to run each time.

The simulation time for this project is only 0.9 seconds, allowing the large number of data-sets needed to train a neural network to be created in a relatively short period of time. This work will provide a method for experimentation which will not only use GA's and Neural Networks, but will provide a model through which the two may be fully integrated, varied, and tested to provide successful results.

# 3. GENETIC ALGORITHMS, NEURAL NETWORKS, RULE SET REPLACEMENT

## 3.1 Genetic Algorithms

Genetic algorithms (GA's) provide a method for quickly finding near-optimal solutions to problems in a small amount of time. Due to the way GA's work, they are less likely than other methods to find only local optima, which makes them extremely useful in solving many types of problems.

Genetic Algorithms were developed by John Holland at the University of Michigan (Goldberg, 1989). Genetic algorithms are based on the theory of evolution. This theory is that if a particular individual is more fit for its environment than its counterparts, it will be more successful and reproduce more often than its less fit siblings. As time progresses mutations will occur which may or may not create more fit individuals than those that the mutated ones came from. If this mutation does create a more fit individual then its genes will slowly take over. This mutation operation keeps the GA from getting "stuck" in a local maximum or minimum by randomly changing one of the genes so that a different point in space is tried. This allows the entire realm of possibilities to be examined for possible solutions in the search space, making GA's a very powerful tool.

Davis (1991) explains the basic principles of genetic algorithms. He explains that the basic unit of a genetic algorithm is the gene and that a string of genes linked together form a chromosome which represents a single solution to the problem being modeled. The ability to design problem-specific genes and chromosomes allows genetic algorithms to be applied to almost any problem, as long as the chromosome contains information which represents a distinct solution to the problem being analyzed. He also discusses several methods for combining the chromosomes through crossover to produce offspring, much as the genes of two parents combine to produce a child. This may be achieved through several different methods depending on what aspects of the chromosome are important. A crossover method may be based on position in the string, relative order in the string, absolute order in the string, or some other criteria.

Yamamura et al. (1994) states that crossover is the most characteristic of all the techniques utilized in genetic algorithms. In their paper they start with a two bit chromosome (the minimum chromosome length possible in a genetic algorithm), and utilizes the genetic algorithm to converge on the optimal point in a theoretical square, where each corner of the square has a different fitness value. Next they chose a large chromosome to analyze deference equations and again demonstrated that crossover allowed the genetic algorithm to converge optimally on the solution.

The number of chromosomes, or sets of genes, in a particular GA is called the population of that GA. Each chromosome represents a unique solution to the problem being solved, thus the larger the population size used in the GA, the more solutions are tried in a single generation. By having a large population the variation among the chromosomes will also be higher and the GA will not tend to converge as quickly. In problems where there are a significant number of local minima the population size must be higher in order to keep the GA from converging until the global minimum is found.

The number of generations for the GA is the number of times that the crossover, mutation, and ranking operators are performed. These correspond roughly with the number of generations in a biological system, where the children of one generation are considered the next generation. The number of actual solutions searched depends on both the population and the number of generations, since the larger the population size, the more solutions are tried in one generation, and the more generations are run the more total solutions are tried.

In order for any genetic algorithm to work, there must be some sort of feedback function which, when given a specific chromosome, returns some value corresponding to the fitness of that chromosome. This fitness function may be anything which can yield a result; this may be a person expressing personal bias concerning the solution proposed by each chromosome, a mathematical function performing an operation on the chromosome, or a simulation which uses the chromosome to set some system parameters needed for the simulation to run. If possible, the feedback function needs to run quickly, since GA's by definition need to be able to process many possible solutions quickly.

As an illustration of a genetic algorithm, suppose a GA was to be used to find the values which optimized some unknown mathematical function which had four variables. First a chromosome containing four genes would be created, one for each variable. Each gene would then be initialized at a level within the boundaries for its respective variable, say [0,1,2,3,4,5] for each variable. Once all the chromosomes in the population were initialized, the chromosomes would be entered one at a time into the fitness function. Assuming the fitness function was $X_1 + X_2 + X_3 + X_{4,}$ (this function would actually be hidden from the GA, only the results of a chromosome would be seen), the best chromosomes would be those which had the highest numbers in each of their genes. A chromosome of [1,0,2,3] would receive a fitness value of six, while a chromosome of

[3,5,4,2] would receive a value of 14.

Once all of the chromosomes were tested through the function, they would then be ranked according to their respective fitness values, and then the crossover function would be performed. Assuming that 1-point crossover was specified, and that the population size was four, the top two solutions would be mated.

| | | | | |
|---|---|---|---|---|
| If the total population was: | [1,0,2,3] | [3,5,4,4] | [4,3,5,1] | [0,5,3,1] |
| The resulting fitnesses would be: | 6 | 16 | 13 | 9 |

| | | | | |
|---|---|---|---|---|
| After Ranking: | [3,5,4,4] | [4,3,5,1] | [0,5,3,1] | [1,0,2,3] |

If the crossover point selected was two, the top two chromosomes would be split between the second and third genes and the genes to the right of the split would be exchanged.

| | | |
|---|---|---|
| Parents: | [3,5,\| 4,4] | [4,3,\| 5,1] |
| Children: | [3,5,\| 5,1] | [4,3,\| 4,4] |

After the crossover function was completed, the mutation operator would be executed. For each gene of the children chromosomes, there would be a probability of mutation, such as .01, or 1%. If a gene was selected for mutation, it would be replaced by a random value from within its range. If for instance the third gene of the first child was selected, the five would be changed to another value, such as two.

| | | | | |
|---|---|---|---|---|
| The Population would now be: | [3,5,4,4] | [4,3,5,1] | [3,5,2,1] | [4,3,4,4] |
| The resulting fitnesses would be: | 16 | 13 | 11 | 15 |
| After Ranking | [3,5,4,4] | [4,3,4,4] | [4,3,5,1] | [3,5,2,1] |

This process would be repeated until the specified number of generations was completed, or until some specified amount of time has passed. The genes used here were the actual numbers used in the function, but this is not necessary. In some cases a gene may be a letter, number, or even a description. The only requirement is that no matter what the gene structure is, the chromosome represents a unique solution to the problem being solved, and the method used to give a fitness value must be able to interpret the solution that the chromosome represents.

The procedure used by genetic algorithms is shown in Figure 3.1.

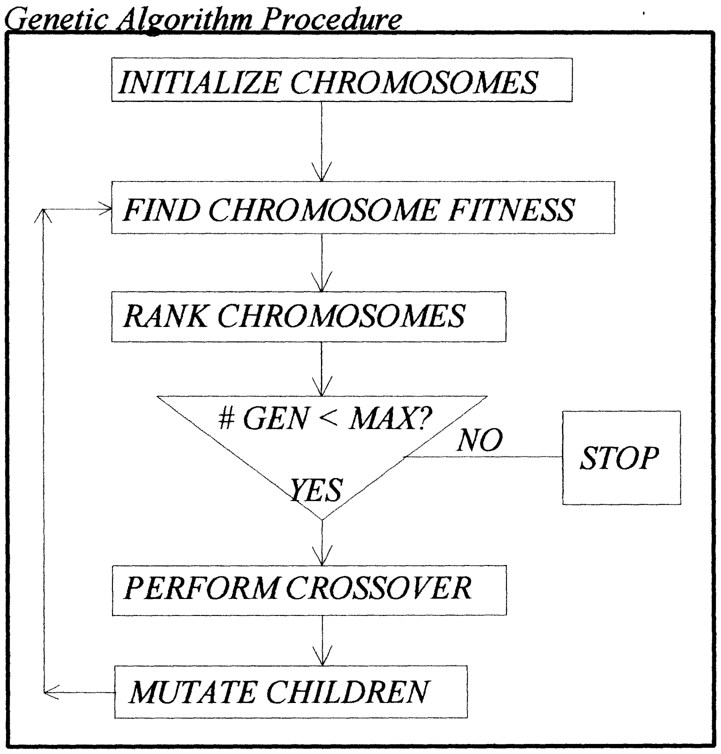


*Figure 3.1*

## 3.2 Neural Networks

For the past thirty-five years neural networks have been used to find relationships among data. The key to their success is that neural networks learn directly from data. Thus, no previous knowledge on the part of the user of the network is needed to make the network perform well. This vastly increases the power of neural networks because in many instances the relationships between the inputs and outputs are either totally or partially unknown prior to the implementation of the neural networks.

An artificial neural network is a set of interconnected computational elements. Each node is a computational element which receives data via input links, modifies the data according to some function, and sends the data out through an output link(s). [illegible] then modified in the link itself when the data is multiplied by a weight assigned to that specific link. This modified data is then fed into another node where it is further modified along with data from several other links. Through the interconnection of nodes and links, the network is able to process massive amounts of data at once. By modifying the weights for each link during the process of training, the network is able to find relationships among data and thus it "learns" from the data.

A typical neural network consists of three types of layers, the input layer, the hidden layer(s), and the output layer. Each of these layers is made up of nodes which perform different functions and purposes. The nodes in the "input layer" take the inputs to the system and send these inputs along several links to nodes in the hidden layer. These nodes usually have a simple transfer function so that the outputs equal the inputs into the node. The nodes in the hidden layer perform most of the operations on the data. There may be from zero to many hidden layers in a network. This layer is called the "hidden" layer because these nodes cannot be accessed directly from outside of the system, and thus the data they process is not seen as input or output from the system. The final layer is the output layer. This layer contains classification nodes which classify the data and present it in an understandable manner as output from the system. See Figure 3.2.

*Simple Neural Network*

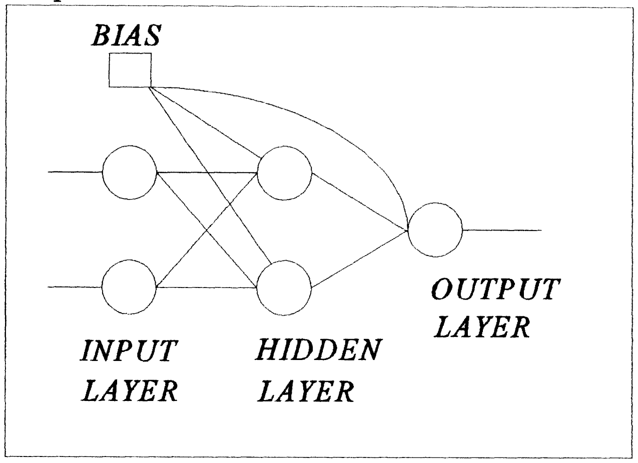


*FIGURE 3.2*

Just as genetic algorithms are based on a biological process, so are neural networks. Artificial neural networks are based on the neurons in the brain. In the brain information is fed into a node along "synapsis" or connections between nodes. The connections which are fired more often in the brain change to allow these firings to occur more easily. These processes correspond to the processes of a neural network. The nodes in a neural network have functions which allow them to process data using simple mathematical formulas. The nodes of the neural network are connected by weighted connections which represent the synapsis of the brain. In the brain, a synapse fires easier the more often it is fired, while the neural network changes the weights in the connections to provide an increase or decrease in the magnitude of the messages sent along the connections. In the brain the signals coming into a node are processed according to a function which closely resembles the sigmoid function. Because of this, many neural networks also make use of the sigmoid function for the processing elements (nodes) of the network.

There are several methods for training neural networks. Some of these methods, such as backpropogation, change the weights of the interconnections between the nodes, while other methods change both the weights and the architecture of the network itself. Each method has its own benefits and drawbacks, and different methods are obviously better suited to different types of tasks.

The main strength of neural networks is also the cause of their main weakness. Neural networks can perform pattern recognition on incomplete or imperfect data, unlike other techniques which require full and accurate definitions of the data. The way neural networks do this is not totally understood, making a network seem like a black-box which magically produces results. This allows neural networks to be used in situations far beyond the capabilities of other methods, but it also creates a major weakness. Since the reasoning behind why a neural network chooses a particular solution is unknown, the result may be totally incorrect, but there is no way of knowing unless either the solution is already known, or the solution produced by the network is implemented and the actual results are checked. One way around this difficulty is to test the network with a significant number of cases where the solution is already known in order to see how reliable the network is at predicting results. This "known" data is referred to as a test set, and by running a network on a test set, the ability of the network to accurately predict results can be tested. Obviously the size of the test set is determined by the amount of time needed to create the set, and the importance and/or consequences that will result from an inaccurate prediction on the part of the network.

Due to the fact that a network learns from the data used as inputs into the system, it is very important to choose inputs which contain data critical to the decision process. If a network were to be trained to decide if a person was male or female, the inputs should not be the color of the person's eyes or the number of fingers he or she has, since this data would contain no information concerning the person's gender. In many cases it is not easy to determine which inputs will be the most effective without trying different combinations and observing the results of these selections.

## 3.3 Rule Set Replacement Technique

When the data collected from the enumerations of the 10 orders was analyzed, it became obvious that there were several sets of rules which would yield the same results for a given order. This meant that along with learnable variations of rule-sets due to different orders, there was also a significant amount of noise in the data. This noise is present in the data due to the fact that the genetic algorithm will return a rule-set that yields the best result, and if there are several rule-sets which produce the best result the genetic algorithm will choose one at random. This randomness in the data creates noise which makes the task of the neural network much more difficult.

In order to reduce the amount of noise present in the data, it was decided that a procedure must be developed to at least reduce this noise, if not totally eliminate it. The technique decided upon is one in which each of the first 100 order-rule set choices will be tested with every rule-set chosen by the GA for these 100 orders. This means that the rule-set chosen by the GA for the first order, will be tried on each of the first 100 orders, and the number of times it produces as good as or better results than the rule-set chosen for each order by the GA, will be tallied.

Once the rule-sets for each order in the first 100 have been tried on all of the other 99 orders, the rule-set with the most successes will be used to replace the rule-sets in each of the 1000 orders where the new rule-set performs as good as or better than the rule-set chosen for the order by the GA. In this manner we hope to reduce the amount of noise in the data and allow the neural network to learn from the data and not from the noise in the data.

# 4. RESEARCH AND SOFTWARE ENVIRONMENT

Due to the varied nature of the types of programs needed to complete this project, different methods were used for these different parts. All of the software used was run on a Pentium-75 MHz IBM-based personal computer. For the enumerations, Genetic Algorithms, information extraction, and the Rule-Set Replacement tasks, C++ programs were written so that these tasks could be executed quickly and efficiently. For the simulation itself, AutoSched was used, running in ASAP mode. This section describes the creation and integration of these different programs.

## 4.1 AutoSched

As previously stated, the simulation of the job-shop was created using AutoSched. This package uses data from different text files in order to run the simulation. Much of the information in these text files can be modified without entering the program itself, and the simulation can be re-executed without recompilation the simulation. Once the simulation executes, AutoSched creates an output file which may be read from outside the simulator.

The files used by the simulator to execute the simulation include: an order file, station file, product file, routing file, and others. The order file contains the data regarding the parts ordered, the arrival and due dates of the parts ordered, and the quantity of each part ordered. The station file contains data concerning each of the machines including the machine type and the rules used by each machine. The product file includes information for each part type including the part name and the name of the routing for each part. The routing file contains a routing for each part specified. This routing includes the order of the processes for the parts and the amount of time spent at each machine for processing.

Many output files may be specified when using AutoSched, and many different measurements may be taken. For this project, the only output file utilized was the file lotrep, or lot report file. This file contained the completion dates/times for each of the lots in the order file, in the order in which the lots were completed.

## 4.2 Module Creation

Once the simulation itself was completed, modules were created to control each of the functions required to run the simulation from within another program. These procedures were done in modules so that they could be used over-and-over in the different programs without the need to rework the process each time.

The first module created was one to automatically create a new order file randomly. Each order created contains 20 lots, and each lot is an order for one of the 8 parts at a quantity of 10 parts. This process was done by accessing the 'order.txt' file and modifying the data within the file using C++.

The second module created accesses the station file and changes the scheduling rules for each machine. AutoSched contains 16 different scheduling rules, and each of these rules was input into an array. When the module is called, it receives an array of four integers which represent the rules required for the next run. The corresponding rules from the rule-array are then used to replace the previous rules in the station file.

The last two modules created run the simulation and get the results. The module to run the simulation automatically moves to the correct directory and runs the simulation through a system command. The final module accesses the output file lotrep, reads the data in it, converts the data into minutes for the make-span, and returns this time as a float to the referencing function.

## 4.3 Genetic Algorithm Development

In order to find optimal, or near optimal solutions to the system quickly, a genetic algorithm was used to perform a search of the space of all possible rule selections given a set of inputs. Since there were sixteen rules in the simulation package used (there were actually many more but the rest were just special case variations on the 16 standard rules), and four machines, a chromosome containing four genes was chosen. This allowed each gene to represent the scheduling rule for its respective machine. The genes could each contain 1 number from 0 to 15 which represented the actual rule used. The rules that were available for testing were:

- CR (Critical Ratio)
- EDD (Earliest Due Date)
- FIFO (First In First Out)
- HP (Highest Priority)
- HV (Highest Value)
- HXT (Highest XTheoretical)
- LBA (Least Balance Ahead)

- LLA (Least Lots Ahead)
- LP (Lowest Priority)
- LPA (Least Pieces Ahead)
- LPR (Least Percentage of processing time Remaining)
- LTR (Least processing Time Remaining)
- LV (Lowest Value)
- LXT (Lowest XTheoretical)
- SPT (Shortest Processing Time)
- SSU (Same SetUp)

The genetic algorithm first creates a population of chromosomes by assigning a random integer from 0 to 15 to each gene of each chromosome in the population. The number in a specific gene represents which scheduling rule of the 16 is to be used by the respective machine in the simulation. Each chromosome is then used to set the rules in the file which stores the simulation data on which rules are used by which machines, and the simulation is run. Once the run is complete, the performance criteria of the simulation is retrieved by the program and assigned to the chromosome containing that specific set of rules. The process is then repeated for every chromosome in the set. Once all of the chromosomes have been used to run the simulation, they are ranked according to their "fitness value" or performance criteria measure.

Once the chromosomes are ranked according to how well they performed in the simulation, the top half of the chromosomes are mated among each other and the offspring replace the lower half of the chromosomes. The crossover operators tested were 1 and 2-point crossover. In this type of crossover, the mating genes are split in identical locations along the chromosomes of the two genes, and the gene sequences to the right of (or between) the split(s) are traded. This process yields two offspring per crossover, and thus when the first half of the chromosomes are mated (after they have been ordered according to rank), the top half of the chromosomes are mated. This means that the crossover function occurs between the 1->2, 3->4, ... n/2-1->n/2 chromosomes. The choice of whether to use 1 or 2-point crossover will be decided through the design of experiments for the GA.

The mutation operators used in this genetic algorithm will be tested originally at rates of .01, .05, .1, and .5. This operation is only performed on the second half of the population since it is desirable to retain the top half of the chromosomes and not risk "throwing away" a good set of genes on mutation. In the second half of the population however, if a gene is selected for mutation the number in that gene is replaced with a random number from 0 to 15. In this manner a mutation will randomly change the scheduling rule used in that specific gene to some other rule. Once mutation is complete, the new set of chromosomes continues to cycle through testing, ranking, crossover, and mutation until either the maximum number of generations (iterations) is met or some predefined performance level is achieved. The choice of the final mutation rate will depend on the results of the design of experiments for the GA.

The population used in the GA will be tested at levels of 12, 25, and 50. This population is the total number of chromosomes kept for testing. Once the set of chromosomes have been ranked according to their fitness, the top half of the chromosomes will be mated with each other via the crossover operator discussed above, and the resulting children (the lower half of the population), will then be tested and the entire population will again be ranked according to the fitness. The population size will be finalized based on the results of the design of experiments for the GA.

The number of generations for the GA will be tested at levels of 50, 100, and 150. The number of generations is the number of times the crossover, mutation, and ranking operators will be performed in the GA. The final population size will also be determined from the results of the design of experiments for the genetic algorithm.

GA Design of Experiments

| *Factors* | *Levels* |
|---|---|
| *Mutation Rate* | *.01, .05, .1, .5* |
| *Population* | *12, 25, 50* |
| *Generations* | *50, 100, 150* |
| *Crossover Points* | *1, 2* |

*Total Number of Runs = 72*

*Figure 4.1*

## 4.4 Rule Set Replacement Development

After the GA has found solutions for 1000 orders, the rule-set replacement technique described in chapter 3 will be performed on the data to determine which rules may be replaced by which other rules to help reduce the noise present in the data. This will be done by utilizing the rule-change and run-simulation modules from within a C++ program.

## 4.5 Neural Network Development

Once the data has been obtained and modified via the previous methods, data will be extracted from the order files which will be used to train the networks. The data that will be used to train the network will be created by integrating the data in the order files with the data from the process steps used to create the simulation. The different sets of data which will be used to train the networks will include:

$$INPUT1 = \frac{\frac{N*P}{M}}{\sum_{i=0}^{N}\sum_{j=0}^{M} T_{ij}}$$

Where:$T_{im}$ = process time of order I on machine m

L = number of orders

M = total number of machines

N = total number of orders

P = number of tasks to be performed

$$INPUT2 = \frac{\sum_{i=0}^{N} \frac{(DueDate[i] - \sum_{j=0}^{M} T_{ij})}{\sum_{j=0}^{M} T_{ij}}}{P}$$

$$INPUT3 = STDDEV\left(\frac{(DueDate[i] - \sum_{j=0}^{M} T_{ij})}{\sum_{j=0}^{M} T_{ij}}\right)$$

$$INPUT4 = \frac{\sum_{j=0}^{M} \frac{\sum_{i=1}^{N} T_{ij}}{MAX \sum_{i=1}^{N} T_{ij} | j=1..M}}{M}$$

$$INPUT5 = STDDEV\left(\frac{\sum_{i=1}^{N} T_{ij}}{MAX \sum_{i=1}^{N} T_{ij} | j=1..M}\right)$$

$$INPUT6_j = \frac{\sum_{i=0}^{N} T_{ij}}{\sum_{i=0}^{N} \sum_{j=0}^{M} T_{ij}}$$

NOTE: Input 6 is machine specific (thus the subscript).

## 4.6 Testing Procedure

Once each of the different inputs has been used to train their respective networks, the networks will be tested to see which input configuration is the most successful at predicting good rule choices. The method for testing the network will not be the traditional technique, where the network is run on an order file for which the solution is known. While this would work for most situations, it will not work in this one for the same reason that the rule-set replacement technique was used on the data before training. Since there is no unique solution which produces the best result in the data, the result of the network may work just as well as the solution found by the genetic algorithm.

Because of this the neural network will not be tested on its ability to regurgitate rules, it will be tested on its ability to present rule choices which will perform well in the simulation. When the GA is used to find good rule choices for each order, it will record the solution found and the value of the performance measure tested (in this case the Make-Span for the rule-set chosen). In the testing phase of the project the neural network will be given inputs from an order file for which a good value of the performance measure is known. The rule-set which the neural network produces will then be used to run the simulation and the resulting value of the performance measure will be checked against the value found to be "good" by the GA. For each instance in the test set when the rule-set produced by the neural network results in a value as good as or better than the value found by the GA the ranking for that particular network will be incremented. Once each of the networks has been tested on the test sets in this manner, the network which has the highest number of successes will be selected as the best network and the inputs used to train it will be determined to be the best.

# 5. EXPERIMENTATION AND RESULTS

## 5.1 Order Generation

The first step in the experiment was to create the simulation of the system to be used for experimentation. This system consisted of the four machines previously defined with the eight process plans representing eight different parts to be produced. The order file used by the simulation was created using a C++ program which randomly created 20 lots in the order file. Each lot consisted of an order of 10 pieces of a specified part. Thus, an entire order contained 20X10 or 200 parts. Once this step was completed, the output of the simulation was set to the finish time of each of the lots in the order file. In this way the response function module could enter the output file and retrieve the largest time, which represented the makespan of the lots produced. A sample order is shown in Figure 5.1. A sample output file is shown in Figure 5.2.

Sample Order File

```
ORDER       LOT   PART  PIECES START        DUE            TRACE
1     1     P2    10    1/1/96  8:0:0  1/1/96 11:38:0      3
2     2     P1    10    1/1/96  8:0:0  1/1/96 9:57:0       3
3     3     P2    10    1/1/96  8:0:0  1/1/96 12:6:0       3
4     4     P5    10    1/1/96  8:0:0  1/1/96 10:25:0      3
5     5     P5    10    1/1/96  8:0:0  1/1/96 10:33:0      3
6     6     P4    10    1/1/96  8:0:0  1/1/96 11:19:0      3
7     7     P7    10    1/1/96  8:0:0  1/1/96 10:40:0      3
8     8     P4    10    1/1/96  8:0:0  1/1/96 11:8:0       3
9     9     P5    10    1/1/96  8:0:0  1/1/96 10:54:0      3
10    10    P4    10    1/1/96  8:0:0  1/1/96 11:16:0      3
11    11    P4    10    1/1/96  8:0:0  1/1/96 11:20:0      3
12    12    P3    10    1/1/96  8:0:0  1/1/96 12:24:0      3
13    13    P6    10    1/1/96  8:0:0  1/1/96 12:35:0      3
14    14    P6    10    1/1/96  8:0:0  1/1/96 11:57:0      3
15    15    P8    10    1/1/96  8:0:0  1/1/96 13:15:0      3
16    16    P8    10    1/1/96  8:0:0  1/1/96 13:8:0       3
17    17    P2    10    1/1/96  8:0:0  1/1/96 11:42:0      3
18    18    P2    10    1/1/96  8:0:0  1/1/96 11:40:0      3
19    19    P3    10    1/1/96  8:0:0  1/1/96 12:55:0      3
20    20    P5    10    1/1/96  8:0:0  1/1/96 10:42:0      3
```

Figure 5.1

Sample Output File

```
CYCLEMAX

~Report time: 01/01/96 23:20:00.
 00:15:20:00
 00:21:20:00
 01:00:40:00
 01:03:50:00
 01:05:10:00
 01:06:40:00
 01:07:10:00
 01:08:00:00
 01:08:30:00
 01:09:20:00
 01:10:30:00
 01:12:00:00
 01:14:40:00
 01:15:10:00
 01:18:30:00
 01:19:50:00
 01:21:50:00
 01:23:10:00
 02:05:10:00
 02:05:10:00
```

Figure 5.2

## 5.2 Enumeration Results

Once the simulation was completed, enumerations of ten order files were performed. The enumerations consisted of first setting the orders in the order file, and then running every combination of rules on the four machines. Since each simulation call required approximately 0.9 seconds, this procedure took about 18 hours per enumeration to complete. These enumerations were performed so that the GA and the neural networks could be tested to see whether the solutions obtained were actually the best possible or not. Each enumeration consisted of 65536 rule-sets, where each rule-set is a unique set of four dispatching rules, one for each machine. Each of the enumeration files was then analyzed in order to determine which of the enumerations would be the hardest for a GA to solve.

After each of the ten enumerations were completed, the data was sorted into bins of 100 (ie 2000-2100 min., 2100-2200 min., etc.) for each enumeration and this data was graphed. The extra sorting step was required due to the fact that the file was otherwise too large to be read by any of the commercial graphing packages available. The resulting graphs are in the appendix.

With the graphs of the ten enumerations completed, the sixth order file was determined, by visual inspection, to be the best enumeration to use to test the genetic algorithm. The sixth enumeration was chosen due to the fact that it contained the smallest set of solutions which yielded the optimum result of the shortest make-span. The graph of that enumeration is shown in Figure 5.3. All 10 enumeration graphs are shown in the Appendix.

Enumeration of Order #6

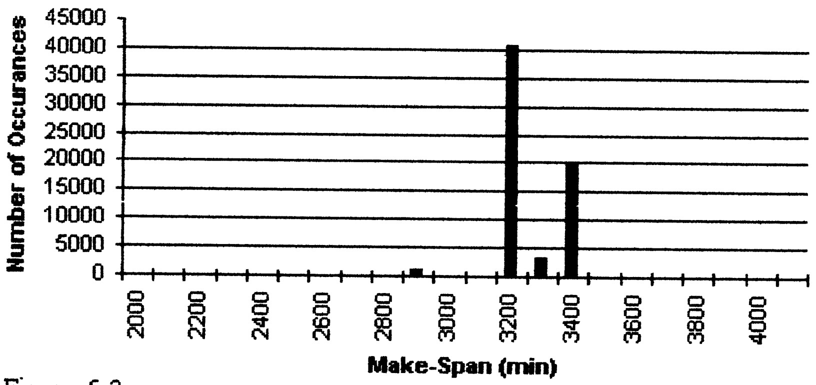


Figure 5.3

## 5.3 Genetic Algorithms Results

The genetic algorithm was tested by running a design of experiments (DOE) with the GA, using enumeration number six as the testing problem. This enumeration was chosen because it had the smallest number of "optimal" solutions. The independent variables of the GA (population size, number of generations, number of crossover points, and mutation rate) were set to the different levels defined in the DOE, and the dependant variable (makespan) was used to determine which levels of the independent variables produced the best results. When the design of experiments was performed on the Genetic Algorithm using enumeration six, the only significant source of variation found was between a mutation rate of .01 and .1, with .01 showing the best results. This led to further experimentation in which the mutation rate was changed in .01 increments from .01 to .15 with all other variable held at the lowest level. This process was repeated four times to reduce the random noise due to the genetic algorithm, and the results showed .01

and .12 to be the best choices. Of these two .12 was selected because it reflected a better rate for mutation to actually be performed, a factor which has repeatedly been stressed as necessary for genetic algorithms to work effectively in varied environments.

With this step completed the GA was then used to find solutions for 1000 orders. This process involved randomly creating the lots in the order file used by the simulation, and then running the GA on the simulation. The GA program stored the order file along with the best result obtained and the set of four rules used to create the best result. This process required approximately 4 days of computing time to complete.

## 5.4 Rule Set Replacement Results

Once the GA step was complete, the resulting rule choices were tested and modified using the Rule-Set Replacement technique described in Chapter 4. This briefly consists of trying each set of rules selected for the first 100 orders, on all of the other first 100 orders. The set of four rules which produces a result which is "just as good" as the result obtained using the rules selected by the GA in the most cases, was used to replace the rules for every order in which these "just as good" results are achieved. In this manner the orders for which the choice of rules is not very critical will all use the same set of rules, while the orders for which the rule selection is more critical will keep their original rules. This allowed the neural network to learn variations among the more critical orders, while not being hindered by as much noise from the non-critical differences.

## 5.5 Neural Networks Results

Two approaches were tried for the neural network. The first approach was to use one very large network which would predict the rules for all the machines at the same time. The reason this large network was used was in case there was information on the interactions among the four machines that was important in determining which dispatching rules should be used for a given order. The second approach involved using four separate networks for the prediction. In this approach each network was assigned to a specific machine and was used only to predict the rule for that machine. The inputs for this network consist of each of the inputs described in chapter 4 including all four of the machine specific inputs described.

The outputs from both networks were in a decision tree format. This means that rather than training the network to give an "average" rule as output, each rule receives a certain amount of value. This allows giving a fitness of each rule to be used on the problem rather than try to select an "average" rule (The "averaging" phenomenon occurs when only one output node is present, in which case if certain factors indicated using rule 1, and other factors indicated rule 15, then rule 8 would be selected rather than either of the rules desired.). For each machine modeled, this type of architecture leads to sixteen output nodes.

The first approach utilized a relatively large network. This network contained 9 input nodes, each of which corresponded to order-specific data, and 4 of which corresponded to data on the four machines separately. The number of hidden layers selected was 2, with 16 nodes in each layer. The number of layers and the number of nodes used was determined by searching across the possibilities until a suitable solution was found. Since the outputs were in a tree format, and all four machine rules were represented in the output of this network, there were 64 output nodes in this network.

Each of the networks used in the second approach contained 6 input nodes and 16 output nodes. The number and size of the hidden layers were varied until a network containing 2 hidden layers, with 8 nodes in each layer, were selected as providing the best results. This size allowed for several degrees of freedom while not producing a network which was too large to be able to learn well.

## 5.6 Neural Networks Testing

The data obtained from the 1000 GA runs was separated as follows. The first 500 were used to train/test the neural network. Of this first 500, 375 were used to train the network, and the other 125 were used to test the neural network. The other 500 order/rule pairs were broken into sets of 100. Each set of 100 was used to further test the generalization capabilities of the networks.

The actual procedure for testing the neural networks consisted of using the network to predict a good set of rules for the order, then running the selected rules in the simulation using the order file for which the rules were selected. The simulation then provides the value of the performance measure when the selected rules are implemented on the system with the given order. Once this was done, the resulting value for the performance measure was tested against the performance measure found by the GA for the order used. If the network produced a set of rules which performed as good as or better than the rules selected by the GA, then the network was counted successful for that order. Once this was completed for each order in the test set for each network trained, the network type which predicted a good solution in the most cases was selected as the best network.

## 5.7 Results

When the networks were tested to evaluate their effectiveness, both types performed at nearly the same level. The success rate of the network evaluated is the percentage of cases in which the solution output by the neural network performs at least as good as the solution found by the genetic algorithm, in terms of the specific performance measure chosen, when the genetic algorithm was executed on the specific set of orders into the proposed system. The data generated by the genetic algorithm was separated into several groups for training and testing. The first 500 data points were separated into two groups, one consisting of 375 points (75% of the first 500) and 125 (25% of the first 500). This 75-25% setup was determined to produce the best ratio for training the network. While training the network, the 375 data points were used to train the network and the 125 data points were used during the training session to ensure that the network was not "learning" the noise present in the first 375 data points. This was achieved using a "test-for-best" setup in which every 1000 iterations of the network was tested on the 125 data points to see if any real improvements were being achieved.

The second five hundred data sets were separated into five groups. These five groups of 100 points were used to test the actual capabilities of the network to perform the prediction task adequately. When evaluated in reference to these criteria, both of the networks performed at approximately the same level of accuracy.

The configuration consisting of a separate network for each machine in the system provided rule selections which performed as good as those determined by the genetic algorithm in 55.6 percent of the test cases. Evaluating each of the test groups of 100 as a “batch” gave a standard deviation of 18.3 across the five groups. When a 95 percent confidence interval was computed using these values, a half width of 5.31 was found yielding a confidence interval of [50.29, 60.91]. These values can be considered as percentages or actual values since the test groups were of size 100.

The configuration consisting of one large network for the entire system yielded a slightly higher average of 56.6 percent. It also had a slightly higher standard deviation of 4.83. Due to the higher standard deviation the half width of the 95 percent confidence interval was 5.99. This yielded a confidence interval of [50.61, 62.59].

From this data no inference can be made concerning which configuration is better at determining scheduling rules, because the confidence intervals on each of the configurations overlap. This means that with this level of information the true mean success rates for the two configurations may be the same, or either one’s true mean success rate may be greater than or less than the other’s.

When the two types of networks were executed on the enumerated data files, the following results were obtained. These results were found by comparing the makespan value produced by the simulation when the four rules selected by the respective networks were implemented, with the results obtained when all possible rule combinations were tested on the ten cases. With the single network configuration the neural network found the optimal solution in four of the ten cases. In the other six cases the network provided a set of rules that produced the second-best results in the enumeration of all possible sets. The configuration containing four separate networks produced optimal rule choices in three of the ten test cases, and next to optimal results in the remaining seven cases.

## 5.8 Order Size Testing

Once this testing was completed, further testing of the small order size assumption was performed. In the process of performing the prescribed methodology for solving the problem, enumerations of all possible assignments of dispatching rules to machines were completed for ten different orders. For each of these enumerations, an order file was created containing 20 parts, and then a C++ program was written to run the simulation using the order file with every possible rule combination. Since the simulation package used contained 16 different dispatching rules, this meant that the simulation had to be called 65,536 times, or $4^{16}$ times. After each simulation run, the resulting makespan time for that order and specific choice of rules was recorded in a file.

These enumerations were initially run to allow the genetic algorithm and neural networks to be tested on problems with known answers. By having a certain number of solutions with known results, the effectiveness of the final results could be validated. When the enumerations were completed and analyzed, however, certain assumptions which were made in the formulation of the solution were brought into question. The enumeration files contained a limited number of unique solutions for many of the order files used. This means that while there were 65,536 times recorded in the enumeration file, thousands of them were the exact same number. Originally the data in the enumeration files was grouped into "bins" of size 100, and the number of times which fell into each of these bins were graphed to show the resulting patterns. The number of unique solutions found using this technique varied from 2 to 6, and the question was raised as to whether this limited number of results was due in part to the small order used.

To determine whether the order size and number of solutions gained by varying the dispatching rules were linked, it was decided that enumerations of the system utilizing orders containing more parts would be performed, and the resulting data again analyzed. The order sizes selected for further testing were 40 parts, 60 parts, and 80 parts per order, along with one of the original enumerations on the system with an order containing 20 parts.

In order to run the simulation using larger order sizes, the length of run of the simulation was increased to accommodate the longer running times. Next the order creation program was modified to create three different order files: one containing 40 parts, one containing 60 parts, and one containing 80 parts. These files were placed into the simulation one at a time and an enumeration of all possible combinations of rules on the four machines was run for each file. The results of each of the rule combinations for each order size were placed into three different files. Once these enumerations were completed, a program was written to read the file created by the enumeration and produce a file which contained each unique time in the enumeration file along with the number of occurrences of that specific time in the enumeration file. This program was then executed on the data files created by the enumeration program. In order to maintain continuity in the analyzation of the data, this "sorting" program was also run on one of the original ten data files containing data from the enumeration of an order file which contained only 20 parts.

Once the enumerations of all possible dispatching rule choices were completed for orders of 40, 60, and 80 parts per order, the data was analyzed using the program described above. The resulting data contains specific makespan values, along with the number of times these times occurred in the course of the enumeration. This data was then sorted by time and the following are these sorted values. Note that the number of distinct solutions varies with order sizes.

Table 5.1 Results of Order Size Tests

Results of Enumeration for 20 Parts in the Order File

| MakeSpan (minutes) | Frequency of Occurrence |
|---|---|
| 2880 | 1 |
| 2898 | 1246 |
| 3130 | 5765 |
| 3140 | 99 |
| 3160 | 30415 |
| 3180 | 4554 |
| 3190 | 18 |
| 3200 | 668 |
| 3220 | 733 |
| 3240 | 1082 |
| 3260 | 212 |
| 3280 | 565 |
| 3300 | 18290 |

| | |
|---|---|
| 3320 | 370 |
| 3340 | 1512 |

Results of Enumeration for 40 Parts in the Order File

| MakeSpan (minutes) | Frequency of Occurrence |
|---|---|
| 5778 | 19807 |
| 6130 | 4432 |
| 6140 | 28 |
| 6160 | 35003 |
| 6170 | 136 |
| 6180 | 3949 |
| 6190 | 16 |
| 6200 | 1076 |
| 6220 | 1087 |
| 6360 | 2 |

Results of Enumeration for 60 Parts in the Order File

| MakeSpan (minutes) | Frequency of Occurrence |
|---|---|
| 9330 | 4962 |
| 9340 | 25 |
| 9360 | 47086 |
| 9380 | 225 |
| 9400 | 64 |
| 9420 | 917 |
| 9440 | 66 |
| 9460 | 4740 |
| 9480 | 706 |
| 9500 | 6669 |
| 9520 | 35 |
| 9560 | 41 |

Results of Enumeration for 80 Parts in the Order File

| MakeSpan (minutes) | Frequency of Occurrence |
|---|---|
| 3859 | 1 |
| 4819 | 65533 |
| 26003 | 2 |

The enumeration on the order file containing 20 parts resulted in 15 distinct makespan times. The enumeration on the file containing 40 parts resulted in 10 distinct makespan values. The enumeration on the order containing 60 parts yielded 12 unique makespan values. Finally the enumeration on the order of 80 parts showed only 3 different makespan values. Therefore the order size did not appear to influence the complexity of results obtained.

The results are discussed in Chapter 6.

# 6. CONCLUSIONS/RECOMMENDATIONS

The work in this thesis has provided a great deal of insight into the problems associated with using GA's and Neural Networks to select dispatching rules. In this section the results of the project are reviewed, and possibilities for further refinement, improvement, and adaptation into other areas are presented in order to allow this work to be extended and applied to a broader range of problems.

## 6.1 Conclusions

This thesis applied genetic algorithms and artificial neural networks to the problem of scheduling parts through a job shop system. The approach first used genetic algorithms to select the best dispatching rules to use for different combinations of orders into the system. Second a noise reduction technique was used to reduce the randomness present in the data. Finally a portion of the data was used to train two neural networks. Once this procedure was completed the neural network was tested using the rest of the data. The following conclusions were drawn from information obtained during the different steps.

The genetic algorithm allowed results to be obtained quickly, and provided the necessary data very quickly. When the Rule Set Replacement Technique was executed however, there were a few cases (5 to 10 per hundred) in which a set of rules was found which produced a result better than the one found by the genetic algorithm. This implies that the GA was not finding the optimal solution in these cases, so in further research a more extensive designing of the genetic algorithm is suggested in which the GA is executed on two or three problems with known solutions in order to prove optimality.

While the neural networks had some difficulty in learning to predict scheduling rule choices, the rule selection technique used allowed the networks to learn a great deal from the inputs given. This technique removed a significant amount of noise by consolidating many of the solutions into one solution. This allowed the network to learn better from the data presented. However, it also tended to make the network biased toward the solution found by the Rule Set Replacement Technique. Methods for reducing/eliminating this difficulty are discussed in the recommendations section.

The neural networks predicted equally well on the problem whether in a single or multiple network configuration. This result may have been influenced to a small degree by the problem resulting from the Rule Set Replacement Technique mentioned above, but in any case this implies that further research should concentrate more on inputs selected than on the specific architecture used, unless of course other factors lend themselves to this choice as is mentioned in the recommendations section.

The testing to determine whether the small order size affected the simplicity of the results obtained, was performed as described in Chapter five. The results indicate that the size of the order file has little influence on the number of makespan times recorded; thus the results obtained from this procedure can be applied to larger orders than those used for this specific system. It is important to note that while only 3 distinct makespan values were found when using the order containing 80 parts, this does not necessarily indicate that larger orders are more likely to have a smaller number of times. In fact when enumerations were completed on ten different orders which contained 20 parts each, the number of distinct values varied from 2 to 15. Thus while there is evidence to support the conclusion that there is a large amount of variation among individual order sizes, no conclusions should be drawn about any linking of order size and number of unique resulting times.

What this means in terms of this thesis is that the results of this thesis can be applied to problems containing larger order sizes than were used in the thesis itself. This generalization capability greatly increases the effectiveness of this work and will permit further work to be done in the way of applying this work to problems of other sizes and types.

## 6.2 Recommendations

This project has provided a good deal of information which will allow further research/implementation to be successfully completed. Several recommendations are made in this section and while they may not provide perfect solutions, they should allow more accurate results to be obtained in this and a variety of problems.

In order to further assist the networks in learning from the supplied data, the Rule Set Replacement Technique should be extended to perform second and third iterations. In order to do this, the data would need to first be obtained using a genetic algorithm or other information generation technique. Next the rule selections could be tried on the other cases as described earlier and the cases in which the result was within an acceptable error value $\epsilon$, then that set of rules would be used to replace the original set. This error value $\epsilon$ would allow selections which were not quite as good, but within a close enough range as to be acceptable, to be chosen in order to reduce the amount of learning necessary for the neural network to successfully solve the problem. Next the remaining data points would be tested again among each other and the set of rules which produced acceptable values in the most cases would be used to replace the rules in those cases. This procedure could continue on the remaining populations until there were no more cases left in the unmodified set of points. This procedure is shown in the following figure. This procedure would be time consuming but should further allow the noise in a given problem to be reduced, thus allowing the neural network to learn much more easily and successfully.

Expanded Rule Set Replacement Technique

1 **GENERATE DATA**

2 **TEST RULE SETS ON ALL OTHER CASES**

3 **CHOOSE SET OF RULES WHICH PRODUCES ACCEPTABLE RESULTS IN THE MOST CASES**

4 **REPLACE SET OF RULES IN ACCEPTABLE CASES**

5 **PERFORM STEPS 3,4,5 ON UNCHANGED CASES UNTIL NO SET OF RULES IN THE UNCHANGED SET PRODUCES ACCEPTABLE RESULTS IN ANY OF THE OTHER UNCHANGED CASES**

Figure 6.1

Another possibility which would allow the problem to be implemented into industry more effectively would be to add on a real-time training module which would begin training the network as soon as data was reviewed, and would continuously improve itself by predicting and learning from these predictions on a day-to-day basis. In an actual manufacturing environment this would allow the network to learn over time and continuously improve its predictive capabilities using the actual results of using its selections to train itself.

In order to allow the research to be implemented into other types of facilities, further research is needed in rolling horizon problems, perhaps using some type of internal feedback network such as those used to learn pieces of music. By using this type of network in a continuously changing environment, the network may be able to learn to include possible future inputs in its decision making process. This will require significantly more time to perform however, due to the fact that the procedure for obtaining training data will be much more difficult.

The system described did not include prior loading of the machines in order to keep the programming to a minimum. This could also be done using a random assignment or some other technique to load the system before running a set of orders through the system. If this is done, information concerning these loadings must be included in some form as inputs into the system. This will allow the network to include this information in its selection of dispatching rules.

In problems containing a greater number of machines than this system contained, one single network may become too large to train effectively. In a situation where several computers may be run in parallel, the multiple network configuration would allow the different networks to be trained at the same time. In the case of a single computer, the larger single-network configuration may work better due to the simplicity in setting it up, and the fact that it would take less time to train one large network than several marginally smaller ones.

Performance measures other than makespan should be simple to use given the procedure used in this thesis. The difficulty will come in implementing any performance measure in a rolling horizon, continuous problem. For these problems make-span is probably not the best choice since it works the best in a fixed horizon, empty start and finish, problem. So long as the same performance measure is used throughout however, the choice of performance measure should have little bearing on the effectiveness of the network.

In order to apply this technique to the rolling horizon problem, certain modifications would be necessary. These modifications would be needed because in the rolling horizon problem, a new set of rules would need to be used each time the orders into the system changed. One possibility would be to generate orders with time lags placed in the order file so that all of the orders do not arrive at the same time. In this manner the same inputs could be used to train the network, perhaps with additional ones to compensate for the time differences. If this approach were to be used, the network would need to be executed every time a new order was placed into the system, thus allowing the dispatching rules to change with the orders.

In order to fully implement his type of solution, machine loading prior to system startup would need to be considered. If parts were allowed to be partially completed before the system started, then the simulation would reflect the non-terminating nature of the system, and the GA would be forced to account for this in its selection of dispatching rules.

## REFERENCES

Ackoff, R.L. (1979), "The future of Operational Research is Past", *Journal of the Operational Research Society.* 30 (3), 93-104.

Ballakur, A. and Steudel, H.J. (1984) "Integration of Job Shop Control Systems: A State-of-the-Art Review", *Journal of Manufacturing Systems*, 3, 71-79.

Blackstone, J.H. Jr., Phillips, D., and Hogg, G. (1982) "A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations" *International Journal of Production Research,* 20, 27-45

Caskey, Kevin R., 1993, Genetic Algorithms and Neural Networks Applied to Manufacturing Scheduling. *University of Washington.*

Chiu, Chanshing, 1994, A Learning-Based Methodology for Dynamic Scheduling in Distributed Manufacturing Systems. *Purdue University.*

Conway, R.W. (1965) "Priority Dispatching and Work-In-Process Inventory in a Job Shop", *Journal of Industrial Engineering*, 16 (2), 123-130, and 16 (4), 228-237.

Dar-El, E.M. and Wysk, R.A. (1982) "Job Shop Scheduling - A Systematic Approach", *Journal of Manufacturing Systems,* 1 (1), 77-88.

Davis, L., 1991, Handbook of Genetic Algorithms. Van Nostrand Reinhold, USA.

Elmaraghy, H.A. (1982) "Simulation and Graphical Animation of Advanced Manufacturing Systems", *Journal of Manufacturing Systems,* 1 (1), 53-63.

Goldberg, E., 1989, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley Publishing Company, USA.

Hershauer, J.C., and Ebert, J. (1975) "Search and Simulation Selection of a Job Shop Scheduling Rule", *Management Science*, 21 (7), 833-843.

Kim, Y. (1990) "A Comparison of Dispatching Rules for Job Shops with Multiple Identical Jobs and Alternative Routings", *International Journal of Production Research*, 28 (5), 953-962.

McCartney, J. and Hinds, B.K. (1981) "Interactive Scheduling Procedures for Flexible Manufacturing Systems", *Proceedings of 22nd International Machine Tool Design and Research Conference,* Manchester, Sept. 1981, 47-54.

Minagawa, Masaaki, and Yukinori, K. (1992), "Scheduling flow-shops with alternative resources: a GA based heuristic approach." *Proc. of the 1992 Japan - USA Symposium on Flexible Automation*, San Francisco, CA, 1151-1154.

Montazeri, M. and Van Wassenhove L. N. (1990) "Analysis of Scheduling Rules for an FMS", *International Journal of Production Research*, 28 (4), 785-802.

O'Grady, P.J. and Menon, U. (1986), "A Concise Review of Flexible Manufacturing Systems and FMS Literature", *Computers in Industry*, 7 (2), 155-167.

Rabelo, L., Yih, Y., Jones, A., and Tsai, J., 1993, "Intelligent Scheduling for Flexible Manufacturing Systems." *Proc. of the IEEE International conf. on Robotics and Automation*, Atlanta, GA, 810-815.

Sittisathanchai, S., 1994, "Genetic Scheduler for Job Shops." *Proc. of the Neural Networks in Eng. Conf.*. St. Louis, MO, 351-356.

Stecke, K.E. and Solberg, J.J. (1981), "Loading and Control Policies for a Flexible Manufacturing System", *International Journal of Production Research.* 19 (5), 481-490.

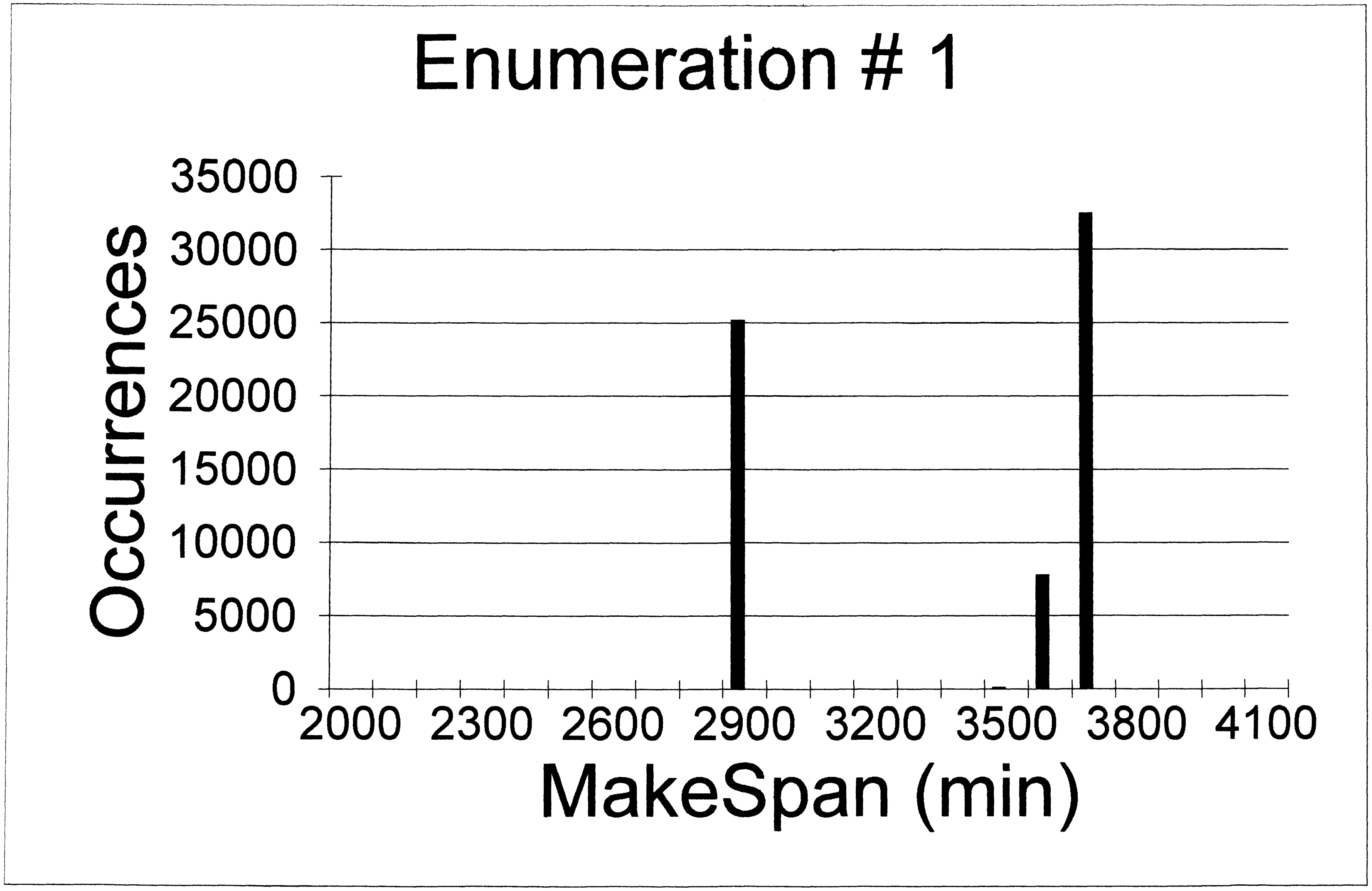
Suri, R. (1988), "A New Perspective on Manufacturing Systems Analysis", In: *Design and Analysis of Integrated Manufacturing Systems,* edited by W. Dale Compton, National Academy Press.

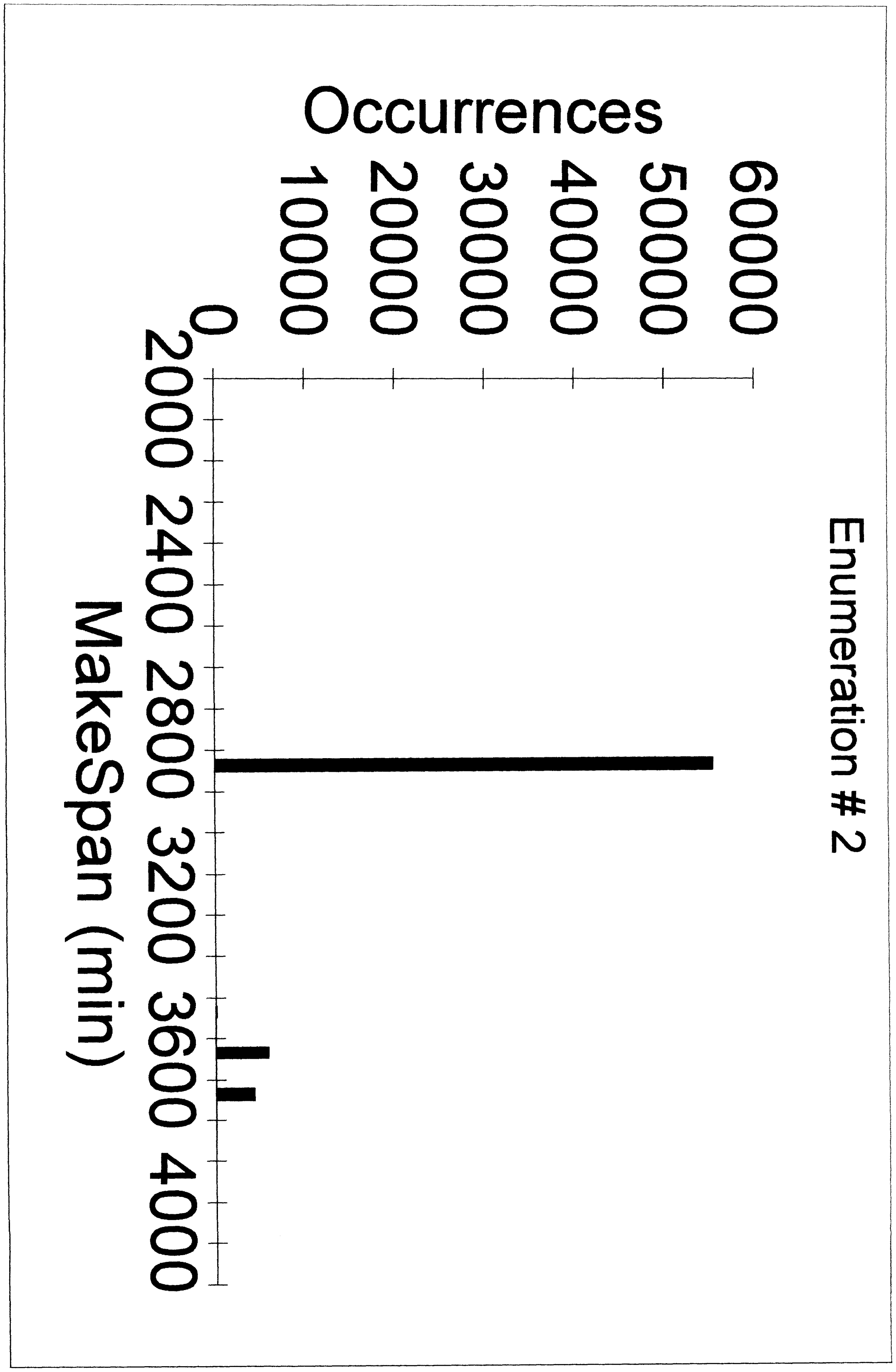
Syswerda,G.,"Schedule Optimization Using Genetic Algorithms," Handbook of Genetic Algorithms. L. Davis (ed.), Man Nostrand Reinhold, New York, 1990.

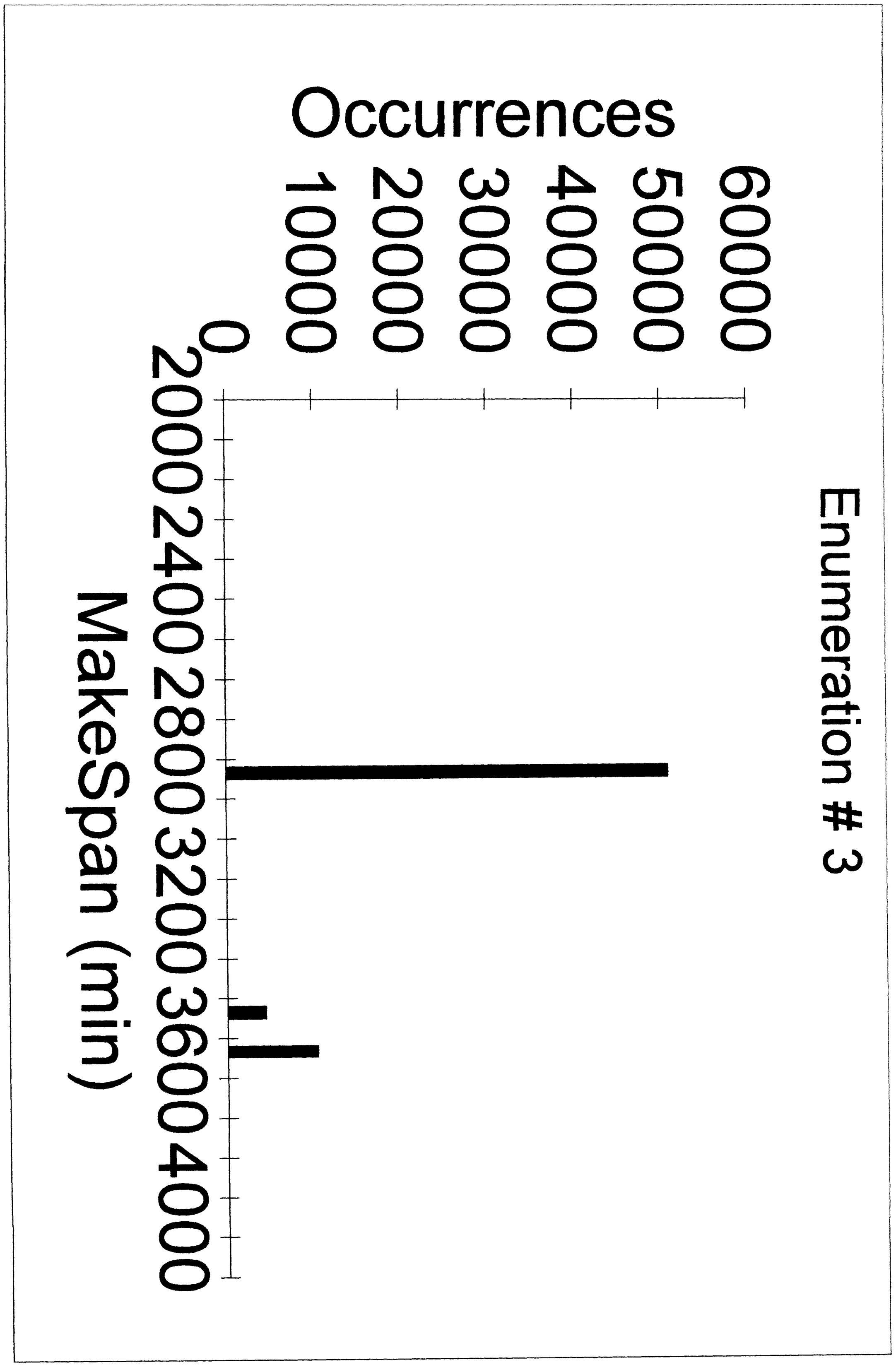
Whitley, D. Starkweather, T., and Fuquay, D.,"Scheduling Problems and the Traveling Salesman Problem: The Genetic Edge Recombination Operator," Proceedings of the Third International Conference on Genetic Algorithms, J. Shaeffer (ed.), Morgan Kaufman, 1989.

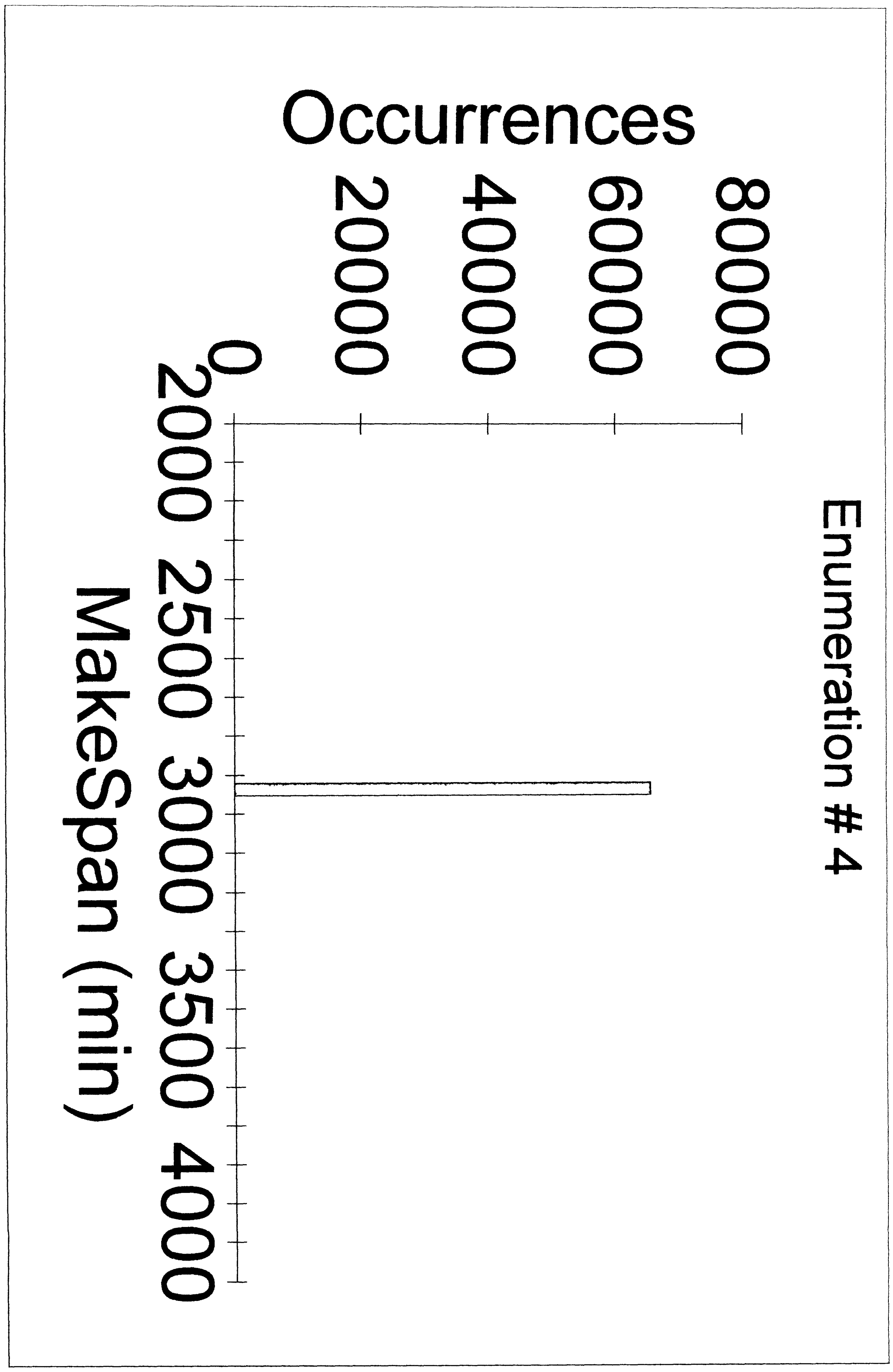
Yamamura, M., Satoh, H., and Kobayashi, S., 1994, "Analysis of Crossover's Effect on Genetic Algorithms." *Proc. of the 1st IEEE conf. on Evolutionary Computation.* Orlando, FL, 613-618.

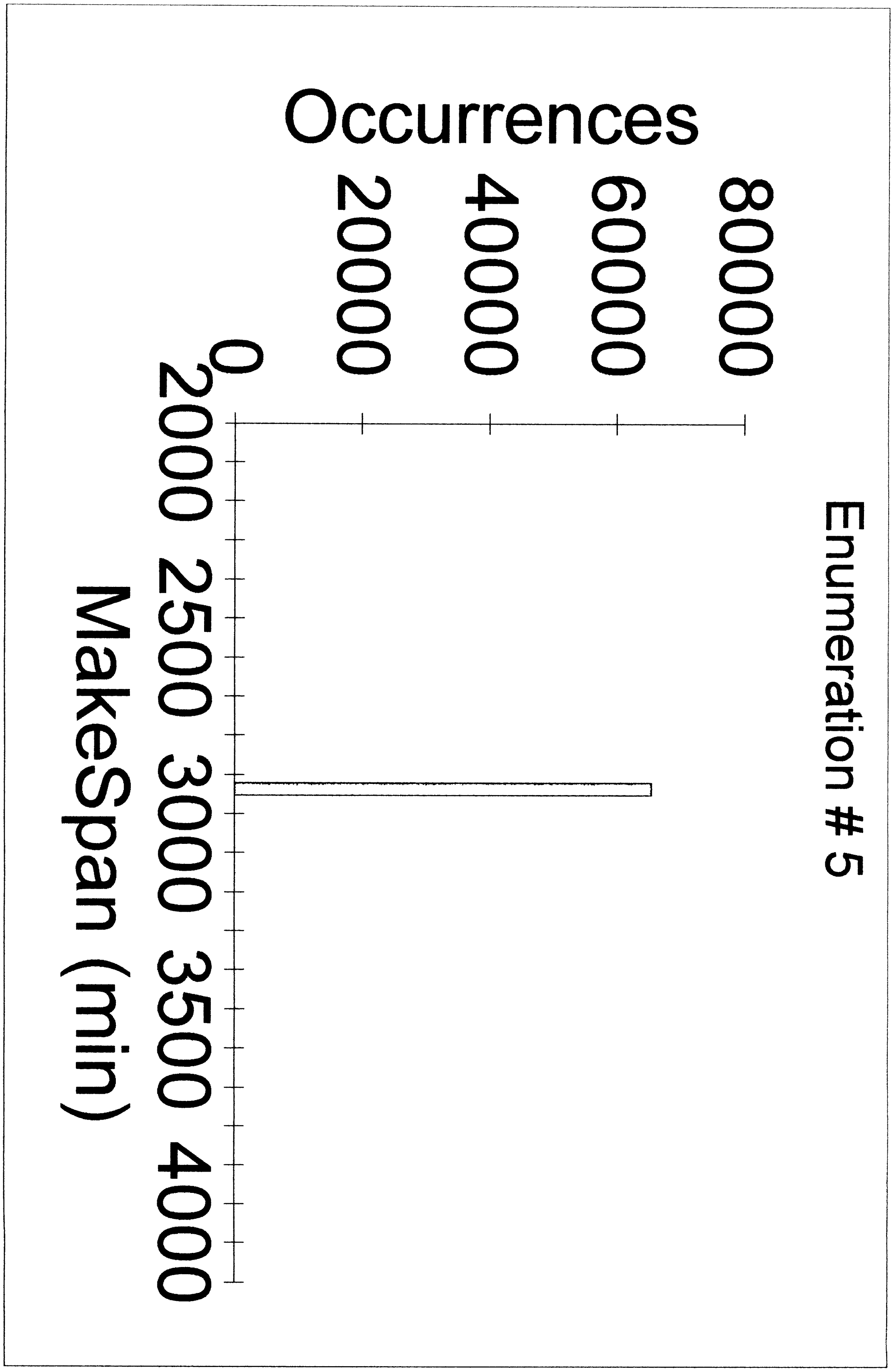
## APPENDIX

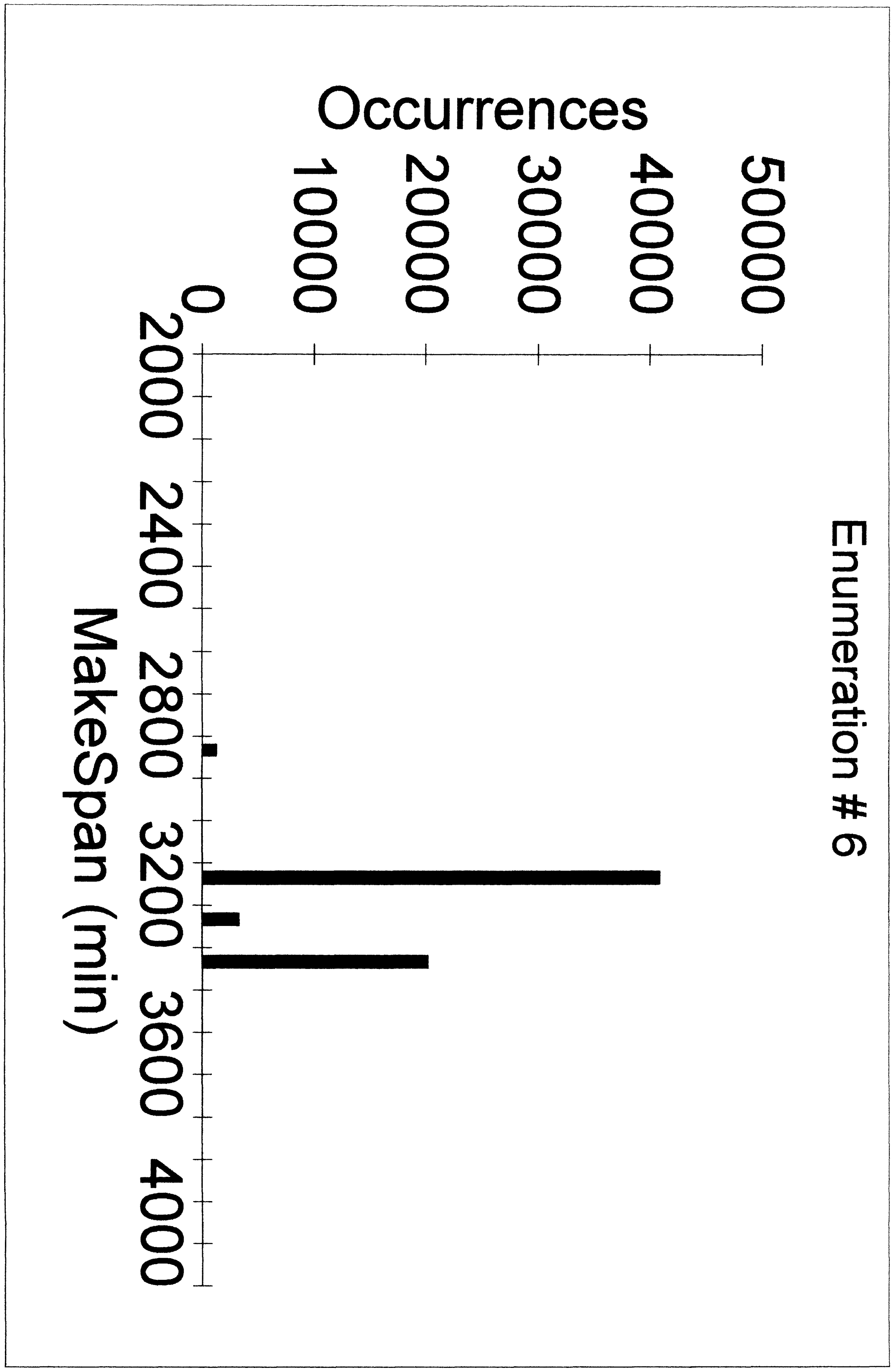

Enumeration # 1
Occurrences
35000
30000
25000
20000
15000
10000
5000
0
2000
2300
2600
2900
3200
3500
3800
4100
MakeSpan (min)

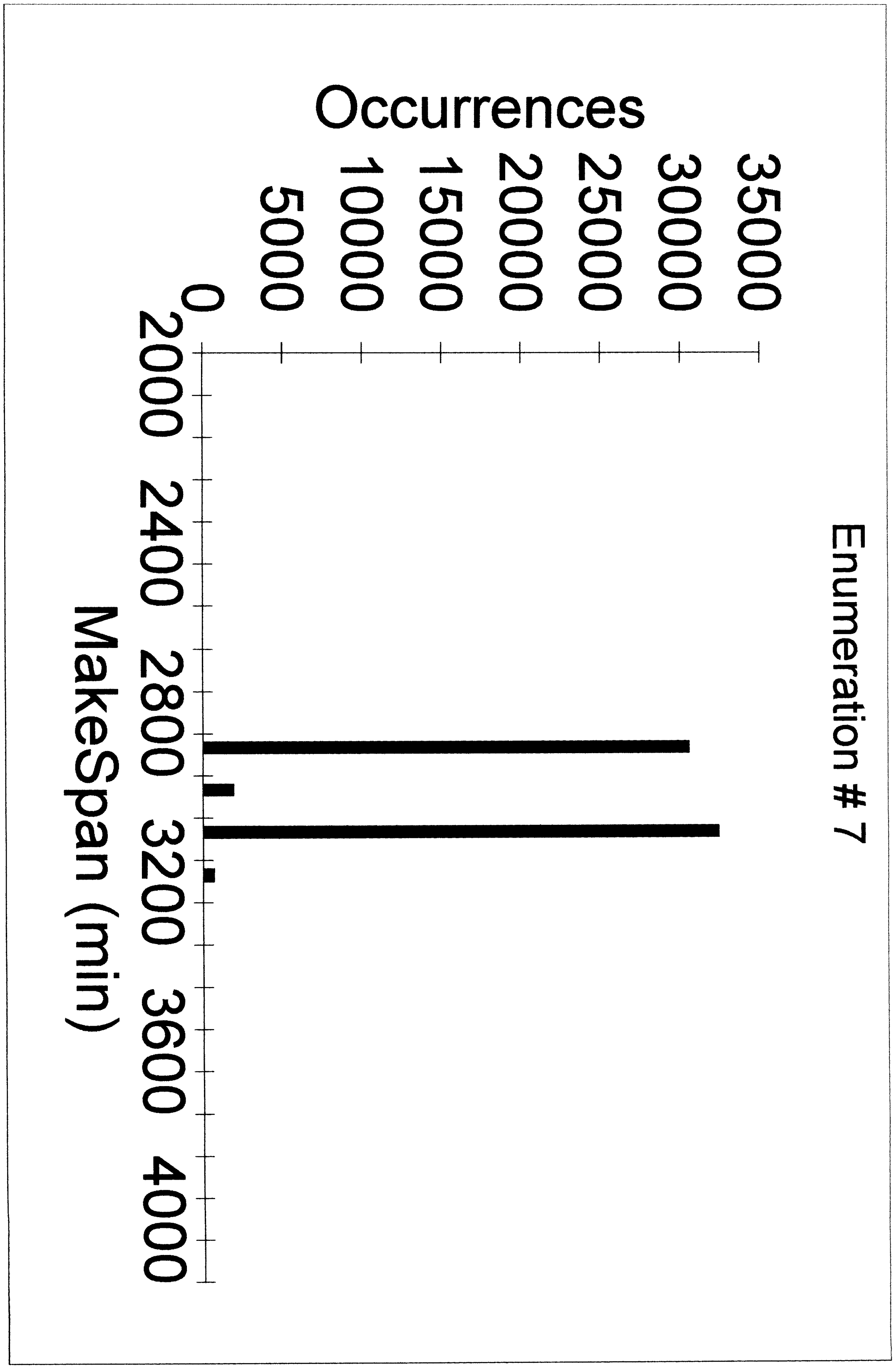

Enumeration # 2
Occurrences
0
100000
200000
300000
400000
500000
600000
2000
2400
2800
3200
3600
4000
MakeSpan (min)

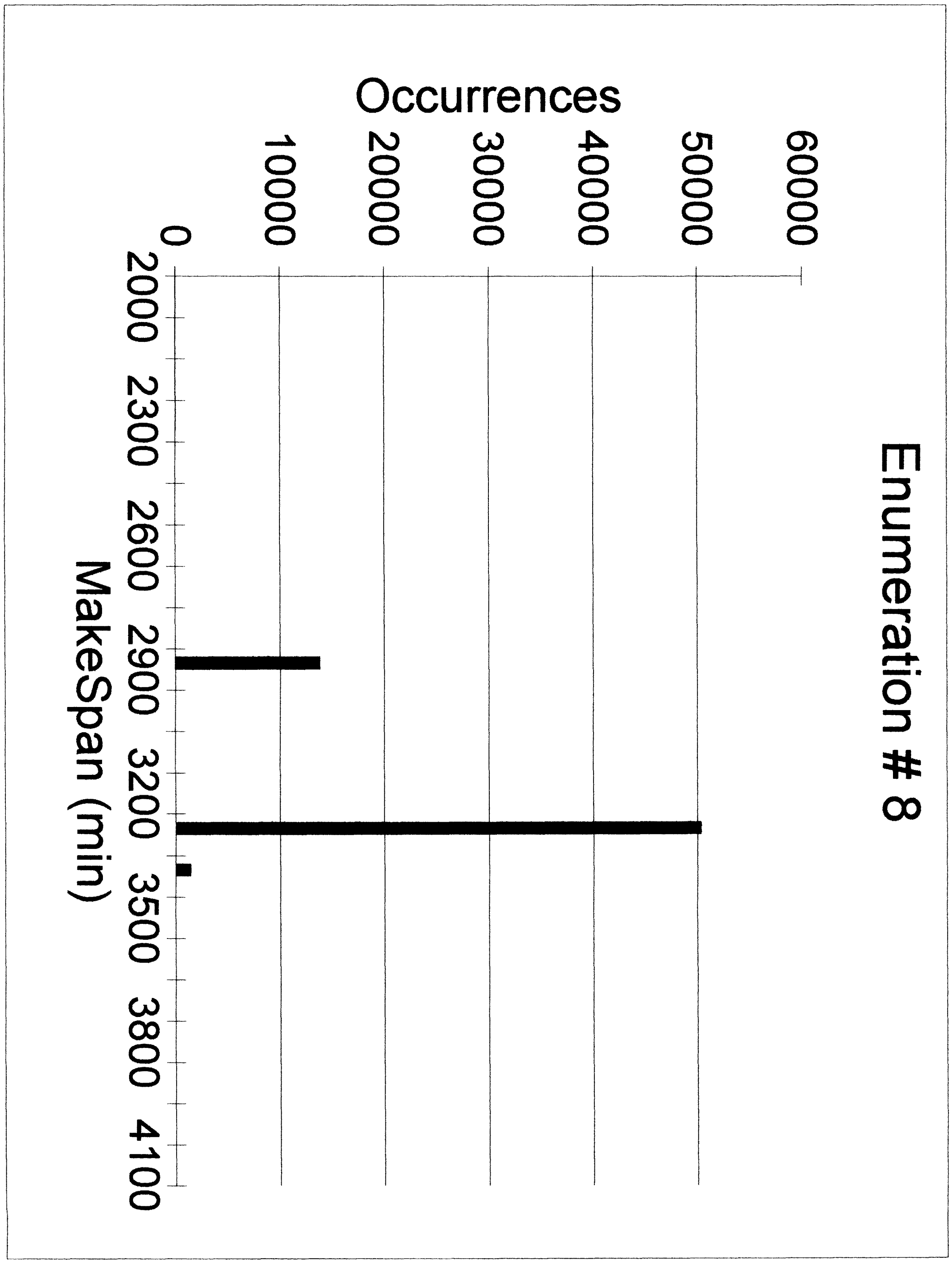

Enumeration # 3
Occurrences
0
100000
200000
300000
400000
500000
600000
MakeSpan (min)
2000
2400
2800
3200
3600
4000

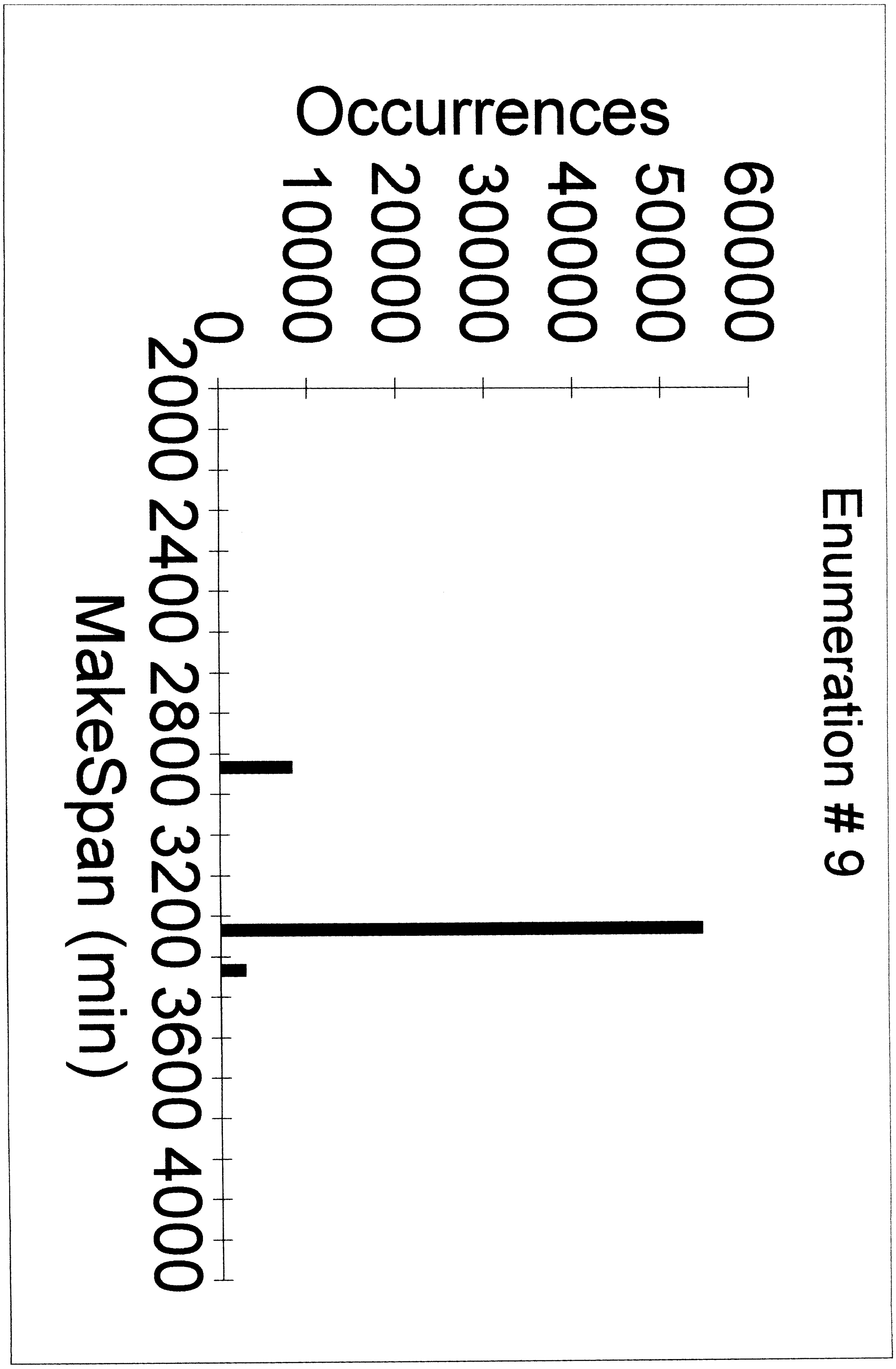

Enumeration # 4
Occurrences
0
20000
40000
60000
80000
2000
2500
3000
3500
4000
MakeSpan (min)

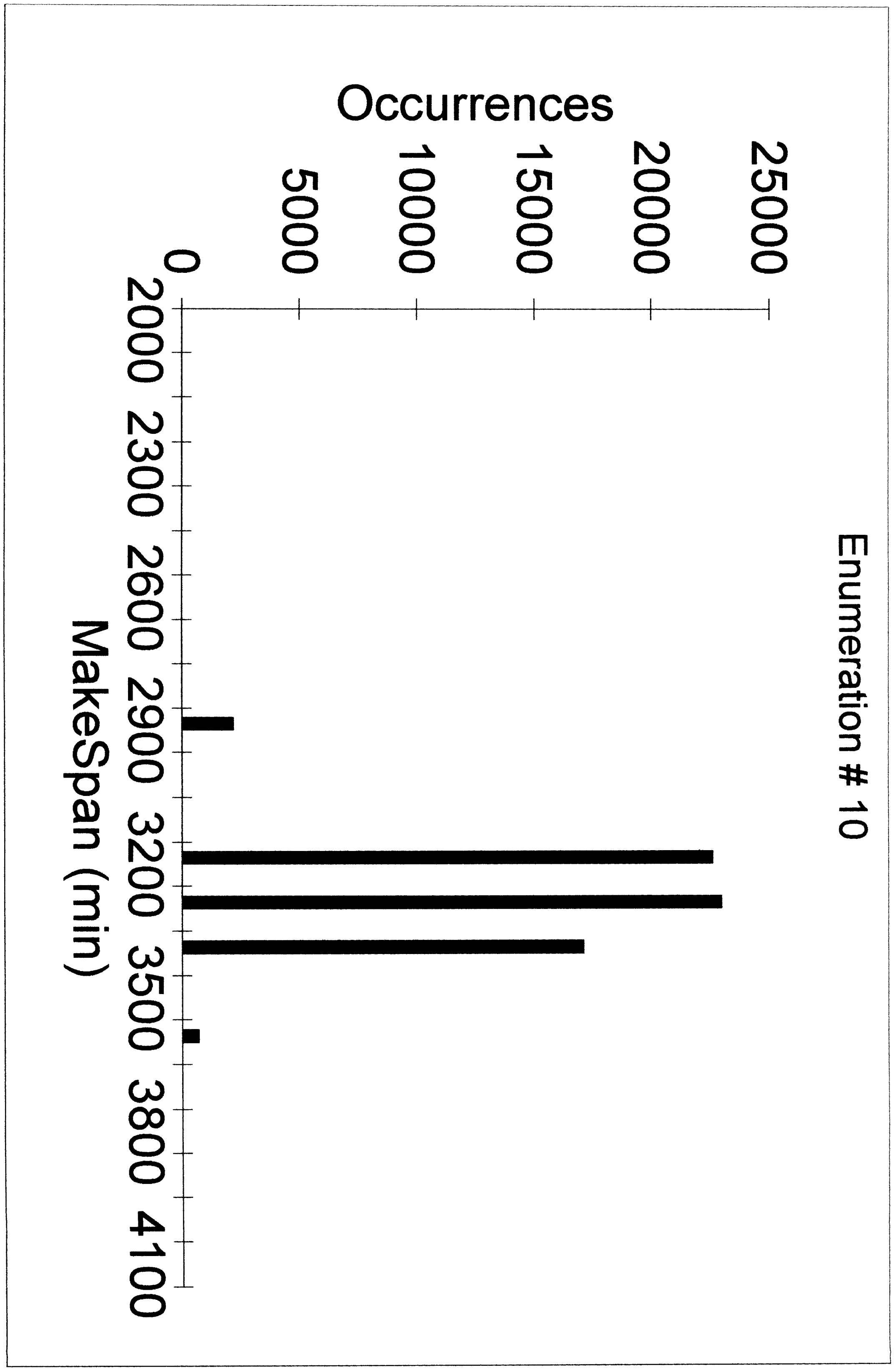

Enumeration # 5
Occurrences
0
200000
400000
600000
800000
MakeSpan (min)
2000
2500
3000
3500
4000


Enumeration # 6
Occurrences
0
10000
20000
30000
40000
50000
2000
2400
2800
3200
3600
4000
MakeSpan (min)


Enumeration # 7
Occurrences
0
5000
10000
15000
20000
25000
30000
35000
2000
2400
2800
3200
3600
4000
MakeSpan (min)


Enumeration # 8
Occurrences
0
10000
20000
30000
40000
50000
60000
2000
2300
2600
2900
3200
3500
3800
4100
MakeSpan (min)


Enumeration # 9
Occurrences
0
100000
200000
300000
400000
500000
600000
2000
2400
2800
3200
3600
4000
MakeSpan (min)


Enumeration # 10
Occurrences
0
5000
10000
15000
20000
25000
2000
2300
2600
2900
3200
3500
3800
4100
MakeSpan (min)