Evaluation of partitioning schemes of the nested partitions method in the context of simulation-based optimization

by

Jagpreet Chhatwal

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:
Sigurdur Ólafsson, Major Professor
Sarah M. Ryan
Tapabrata Maiti

Iowa State University

Ames, Iowa

2004

Copyright © Jagpreet Chhatwal, 2004. All rights reserved.

Graduate College
Iowa State University

This is to certify that the master's thesis of

Jagpreet Chhatwal

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF TABLES v

LIST OF FIGURES vi

ACKNOWLEDGEMENTS vii

ABSTRACT viii

CHAPTER 1. Introduction

1.1 Continuous Decision Variables 2

1.2 Discrete Decision Variables 4

CHAPTER 2. Simulation Optimization

2.1 Simulation Optimization for Discrete Systems 7

2.2 Problem Setting 8

2.3 Metaheuristics 8

2.3.1 Tabu Search 9

2.3.2 Simulated Annealing 10

2.3.3 Genetic Algorithm 11

CHAPTER 3. Nested Partitions Method

3.1 NP Algorithm 12

3.2 Black Box Model for Simulation Optimization 15

3.3 Partitioning in NP Method 16

3.3.1 Importance of Partitioning Scheme 17

3.3.2 Intelligent Partitioning 18

3.3.3 NP Algorithm with Intelligent Partitioning 21

CHAPTER 4. Case Study: A Job Shop Model

4.1 Problem Definition 24

4.2 Entropy Calculations 25

4.3 Intelligent Partitioning Schemes 30

CHAPTER 5. Conclusions and Future Work 37

APPENDIX A - Code of NP Method with Intelligent Partitioning in C 38

APPENDIX B – Program Output for One Iteration of the NP Method 55

REFERENCES 60

LIST OF TABLES

Table 4.1 Mean Processing Times of Machines in Each Work Station................ 23

Table 4.2 Category division of sample points... 27

Table 4.3 Categories having sample points with statistically different performance estimates.. 28

Table 4.4 Average Entropy Values of Workstations.................................... 28

Table 4.5 Performance of different partitioning schemes.............................. 32

Table 4.6 Solution of NP Method Using Intelligent partitioning scheme............... 34

Table 4.7 Sample variance in performance estimate of profit for different partitions.. 35

LIST OF FIGURES

Figure 4.1 Manufacturing System Consisting of Four Work Stations.................. 23

Figure 4.2 Entropy comparisons of workstations using sample size of 40............. 29

Figure 4.3 Entropy comparisons of workstations using sample size of 60............. 29

Figure 4.4 Entropy comparisons of workstations using sample size of 100............ 30

Figure 4.5 Steps of Nested Partitions Method.. 31

Figure 4.6 Performance comparison of partitioning schemes............................ 33

ACKNOWLEDGEMENTS

I would especially like to thank Dr. Olafsson, my major professor, for his guidance and help given during the length of my studies at Iowa State University. I sincerely appreciate his efforts in guiding and encouraging me for further studies. I will miss him as my adviser during my doctoral studies.

I would like to express thanks to Dr. Sarah Ryan, my committee member, for laying a strong foundation of stochastic systems in me, which motivated me for this thesis' topic and higher studies in the same field. In addition, I am highly thankful to Dr. Tapabrata Maiti, my minor representative of statistics, for his invaluable guidance and making me fascinate this subject.

I am thankful to my parents for their moral encouragement and financial assistance, which made this journey possible. Above all, I am highly indebted to God, Who is the driving force of all.

ABSTRACT

A new generic partitioning scheme of the nested partitions (NP) method in the context of simulation optimization is evaluated in this thesis. A heuristic, which partitions the feasible region "intelligently", is applied on a discrete-event simulation model of a manufacturing system whose objective is to maximize total profits.

The basic idea of NP method is to divide the feasible region into partitions and evaluate each region's performance using sampling. Based on performance evaluation, the most promising region is selected for the next iteration. The efficiency of NP method relies heavily on partitioning, if done effectively, can decrease computational time. To develop a generic intelligent partitioning scheme, the idea of diversity known from information theory is applied. Numerical results show that the efficiency of the NP method depends on the partitioning scheme of the feasible region. In addition, intelligent partitioning shows good results, but doesn't always guarantee high computational efficiency.

CHAPTER 1

Introduction

Optimization under uncertainty has been studied since long time, but due to recent advancements in computing techniques, it is nowadays solved with a different prospective. Simulation is now widely used for this kind of problem. Much research has already been done in the area of optimization via simulation and there is lot more to come in the near future. The main advantage of using simulation is that the performance of a complex system can be estimated from the output of a simulation model.

Evaluating the performance of every feasible point using optimization under uncertainty is very time consuming. Even though many algorithms have been developed, there are some difficulties when applying these problems to the real world. Solving complex problems requires great amount of time. Hence, there is a need for efficient algorithms which can reduce some amount of work.

An algorithm discussed in this thesis is based on the Nested Partitions (NP) method and the main focus is on partitioning schemes. It is shown how partitioning in different ways changes the amount of work required to measure the performance of the system under consideration. Therefore, partitioning in an effective way is important when using NP method. Implementing the NP method is quite problem dependent, and in particular, partitioning schemes that have been devised in the past have drawn heavily on specific structure related to the application itself. It requires substantial effort on the practitioner using this method to devise effective partitioning schemes. In this thesis, a generic partitioning

scheme, based on the idea of entropy from information theory, has been implemented in the context of simulation-based optimization. This feature can be incorporated in a black box model of simulation optimization.

There are many methods of optimization under uncertainty. Using these methods depends upon the structure of the problem. For example, gradient estimation, sample path optimization, and stochastic optimization are applicable when the feasible decision variables are continuous. On the other hand random search and statistical methods are applicable for discrete decision variables. Based on above differences we can divide simulation optimization in two main categories; continuous parameter simulation optimization and discrete parameter simulation optimization. Further details are given below.

1.1 Continuous Decision Variables

Until recently most techniques were developed for continuous input parameters. There are several common methods used for continuous input parameters which are categorized as gradient-based methods. In these methods simulation is used to obtain the estimate of the gradient of the expected system performance with respect to the (continuous) parameter.

The classical stochastic optimization methods are based on an iterative search in the direction of time of the gradient. This was originally suggested by Robbin-Monro and Keifer-Wolfowitz in the 1950s (Robinson *et al.*, 1951). The Robbins-Monroe algorithm is simply a root finding procedure for functions whose values are not known but observed with noise. The Robbins-Monroe algorithm estimated the gradient directly; whereas, the Keifer-Wolfowitz algorithm uses finite differences to the derivative. In both cases, the primary

implementation problem determines the step size. One of the problems with the Robbins-Monroe algorithm is that when it is applied to the solve optimization problems with minimization as objective with an unbounded feasible set, the convergence of the algorithm is not guaranteed when the objective function grows faster than quadratically in the decision parameter. Andradóttir (1996) has proposed an alternative approach that addresses this problem using scaling.

The most straightforward approach for the gradient estimation methods is the finite difference method (Glynn, 1989). This method is simple to implement and generally applicable, but it has several difficulties when applied to practical problems. One of the big difficulties is that too much time is spent calculating the gradient. The other difficulty is the gradient estimate obtained using finite differences are generally biased. Trying to reduce bias leads to another difficulty; large variance in the estimation. To reduce variance methods such as CRN (Common Random Number) have been suggested.

Another approach to continuous parameter simulation optimization is sample path optimization. It involves approximating the original simulation optimization problem with a deterministic optimization problem. It uses simulation to generate one sample path and it yields and an estimated optimal solution that depends on the sample path that the approximate deterministic optimization problem is based on. Several researchers have studied simulation optimization approaches of this form. More details about different simulation optimization techniques can be found in Andradóttir (1998).

1.2 Discrete Decision Variables

In this section a brief review of simulation optimization techniques is given when the feasible region is discrete. Random search methods are mostly used for solving this category of problems. These methods do not guarantee to converge; hence are also called heuristics. In early research literature, specialized heuristics were typically developed to solve complex combinatorial optimization problems. This required a new approach for every new problem and information obtained from one problem did not always generalize well to different class of problems. On the other hand, with the emergence of more general solution strategies or metaheuristics, the main challenge has become to adapt these metaheuristics to a particular problem or class of problem. Some of the commonly used metaheuristics are tabu search simulated annealing, and genetic algorithms. More details of these methods can be found in section 2.3. An algorithm discussed in this thesis is based on nested partitions method. Other than random search methods, statistical selection methods are commonly used.

The most popular statistical methods used are ranking and selection (R&S) and multiple comparison procedures (MCPs), which are applicable when the input parameters are discrete and the number of designs to be compared are both discrete and small (say between 2 and 30). Simulations are performed to compare two or more system designs. Ranking and selection are selection methods developed to select the best system or a subset that contains the best system design from a set of competing alternatives (Goldman and Nelson 1994). Multiple comparison procedures use pairwise comparison to derive relationships among all designs. In other words, R&S yield the best system while MCPs yield information about relationships among the alternative solutions. Sanchez (1997) gives an overview of R&S with samples. Wen and Chen (1994) present single-stage sampling procedures for different

MCPs. Goldman and Nelson (1994, 1998) provide comprehensive reviews of R&S and MCPs.

The remainder of this thesis is organized as follow: Chapter 2 defines simulation optimization problem; how it is different from other optimization problems. In addition, commonly used metaheuristics for simulation optimization are explained briefly. In Chapter 3, nested partitions method is explained in detail, and importance of partitioning the feasible region in an efficient way. Chapter 4 discusses the results of partitioning schemes in the context of simulation-based optimization of a manufacturing system. A job shop model is considered where the objective is to maximize the profit by finding optimal number of machines in each workstation and finally, in Chapter 5, the conclusions and future research is proposed.

CHAPTER 2

Simulation Optimization

Optimizing complex and large discrete systems like manufacturing systems, supply chain management, determining policies in inventory systems and financial systems itself is a large problem. It is not possible to express them in a simple analytical or mathematical form. For this reason, simulation is often used to estimate the performance of a stochastic system for different set of parameters. Simulation optimization is defined as finding the best set of decision parameters for a system, where the performance is evaluated based on the output of a simulation model of this system.

Stochastic optimization and simulation optimization are very closely related to each other. In simulation optimization the focus is on output from stochastic discrete-event simulation models, whereas stochastic optimization is generally more broadly defined, encompassing any system that has stochastic behavior. Simulation requires much more computation work as compared to an optimization problem. Even a single replication of a model requires more work than many large linear programming problems. Therefore, even a considerably small reduction in number of simulation runs can save large amount of time.

Simulation optimization is relatively difficult problem to solve. Its main reason is that the performance cannot be determined but has to be estimated. Even when there is no uncertainty, optimization can be difficult when the number of design variables is large and little is known about the structure of the problem. Hence, noise is the performance adds additional complexity. Because of these estimates it is not possible to conclusively determine

if one design is better than other. A solution to this problem is to consider to designs different only if they are statistically different from each other. In addition, more simulation replications can give better estimates but at the expense of time.

2.1 Simulation Optimization for discrete systems

One of the disadvantages of simulation historically is that it was not an optimization technique. Relatively small number of system configurations was estimated and best out of them was chosen. But with the advent of fast processors and computing techniques, simulation is now widely used for optimization. There has been a great deal of work on simulation optimization in research literature, and more recently optimization routines have been incorporated into several commercial simulation packages.

Estimating the performance of system for every set of parameters using optimization under uncertainty is a very time consuming process. Various search methods can be found in literatures that have been applied to this class of problem. Andradóttir (1995, 1996) has developed two random search methods for discrete parameter simulation optimization. One of them is locally convergent while other is globally convergent. Gong, Ho, and Zhai (1992) have analyzed a method for discrete simulation optimization called the stochastic comparison method.

Simulated annealing has been proposed by Haddock and Mittenthal (1992). Many other heuristics designed for simulation optimization can be found in Andradóttir (1998). Other than that, tabu search, and genetic algorithm are widely used for this class of problems, which can be found in detail under section 2.3.

2.2 Problem Setting

A simulation optimization problem is like an optimization problem where the performance is an output of a simulation model instead of an analytical function. Different components of the problem can be defines as:

- decision variables
- objective function, and
- constraints

Here, decision variables are denoted by θ and the constraints are represented by these variables to be contained in some feasible region Θ, that is $\theta \in \Theta$. The objective function is a real valued function defined on these variables $f : \Theta \rightarrow \mathbf{R}$, but due to the stochastic nature and complexity of the system an analytical expression doesn't exist for $f(\cdot)$, and it must be estimated using the a function of a stochastic simulation output, say $L(\theta)$. Typically this might be an unbiased estimate of the true objective function, that is $f(\theta) = E[L(\theta)]$.

2.3 Metaheuristics

Metaheuristics are general solution strategies that can be applied to a particular class of problems. These are generally designed for solving complex optimization problems where other methods can't be used effectively or efficiently. The practical advantage of metaheuristics lies in both their effectiveness and general applicability. The metaheuristics approach to solve complex problems is to start with an initial solution or an initial set of solutions, and then applying an improving search guide by certain principles. Tabu search, genetic algorithms, and simulated annealing are some of the commonly used metaheuristics.

Although these methods are generally designed for combinatorial optimization in the deterministic context and may not have guaranteed convergence, they have been quite successful when applied to simulation optimization. However, there practical implementation in simulation optimization do not adequately account for the presence of simulation noise. Several simulation optimization packages that have been developed for commercial simulation software use above mentioned metaheuristics as their primary search methods. For example genetic algorithms are used in AutoStat and SimRunner, and tabu search, genetic algorithms, and scatter search are all used in Optquest.

The main concern of using these metaheuristics for simulation optimization is that they do not account for noise per se. These methods move from one solution to another better solution, or one set of solutions to another set of better solutions, and thus improving the solution quality until the method terminates. However, when simulation is used to estimate the performance, determining what constitutes a better solution becomes an issue. Here, solution or set of solutions is considered better only if the improvement is statistically significant.

2.3.1 Tabu Search

Tabu search was introduced by Glover (1989, 1990) to solve combinatorial optimization problems and it has been widely used effectively for simulation optimization. The main idea of this algorithm is to make moves from solution to solution and tabu certain moves, that is they cannot be visited as long as they are in the tabu list. Tabu list is dynamic, and after every move, the latest solution or the move that resulted in that solution is added to the list and the oldest solution or the move is removed from the list. Another characteristic of

tabu search is that the search always selects the best non-tabu solution from the neighborhood, even if it is worse than the current solution. This helps the search in breaking local optima, and the tabu list ensures that the search doesn't revert back. Although tabu search has been widely used in commercial packages for simulation optimization but it doesn't account for simulation noise.

2.3.2 Simulated Annealing

Simulated annealing, introduced by Kirkpatrick *etal.* (1983), is one of the oldest metaheuristics. Unlike tabu search, simulated annealing does not evaluate the entire neighborhood in every iteration. Instead, it randomly chooses only one solution from the current neighborhood and evaluates its costs. That means simulated annealing requires more iterations to find the best solution than tabu search. Another disadvantage is that it does not have memory, and hence it may revisit a recent solution. As a solution-to-solution search method, in each step it selects a candidate $\theta^c \in N(\theta_k)$ from the neighborhood of the current solution $\theta_k \in \Theta$. The definition of the neighborhood is determined by the user. If the candidate is better than the current solution it is accepted, but if it is worse it is not automatically rejected, but rather accepted with probability

$$\text{P [Accept}\,\theta^c] = e^{f(\theta_k)-f(\theta^c)/T_k}$$

where $f:\Theta \rightarrow \mathbf{R}$ is an objective function to be minimized, and T_k is a parameter called the temperature. The probability of acceptance is high is the performance difference is small and T_k is large. The key to simulated annealing is to specify a cooling schedule $\{T_k\}_{k=1}^{\infty}$, by which the temperature is reduced so that initially inferior solutions are selected with a high enough

probability so local optimal are escaped, but eventually it becomes small enough so that algorithm converges.

2.3.3 Genetic Algorithm

Genetic algorithm is set based or population based unlike tabu search and simulated annealing, which are solution-to-solution based algorithms. It is based on the idea of survival of the fittest and resembles an evolutionary process where two fit individuals are allowed to reproduce to generate offspring that resemble the parents (Goldberg 1989). In each step, a subset of the current set of solutions is selected based on their performance and these solutions are combined into new solutions. The operators used to create the new solution are survival, where a solution is carried to the next iteration without change, crossover, where the properties of two solutions are combined into one, and mutation, where a solution is modified slightly. The same process is then repeated with same set of solutions. Out of the above three steps, only the selection depends upon simulation performance.

Another random search metaheuristics is the Nested Partitions (NP) method (Shi and Ólafsson, 1997) which is explained in detail in the next chapter.

CHAPTER 3

Nested Partitions Method

Nested partitions method is one of the recently developed metaheuristics used for combinatorial optimization problems. Introduced by Shi and Ólafsson (2000a), it can be easily adapted to simulation optimization problems (Shi and Ólafsson (2000b). The key idea behind this method lies in systematically partitioning the feasible region into subregions, evaluating the potential of each region, and then focusing the computational effort to the most promising region. This process is carried out iteratively with each partition nested within the last. The computational effectiveness the NP method relies heavily on the partitioning, which if carried out in a manner such that good solutions are clustered together, can reach a near optimal solution very quickly.

3.1 NP Algorithm

NP method is used for solving optimization problems (Shi and Ólafsson, 2000a), which can be represented in the form

$$\min_{\theta \in \Theta} f(\theta), \tag{1}$$

where Θ is a finite feasible region. In simulation optimization context $f : \Theta \to \mathbf{R}$ is a performance function that is subject to noise. In other words, for any feasible point $\theta \in \Theta, f(\theta)$ cannot be evaluated analytically. Often $f(\theta)$ is an expectation of some random estimate of the performance of a complex stochastic system given a parameter θ, that is $f(\theta) = E[L(\theta)]$. Here $L(\theta)$ is a random variable which depends on the parameter $\theta \in \Theta$ and

is a discrete event simulation estimate of the performance. It is assumed that the feasible region is n-dimensional, that is, $\Theta \subset \mathbf{R}^n$ such that

$$\Theta = \{\theta = (\theta_1, \theta_2, ..., \theta_n) : \theta_i \in \Theta_i\}, \qquad (2)$$

where Θ_i is some finite set. Also, let $m(\theta_i) = |\Theta_i|$ be the number of values that the i-th dimensional variable can take.

In Ólafsson (1999) this method is further improved by drawing on ideas from statistical sampling techniques traditionally used for comparing few alternatives with a global optimization framework aimed at large-scale optimization. The main components of NP method are:

- **Partitioning**: at each iteration, the feasible region is partitioned into subsets that cover the feasible region but concentrate the search in what is believed to be the most promising region.
- **Random Sampling**: to evaluate each of the subsets, a random sample of solutions is obtained from each subset and used to estimate the performance of the region as whole.

This method can be understood as an optimization framework that uses partitioning to divide the design space into regions that can be analyzed individually and then aggregates the results from each region to determine how to continue the search, that is, how to concentrate the computational effort. In other words, the NP method adaptively samples from the entire space of possible feature subsets and concentrates the sampling effort by systematic partitioning of this space. The computational efficiency of the NP method relies

heavily on the partitioning, which, if carried out in a manner such that good solutions are clustered together, can reach a near optimal solution very quickly.

In the *k*-th iteration of the NP algorithm, a region $\sigma(k) \subseteq \Theta$ is considered most promising. It means that the algorithm assumes it to be the region that most likely contains the optimal solution, and thus the computation effort should be concentrated in this region. As in the beginning nothing is known about the location of the optimal solution, the algorithm is initiated with $\sigma(0) = \Theta$. The most promising region is then partitioned into $\boldsymbol{M}$ subsets or subregions and the entire surrounding region $\Theta \setminus \sigma(k)$ is aggregated into one. Thus in each iteration $\boldsymbol{M}$ + 1 disjoint subsets that cover the entire feasible region are considered. Each of these $\boldsymbol{M}$ + 1 regions is sampled using some random sampling scheme to obtain the sets of solutions, and then simulated performance function values at randomly selected points are used to estimate the promising index for each region. This index determines the most promising region in the next iteration by taking the subregion scoring highest on the promising index. If the surrounding region rather than subregion is found to have the highest promising index, the method backtracks to previous solution corresponding to the larger region. The new most promising region is partitioned and sampled in a similar manner. This generates a sequence of set of partitions, with each partition nested within the last, is referred as partition tree. The distance of the current promising region from the top of the tree, which corresponds to the minimum number of iterations it takes to move to this regions is called the depth of the region. Once the maximum depth is reached, that is the region that will not be partitioned further, the algorithm terminates. In the context of job-shop model, this maximum depth is equal to the number of workstations that need to be fixed.

Performance of each region has a noise because of simulation. Therefore, selecting the most promising region has noise and this region is selected with less confidence as compared to deterministic optimization context. But selecting most promising region in deterministic optimization itself has noise. Hence, there are two sources of randomness:

- There is a sampling error due to a small sample of solutions being used to estimate the overall performance of a subregion which often contains large set of feasible points.
- For each of the sample solutions, the performance is estimated using simulation, and is hence noisy.

The nested partitions method is not affected much by noise. The reason for this robustness is that this method includes a built-in mechanism for recovering from incorrect moves. Sometimes the simulation noise may cause the algorithm to move to the wrong subregion. However, as the search progresses the surrounding region continues to be sampled, which allows the algorithm to recover from incorrect moves through backtracking. It is shown in the result that irrespective of noise level statistically same solution is found.

3.2 Black Box Model for Simulation Optimization

One of the current difficulties in implementing many simulation optimization algorithms for practical problems appears to be that they can be quite complicated and require substantial knowledge on part of the user. To bridge this gap, Ólafsson and Kim (2001) present a framework for a black-box simulation-based optimization package that is intended for use by simulation practitioners. As the general problem of simulation-based optimization is extremely hard, the framework combines elements from a variety of methods

that have been found to be effective in this context, including random search, adaptive sampling, and ranking-and-selection. Implementation of NP method is quite problem dependent and, partitioning schemes in the past have drawn heavily on specific structure related to the application itself. This however, requires substantial effort on part of the practitioner using this method. Ólafsson and Kim, 2001 presents a new framework for automating these decisions,

The main components of the method proposed for the black-box implementation are:

- Intelligent partitioning
- Guided random sampling
- Guided local search

3.3 Partitioning in NP Method

Partitioning plays important role in simulation optimization using NP method. In these problems, besides finding a good solution, one also needs to keep in mind the number of simulation replications. For a complex problem, each replication is very time consuming, therefore it is imperative to direct the algorithm to move in the direction that requires fewer replications. It has been shown in the results that different partitioning schemes can affect number of simulation runs greatly. The amount of extra work done on partitioning schemes is very less as compared to the work done in finding optimal solution without considering any intelligent partitioning. Hence, this area requires considerable attention for the efficacy of NP method.

3.3.1 Importance of Partitioning Scheme

The selected partition imposes a structure on the feasible region. When the partitioning is done in such a way that good solutions are clustered together in the same subsets and then those subsets are selected by the algorithm with the little effort. On the other end of the spectrum, if the optimal solution is surrounded by solutions of poor quality it is unlikely that the algorithm will move quickly towards those subsets. This can be made more rigorous by looking at how the minimum probability P* of moving in the right direction can be guaranteed. The amount of computational effort is directly proportional to the variance of the solution in each region (Ólafsson and Kim, 2001). Therefore, it is advantageous to cluster together similar solutions, that is, the diversity of the solutions with respect to the simulation estimates of the objective function values, should be as small as possible.

Another concept that makes partitioning important is the overlap between the subsets. Assume $\Theta_g \subset \Theta$ contains an optimal solution whereas, $\Theta \setminus \Theta_g$ does not. Define

$$r(\Theta_g) = \frac{|\Theta_g| - |\theta \in \Theta_g : f(\theta) < \min_{\alpha \in \Theta \setminus \Theta_g} f(\alpha)|}{\Theta_g}.$$

Smaller the overlap between the 'good' set Θ_g and the 'poor' set of solutions $\Theta \setminus \Theta_g$, the larger the value of the above equation. Furthermore, it can be shown that the number $N(\Theta_g)$ of uniformly selected samples required to select the region Θ_g that contains the optimal solution correctly with probability at least ψ, grows exponentially with this overlap (Ólafsson and Shi, 2002):

$$N(\Theta_g) = \begin{cases} \left\lceil \dfrac{\log(1-\psi)}{\log(r(\Theta_g))} \right\rceil, & \mathrm{r}(\Theta_g) > 0, \\ 1 & \mathrm{r}(\Theta_g) = 0. \end{cases}$$

Therefore, from the above theoretical results, it is clear that the good partitioning scheme should have the following two properties: (i) partition such that variance between each region is minimized, and (ii) to partition such that overlap between the region that should be selected and the other regions is minimized.

Incorporating structure into the partitioning is critical for the efficiency of the method and the theoretical derivations that justify this can be found in Ólafsson (1999) and Ólafsson and Shi (2002). However, it is equally important to devise a methodology that can automate or semi-automate the process of partitioning. With this the practitioner does not need to understand every detail of the complex problem to form a partitioning scheme. Intelligent partitioning scheme has been devised based on the above mentioned characteristics (Ólafsson and Kim, 2001), which can be seen in the next section.

3.3.2 Intelligent Partitioning

To develop a generic intelligent partitioning scheme, idea of diversity known from information theory and its applications to areas such as machine learning (Mitchell 1997) is applied. Assuming that the solution of the optimization problem can be written as $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$, then the generic partitioning scheme will be to fix one of these n values at a time. Thus, for example Θ can be partitioned into $m(\theta_i)$ subsets defined by

$$\sigma_j = \{\theta \in \Theta : \theta_i = \theta_{ij}\},$$

where $\theta_{ij} \in \Theta_i$ for j=1,2,…,$m(\theta_i)$. The only decision to be made is the order in which these variables should be fixed first, and so forth. Each order of decision variables creates a different partition, implying *n!* different partitions. Among those, the best partition is the one that has the least overlap between the region containing the global optimum and regions not containing the global optimum, or alternatively minimizes the sum of the variance in each region. To use traditional diversity measures each solution is classified into fixed categories. It is done in such a way that solutions in each category are statistically same and for any two categories there is at least one solution in each such that those two have different statistical parameters. The boundaries between the categories are determined from the simulation error. Since, it is not possible to do this for every possible solution, therefore, a sample of solutions is considered.

Following is the detailed algorithm for intelligent partitioning as incorporated in NP method, introduced by Ólafsson and Kim (2001).

Step 1 Start by using random sampling to obtain a set of M_0 sample solutions.

Step2 Simulate the performance $L(\theta)$ of each one of these sample solutions, and record the average standard error s^2.

Step 3 Construct $g(s^2)$ intervals or categories for the sample solutions such that each solution in the sample interval has statistically equivalent performance, but for different intervals there are at least two solutions that make them distinguishable.

Step 4 Let $i = 1$.

Step 5 Fix $\theta_i = \theta_{ij}$, $j = 1,2,\ldots m(\theta_i)$.

Step 6 Drawing on the idea of entropy (Mitchell 1997) calculate the following quantity for each dimension θ_i of the feasible region:

$$E(\theta_i) = \sum_{j=1}^{m(\theta_i)} \frac{M(\theta_{ij})}{M_0} \cdot \sum_{l=1}^{g(s^2)} - p_{lj} \log_2 p_{lj}$$

where p_{lj} is the proportion of samples with $\theta_i = \theta_{ij}$ and fall in category *l,* and

$M(\theta_{ij})$ = Number of samples for which $\theta_i = \theta_{ij}$

Step 7 If $i = n$, continue to step 8; otherwise let $i = i + 1$ and go back to step 5.

Step 8 A high entropy value in the above equation indicates high diversity, so it is desirable to partition by fixing the lowest entropy dimensions first. Thus, we order the dimensions according to their entropy values

$$E(\theta_{[1]}) \leq E(\theta_{[2]}) \leq \ldots \leq E(\theta_{[n]}),$$

and let this order determine the *intelligent partitioning.*

The above defined method is quite generic and can be applied to any problem which can be formulated according to equation (2) above. In addition it also automates the process of identifying good partitions. The value of M_0 i.e. number of sample solutions used to build partitioning depends upon how much computation time should be used to assure the quality of the partitioning. But, it should be large enough so that entropies of each partition tend to stabilize. It has been shown in the results that if M_0 is not large enough, entropies have lot of noise and order of each partition is not fixed for different iterations of NP method.

3.3.3 NP algorithm with Intelligent Partitioning

The following algorithm is used for comparing intelligent partitioning scheme with other possible partitioning schemes:

Step 1 Apply algorithm *Intelligent Partitioning* to obtain a ranking:

$$E(\theta_{[1]}) \le E(\theta_{[2]}) \le \ldots \le E(\theta_{[n]}),$$

Step 2 Let k=0 and $\sigma(0) = \Theta$.

Step 3 Given the current most promising region $\sigma(k)$, partition $\sigma(k)$ into

$$M = m(\theta_{[d(\sigma(k))]})$$

subregions $\sigma_1(k),\ldots,\sigma_M(k)$ by fixing $\theta_{[d(\sigma(k))]}$, that is

$$\sigma_j(k) = \{\theta \in \sigma(k) : \theta_{[d(\sigma(k))]} = \theta_j\}$$

and aggregate the surrounding region

$$\sigma_{M+1}(k) = \Theta \setminus \sigma(k)$$

into one region (which can be empty).

Step 4 Use a uniform sampling procedure to select N_j points

$$\theta^{j1}, \theta^{j2}, \ldots, \theta^{jN_j},$$

from each of the regions $\sigma_j(k)$, j= 1, 2, …, M+1.

Step 5 Use discrete event simulation of the system to obtain a sample performance

$$L(\theta^{j1}), L(\theta^{j2}), \ldots, L(\theta^{jN_j}),$$

for each of the regions $\sigma_j(k)$, j= 1, 2, …, M+1.

Step 6 Calculate the estimated promising index $\hat{I}(\sigma_j)$ of each region,

$$\hat{I}(\sigma_j) = \min_{i \in \{1,2,\ldots N_j\}} L(\theta^{ji}), j = 1, 2, \ldots, M+1.$$

Step 7 Select the index of the region with the best promising index,

$$\hat{j}_k \in \arg \min_{j=1,2,\ldots M+1} \hat{I}(\sigma_j).$$

If more than one region is equally promising, the tie can be broken arbitrarily. If this index corresponds to a region that is a subregion of $\sigma(k)$, then let this be the most promising region in the next iteration. Otherwise if the index corresponds to the surrounding region, backtrack to larger region containing the most promising region. That is let

$$\sigma(k+1) = \begin{cases} \sigma_{\hat{i}_k}(k), & \text{if } \hat{i}_k < M+1 \\ s(\sigma(k)), & \text{otherwise.} \end{cases}$$

Step 8 If $d(\sigma(k)) = n,$ that is, all dimensions have been fixed then go to step 8; otherwise go back to step 3.

Step 9 If $\theta_{[1]}, \theta_{[2]}, \ldots, \theta_{[n]}$ can be arranged with a new permutation, go back to step 2; otherwise stop.

CHAPTER 4

Case Study: A Job Shop Model

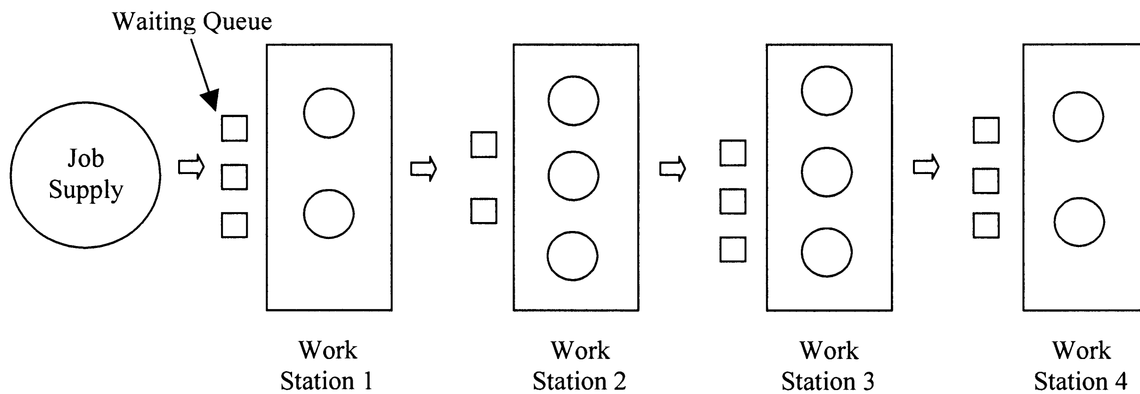


Figure 4.1: Manufacturing System Consisting of Four Work Stations

A job-shop model is considered to numerically evaluate the results of intelligent partitioning scheme as proposed by Ólafsson and Kim (2001). ANSI-standard version of C (Kernighan and Ritchie, 1988) is used to model and evaluated partitioning schemes. Part of the job-shop model code is taken from Law and Kelton (2000), which uses simlib, a C-based simulation library of inbuilt functions.

Table 4.1: Mean Processing Times of Machines in Each Work Station

Work Station	**Exponential mean processing time for a machine (in hours)**
1	0.333
2	0.8
3	0.55
4	0.15

Model Parameters: -

Exponential mean Inter-arrival time for jobs = 0.25 hours

Maximum number of machines in each work station = 8

Cost of each machine = $15,000

Profit of each job = $400

Length of the simulation = 240*8 hours (240 days assuming 8 hours shift)

Number of simulation runs during search phase = 2 (to add deliberate noise in the model)

Number of simulation runs for entropy calculation = 20

Number of sample samples used for entropy calculation (M_0) = 100

Number of sample points used to measure performance of each sub-region = 5

Number of sample points used to measure performance of the surrounding region = 30

4.1 Problem Definition

A simple job-shop model is considered as shown in figure 4.1 which consists of four workstations (WS), each having variable number of machines. The problem considered here is simple and can easily be solved using queuing theory, but the focus here is on the behavior of the performance based on different partitioning schemes. Jobs arrive with the above mentioned mean arrival time and are processed through each workstation. Each workstation has identical machines in parallel with a fixed mean service time. The objective is to find the optimal number of machines in each workstation that maximizes the total profit over a fixed simulation run. The objective function random variable is defined as follow:

$$f = (\$400 * \text{throughput}) - (\$25000 * \text{total number of machines})$$

For this example NP method is used to find optimal number of machines in each workstation using different partitioning scheme. The concept of entropy or information value is used to find intelligent partitioning scheme, as proposed by Ólafsson and Kim (2001). Results show that intelligent partitioning using order of entropy gives good results.

4.2 Entropy Calculations

From information theory's perspective, entropy is defined as the minimum number of bits of information needed to encode the classification of an arbitrary member of some region. To find the entropy of each workstation, 100 sample points are obtained using random sampling and their corresponding performances are estimated using simulation. Based on the average sample error, these points are divided into different intervals or categories. The maximum value of entropy depends upon the number of categories in which 100 sample points can be divided, such that, solutions (here, profit value) in each category are statistically identical, and for any two different categories, estimated performance of each sample point is statistically different from the estimated performance of each sample point of another category. The maximum possible value of entropy is $\log_2(c)$, where c is the number of categories in which sample points can be divided. To explain the concept of entropy, a simple data set is considered consisting of 4 workstations, with maximum 4 machines in each. Table 4.2 shows 20 random sample points, their performance, and how they are designated to different categories. This table shows performance for different model parameters.

$$E(\theta_i) = \sum_{j=1}^{m(\theta_i)} \frac{M(\theta_{ij})}{M_0} \cdot \sum_{l=1}^{g(s^2)} - p_{lj} \log_2 p_{lj}$$

$M_0 = 20$;

$j = 1, 2, 3,$ and 4;

$l = 1, 2,$ and 3;

For $i = 1$:

$p_{11} = 0$; $p_{21} = 7/7$; $p_{31} = 0$; $M(\theta_{11}) = 7$;

$p_{12} = 6/6$; $p_{22} = 0$; $p_{32} = 0$; $M(\theta_{12}) = 6$;

$p_{13} = 0$; $p_{23} = 0$; $p_{33} = 2/2$; $M(\theta_{13}) = 2$;

$p_{14} = 4/5$; $p_{24} = 0$; $p_{34} = 1/5$; $M(\theta_{14}) = 5$;

$E(\theta_1) = 0.1804$

Similarly, $E(\theta_2)$, $E(\theta_3)$, and $E(\theta_4)$ are calculated.

Notations: -

$E(\theta_i)$ - Entropy of workstation i

$m(\theta_i)$ - Maximum number of machine in workstation i

θ_{ij} - Number of machines in workstation i and in partition j

$m(\theta_i)$ - Number of partitions in i-th workstation

p_{lj} - Proportion of samples with $\theta_i = \theta_{ij}$ and fall in category l

$g(s^2)$ - Total categories or intervals

$M(\theta_{ij})$ - Number of samples for which $\theta_i = \theta_{ij}$

Table 4.2: Category division of sample points

Sample Point	**Number of Machines**				**Profit ($)**	**Category**
	WS 1	**WS 2**	**WS 3**	**WS 4**		
1	2	2	1	1	1,349,893.38	**I**
2	2	3	1	2	1,327,484.50	
3	2	3	2	1	1,316,422.25	
4	2	3	2	1	1,316,382.25	
5	2	3	3	1	1,291,746.63	
6	4	2	3	1	1,251,475.63	
7	4	2	1	4	1,241,244.38	
8	4	4	2	2	1,236,960.00	
9	2	3	4	3	1,233,142.25	
10	4	2	4	3	1,192,111.13	
11	1	2	1	2	998,559.94	**II**
12	1	4	1	2	953,453.31	
13	1	3	3	1	946,275.56	
14	1	3	2	3	926,040.06	
15	1	2	3	3	923,826.69	
16	1	4	1	4	900,266.69	
17	1	3	4	4	852,840.06	
18	3	1	1	1	617,320.00	**III**
19	3	1	4	3	493,591.13	
20	4	1	4	2	490,751.13	

Table 4.3 shows 7 categories in which 100 sample points are divided, having statistically different performance estimates with 99% confidence.

Table 4.3: Categories having sample points with statistically different performance estimates

Interval	Profit Range ($)
1	2879680 - 2688640
2	2688640 - 2499420
3	2499420 - 2110980
4	2110980 - 1751080
5	1751080 - 1266400
6	1266400 - 828159
7	828159 - 612259

Table 4.4 below, shows the average entropy of each workstation calculated using 100 sample points. Workstation 2 has the lowest entropy value, implying there is least diversity in solutions if number of machines in workstation 2 is fixed first. In other words, lowest information is needed to decide number of machines in workstation 2. Therefore, number of machines in workstation 2 is decided first, followed by workstation 3, workstation 1, and finally workstation 4.

Table 4.4: Average Entropy Values of Workstations

Work Station	Average Entropy
1	1.7862
2	1.1414
3	1.6228
4	2.0815

Number of sample points (M_0) for entropy calculation depends upon application considered. For this problem, Figure 4.2, Figure 4.3, and Figure 4.4 show how entropy changes with iterations of NP algorithm for different number of sample points. Entropies tend to stabilize as number of sample points increases. Hence, the order of entropies, also stabilize with number of sample points. Value of M_0 should be large enough, so that order of entropies doesn't change for different iterations of NP algorithm.

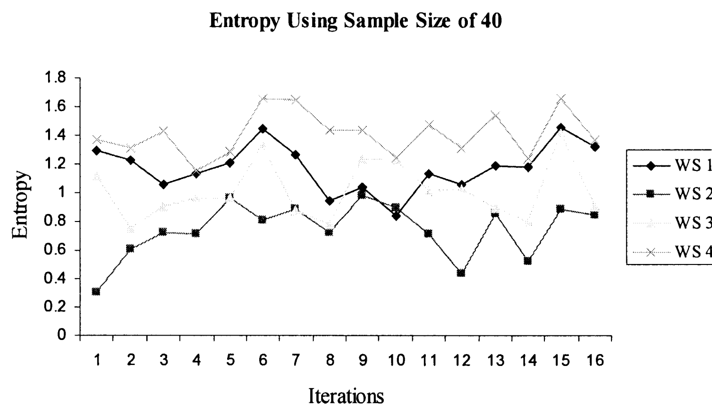


Figure 4.2: Entropy comparisons of workstations using sample size of 40

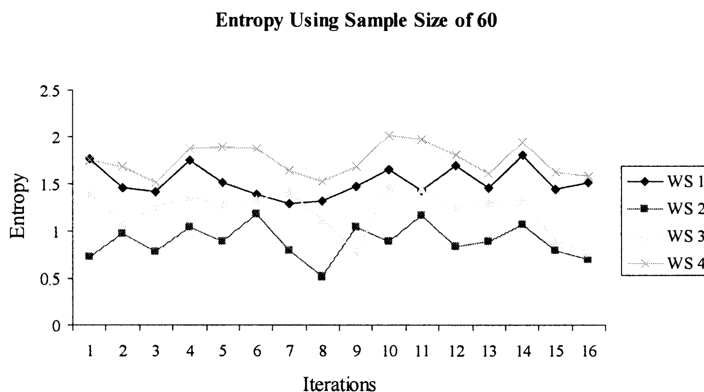


Figure 4.3: Entropy comparisons of workstations using sample size of 60

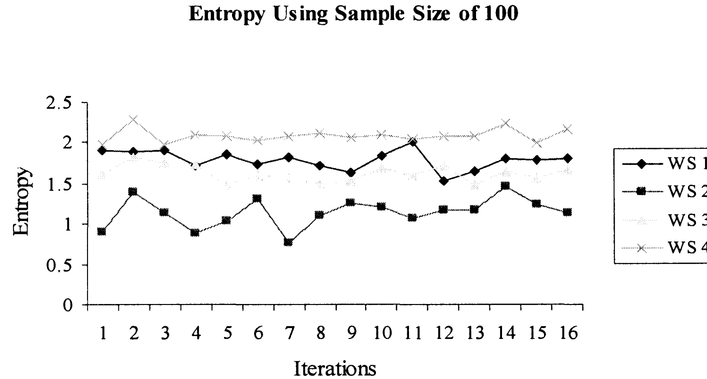


Figure 4.4: Entropy comparisons of workstations using sample size of 100

4.3 Intelligent Partitioning Scheme:

Table 4.5 shows average simulation runs for all 24 orders with which workstations can be fixed. These results show average values of 16 replications of NP method. Order of fixing number of machines in each work station according to intelligent partitioning is:

Work Station 2 → Work Station 3 → Work Station 1 → Work Station 4

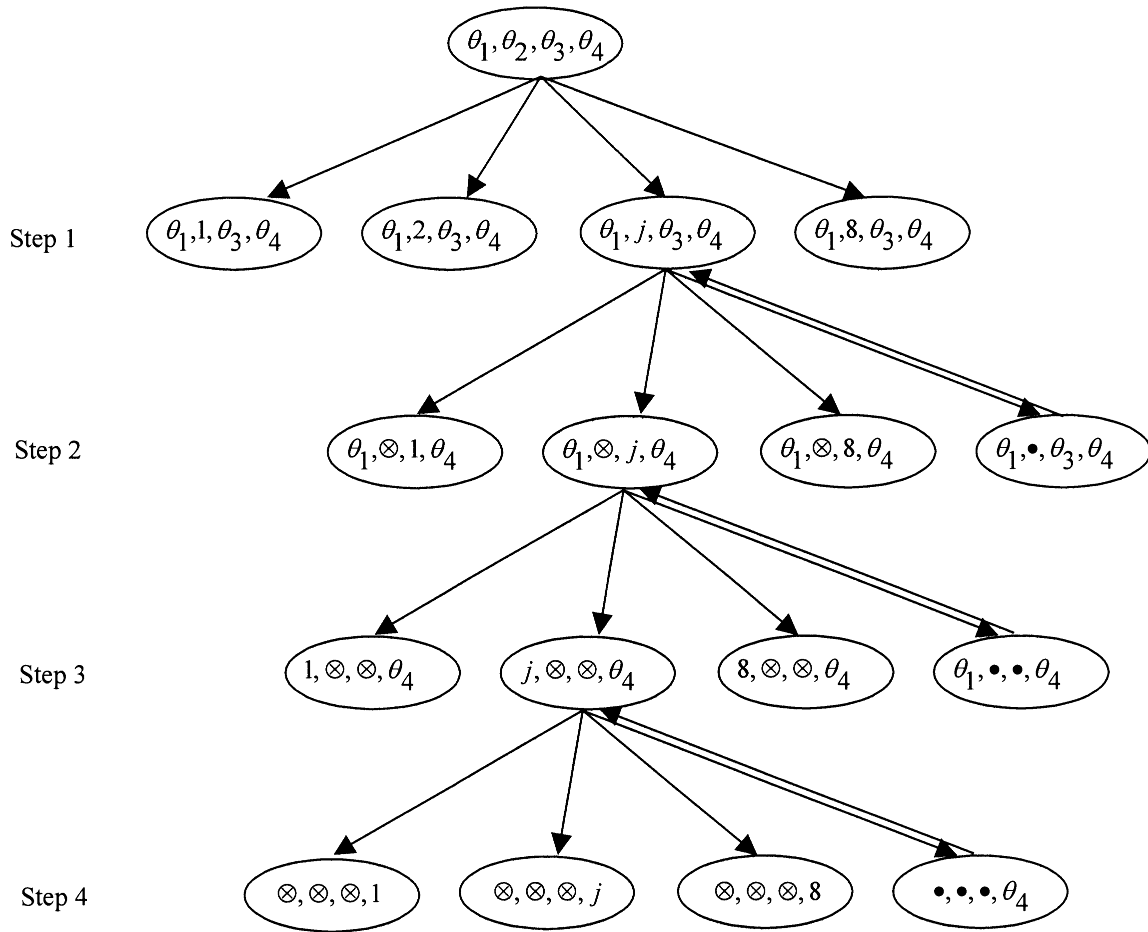
and the average simulation-model runs is 402. The best order is

Work Station 3 → Work Station 2 → Work Station 1 → Work Station 4

with 373 average number of simulation-model runs, and worst order is

Work Station 4 → Work Station 2 → Work Station 1 → Work Station 3.

Average number of simulation runs or steps taken by the NP algorithm to find "good" solution using partitioning scheme as directed by entropy is much better than the worst case partitioning. Figure 4.5 shows different steps of NP method to fix machines in workstation.



where,
$\otimes$ - Machines fixed
$\bullet$ - Complementary set of machines

Figure 4.5: Steps of Nested Partitions Method

Table 4.5: Performance of different partitioning schemes

Order Number	Order of Fixing Workstations				Profit ($)	Simulation Runs	
	WS1	WS2	WS3	WS4		Average	Standard Deviation
1	1	2	3	4	2888854	422	182.3
2	1	2	4	3	2877730	414	191.2
3	1	3	2	4	2892208	473	166.2
4	1	3	4	2	2884976	423	95.3
5	1	4	2	3	2879086	473	151.8
6	1	4	3	2	2889774	420	105.7
7	2	1	3	4	2883698	409	91.0
8	2	1	4	3	2890440	456	145.3
9	2	3	1	4	2891768	400	80.4
10	2	3	4	1	2875954	398	58.1
11	2	4	1	3	2890268	409	80.7
12	2	4	3	1	2884490	409	109.6
13	*3*	*1*	*2*	*4*	2880649	*402*	71.4
14	3	1	4	2	2866228	448	168.6
15	3	2	1	4	2871216	373	66.6
16	3	2	4	1	2881508	519	243.2
17	3	4	1	2	2872556	462	225.2
18	3	4	2	1	2881750	448	126.3
19	4	1	2	3	2883570	404	65.4
20	4	1	3	2	2862968	504	197.7
21	4	2	1	3	2882036	409	76.5
22	4	2	3	1	2891316	412	127.8
23	4	3	1	2	2871354	431	91.2
24	4	3	2	1	2889942	384	50.4

Mean and standard deviation of 'average simulation runs' for 24 cases is 429.25 and 36.55 respectively. Mean and standard deviation of profit is $2,881,847 and $8370.6 respectively. This implies, on the long-run, NP method with random partitioning scheme will take 429.25 simulation runs with profit of $2,881,847 where as, with intelligent partitioning

scheme it will take 402 simulation runs and estimated profit value is $2,880,649. 99% confidence interval of average simulation runs and profit is [424, 434] and [2880443, 2883252] respectively. Hence, the difference between average simulation runs is statistically significant whereas, difference between profits is statistically insignificant. It shows that performance measure of the NP method depends on the partitioning schemes only marginally. The possible reason for this is that the NP method has an in-built recovering mechanism to backtrack to the higher region from the present region. Moreover, the number of simulation runs can have significant differences using different partitioning schemes. Figure 4.6 shows the comparison of simulation runs of 3 partitioning schemes of the NP method for different replications. Results of the NP method using partitioning scheme determined by the entropy order for different replications are shown in Table 4.6.

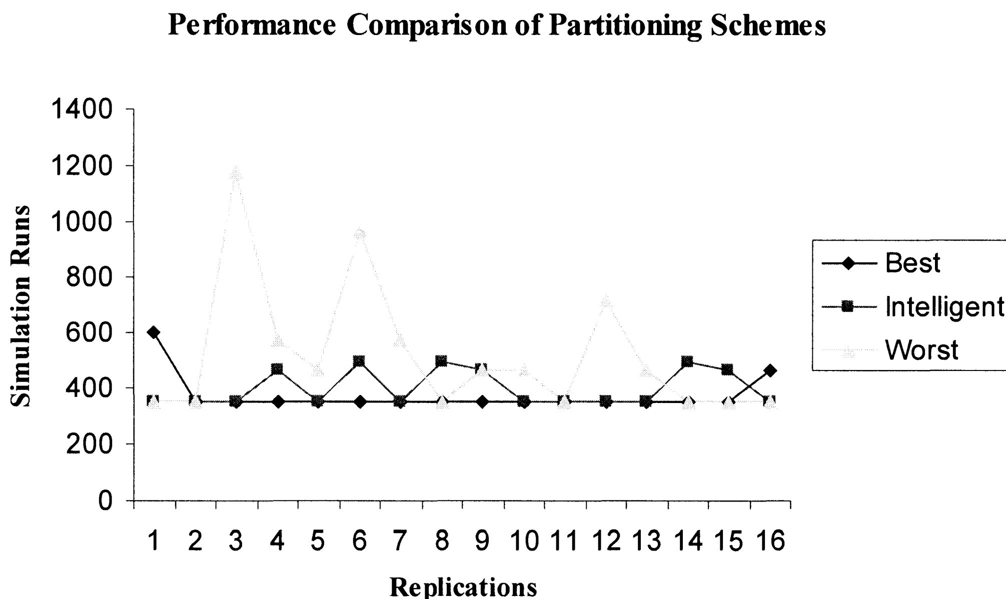


Figure 4.6: Performance comparison of partitioning schemes

It can be seen from the table 4.6 that for this partitioning scheme; 10 out of 16 times NP algorithm finds near-optimal solution without any backtracking. Using exhaustive simulation runs for good solutions, optimal solution was found to be:

Workstation 1 - 2 machines

Workstation 2 - 4 machines

Workstation 3 - 3 machines

Workstation 4 - 1 machine

Expected profit for the above configuration of machines is $2,912,140.

Table 4.6: Solution of NP Method Using Intelligent partitioning scheme

Replications	**Number of Machines**				**Profit ($)**	**Average Simulation Runs**
	WS1	**WS2**	**WS3**	**WS4**		
1	2	4	3	1	2912740	351
2	2	4	3	1	2909380	351
3	3	4	4	1	2893060	351
4	3	4	4	1	2904100	491
5	3	4	6	1	2862060	351
6	2	4	3	1	2912240	351
7	2	5	3	1	2898440	461
8	2	5	4	2	2877300	571
9	5	4	3	2	2862300	461
10	6	5	3	1	2832740	461
11	2	4	4	1	2916040	351
12	2	5	4	1	2882100	461
13	3	4	3	1	2907580	351
14	3	5	3	1	2896920	351
15	2	4	5	2	2861080	351
16	4	5	3	3	2850980	351

It is intuitive that low entropy implies low variability in the performance of the solution. Level of variability in the performance function for different orders of fixing workstations is shown in table 4.7. Assuming each workstation is fixed first; sample variance is calculated using idea of correlated sampling from Nelson-Matejcik (1995). The following equation is used to measure variability:

$$S^2(k) = K_0 \sum_{j=1}^{M} \sum_{i=1}^{n_0} \left(X_{ij}(k) - \overline{X}_{i\bullet}(k) - \overline{X}_{\bullet j}(k) + \overline{X}_{\bullet\bullet}(k) \right)^2$$

where,

$$K_0 = \frac{2}{(M-1)(n_0-1)},$$

$$\overline{X}_{i\bullet}(k) = \frac{1}{M} \sum_{j=1}^{M} X_{ij}(k),$$

$$\overline{X}_{\bullet j}(k) = \frac{1}{n_0} \sum_{i=1}^{n_0} X_{ij}(k),$$

$$\overline{X}_{\bullet\bullet}(k) = \frac{1}{M \cdot n_0} \sum_{j=1}^{M} \sum_{i=1}^{n_0} X_{ij}(k),$$

Notations: -

X_{ij} - Estimated performance of a sample point

M – Number of partitions

n_0 – Number of sample points

Table 4.7 shows the estimated sample variance of difference of sample means using 8 partitions (M), and 12 sample points from each partition (n_0).

Table 4.7: Sample variance in performance estimate of profit for different partitions

Workstation Fixed	Sample Variance
1	1,142,149,182,243
2	463,174,750,858
3	898,057,632,489
4	1,020,633,138,030

Differences in variance suggest that there is a need for two-stage sampling as discussed in Ólafsson (2004). Number of sample points needed to estimate the performance of each region is given by the following equation: -

$$N_j(k) = \max\left\{ n_0, \left\lceil \frac{g^2 S^2(k)}{\epsilon^2} \right\rceil \right\}$$

where,

N_j - Total number of sample points from region j

n_0 – Initial number of sample points

S^2 – Variance of the region estimated using n_0 sample points

ϵ - Indifference zone

g^2 – Constant depending upon n_0 and minimum probability of making correct selection

Therefore, max $\{0, (N_j - n_0)\}$ more sample points are needed from each region to estimate its performance.

CHAPTER 5

Conclusions and future work

In this thesis a new intelligent partitioning scheme for the Nested Partitions method is implemented on a manufacturing system. Importance of partitioning schemes is emphasized, particularly for simulation-based optimization problems. For the numerical evaluations, a job-shop model is optimized to test intelligent partitioning scheme.

Though intelligent partitioning shows good results, but it doesn't guarantee that algorithm will take fewer steps to find optimal solution. As results vary largely with different partitioning schemes, hence, if done appropriately, it can save huge amount of computational time. It is seen that the average number of simulation runs taken by the NP method using intelligent partitioning scheme is lower as compared to random partitioning.

More research is needed to identify partitioning schemes that can guarantee good solutions. Because of the fixed computing-budget, there is a trade-off between obtaining high quality estimates and allowing the search to explore the feasible region for more solutions. Research is needed in deciding the appropriate proportion of computing efforts to be spent for each task.

APPENDIX A – Code of NP Method with Intelligent Partitioning in C

```
#define EVENT_ARRIVAL           1 /* Event type for arrival of a job to the
                                     system. */
#define EVENT_DEPARTURE         2 /* Event type for departure of a job from
                                     a particular station. */
#define EVENT_END_SIMULATION  3 /* Event type for end of the simulation.*/
#define STREAM_INTERARRIVAL    1 /* Random-number stream for interarrivals.
                                     */
#define STREAM_JOB_TYPE         2 /* Random-number stream for job types. */
#define STREAM_SERVICE          3 /* Random-number stream for service times.
                                     */
#define STREAM_MACHINE          5 /* Random-number stream for number of
                                     machine. */
#define MAX_NUM_STATIONS        5 /* Maximum number of stations. */
#define MAX_NUM_JOB_TYPES       3 /* Maximum number of job types. */
#define NUM_RUNS                2 /* Number of simulation runs in sub
                                     regions. */
#define NUM_RUNS2              20 /* Number of simulation runs for entropy
                                     calculation. */
#define NUM_RUNS3              20 /* Number of simulation runs for most
                                     promising solution */
#define NUM_SAMPLES           100 /* Sample size to find number of
                                     partitions */
#define MAX_MACHINES            8 /* Upper Constraint on number of machines
                                     in each station */
#define N_points                5 /* Number of sample points to measure
                                     performance of each sub region*/
#define S_region               30 /* Number of samples from surrounding
                                     region*/

/* Declare non-simlib global variables. */
int   num_stations, num_job_types, i,j,l,k,p,
      num_machines[NUM_SAMPLES+10][MAX_NUM_STATIONS + 1],
```

```
        total_machines[NUM_SAMPLES+10],num_tasks[MAX_NUM_JOB_TYPES+1],
        intervals,count_runs=0, route[MAX_NUM_JOB_TYPES +1][MAX_NUM_STATIONS
        + 1],index[NUM_SAMPLES+10], rank[MAX_NUM_STATIONS+1], iteration,
        num_machines_busy[MAX_NUM_STATIONS + 1],job_type,
        task,num_machines_s[NUM_SAMPLES+10][MAX_NUM_STATIONS + 1],
        fix_num_machines[MAX_NUM_STATIONS + 2];

float mean_interarrival, length_simulation, prob_distrib_job_type[26],
        prob_distrib_sample[26], throughput[MAX_NUM_JOB_TYPES +1],
        profit[MAX_NUM_JOB_TYPES+1][NUM_SAMPLES+10], T_VALUE,
        mean_service[MAX_NUM_JOB_TYPES +1][ MAX_NUM_STATIONS + 1],
        profits[NUM_SAMPLES+10], sample_profit[MAX_NUM_JOB_TYPES+1],
        avg_thru[MAX_NUM_JOB_TYPES+1][NUM_SAMPLES+10], mean_sample_profit,
        performance_index[MAX_MACHINES+2], Entropy_s[MAX_NUM_STATIONS+1],
        Ent[MAX_NUM_STATIONS+1], Entropy[MAX_NUM_STATIONS+1];

FILE  *infile, *outfile;

/* Declare non-simlib functions. */

void     arrive(int new_job);
void     depart(void);
void     objective(int runs_o);
void     model(int runs_m);
void     sample(int sample_size);
void     sorting(int points);
double   stdev(float *pro);
void     group(void);
void     entropy(void);
void     search(void);
int      surrounding_region(int fix);
void     optimal(void);


#include "simlib.h"               /* Required for use of simlib.c. */
#include "myheader.h"
```

```
main()  /* Main function. */
{
        /* Open input and output files. */
       infile  = fopen("jobshop.in",  "r");
       outfile = fopen("jobshop.out", "w");

       for(iteration=1;iteration<=16;iteration++){

             /*clock starts on program execution*/
             clock_t start=0,finish=0;
             double total=0;
             start = clock();
             if(NUM_SAMPLES<=40)
                   T_VALUE=2.708;
             else if(NUM_SAMPLES>40)
                   T_VALUE=2.660;
             count_runs=0; /* Initialize simulation runs */

             /* Read input parameters. */
             fscanf(infile, "%d %d %f %f", &num_stations, &num_job_types,
                   &mean_interarrival, &length_simulation);
             for (i = 1; i <= num_job_types; ++i)
                   fscanf(infile, "%d", &num_tasks[i]);
             for (i = 1; i <= num_job_types; ++i) {
                   for (j = 1; j <= num_tasks[i]; ++j)
                         fscanf(infile, "%d", &route[i][j]);
                   for (j = 1; j <= num_tasks[i]; ++j)
                         fscanf(infile, "%f", &mean_service[i][j]);
             }
             for (i = 1; i <= num_job_types; ++i)
                   fscanf(infile, "%f", &prob_distrib_job_type[i]);

             /* Probability distribution of machines */
             for (i = 1; i <= MAX_MACHINES; ++i)
                   prob_distrib_sample[i]=(float)i/MAX_MACHINES;

             /* Write report heading and input parameters. */
```

```
fprintf(outfile, "Job-shop model\n\n");
 fprintf(outfile, "Number of work stations%21d\n\n",
 num_stations);
fprintf(outfile, "\n\nNumber of job types%25d\n\n",
num_job_types);
fprintf(outfile, "Number of tasks for each job type      ");
for (i = 1; i <= num_job_types; ++i)
      fprintf(outfile, "%5d", num_tasks[i]);
fprintf(outfile, "\n\nDistribution function of job types  ");
for (i = 1; i <= num_job_types; ++i)
      fprintf(outfile, "%8.3f", prob_distrib_job_type[i]);
fprintf(outfile, "\n\nMean interarrival time of jobs%14.2f
hours\n\n", mean_interarrival);
fprintf(outfile, "Length of the simulation%20.1f eight-hour
days\n\n\n", length_simulation);
fprintf(outfile, "Job type     Work stations on route");
for (i = 1; i <= num_job_types; ++i) {
      fprintf(outfile, "\n\n%4d        ", i);
      for (j = 1; j <= num_tasks[i]; ++j)
            fprintf(outfile, "%5d", route[i][j]);
}
fprintf(outfile, "\n\n\nJob type     ");
fprintf(outfile, "Mean service time (in hours) for successive
tasks");
for (i = 1; i <= num_job_types; ++i) {
      fprintf(outfile, "\n\n%4d    ", i);
      for (j = 1; j <= num_tasks[i]; ++j)
            fprintf(outfile, "%9.2f", mean_service[i][j]);
}
fprintf(outfile,"\n\nNumber of simulation runs(replications)
%10d\n",NUM_RUNS);
fprintf(outfile,"\nMaximum number of machines in each station
= %5d\n",MAX_MACHINES);
fprintf(outfile,"\nNumber of points from each region
(N_points) =%5d\n",N_points);
fprintf(outfile,"\nNumber of points from surrounding region
(S_region) =%5d\n",S_region);
```

```
            fprintf(outfile,"\nNumber of sample points for entropy
            calculation(NUM_SAMPLES) =%5d\n\n",NUM_SAMPLES);
            sample(NUM_SAMPLES);
            sorting(NUM_SAMPLES);
            group();
            entropy();
            search();
            optimal();
            fprintf(outfile,"\n\nToal simulations runs = %d", count_runs);

            /*clock stops at the end of program*/
            finish = clock();
            total = (double) (finish - start) / (double) CLOCKS_PER_SEC;
            fprintf(outfile,"\nTotal execution time = %f\n\n\n\n",total);
      }
      fclose(infile);
    fclose(outfile);
    return 0;
}

void model(int runs_m)
{
      /* Initialize throughput of each job to zero */
      for(i=1;i<=num_job_types;i++)
            throughput[i]=0;

      /*Run model for specified number of NUM_RUNS*/
      for(i=1;i<=runs_m;i++){

      /* Initialize all machines in all stations to the idle state. */
            for (j = 1; j <= num_stations; ++j)
                  num_machines_busy[j] = 0;

            /* Initialize simlib */
            init_simlib();

            /* Set maxatr = max(max number of attributes per record, 4) */
```

```
            maxatr = 4;  /* NEVER SET maxatr TO BE SMALLER THAN 4. */

            /* Schedule the arrival of the first job. */
            event_schedule(expon(mean_interarrival, STREAM_INTERARRIVAL),
                                 EVENT_ARRIVAL);

            /* Schedule the end of the simulation. */
            event_schedule(8 * length_simulation, EVENT_END_SIMULATION);

            /* Run the simulation until it terminates after an end-
            simulation event (type EVENT_END_SIMULATION) occurs. */
            do {
                  /* Determine the next event. */
                  timing();
                  /* Invoke the appropriate event function. */
                  switch (next_event_type) {
                        case EVENT_ARRIVAL:
                              arrive(1);
                              break;
                        case EVENT_DEPARTURE:
                              depart();
                              break;
                        case EVENT_END_SIMULATION:
                              break;
                  }

            /* If the event just executed was not the end-simulation event
            (type EVENT_END_SIMULATION), continue simulating.  Otherwise,
            end the simulation. */

            } while (next_event_type != EVENT_END_SIMULATION);
      }
}


void arrive(int new_job)
```

```
/* Function to serve as both an arrival event of a job to the system, as
well as the non-event of a job's arriving to a subsequent station along
its route. */
{
    int station;

    /* If this is a new arrival to the system, generate the time of the
    next arrival and determine the job type and task number of the
    arriving job. */
    if (new_job == 1) {
        event_schedule(sim_time + expon(mean_interarrival,
        STREAM_INTERARRIVAL), EVENT_ARRIVAL);
        job_type = random_integer(prob_distrib_job_type, STREAM_JOB_TYPE);
        task     = 1;
    }

    /* Determine the station from the route matrix. */
    station = route[job_type][task];

    /* Check to see whether all machines in this station are busy. */
    if (num_machines_busy[station] == num_machines[1][station]) {
      /* All machines in this station are busy, so place the arriving job
      at the end of the appropriate queue. Note that the following data
      are stored in the record for each job:
             1. Time of arrival to this station.
             2. Job type.
             3. Current task number. */
        transfer[1] = sim_time;
        transfer[2] = job_type;
        transfer[3] = task;
        list_file(LAST, station);
    }
    else {
            /* A machine in this station is idle, so start service on the
            arriving job (which has a delay of zero). */
        sampst(0.0, station);                      /* For station. */
        sampst(0.0, num_stations + job_type);     /* For job type. */
```

```
        ++num_machines_busy[station];
        timest((float) num_machines_busy[station], station);

        /* Schedule a service completion.  Note defining attributes beyond
      the first two for the event record before invoking event_schedule.*/
        transfer[3] = job_type;
        transfer[4] = task;
        event_schedule(sim_time + expon(mean_service[job_type][task],
        STREAM_SERVICE), EVENT_DEPARTURE);
    }
}

void depart(void)
/* Event function for departure of a job from a particular station. */
{
    int station, job_type_queue, task_queue;

    /* Determine the station from which the job is departing. */
    job_type = transfer[3];
    task     = transfer[4];
    station  = route[job_type][task];

    /* Check to see whether the queue for this station is empty. */
    if (list_size[station] == 0) {

        /* The queue for this station is empty, so make a machine in this
           station idle. */
        --num_machines_busy[station];
        timest((float) num_machines_busy[station], station);
    }
    else {

        /*The queue is nonempty, so start service on first job in queue.*/
        list_remove(FIRST, station);

        /* Tally this delay for this station. */
        sampst(sim_time - transfer[1], station);
```

```
        /* Tally this same delay for this job type. */
        job_type_queue = transfer[2];
        task_queue     = transfer[3];
        sampst(sim_time - transfer[1], num_stations + job_type_queue);

        /* Schedule end of service for this job at this station.  Note
        defining attributes beyond the first two for the event record
        before invoking event_schedule. */
        transfer[3] = job_type_queue;
        transfer[4] = task_queue;
        event_schedule(sim_time + expon(mean_service[job_type_queue]
        [task_queue], STREAM_SERVICE),EVENT_DEPARTURE);
}

    /* If the current departing job has one or more tasks yet to be done,
    send the job to the next station on its route. */
    if (task < num_tasks[job_type]){
        ++task;
        arrive(2);
      }
      else ++throughput[job_type];
}


void sample(int sample_size)
/* Estimate's the performance of a sub-region from sample points. */
{
      for(l=1;l<=sample_size;l++){
            total_machines[l]=0;
            for (j = 1; j <= num_stations; ++j){
                  num_machines[l][j] = random_integer(prob_distrib_sample,
                  STREAM_MACHINE);
                  total_machines[l]=total_machines[l] + um_machines[l][j];
            }
            model(NUM_RUNS2);
            count_runs++;
```

```
            objective(NUM_RUNS2);
        }
}

void sorting(int points)
/* Function to sort performance estimates in decreasing order */
{
        register int a,b;
        register float t;
        for(a=1;a<=points;++a)
                profits[a]=profit[1][a];
        for(a=1;a<=points;++a)
                for(b=points;b>a;--b){
                        if(profits[b-1]<profits[b]){
                                t=profits[b-1];
                                profits[b-1]=profits[b];
                                profits[b]=t;
                        }
                }
        for(a=1;a<=points;++a)
                for(b=1;b<=points;++b)
                        if(profits[a]==profit[1][b]){
                                for (j = 1; j <= num_stations; ++j)
                                        num_machines_s[a][j]=num_machines[b][j];
                                break;
                        }
}

double stdev(float *pro)
/* Function returns standard error of the performance to categorize sample
points */
{
        register float mean,sum1=0,sum2=0;
        for(i=1;i<=NUM_SAMPLES;i++)
                sum1+=pro[i];
        mean=sum1/NUM_SAMPLES;
        for(i=1;i<=NUM_SAMPLES;i++)
```

```
            sum2+=(pro[i]-mean)*(pro[i]-mean);
      return sqrt(sum2/NUM_SAMPLES-1);
}

void group(void)
{
      double t;
      k=1; index[k]=1;
      for(l=1;l<NUM_SAMPLES;l++){
            t=(profits[index[k]]-profits[l])/stdev(profits)*
            sqrt(NUM_SAMPLES);
            if(t>=T_VALUE){
                  k++; index[k]=l;
            }
      }
      intervals=k;
      index[k+1]=NUM_SAMPLES+1;
}

void entropy(void)
/* Function calculates entropies of each workstation */
{
      int a,b,s[NUM_SAMPLES][MAX_MACHINES],count,SS[MAX_MACHINES];
      float p[NUM_SAMPLES][MAX_MACHINES],M[MAX_NUM_STATIONS][MAX_MACHINES]
      ,I[MAX_NUM_STATIONS][MAX_MACHINES], t;
      for(i=1;i<=num_stations;i++){
            Entropy[i]=0;
            for(j=1;j<=MAX_MACHINES;j++){
                  k=1;count=0;
                  do{
                        for(l=index[k];l<index[k+1];l++){
                              if(num_machines_s[l][i]==j)
                                    count++;
                        }
                        s[k][j]=count;
                        k++; count=0;
                  } while(k<=intervals);
```

```
        }
        for(j=1;j<=MAX_MACHINES;j++){
                SS[j]=0;
                for(l=1;l<=intervals;l++)
                        SS[j]+=s[l][j];
        }
        for(j=1;j<=MAX_MACHINES;j++)
                for(l=1;l<=intervals;l++){
                        if(SS[j]==0){
                                p[l][j]=0;
                                continue;
                        }
                        p[l][j]=(float)s[l][j]/SS[j];
                }
        for(j=1;j<=MAX_MACHINES;j++){
                Ent[j]=0;I[i][j]=0;
                M[i][j]=(float)SS[j]/NUM_SAMPLES;
                for(l=1;l<=intervals;l++){
                        if(p[l][j]==0)
                                continue;
                        I[i][j]-=p[l][j]*log(p[l][j])/log(2);
                }
                Ent[j]=M[i][j]*I[i][j];
                Entropy[i]+=Ent[j];
        }
        Entropy_s[i]=Entropy[i];
}
for(a=1;a<=num_stations;++a){
        for(b=num_stations;b>=a;--b){
                if(Entropy_s[b-1]>Entropy_s[b]){
                        t=Entropy_s[b-1];
                        Entropy_s[b-1]=Entropy_s[b];
                        Entropy_s[b]=t;
                }
        }
        fprintf(outfile,"\nEntropy_s[%d] = %f",a,Entropy_s[a]);
}
```

```
	for(a=1;a<=num_stations;++a)
		for(b=num_stations;b>=1;--b){
			if(Entropy[a]==Entropy_s[b]){
				rank[a]=b;
				Entropy_s[b]=0;
				break;
			}
		}
	/* Assign ranks to entropies */
	for(a=1;a<=num_stations;++a)
		fprintf(outfile,"\nrank[%d]  = %d",a,rank[a]);
}

void search(void)
{
	int a,w=0,q=1,t,rnd,total_points,temp[MAX_MACHINES+2];
	float prob_distrib_surr[2];
	for(j=1;j<=2;j++)
		prob_distrib_surr[j]=(float)j/2;
	do{
		if(q==num_stations)
			total_points=5;
		else
			total_points=N_points;
		for(p=1;p<=MAX_MACHINES;p++){
			for(l=1;l<=total_points;l++){
				total_machines[l] = 0;
				for(j=1;j<=num_stations;j++){
					if(rank[j]==q){
						num_machines[l][j] = p;
						total_machines[l]=total_machines[l] +
						num_machines[l][j];
					}
					else if(rank[j]<q){
					num_machines[l][j]=fix_num_machines[j];
					total_machines[l]=total_machines[l] +
					num_machines[l][j];
```

```
                }
                else if(rank[j]>q){
                num_machines[l][j] = random_integer
                (prob_distrib_sample,STREAM_MACHINE);
                total_machines[l]=total_machines[l] +
                num_machines[l][j];
                }
            }
            model(NUM_RUNS);
            count_runs++;
            objective(NUM_RUNS);

        } /* End of total_points for loop */

        /* Finding performance index of each region*/
        for(a=1;a<=total_points;++a)
            profits[a]=profit[1][a];
        for(a=total_points;a>1;--a){
                if(profits[a-1]<profits[a])
                    profits[a-1]=profits[a];
            }
        temp[p]=p;
        performance_index[p]=profits[1];
        fprintf(outfile,"\nPerformance index of region[%d] =
        %f\n",p,performance_index[p]);
}                               //end of MAX_MACHINES loop
for(a=MAX_MACHINES;a>1;--a){
        if(performance_index[a-1]<performance_index[a]){
            performance_index[a-1]=performance_index[a];
            temp[a-1]=temp[a];
        }
}
fprintf(outfile,"\nMax. performance  = %f\n",
performance_index[1]

/* Searching surrounding region */
if(q>1){
```

```
for(l=1;l<=S_region;l++){
      do{
            t=0;
            total_machines[l]=0;
            for(j=1;j<=num_stations;j++){
                  if(rank[j]>=q)
                  num_machines[l][j] = random_integer
                  (prob_distrib_sample,STREAM_MACHINE);
            rnd=random_integer(prob_distrib_surr,4);
                  if(rank[j]<q && (rnd==1 || q==2)){
                  num_machines[l][j] =
                  surrounding_region(j)
                  }
                  else if(rank[j]<q && rnd==2 && q!=2){
                  num_machines[l][j] = random_integer
                  (prob_distrib_sample,STREAM_MACHINE);
                  t++;
                  }
                  total_machines[l] = total_machines[l]
                  + num_machines[l][j];
            }
      }while(t==q-1);
      model(NUM_RUNS);
      count_runs++;
      objective(NUM_RUNS);
}

/* Finding performance index of surrounding region*/
for(a=1;a<=S_region;++a)
      profits[a]=profit[1][a];
for(a=S_region;a>1;--a){
            if(profits[a-1]<profits[a])
                  profits[a-1]=profits[a];
      }
temp[MAX_MACHINES+1]=MAX_MACHINES+1;
performance_index[MAX_MACHINES+1]=profits[1];
```

```
                fprintf(outfile,"\nPerformance index of surrounding
                region = %f\n",
            }
            else
                performance_index[MAX_MACHINES+1]=0;
            if(performance_index[MAX_MACHINES+1]<performance_index[1]){
                for(j=1;j<=num_stations;j++)
                    if(q==rank[j]){
                        fix_num_machines[j]=temp[1];
                        fprintf(outfile,"Promising No. of machines
                        in WS %d = %d",j,fix_num_machines[j]);
                        break;
                    }
                q++;
            }
            else{
                q--;
                fprintf(outfile,"\n\n\n    BACKTRACK: q = %d\n\n",q);
            }
            w++;
            if(w>15 ){
                fprintf(outfile,"\nInfinite do loop");
                exit(0);
            }
        }while(q<=num_stations);      //q for loop ends
}

int surrounding_region(int fix)
/* Uniform sampling of points from surrounding region */
{
        int b=0; /* Initialize variables */
        do{
            b=random_integer(prob_distrib_sample,STREAM_MACHINE);
            }
        }while(b==fix_num_machines[fix]);
        return b;
}
```

```
void optimal(void)
/* Function estimates the performance of near optimal solution */
{
      total_machines[1]=0;
      fprintf(outfile,"\n\nMost promising number of machines :-\n");
      for (j = 1; j <= num_stations; ++j){
            num_machines[1][j]=fix_num_machines[j];
            fprintf(outfile, "%5d", fix_num_machines[j]);
            total_machines[1] = total_machines[1] + num_machines[1][j];
      }
      total_machines[1]);
      for(l=1;l<=1;l++){
            model(NUM_RUNS3);
            count_runs++;
            objective(NUM_RUNS3);
            fprintf(outfile,"Most promising Profit = %f", profit[1][1]);
      }
}

void objective(int runs_o)  /* Performance estimate function. */
{
      fprintf(outfile, "\nNumber of machines in each station[%d]",l);
      for (j = 1; j <= num_stations; ++j)
            fprintf(outfile, "%5d", num_machines[l][j]);
      for (i = 1; i <= num_job_types; ++i){
            avg_thru[i][l]=throughput[i]/runs_o;

            /* Objective function*/
            profit[i][l]=400*avg_thru[i][l]-15000*total_machines[l];
            fprintf(outfile,"\n\nAverage throughput of job type %d = %f",
            i, avg_thru[i][l]);
            fprintf(outfile,"\n\nAverage profit of job type %5d =   %f",
            i,profit[i][l]);
            sample_profit[i]+=profit[i][l];
      }
}
```

APPENDIX B – Program Output for One Iteration of the NP Method

Job-shop model

Number of work stations	4
Number of job types	1
Number of tasks for each job type	4
Distribution function of job types	1.000
Mean inter-arrival time of jobs	0.25 hours
Length of the simulation	240.0 eight-hour days

Job type   Work stations on route

1   1 2 3 4

Job type   Mean service time (in hours) for successive tasks

1   0.33   0.50   0.20   0.25

Number of simulation runs   3

Maximum number of machines in each station =   8

Number of points from each region (N_points) =   12

Entropy[1] = 0.704993

Entropy[2] = 0.923220

Entropy[3] = 0.500000

Entropy[4] = 0.500000

rank[1] = 2

rank[2] = 3

rank[3] = 4

rank[4] = 1

Performance index of region[1] = 1358266.750000

Performance index of region[2] = 1344266.750000

Performance index of region[3] = 1335400.000000

Performance index of region[4] = 1373200.000000

Performance index of region[5] = 1342266.750000

Performance index of region[6] = 1309666.750000

Performance index of region[7] = 1269000.000000

Performance index of region[8] = 1264466.750000

Max. Performance = 1373200.000000

Most promising number of machines in WS 4 = 4

Performance index of region[1] = 974800.000000

Performance index of region[2] = 1362933.250000

Performance index of region[3] = 1350066.750000

Performance index of region[4] = 1290333.250000

Performance index of region[5] = 1343266.750000

Performance index of region[6] = 1302333.250000

Performance index of region[7] = 1295800.000000

Performance index of region[8] = 1287666.750000

Max. Performance = 1362933.250000

Performance index of surrounding region = 1396733.250000

BACKTRACK: q = 1

Performance index of region[1] = 1363266.750000

Performance index of region[2] = 1361733.250000

Performance index of region[3] = 1323733.250000

Performance index of region[4] = 1321600.000000

Performance index of region[5] = 1260333.250000

Performance index of region[6] = 1303866.750000

Performance index of region[7] = 1293866.750000

Performance index of region[8] = 1249800.000000

Max. Performance = 1363266.750000

Most promising number of machines in WS 4 = 1

Performance index of region[1] = 1012066.750000

Performance index of region[2] = 1355733.250000

Performance index of region[3] = 1361933.250000

Performance index of region[4] = 1370600.000000

Performance index of region[5] = 1352533.250000

Performance index of region[6] = 1314400.000000

Performance index of region[7] = 1280800.000000

Performance index of region[8] = 1280400.000000

Max. Performance = 1370600.000000

Performance index of surrounding region = 1356333.250000

Most promising number of machines in WS 1 = 4

Performance index of region[1] = 671200.000000

Performance index of region[2] = 1386600.000000

Performance index of region[3] = 1375400.000000

Performance index of region[4] = 1376200.000000

Performance index of region[5] = 1335533.250000

Performance index of region[6] = 1344533.250000

Performance index of region[7] = 1328600.000000

Performance index of region[8] = 1311333.250000

Max. Performance = 1386600.000000

Performance index of surrounding region = 1393933.250000

BACKTRACK: q = 2

Performance index of region[1] = 1063133.250000

Performance index of region[2] = 1414600.000000

Performance index of region[3] = 1373200.000000

Performance index of region[4] = 1330866.750000

Performance index of region[5] = 1368066.750000

Performance index of region[6] = 1348466.750000

Performance index of region[7] = 1295666.750000

Performance index of region[8] = 1302666.750000

Max. Performance = 1414600.000000

Performance index of surrounding region = 1358133.250000

Most promising number of machines in WS 1 = 2

Performance index of region[1] = 689333.375000

Performance index of region[2] = 1417933.250000

Performance index of region[3] = 1407066.750000

Performance index of region[4] = 1414733.250000

Performance index of region[5] = 1373533.250000

Performance index of region[6] = 1360800.000000

Performance index of region[7] = 1353733.250000

Performance index of region[8] = 1333333.250000

Max. Performance = 1417933.250000

Performance index of surrounding region = 1331733.250000

Most promising number of machines in WS 2 = 2

Performance index of region[1] = 1415266.750000

Performance index of region[2] = 1403466.750000

Performance index of region[3] = 1389000.000000

Performance index of region[4] = 1389400.000000

Performance index of region[5] = 1350466.750000

Performance index of region[6] = 1341266.750000

Performance index of region[7] = 1332133.250000

Performance index of region[8] = 1319400.000000

Max. Performance = 1415266.750000

Performance index of surrounding region = 1354066.750000

Most promising number of machines in WS 3 = 1

Most promising number of machines in each workstation:-

2 2 1 1

Most promising Profit = 1410340.000000

Toal simulations runs = 865

Total execution time = 388.0 seconds

REFERENCES

Andradóttir, S. 1995. "A method for discrete stochastic optimization," *Management Science,* **41**, 1946-1961.

Andradóttir, S. 1996. "A global search method for discrete stochastic optimization," *SIAM J. Optimization,* **6**, 513-530.

Andradóttir, S. 1998. "A Simulation Optimization Techniques," in J. Banks (ed.), *Handbook of Simulation,* 303-333.

Glower, F. 1989. Tabu Search – Part I. *ORSA Journal on Computing* **1**: 190-206.

Glyn, P.W. 1989. "Optimization of Stochastic Systems Via Simulation," in *Proceedings of the Winter Simulation Conference,* 90-105.

Goldsman, D., and Nelson, B.L. 1994. "Ranking, selection, and multiple comparisons in computer simulation," in *Proceeding of the Winter Simulation Conference,* 192-199.

Goldsman, D., and Nelson, B.L. 1998. "Statistical screening, selection, and multiple comparison procedures in computer simulation," in *Proceeding of the Winter Simulation Conference,* 159-166.

Goldberg, D.E. 1989. "*Genetic Algorithm in Search, Optimization, and Machine Learning,* Addison-Wesley.

Gong, W.-B., Ho, Y.-C., and Zhai, W. 1992. "Stochastic comparison algorithm for discrete optimization with estimation" in *Proceedings of the 31st Conference on Decision and Control,* 795-800.

Haddock, J., and Mittenthal. J. 1992. "Simulation Optimization using simulated annealing," *Comput. Ind. Engng*, **22**, 387-389.

Kernighan, B.W., and Ritchie, D.M. 1988. *The C Programming Language*, Prentice Hall.

Kirkpatrick. S., Gelatt, C.D. Jr., and Vecchi, M.P. 1983. "Optimization by Simulated Annealing," *Science,* **220**, 671-680.

Law, A.M., and Kelton W.D. 2000. *Simulation Modeling and Analysis*, McGraw-Hill, Inc.

Mitchell, T.M. 1997. *Machine Learning,* McGraw-Hill, Inc.

Nelson, B.L., and Matejcik, F.J. 1995. "Using common random numbers for indifferences-zone selection and multiple comparisons in simulation," *Management Science*, **41**, 1935-1945.

Ólafsson, S. 1999. "Iterative ranking-and-selection for large-scale optimization," in *Proceedings of the Winter Simulation Conference,* 479-485.

Ólafsson, S., and Kim, J. 2001. "Towards a framework for black-box simulation optimization," in *Proceedings of the Winter Simulation Conference,* 300-306.

Ólafsson, S., and Shi, L. 2002. "Ordinal comparison via the nested partitions method", *Discrete Event Dynamic Systems: Theory and Applications*, **12**, 211-239.

Ólafsson, S. 2004. "Two-stage Nested Partitions Method for stochastic optimization", *Methodology and Computing in Applied Probability*, **6**, 5-27.

Robinson, H., and Monro, S. 1951. "A stochastic approximation method." *Annals of Mathematical Statistics,* **22**, 400-407.

Sanchez, S.M. 1997. "It is a far, far better mean I find...," in *Proceedings of the Winter Simulation Conference,* 31-38.

Shi, L., and Ólafsson, S. 1997. "An Integrated Framework for Deterministic and Stochastic Optimization," in *Proceedings of the Winter Simulation Conference,* 358-365.

Shi, L., and Ólafsson, S. 2000a. "Nested Partitions Method for Global Optimization," in *Proceedings of the Winter Simulation Conference,* 390-407.

Shi, L., and Ólafsson, S. 2000b. "Nested Partitions Method for Stochastic Optimization," *Methodology and Computing in Applied Probability,* **2**, 271-291.

Wen, M.J., and Chen, H.J. 1994. "Single-stage multiple comparison procedures under heteroscedasticity," *American Journal of Mathematical and Management Sciences,* **14**, 1-48.