Analysis of Batching Strategies for Multi-Item Production with Yield Uncertainty

by

Christopher Siow

B.S. Electrical Engineering and Computer Sciences
University of California at Berkeley, 2006

Submitted to the Sloan School of Management
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Operations Research
at the
Massachusetts Institute of Technology

February 2008

Author...
Operations Research Center
December 13, 2007

Certified by...
Stephen C. Graves
Abraham J. Siegel Professor of Management Science
Thesis Supervisor

Accepted by...
Cynthia Barnhart
Professor of Civil and Environmental Engineering, and Engineering Systems
Co-Director, Operations Research Center

Analysis of Batching Strategies for Multi-Item Production with Yield Uncertainty

by

Christopher Siow

Submitted to the Sloan School of Management
on December 13, 2007, in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

ABSTRACT

In this thesis, we investigate the batch sizing problem for a custom-job production facility. More specifically, given a production system that has been assigned several different types of custom jobs, we try to derive batching policies to minimize the expected total time that a job spends in the system. Custom-job production brings a host of challenges that makes batch sizing very difficult – production can only begin when the order arrives, the yield uncertainty probabilities are fairly large, and the production quantities are typically small. Furthermore, deriving an optimal batch sizing policy is difficult due to the heterogeneity of the job types; each job type has a different demand, batch setup time, unit production rate, unit defective probability, and job arrival rate. In addition, further complexity stems from the fact that the batch sizing decisions for each job type are coupled, and cannot be made independently.

Given the difficulties in selecting the batch sizes, we propose an alternative batching method that minimizes the system utilization instead of the expected total job time. The main advantage of this approach is that is allows us to choose the batch size of each job type individually. First, we model the system as an M/G/1 queue, and obtain a closed-form expression for the expected total job time when the demand is restricted to be a single unit. Following which, we show empirically that the minimum utilization heuristic attains near-optimal performance under the unit demand restriction. We then build on this analysis, and extend the heuristic to the general case in which the demand of each job is allowed to be more than a single unit. Finally, we use simulations to compare our heuristic against other alternative batching policies, and the results indicate that our heuristic is indeed an effective strategy.

Thesis Supervisor: Stephen C. Graves
Title: Abraham J. Siegel Professor of Management Science

(This Page Intentionally Left Blank)

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my thesis advisor, Professor Stephen Graves. Working with him on this project has been an extremely enriching experience for me, and I want to thank him for being so enthusiastic in sharing his wealth of knowledge. I really appreciate his invaluable insights when I was at the crossroads of my research, and for his guidance in keeping me on the right track. His advice was vital for the success of this project, and I am very fortunate to have had the opportunity to learn from him.

Thanks to my friends from the ORC, especially Mo, Clay, Bryan, and Charles, for their friendship and hospitality. It is really a pleasure and a privilege to have met every one them, and my experience at MIT would certainly not be the same without them.

All my love goes to my family, without whom I would not be who I am today. It gives me great comfort to know that I am constantly in their prayers. Their words of love and encouragement mean so much to me and give me the strength to persevere. Most of all, I want to thank them for taking pride in my achievements; it makes success so much sweeter.

Last but not least, to my dearest Jean. All these years, it has been her unconditional love and support that has kept me going. Thank you for being a part of my life and walking this journey with me.

(This Page Intentionally Left Blank)

Contents

(This Page Intentionally Left Blank)

List of Figures

(This Page Intentionally Left Blank)

List of Tables

(This Page Intentionally Left Blank)

Chapter 1

Introduction

1.1 Thesis Motivation

The issue of lot sizing is one that can be found in a myriad of industries, and the problem of determining optimal batching policies generates much interest because it affects throughput, production costs, and revenue. Consequently, a variety of lot sizing problems has received widespread attention throughout the years. In this thesis, we consider one such variant. While our analysis was originally aimed at microfabrication facilities that manufacture custom-designed integrated circuit or Micro-Electrical-Mechanical-Systems (MEMS) components, there certainly are other types of production facilities that fit our problem description, and the analysis and results that are presented in the following chapters will still be relevant.

The context is that of a custom-job production facility that receives job orders from customers, and has to deliver the order by a set deadline. The jobs that these facilities undertake are usually custom jobs that are subcontracted from other companies. Frequently, the job order quantity is small, and the product required must be custom-made to the customer requirements. As with almost all custom production facilities, yield is never perfect, and there will be a certain fraction of defective units per production run. The problem that the facility faces is thus to decide on the batch size to produce for each job, so as to meet the deliverable deadline. In many aspects, the problem that this thesis addresses is very different from other batch sizing problems in the following ways:

- The fact that the each job is unique emphasizes the time-sensitivity in delivering the order. Since each custom MEMS product uses a custom blueprint, the number of process steps, the types of process steps required, and even the length of each process step will vary from product to product. This means that production can only begin when a job actually arrives at the facility (make-to-order). In contrast, mass production facilities for companies such as Dell are able to fabricate components beforehand, due in part to the scale of the business, and also the fact that these companies operate in an assemble-to-order or make-to-stock environment.
- The order quantities for custom-job production are usually small, and frequently in the range of between one and ten units. When the order quantity is small, yield uncertainty has a very large effect on production planning and batch sizing because the actual number of good units produced is very unpredictable. This is evident if we consider a factory that produces common household products such as diapers. The production quantities per order are usually very large, and on the order of several thousands. As a result, even with yield uncertainty, the actual realized yield is fairly predictable due to the law of large numbers.
- The yield uncertainty probabilities that we are dealing with can be fairly large. This is simply a consequence of the custom fabrication of hi-tech components. In the microfabrication industry, the yield is typically much lower than compared to other industries due to the sensitive and difficult processing steps required. In addition, since jobs are unique, there is no fixed production template to follow, and this further drives down the yield.

The significance of all these differences is that these custom-job microfabrication facilities are faced with the difficulty of managing a production process that has a highly variable yield to satisfy an order within a tight deadline.

1.2 Thesis Objective

This thesis aims to develop a methodology for batch sizing to allow the before-mentioned production facilities to complete a job order as quickly as possible so as to meet the deadline. Apart from meeting the job deadlines, this would also allow the facility to accept more jobs.

Instead of focusing on the entire production facility, we consider a single machine in the facility. For instance, this single machine might be the bottleneck for the production system. Alternatively, we could think of the single machine as a proxy for the entire production line. Given a certain set of jobs that are assigned to the machine, we attempt to derive an optimal batch sizing policy. We could evaluate the efficacy of a batch sizing policy can be evaluated by several different standards; however, for this research the criterion that we use to compare the performance of various policies is the expected total time that a job would spend in the system.

In simpler cases where the algebra is tractable, we propose an algorithm to find the optimal policy under such conditions. In more difficult cases, we offer a batch sizing heuristic that minimizes the utilization of the machine. When compared to the true optimal policies (in the cases when it is possible to derive them), and other alternative strategies, we show that our heuristic performs very well.

1.3 Literature Review

The earliest papers written on batch sizing under yield uncertainty date back to the late 1950s. In all these papers, the aim is to derive the reject allowance – the extra quantity processed to allow for possible defective units. Since then, the papers that are relevant to this thesis can be broadly classified into three categories.

The first category consists of papers that are concerned with batch sizing decisions under yield uncertainty for single product systems, and when only a single production run is permitted. By considering the costs associated with overage and underage production, Giffler [5] formulated the lot sizing problem as one of finding the optimal reject allowance to minimize the expected costs. In essence, this was a marginal analysis in which the reject allowance was selected to provide the optimal tradeoff between overproduction, and failing to meet the demand. Shortly after, Levitan [12] provided mathematical support for Giffler's results and gave sufficient conditions for the algorithm to achieve optimality.

Although almost all the papers we reviewed used the minimization of costs (or equivalently the maximization of savings) as their objectives in determining the batch sizes, one can easily see how these analyses can be used when the objective is to minimize the expected total time. In many references such as Bowman [1], Llewellyn [14], and White [16], the cost associated with a batch is typically a fixed setup cost, as well as variable production costs proportional to the batch size. This is entirely analogous to the case when the time associated with producing a batch comprises a fixed setup time and a variable time component. Therefore, the "cost" in these cases, is in terms of units of time, rather than dollars. In this thesis, since we are primarily concerned with meeting a deadline, minimizing a time objective is the approach we have adopted.

Like the first category that we have just discussed, the second category also consists of papers that also analyze single-product systems with yield uncertainty. However, the main difference is that multiple production runs are allowed. In 1957, Bowman and Fetter [2] presented a heuristic to determine the lot size for a custom order. The analysis performed is very similar to Giffler [5], and the same tradeoff model was used, only that multiple runs are allowed.

Llewellyn [14] then used normal and Poisson approximations to the binomial yield distribution to calculate reject allowances to ensure the satisfaction of the demand within a specified probability. These approximations were made under the context that demand was large (thus necessitating the use of approximations), and for a single production run. He also considered models that allowed additional runs, but non-zero costs were only assigned to the first few runs. Through use of an example, he concludes that the optimal decision under these models does not change, so the effects from the second run onwards (second-order effects) can be ignored. In this thesis, as in Goode and Saltzman [6], we do not make the assumption that second-order effects can indeed be ignored, and associate appropriate costs with every production run that is performed.

The analysis made by White [16] in 1965 is perhaps one of the closest to the problem addressed by this thesis. The objective is to minimize the total expected setup and variable production costs, and White presents a dynamic programming formulation of his solution. However, it is not clear how this analysis could be extended to the multi-product case, since batch sizing decisions among heterogeneous items will affect each other. This work was further extended by Delfausse and Saltzman [4], who incorporated salvage costs into the objective function.

More recently, Karmarkar [8] presented a method for obtaining the optimal batch size for single-product systems with no yield uncertainty. He models the machine as an M/D/1 queue, and invokes first order optimality conditions to derive the best batch size. More interestingly, unlike the earlier papers that dealt with a cost objective, Karmarkar focused on minimizing the average time a job spends in a system.

Given the extensive literature on single-product production systems with yield uncertainty, the last category of papers addresses the batch sizing problem for multi-product systems when multiple production runs are allowed. While these papers seem to provide a solution to our problem, we note that those we reviewed did not consider yield uncertainty. Also, there are not many papers within this category. In the review of lot sizing literature done by Yano and Lee [17], they state that relatively little work has been done on multi-product systems. This is perhaps due to the difficulty involved in such analysis, since there is a great interdependence in lot sizing decisions due to the heterogeneity of the items (Karmarkar [8]).

There have been several papers that discuss the importance of minimizing delays and queueing time in order to reduce production time. Some reiterate the fact that a large percentage of lead times is due to queueing delays (Jönsson and Silver [7]; Stalk [15]). Others, like Buzacott and Shanthikumar [3], review several models for studying queueing delays in production facilities. Nonetheless, these papers are concerned more with the queueing delay per se, and the relationship between batching decisions and these delays is not explored in depth.

With regard to the effect of batching strategies on delays, two papers offer an extension to the work done by Karmarkar[8]. Karmarkar, Kekre and Kekre [9] extended the analysis to multi-product systems with no yield uncertainty. Similarly, Kuik and Tielemans [10] also propose a numerical method to solve for batch sizes that minimizes the expected waiting time of a job in the queue, rather than its expected total time in the system. However, as we have stated, these papers do not consider the effect of yield uncertainty in the batch sizing strategies.

To summarize, our work differs from previous literature in the fact that we are trying to combine the dual problems of batch sizing for multi-product systems and batch sizing under yield uncertainty; furthermore, we do account for queueing effects, as they depend on the choice of batch size. Through this thesis, we provide an analysis of these problems, and propose some batch sizing policies.

1.4 Thesis Outline

The following chapter presents an analysis of the operation of the machine, as well as a derivation of the expected total time formula using queueing theory. In Chapter 3, we analyze the simplest possible scenario in which all the jobs have the same job parameters (a single job type), and the demand for each job is a single unit. Under these assumptions, we are able to derive an optimal batching policy. We then relax the job homogeneity restriction in Chapter 4, and allowed the machine to handle different types of jobs; we still assume that the demand for each job is a single unit. Here, we introduce our batch sizing heuristic, and highlight its performance through a series of test cases. Next, we present a dynamic programming implementation of our batch sizing heuristic in Chapter

5, which can be used to deal with the cases when demand is not restricted to a single unit. Finally, we conclude with Chapter 6, and put forth some ideas for future work.

(This Page Intentionally Left Blank)

Chapter 2

Analysis of the Multi-Product Production Process

To compare batching strategies, we will use as the criterion the expected total time that a job would spend in the system under a certain policy. That is, the criterion is the expected amount of time from the arrival instance of the job to the time when the system completely satisfies the job demand. In this chapter, we first describe the queueing model used to represent the production system. With the framework in place, we then obtain an expression for the expected total time, which we will then seek to minimize through the choice of batching policy in the following chapters.

2.1 Modeling of the Machine as a Queue with Feedback

Given the problem description, where a multi-product production system has to satisfy demand from jobs of different types, it is natural to analyze the batching problem from a queueing theory standpoint. We assume that the machine can process J kinds of different products. As shown in Figure 2.1, job requests are submitted to the machine, where they enter a single queue and await service. We assume that the the job arrival process is Markovian in nature, which is not an unreasonable assumption if the arrivals are due to many overlapping and uncoordinated product flows (Karmarkar [8]). The queue is cleared in a first-come-first-serve discipline; a job request waits in the queue until all the job requests that arrived before it have received service. Each job request of type j is a demand for a certain number of items of that particular product. To satisfy this demand, the machine will produce a batch of the product.

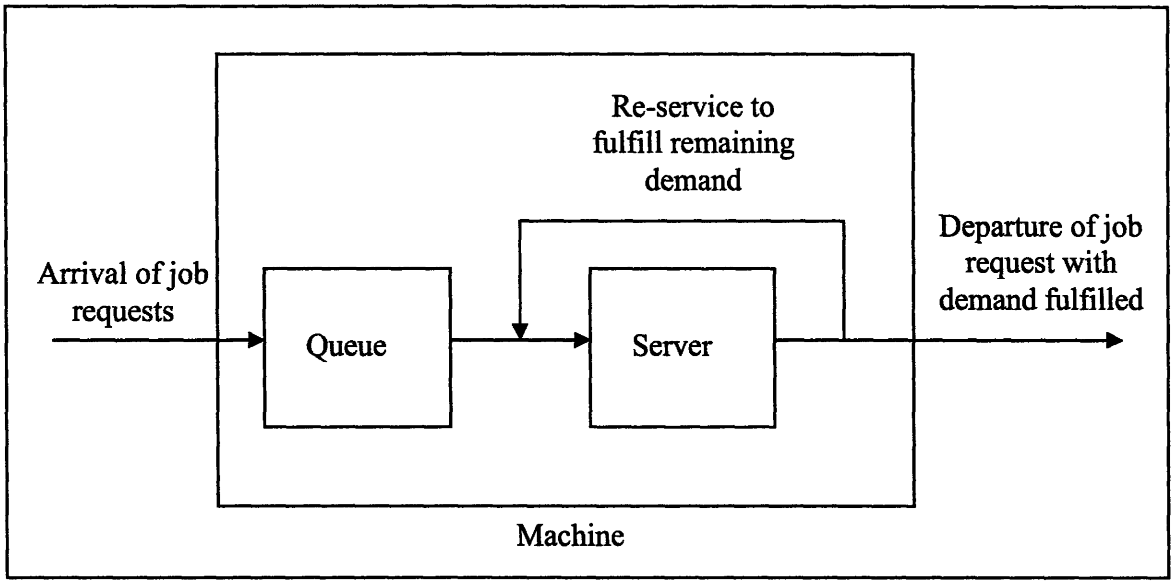
Re-service to fulfill remaining demand
Arrival of job requests
Queue
Server
Departure of job request with demand fulfilled
Machine

Figure 2.1: Representation of the Production System as a Queue with Feedback

However, due to yield uncertainty, the demand of the job request may not be satisfied by a single batch even if the batch size is larger than the number of good units required. We model the random yield for each unit in the batch as a Bernoulli process. In other words, the number of good units that is obtained from a batch of size n is a Binomial random variable with parameters n and p, where p represents the probability that a good unit is produced.

In the event that demand is not satisfied, then the machine server continues producing batches to satisfy the remaining demand of the current job request, before proceeding to the next job request waiting in the queue. We assume that the machine can process at most one batch at a time, and that the service time for the batch depends on the type of job, and the size of the batch. For example, suppose a job arrives for which the initial demand is 5 units, and the machine starts with a batch of size 8. If this initial batch were to yield only 3 non-defective units, then the machine has to process a new batch to satisfy the remaining 2 units of demand. The new batch need not have the same size as the initial batch; for instance, we might decide to set the second batch to size 6 and hope that this will yield at least 2 good units. This process continues until the demand for 5 units is completely satisfied. The decisions that we are concerned with, are thus the batch sizes to use when faced with different job types that require different demand amounts.

As described above, we assume that we always insert a job request with unfulfilled demand to the front of the queue for "re-service" and continue to do so until all of the demand for a job is satisfied. Under the assumption of job homogeneity, the expected total time that a job spends in the system is not affected by the re-service discipline that is adopted. In other words, if the machine handles only a single type of job, the position at which we insert a "failed" job into the queue for re-service does not affect the expected total job time. This is because the expected queue length does not change under various re-service disciplines. Therefore by Little's Law [13], the expected total job time remains invariant. However, when the machine is allowed to handle different types of jobs, then the re-service discipline does affect the expected total job time. To keep the analysis simple and tractable, we will henceforth make the assumption that a job that requires re-service is inserted to the front of the queue until its demand is completely satisfied.

2.2 Expected Total Time for Single-Item M/M/1 Queue

The total amount of time, T, that a job spends in the system can be decomposed into two parts – the amount of time that the job spends waiting in the queue, W, and its total service time, S. We emphasize that it is important to distinguish between the total service time S and the time required for a single service through the server X; because there is yield uncertainty, the total service time will be the sum of a random number of service times, corresponding to the number of batches that are needed before the job requirements are met.

$$\begin{aligned} T &= W + S \\ E[T] &= E[W] + E[S] \qquad (2.1) \end{aligned}$$

Before analyzing multi-item queues with the complexity of general service times, we first concentrate on the simple case where there is only a single job type, and the single service time, X, follows an exponential distribution with a mean of $\bar{x}$. For ease of notation, we will let the service rate, $\bar{x}^{-1} = \mu$. Also, we let γ represent the probability that a job has to undergo re-service, and thus rejoins the queue.

Although the exogenous job arrival rate to the queue is Poisson, the input rate cannot be assumed to be Poisson due to the feedback from the server. However, we can conceptualize each server as a machine without feedback, with the rate at which a job is completed given as $\mu(1-\gamma)$. The factor of $(1-\gamma)$ is due to the fact that the machine processes jobs at a slower rate than the actual job service rate μ, because of the probability of re-service. This is illustrated in Figure 2.2, in which state l represents the event that there are l jobs already in the machine (waiting in the queue and currently being serviced).

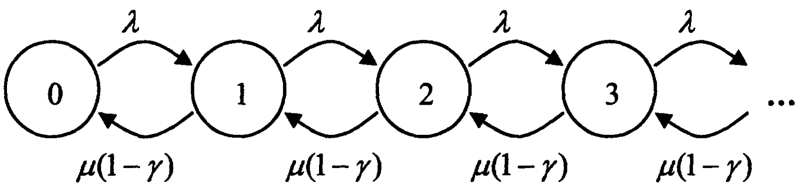

Figure 2.2: State Transition Diagram of the Steady-State M/M/1 Queue with Feedback

To obtain an expression for the expected waiting time, $E[W]$, we have to find the steady-state probability of every state l. This is done by writing the steady-state balance equations:

$$\pi_l \lambda = \pi_{l+1} \mu(1-\gamma) \qquad l = 0,1,2,... \tag{2.2}$$

Together with the fact that the steady-state probabilities must sum to 1; $\sum_{l=0}^{\infty} \pi_l = 1$, some recursive substitution and algebra yields:

$$\pi_l = \left(1 - \frac{\lambda}{\mu(1-\gamma)}\right)\left(\frac{\lambda}{\mu(1-\gamma)}\right)^l \qquad l = 0,1,2,... \tag{2.3}$$

Suppose a new job request arrives, joins the queue, and there are L prior jobs ahead of it in the queue. We can model the number of job requests that have to receive service before the new job as a Markov chain shown in Figure 2.3.

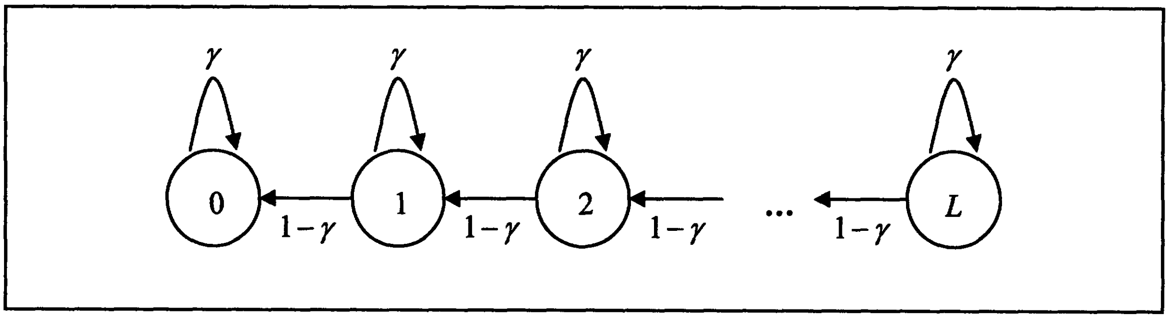

Figure 2.3: Markov Chain Representing the Number of Prior Job Requests

Assuming that there are L jobs that will receive service before the newly-arrived job request, then the expected waiting time until the new job receives service is then the expected time to absorption from state L to state 0. If we let w_l represent the expected waiting time given that we are starting from state l, we can write a series of recursive formulas:

$$w_0 = 0$$

$$w_l = \frac{1}{\mu}\left(1 + \gamma w_l + (1-\gamma) w_{l-1}\right) \qquad l = 1,2,3,...,L$$

By recursive substitution, we can write a closed-form expression for w_l

$$w_l = \frac{1}{\mu}\left(\frac{l}{1-\gamma}\right) \tag{2.4}$$

Combining Equations 2.3 and 2.4, we can then calculate the expected waiting time

$$E[W] = \sum_{l=0}^{\infty} \pi_l w_l$$

$$E[W] = \sum_{l=0}^{\infty}\left(1-\frac{\lambda}{\mu(1-\gamma)}\right)\left(\frac{\lambda}{\mu(1-\gamma)}\right)^l \frac{1}{\mu}\left(\frac{l}{1-\gamma}\right)$$

$$E[W] = \frac{\lambda}{\mu(1-\gamma)(\mu(1-\gamma)-\lambda)} \tag{2.5}$$

We note that due to the assumption of exponential service times, the expected service time of the job is equivalent to the expected waiting time if there is only one prior job; $E[S] = \frac{1}{\mu}\left(\frac{1}{1-\gamma}\right)$. Therefore, summing the components of Equation 2.1, we can obtain the expected total time as

$$E[T] = \frac{\lambda}{\mu(1-\gamma)(\mu(1-\gamma)-\lambda)} + \frac{1}{\mu(1-\gamma)}$$

$$E[T] = \frac{1}{(\mu(1-\gamma)-\lambda)} = \frac{\bar{x}}{((1-\gamma)-\lambda\bar{x})} \tag{2.6}$$

2.3 Expected Total Time for Single-Item M/G/1 Queue

Although using the M/M/1 queue as a model for our production system yields straightforward formulas, the assumption of exponential service times is restrictive since we would like to be able to analyze systems with more general service distributions that depend on the job type and batch size. Hence, we make the first relaxation of the model and consider single-item production systems with general service times, and proceed with an analysis along the same lines as Equation 2.1. We still assume that the job arrival process if Markovian. The waiting time that a new job request experiences is the sum of the service times of all the jobs that are already in the queue, and the residual service time of the job that is being serviced. In this case, it is important to make this distinction between the jobs that are already in the queue, and the job being serviced because we are now not making the assumption that the service times are exponential. Thus, we cannot exploit the memoryless property of the exponential service times, and the probability density function (pdf) of the residual service time is generally not the same as the pdf of the actual service time. Since the server is busy a fraction ρ of the time on average, where ρ is the utilization, we can then write the following.

$$E[W] = E[L] \times E[S] + \rho E[R] \tag{2.7}$$

where L is the length of the queue, and R is the residual service time of the job being serviced, conditioned on there being a job in service.

By considering the queue as a system by itself, we can apply Little's Law to obtain

$$E[L] = \lambda E[W] \tag{2.8}$$

Substituting Equation 2.8 into Equation 2.7,

$$\begin{aligned} E[W] &= \lambda E[W] \times E[S] + \rho E[R] \\ E[W] &= \rho E[W] + \rho E[R] \\ E[W] &= \frac{\rho E[R]}{1-\rho} \end{aligned} \tag{2.9}$$

To obtain the expected residual service time $E[R]$, we observe that the arrival of the most recent job is one of random incidence, and we wish to find the expectation of the time left until the next job in the queue receives service. Larson and Odoni [11] provide a complete derivation of the formula for $E[R]$.

$$E[R] = \frac{E[S^2]}{2E[S]}$$

$$E[R] = \frac{\sigma_S^2 + E[S]^2}{2E[S]} \tag{2.10}$$

Finally, by substituting Equations 2.9 and 2.10 into Equation 2.1,

$$E[T] = \frac{\rho}{(1-\rho)} \frac{E[S^2]}{2E[S]} + E[S] \tag{2.11}$$

Equation 2.11 thus expresses the expected total time that a job spends in the machine as a function of the first and second moments of S, the total service time. One may note that the first term of the sum in Equation 2.11 is in fact the well-known Pollaczek-Khinchin mean value formula.

2.4 Generalization of the Expected Total Time Formula for Multiple Jobs

Although we derived Equation 2.12 in the context of a single-item production system, we can modify it for the case of multiple job types. To obtain the expression for $E[T]$ when the system handles multiple job types, we simply have to let S represent the total service time of an "average" job, and invoke the total expectation theorems. If we let S_j represent the total service time of a job of type j, we then get the following equations

$$E[S] = \sum_{j=1}^{J} E[S_j] \times P(job\ arrival\ is\ type\ j) \tag{2.12}$$

$$E[S^2] = \sum_{j=1}^{J} E[S_j^2] \times P(job\ arrival\ is\ type\ j)$$

$$E[S^2] = \sum_{j=1}^{J} \left(\sigma_{S_j}^2 + E[S_j]^2\right) \times P(job\ arrival\ is\ type\ j) \tag{2.13}$$

$$P(job\ arrival\ is\ type\ j) = \frac{\lambda_j}{\lambda} \tag{2.14}$$

Now consider a job request of job type j that has a demand for d good units, and the production yield is uncertain. In this case, S_j is dependent on two factors:

1. The choice of batch sizes
2. The number of good units that is produced during each batch service

While it is possible to perform an analysis on S_j in the case of arbitrary demand, this is considerably more difficult due to the probabilistic nature of factor 2. In addition, since the yield of a batch is modeled as a Binomial random variable, the number of good units in a batch is also dependent on the batch size. With this yield uncertainty, the demand of a job request may only be partially fulfilled during a particular service, and S_j depends not only on the initial batch size, but also on the subsequent batch sizes. The complexity in considering all the possible combinations leads to difficulty in obtaining the moments of S_j. As a result, obtaining a closed-form expression for the expected total time for an arbitrary demand is mathematically tedious.

However, suppose each job request requires only a single good unit. Then, the algebra is significantly simplified since we know that as long as a batch yields at least 1 good unit, the job request is satisfied. In other words, re-service is only required when the batch yields no good units. With the unit demand assumption, we see that:

$$S_j = X_{j,1} + X_{j,2} + \ldots + X_{j,M}$$

where $X_{j,i}$ is the time taken for the i^{th} service iteration of a job of type j, and M is a geometric random variable representing the number of batch services before the unit demand is satisfied. For a job of type j, we will define n_j as the batch size to be used, and β_j as the probability that a defective unit is produced. We then have the following results:

$$P(M=m)=(\beta_j^{n_j})^{m-1}(1-\beta_j^{n_j}) \qquad m=1,2,...$$

$$E[M]=\frac{1}{1-\beta_j^{n_j}}$$

$$\sigma_M^2=\frac{\beta_j^{n_j}}{(1-\beta_j^{n_j})^2}$$

By the law of iterated expectations,

$$E[S_j]=E[X_j]\times E[M]$$

$$E[S_j]=\frac{\bar{x}_j}{1-\beta_j^{n_j}} \tag{2.15}$$

where $\bar{x}_j$ is the expected service time for one iteration of job j, using a batch size of n_j.

By the law of total variance,

$$\sigma_{S_j}^2=\sigma_{X_j}^2 E[M]+\sigma_M^2 E[X_j]^2$$

$$\sigma_{S_j}^2=\frac{\sigma_{X_j}^2}{1-\beta_j^{n_j}}+\frac{\bar{x}_j{}^2\beta_j^{n_j}}{(1-\beta_j^{n_j})^2} \tag{2.16}$$

Combining Equations 2.12 through 2.16, the first and second moments of S are found as

$$E[S]=\sum_{j=1}^{J}\frac{\lambda_j}{\lambda}\frac{\bar{x}_j}{1-\beta_j^{n_j}} \tag{2.17}$$

$$E[S^2] = \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \left(\frac{\sigma_{X_j}^2}{1-\beta_j^{n_j}} + \frac{\bar{x}_j{}^2 \beta_j^{n_j}}{(1-\beta_j^{n_j})^2} + \left(\frac{\bar{x}_j}{1-\beta_j^{n_j}} \right)^2 \right) \tag{2.18}$$

For multiple job types, the utilization is given by:

$$\rho = \lambda E[S] \tag{2.19}$$

Finally, the general expression for the expected total time that an average job spends in a production system that handles multiple job types, each with unit demand, is obtained from Equations 2.11, 2.17-2.19.

$$E[T] = \frac{\rho}{(1-\rho)} \frac{E[S^2]}{2E[S]} + E[S]$$

$$E[T] = \frac{\lambda E[S^2]}{2(1-\lambda E[S])} + E[S]$$

$$E[T] = \frac{\sum_{j=1}^{J} \lambda_j \left(\frac{\sigma_{X_j}^2}{1-\beta_j^{n_j}} + \frac{\bar{x}_j{}^2 \beta_j^{n_j}}{(1-\beta_j^{n_j})^2} + \left(\frac{\bar{x}_j}{1-\beta_j^{n_j}} \right)^2 \right)}{2\left(1 - \sum_{j=1}^{J} \lambda_j \frac{\bar{x}_j}{1-\beta_j^{n_j}} \right)} + \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \frac{\bar{x}_j}{1-\beta_j^{n_j}} \tag{2.20}$$

(This Page Intentionally Left Blank)

Chapter 3

Batching Strategy for Single-Item, Unit Demand Production Systems

To gain some insight into the effect of batching policies on the expected total time, we first concentrate on the simple case where machines only handle jobs of a single job type, and the demand of each job is for a single good unit. With these simplifications, we are able to obtain solutions analytically, and propose a method to search for the optimal batch size. In addition, we also conduct several studies to observe how changes in the job parameters affect the expected total time and the batch sizing decision.

3.1 Expected Total Time Formula

In the case when there is only a single type of job request arriving at our production system, and there is unit demand, then we can reduce Equation 2.20 to the following equation. We omit the subscript j because there is only one job type.

$$E[T] = \frac{\lambda\left(\frac{\sigma_X^2}{1-\beta^n} + \left(\frac{\bar{x}}{(1-\beta^n)}\right)^2 \left(1+\beta^n\right)\right)}{2\left(1-\lambda\frac{\bar{x}}{1-\beta^n}\right)} + \frac{\bar{x}}{1-\beta^n} \tag{3.1}$$

To further simplify the analysis, we assume that once the batch size is fixed, then the service time is deterministic. Thus, there is no variability associated with a single service time ($\sigma_X^2 = 0$), and we are dealing with an M/D/1 queue. With deterministic service times, we obtain

$$E[T] = \frac{\lambda\left(\frac{\bar{x}}{(1-\beta^n)}\right)^2 \left(1+\beta^n\right)}{2\left(1-\lambda\frac{\bar{x}}{1-\beta^n}\right)} + \frac{\bar{x}}{1-\beta^n} \tag{3.2}$$

Before we proceed to minimize the expected total time, we need to know how the single service time depends on the batch size. We assume that the expected single service time for a batch of size n is comprised of two parts. First, there is a fixed setup time τ, which is invariant of the batch size. Apart from a setup time, the service time also has a variable component that grows linearly with the batch size n, each unit of which takes α units of time on average to produce. Our model for the single service time of batch can then be written as

$$\bar{x} = \tau + n\alpha \tag{3.3}$$

For example, in a MEMS microfabrication system, the setup time of a batch would encompass tasks such as etching, and the growing of silicon dioxide in an oxidation furnace. In industry, these are the process steps in which the entire batch of wafers is processed at the same time. On the other hand, tasks such as probing and masking are usually done individually, and thus contribute to the variable time component.

3.2 Upper and Lower Bounds

Since the batch sizes in any real system must be integers, we immediately see that this is an integer optimization problem. However, the approach that we adopt to solve the single-item unit demand problem is to relax the integrality constraint on the batch sizes, and obtain the optimal non-integer batch size $\hat{n}$. Then, we will show how to obtain the optimal integer batch size n^*, from $\hat{n}$.

In order for the expected total time to be finite, the queuing system must be stable. Mathematically, this implies that the utilization of the system must be strictly less than 1.

$$\rho < 1$$

$$\lambda \frac{\tau + n\alpha}{(1-\beta^n)} < 1 \tag{3.4}$$

The effect of this utilization constraint is that it enforces upper and lower bounds on the size of $\hat{n}$. This is illustrated in Figure 3.1, for an example with input parameters: $\tau = 0.5$, $\alpha = 0.04$, $\beta = 0.4$, and $\lambda = 1$. From the figure, we see that the lower bound on $\hat{n}$ is 0.83, and the upper bound is 12.51.

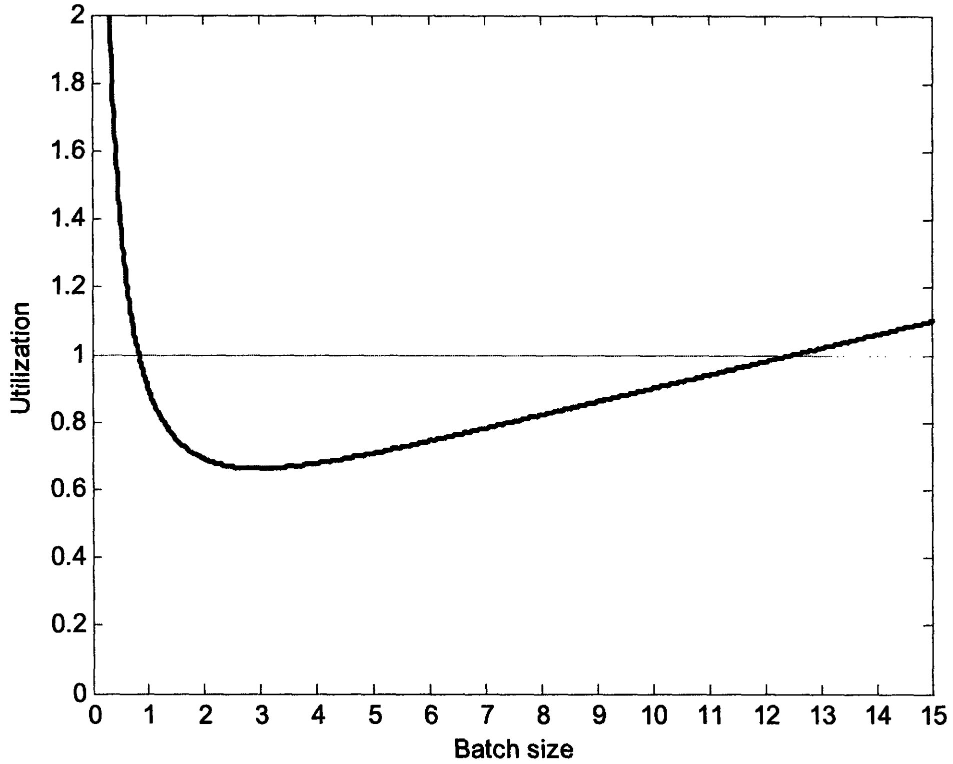

Figure 3.1: Example of the Lower and Upper Bounds on $\hat{n}$
$\tau = 0.5$, $\alpha = 0.04$, $\beta = 0.4$, and $\lambda = 1$

Having defined the set of feasible batch sizes, we then examine the relationship between the expected total time $E[T]$, and batch size n (Figure 3.2). By using batch sizes that are too small, there is a high probability that the demand for a job request will not be fulfilled in a single service, and multiple re-services will be required. Thus, we incur multiple setup times, which causes the total service time of a job to grow very large. However, using large batch sizes is also unfavorable because there is a tendency to produce too many units to satisfy the demand. Although the large batch size makes it highly likely that a single service is sufficient, this drives up the variable time component excessively. This effect is clearly observed in Figure 3.2.

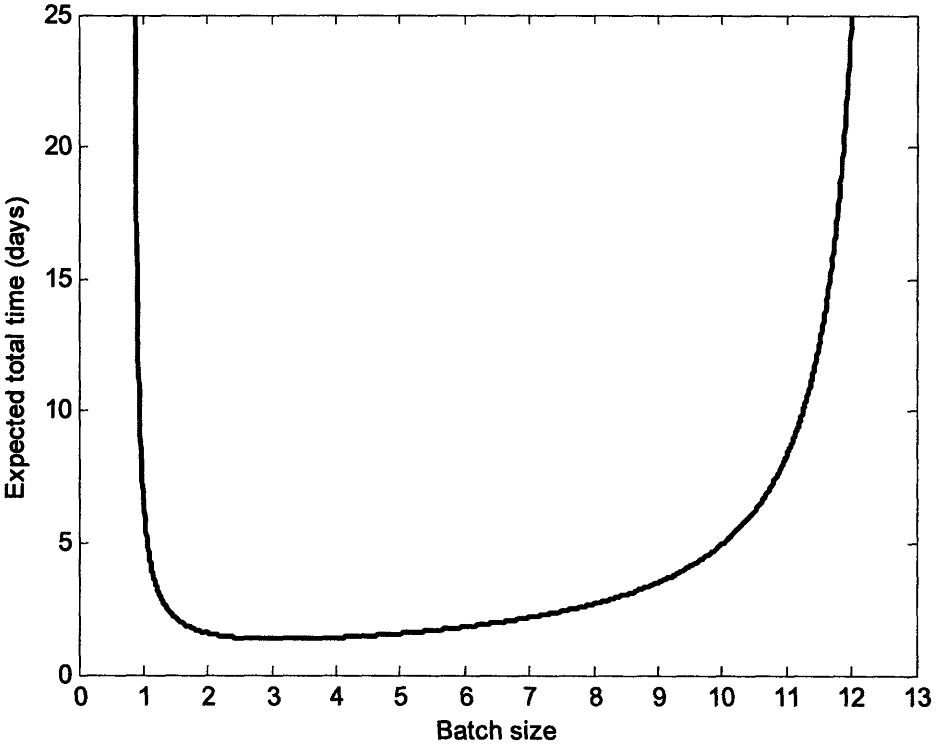

Figure 3.2: Relationship between Expected Total Time and Batch Size
$\tau = 0.5$, $\alpha = 0.04$, $\beta = 0.4$, and $\lambda = 1$

3.3 Bisection Line-Search Algorithm

From our experience with numerical examples, we observe that the expected total time function is well-behaved. $E[T]$, as a function of batch size, appears to be unimodal and has only one minimum point. Therefore, we attempt to find the optimal non-integer batch size by solving the first-order optimality conditions.

Using the substitutions $\bar{s} = \dfrac{\bar{x}}{(1-\beta^n)}$ and $u = 1 + \beta^n$, we rewrite Equation 3.2 as

$$E[T] = \frac{\lambda \bar{s}^2 u}{2(1-\lambda \bar{s})} + \bar{s} \tag{3.5}$$

Taking the first derivative of $E[T]$ with respect to n:

$$\frac{d}{dn}E[T]=\frac{\partial}{\partial\bar{s}}E[T]\frac{d}{dn}\bar{s}+\frac{\partial}{\partial u}E[T]\frac{d}{dn}u \tag{3.6}$$

where

$$\frac{\partial}{\partial\bar{s}}E[T]=\frac{\lambda\bar{s}u}{1-\lambda\bar{s}}+\frac{\lambda^2\bar{s}^2u}{2(1-\lambda\bar{s})^2}+1$$

$$\frac{d}{dn}\bar{s}=\frac{(\tau+n\alpha)\ln(\beta)\beta^n}{(1-\beta^n)^2}+\frac{\alpha}{(1-\beta^n)}$$

$$\frac{\partial}{\partial u}E[T]=\frac{\lambda\bar{s}^2}{2(1-\lambda\bar{s})}$$

$$\frac{d}{dn}u=\ln(\beta)\beta^n$$

To solve the first-order optimality condition given in Equation 3.6, we can perform a bisection line search. The algorithm is outlined as follows.

Algorithm to solve single-item unit demand batching problem:

```
1. Determine the lower and upper bounds on $\hat{n}$ by solving the utilization constraint.
        Let the lower and upper bounds be $n_l$ and $n_u$ respectively.
        Denote $E[T]$ as $h(n)$, a function of the batch size.
        $h'(n)$ is the first derivative, and its value at any value of $n$ can be calculated by
        Equation 3.6

2. Let $\tilde{n} = \frac{n_l + n_u}{2}$ and compute $h'(\tilde{n})$
        If $|h'(\tilde{n})| < \varepsilon$, where $\varepsilon$ is some small user-defined constant (eg. $10^{-6}$)
                Then STOP. Algorithm terminates, $\hat{n} = \tilde{n}$.
        Else,
                If $h'(\tilde{n}) < 0$, then let $n_l = \tilde{n}$
                If $h'(\tilde{n}) > 0$, then let $n_u = \tilde{n}$
                Go to step 2.
```

Since the batch sizes used in real-life production systems must be integer-valued (it makes no sense to process a batch of size 3.42, for example), we then need a method of obtaining the optimal integer batch size n^* from $\hat{n}$. This is easily done, since we have observed that $E[T]$ is unimodal, then we have that $n^* = \lceil \hat{n} \rceil$ or $\lfloor \hat{n} \rfloor$. Therefore, we simply need to evaluate the value of $E[T]$ at these two candidate batch sizes, and pick the better one.

3.4 Results and Observations

Using the bisection line-search algorithm, we examine the change in optimal batch size when we vary the input parameters of the job. Four studies were carried out, with each study corresponding to changing a single input parameter.

The relationship between the expected total time and batch size for various test cases are depicted in Figures 3.3-3.6. For each curve, its minimum point, and corresponding optimal batch size $\hat{n}$ are indicated. These results can be seen from the figures, and are also tabulated in Table 3.1.

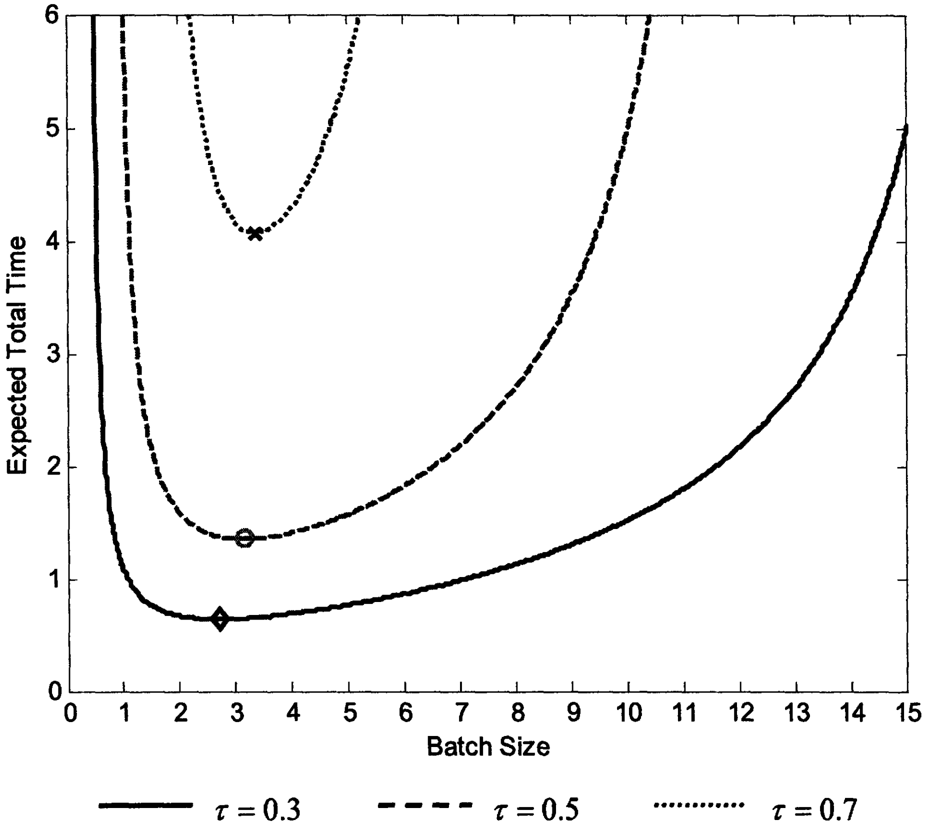

Figure 3.3: Expected Total Time as a Function of Batch Size for τ = 0.3, 0.5, and 0.7 Other input parameters: $\alpha = 0.04$, $\beta = 0.4$, and $\lambda = 1$

3.4.1 Effect of Setup Time τ

First, we note that increasing the setup time increases the expected total time for all values of batch size. This is not surprising given that τ is a fixed setup time that is invariant of batch size. Next, we observe that a job type with a larger setup time will have an optimal batch size that is no smaller than another job type with a smaller setup time. This observation can be explained as such – if we incurred a larger fixed setup time per batch, then it is beneficial to select larger batch sizes to try and reduce the number of re-services.

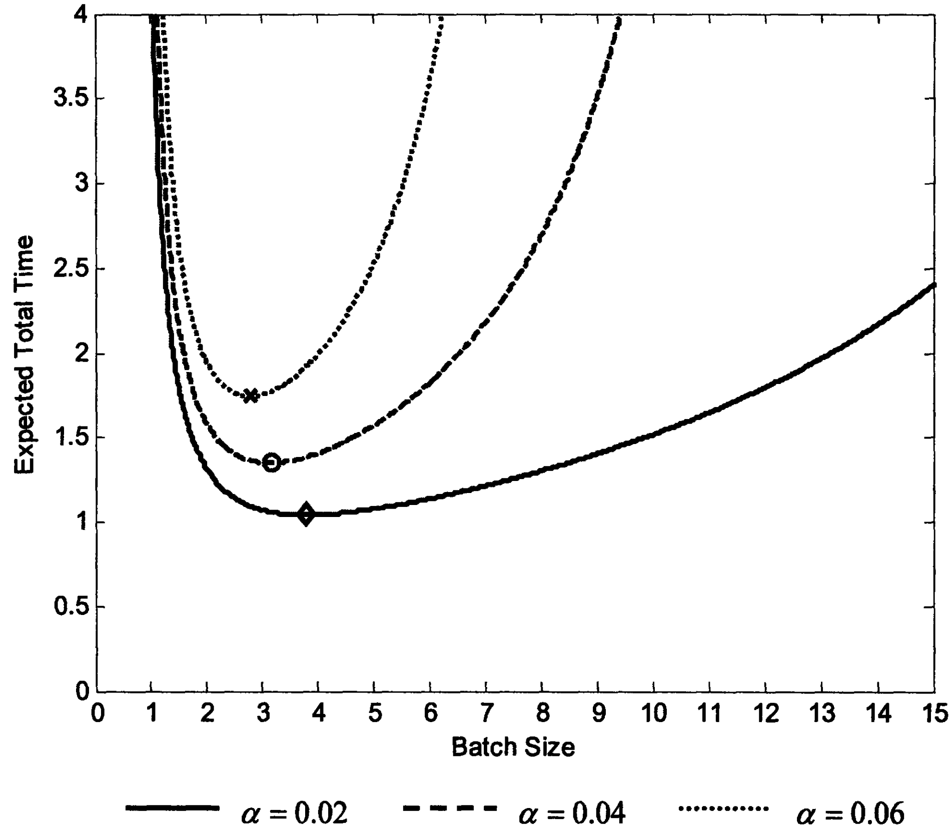

Figure 3.4: Expected Total Time as a Function of Batch Size for α = 0.02, 0.04, and 0.06
Other input parameters: $\tau = 0.5$, $\beta = 0.4$, and $\lambda = 1$

3.4.2 Effect of Unit Production Time α

From Figure 3.4, we see that increasing the unit production time α has a greater effect in increasing the expected total time when batch size is large. At small batch sizes, the dominant factor in the service time is the setup time τ, so the family of $E[T]$ curves appears very similar. In addition, a smaller α causes the optimal batch size to increase. This is indicative of the fact that a smaller α allows the system to produce units at a faster rate, and hence it allows the use of a larger batch size without excessively increasing the service time.

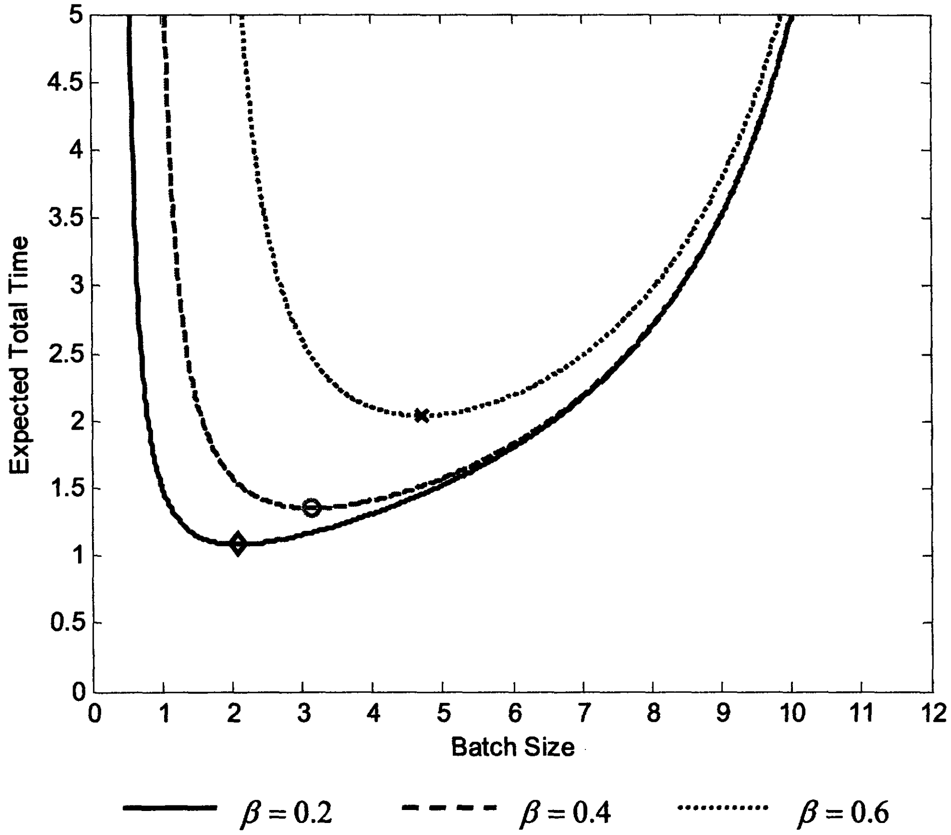

Figure 3.5: Expected Total Time as a Function of Batch Size for β = 0.2, 0.4, and 0.6
Other input parameters: $\tau = 0.5$, $\alpha = 0.04$, and $\lambda = 1$

3.4.3 Effect of Unit Failure Probability β

The effect of increasing the unit failure probability is to increase the optimal batch size. This is obvious since a job with a higher failure rate would require more units on average to satisfy its demand. Thus, we expect a larger optimal batch size so as to reduce the number of re-services. In addition, we see that for larger batch sizes, the effect of β on the expected total time is less pronounced. This is to be expected, since the probability of getting at least one good unit in the first service gets close to 1 if a large batch size is used.

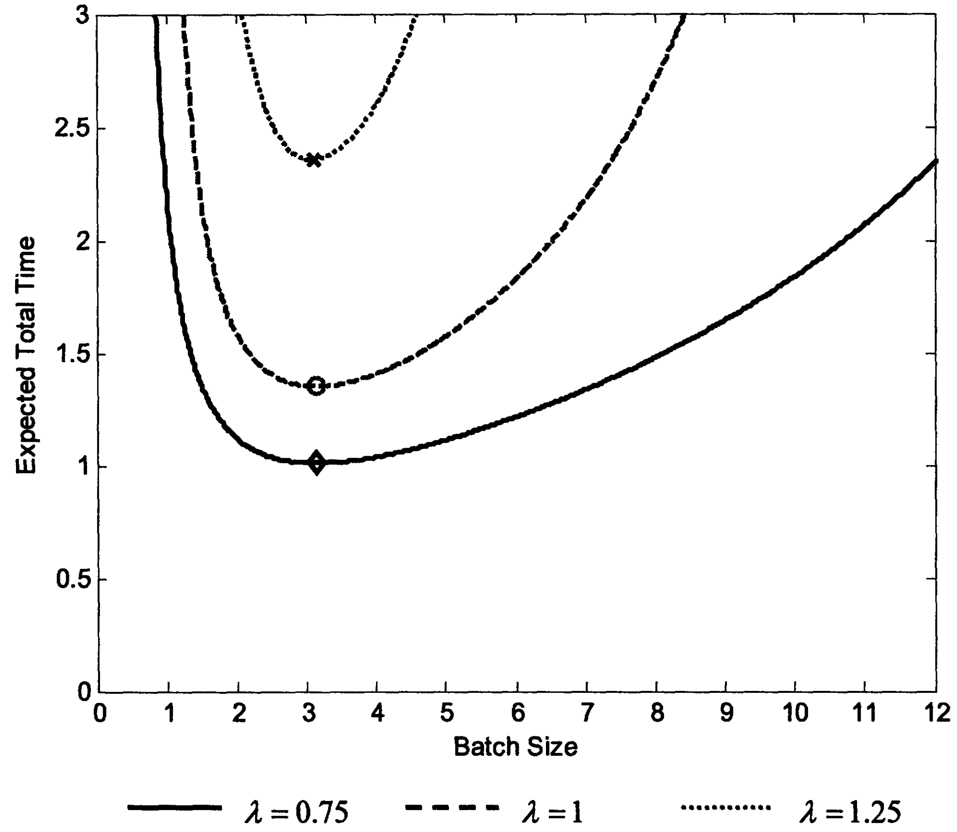

Figure 3.6: Expected Total Time as a Function of Batch Size for λ = 0.75, 1, and 1.25
Other input parameters: $\tau = 0.5$, $\alpha = 0.04$, and $\beta = 0.4$

3.4.4 Effect of Job Input Rate λ

With an increased job input rate, the system utilization increases. As a result, a job arrival will have to spend more time waiting in the queue, thus leading to an increase in expected total time. While the effect of setup time, unit production rate, and unit failure probability are rather straightforward, we observe a surprising result in the test cases where we varied the job input rate. In contrast to the other test cases, the optimal batch size does not change much. A simple explanation for this is as such. The aim of the server is to complete service of its current job as quickly as possible so as to provide service to the next job waiting in the queue. The job input rate only affects the number of jobs waiting in the queue. But regardless of the number of jobs in the queue, the aim of the server is still the same – to satisfy the current job, and start on the next one. Hence, the optimal batch size does not change much.

One interesting point to note is that the job input rate will have an effect on the batch size in the event that it causes the utilization to exceed 1. If the job arrival rate was so high such that the machine was operating extremely high (or unstable) utilization levels, then the batch size may change to maintain the stability of the queue.

Table 3.1: Variation of Optimal Batch Sizes with Change in Input Parameters

	τ	α	β	λ	$\hat{n}$
Base case	0.5	0.04	0.4	1	3.16
Varying τ (Figure 3.4)	**0.3**	0.04	0.4	1	2.71
	0.7	0.04	0.4	1	3.38
Varying α (Figure 3.5)	0.5	**0.02**	0.4	1	3.81
	0.5	**0.06**	0.4	1	2.79
Varying β (Figure 3.6)	0.5	0.04	**0.2**	1	2.10
	0.5	0.04	**0.6**	1	4.72
Varying λ (Figure 3.7)	0.5	0.04	0.4	**0.75**	3.16
	0.5	0.04	0.4	**1.25**	3.11

Chapter 4

Batching Strategies for Multi-Item, Unit Demand Production Systems

Now that we have obtained an optimal batching policy for the single-item, unit demand case, we turn our attention to a more complex problem. In this chapter, we examine the scenario where the demand of each job is still restricted to a single unit, but we now allow the machine to handle multiple job types. As we will see, introducing job heterogeneity makes the analysis slightly more complicated. Despite this, the bisection line-search algorithm that was discussed in Chapter 3 can still be implemented after some modifications. However, we require an alternative batching strategy because the bisection line-search does not generalize well when each job is not restricted to a single unit of demand. The second half of this chapter is thus concerned with deriving an alternative strategy, and demonstrating its effectiveness.

4.1 Expected Total Time Formula

For a multi-item, unit demand production system, the expected total time function that we are trying to minimize is given by Equation 2.20. In addition, we will make the same assumptions as we did in Chapter 3. First, for each job type j, service times are deterministic once the batch size n_j is fixed. Second, the service time is an affine function of batch size. Our objective is to choose batch sizes to minimize the following:

$$E[T]=\frac{\sum_{j=1}^{J}\lambda_j\left(\left(\frac{\bar{x}_j}{(1-\beta_j^{n_j})}\right)^2(1+\beta_j^{n_j})\right)}{2\left(1-\sum_{j=1}^{J}\lambda_j\frac{\bar{x}_j}{1-\beta_j^{n_j}}\right)}+\sum_{j=1}^{J}\frac{\lambda_j}{\lambda}\frac{\bar{x}_j}{1-\beta_j^{n_j}} \tag{4.1}$$

where

$$\bar{x}_j=\tau_j+n_j\alpha_j$$

Following the analysis performed for single-item, unit demand systems, we try to extend it to the multi-item case. One might intuitively expect that the multi-item expected total time function is also unimodal like its single-item counterpart, however we are unable to prove this.

Nevertheless, based on our experience with numerical examples, we expect that the expected total time function is well-behaved. As an example, we plot in Figure 4.1 the expected total time for a system that handles two different types of job requests. The input parameters and their corresponding optimal non-integer batch sizes are recorded in Table 4.1. We observe that the shape of the graph appears unimodal. In fact, the contour intervals that can be seen in both Figures 4.1 and 4.2 provide evidence that the expected total time function is convex in the batch sizes for this example.

Table 4.1: Input Parameters for Two Job Types

	τ_j	α_j	β_j	λ_j	$\hat{n}_j$
Job Type 1	0.5	0.04	0.4	0.7	3.18
Job Type 2	0.3	0.02	0.6	0.5	5.11

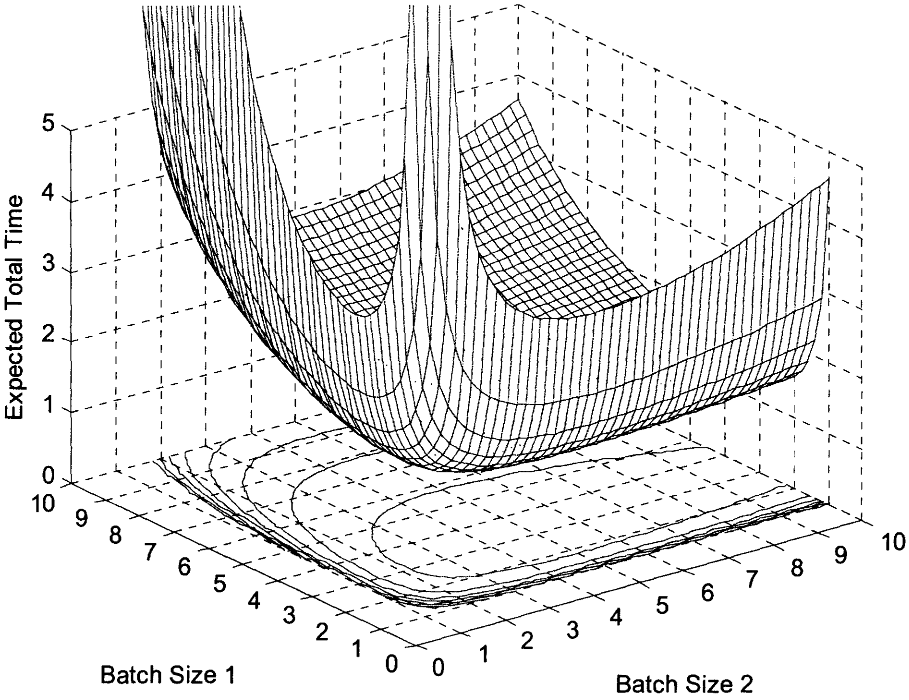

Figure 4.1: Expected Total Time as a Function of Batch Size for 2 Job Types

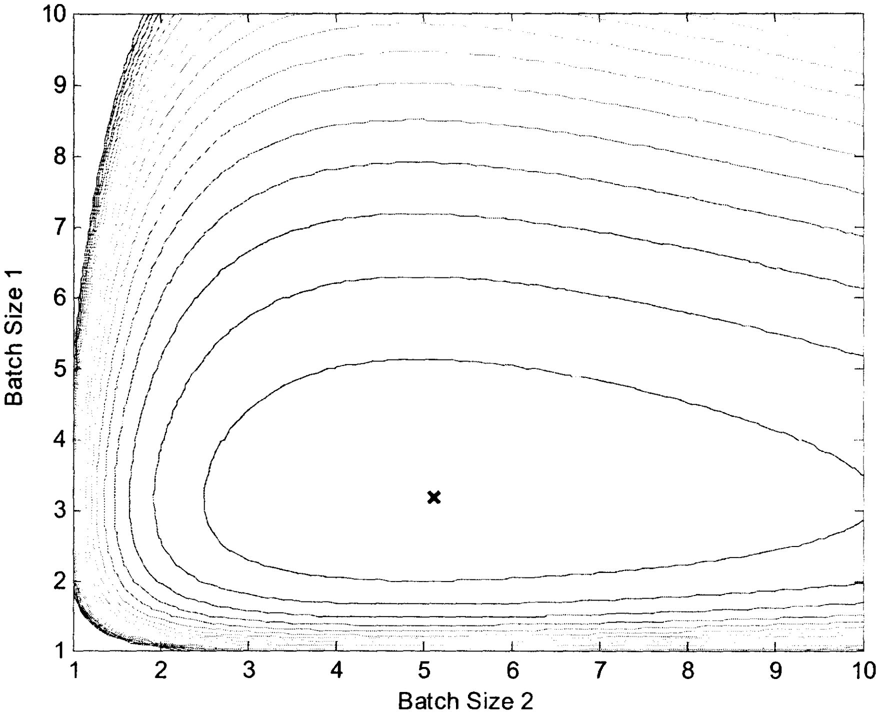

Figure 4.2: Contour Intervals of Expected Total Time for 2 Job Types

4.2 Batching Strategy using First-Order Optimality Conditions

Similar to the case for a single product with unit demand, we attempt to minimize $E[T]$ by satisfying the first order optimality conditions. Using the substitutions $\bar{s}_j = \dfrac{\bar{x}_j}{(1-\beta_j^{n_j})}$ and $u_j = 1 + \beta_j^{n_j}$, we rewrite Equation 4.1 as

$$E[T] = \frac{\sum_{j=1}^{J} \lambda_j \left(\bar{s}_j^2 u_j\right)}{2\left(1 - \sum_{j=1}^{J} \lambda_j \bar{s}_j\right)} + \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \bar{s}_j \tag{4.2}$$

Then, taking the first derivative of $E[T]$ with respect to n_j, we obtain

$$\frac{d}{dn_j} E[T] = \frac{\partial}{\partial \bar{s}_j} E[T] \frac{d}{dn_j} \bar{s}_j + \frac{\partial}{\partial u_j} E[T] \frac{d}{dn_j} u_j \tag{4.3}$$

where

$$\frac{\partial}{\partial \bar{s}_j} E[T] = \frac{\lambda_j \bar{s}_j u_j}{1 - \sum_{k=1}^{J} \lambda_k \bar{s}_k} + \frac{\lambda_j{}^2 \bar{s}_j{}^2 u_j}{2\left(1 - \sum_{k=1}^{J} \lambda_k \bar{s}_k\right)^2} + 1$$

$$\frac{d}{dn_j} \bar{s}_j = \frac{\left(\tau_j + n_j \alpha_j\right) \ln\left(\beta_j\right) \beta_j^{n_j}}{\left(1 - \beta_j^{n_j}\right)^2} + \frac{\alpha_j}{\left(1 - \beta_j^{n_j}\right)}$$

$$\frac{\partial}{\partial u_j} E[T] = \frac{\lambda_j \bar{s}_j{}^2}{2\left(1 - \sum_{k=1}^{J} \lambda_k \bar{s}_k\right)}$$

$$\frac{d}{dn_j}u_j = \ln(\beta_j)\beta_j^{n_j}$$

Solving for the solution to these J partial derivatives is difficult since the equations cannot be solved independently. Observe that it is the utilization $\sum_{k=1}^{J}\lambda_k \bar{s}_k$ in the denominators that causes the lot sizing decisions of different jobs to affect each other. However, if we know the value of utilization beforehand, then the J first-order optimality conditions are decoupled, and we can solve for the optimal batch size of each job independently of the other jobs.

This observation suggests an iterative procedure for solving the first-order conditions. We first define ρ_{guess} to be our estimate of the actual system utilization ρ_{actual}. Since ρ_{actual} is a function of the batch sizes, we will not know its value until we have solved for the batch sizes. However, we have also seen that to solve for the batch sizes, we require the value of ρ_{actual} to decouple the J first-order optimality conditions. The issue then, is that we know neither to begin with. To circumvent this problem, we estimate ρ_{actual} by ρ_{guess} and solve for the batch sizes using the algorithm presented below. After this is done, we use the batch sizes (that were obtained by making the assumption that ρ_{guess} is indeed correct) to calculate the realized system utilization ρ_{actual}. If there is a discrepancy between ρ_{guess} and ρ_{actual}, then we re-run the algorithm again with the revised value as the new value for the estimate of the actual system utilization. This method of calibrating the system utilization is done iteratively until ρ_{guess} and ρ_{actual} converge. A detailed step-by-step description of the algorithm is presented next.

Algorithm to solve multi-item, unit demand batching problem:

1. Initialize $\rho_{guess} = 0$ to start the algorithm.

2. Solve J first-order optimality conditions to obtain batch sizes, $\tilde{n}_1,...,\tilde{n}_J$, for the value of ρ_{guess} via a bisection line-search (Steps 2a and 2b).

For each job type j :

2a. Determine the lower and upper bounds on $\tilde{n}_j$ by solving the utilization constraint $\lambda_j \frac{\tau_j + n_j\alpha_j}{\left(1-\beta_j^{n_j}\right)} = 1$. This method is chosen because it provides us with a convenient way of specifying a lower and upper limit on $\tilde{n}_j$ for the bisection line-search to work.

Let the lower and upper bounds be n_l and n_u respectively.

2b. Let $\tilde{n}_j = \frac{n_l + n_u}{2}$ and compute $g(\tilde{n}_j)$ where

$$g(n_j) = \frac{\partial}{\partial \bar{s}_j} E[T] \frac{d}{dn_j} \bar{s}_j + \frac{\partial}{\partial u_j} E[T] \frac{d}{dn_j} u_j$$

and we substitute ρ_{guess} for $\sum_{k=1}^{J} \lambda_k \bar{s}_k$, such that:

$$\frac{\partial}{\partial \bar{s}_j} E[T] = \frac{\lambda_j \bar{s}_j u_j}{1-\rho_{guess}} + \frac{\lambda_j^{\,2} \bar{s}_j^{\,2} u_j}{2\left(1-\rho_{guess}\right)^2} + 1$$

$$\frac{d}{dn_j} \bar{s}_j = \frac{\left(\tau_j + n_j\alpha_j\right)\ln\left(\beta_j\right)\beta_j^{n_j}}{\left(1-\beta_j^{n_j}\right)^2} + \frac{\alpha_j}{\left(1-\beta_j^{n_j}\right)}$$

$$\frac{\partial}{\partial u_j} E[T] = \frac{\lambda_j \bar{s}_j^{\,2}}{2\left(1-\rho_{guess}\right)}$$

$$\frac{d}{dn_j} u_j = \ln\left(\beta_j\right)\beta_j^{n_j}$$

If $\left|g(\tilde{n}_j)\right| < \varepsilon$, where ε is some small user-defined constant (eg. 10^{-6})

Then STOP. Algorithm terminates.

Else,

If $g(\tilde{n}_j) < 0$, then let $n_l = \tilde{n}_j$

If $g(\tilde{n}_j) > 0$, then let $n_u = \tilde{n}_j$
Go to step 2b.

3. With the batch sizes obtained from Step 2, $\tilde{n}_1,...,\tilde{n}_J$, calculate the actual utilization.

$$\rho_{actual} = \sum_{j=1}^{J} \lambda_j \frac{\tau_j + \tilde{n}_j \alpha_j}{1 - \beta_j^{\tilde{n}_j}}$$

4. If $\left|\rho_{actual} - \rho_{guess}\right| < \delta$, where δ is some small user-defined constant (eg. 10^{-6})
    Then STOP. Algorithm terminates, $\hat{n}_j = \tilde{n}_j \qquad \forall j = 1,...,J$.
  Else,
    Let $\rho_{guess} = \rho_{actual}$
    Go to Step 2.

Once we obtain the optimal non-integer batch sizes $\hat{n}_j$, we then need to obtain an integer batch size n_j^*. This can be done by a simple rounding heuristic, where we round $\hat{n}_j$ to the nearest integer. Alternatively, to find the optimal integer batch sizes requires solution of an integer program, for instance by branch-and-bound.

4.3 Batching Strategy by Minimizing Utilization

To this point, the approach we have adopted to attain the optimal integer batch sizes is to relax the integrality constraints on the batch sizes. However, such an approach, while useful when faced with jobs with unit demand, would not work well when we generalize the model to incorporate general demand. Also, from the analysis performed in the previous section, we have also seen how batching decisions of one job type depend on the batch choices of the other job types.

Although one job type's batch size has an impact on the delay of other jobs, we found from numerical testing that the effect was not great. Hence, we present a heuristic in which we decompose the J inter-dependent batching decisions into J independent ones. This is first done by showing that for each job type j, there exist lower and upper bounds on the optimal integer batch size n_j^*. More importantly, these bounds are independent of the input job parameters and the batch sizes of the other job types. Having reduced the search range for n_j^*, we then show how to select a batch size for job type j independently of the other job types. The collection of these batch sizes then forms our solution vector, which as we will show, performs extremely well as compared to the true optimal solution vector.

4.3.1 Lower and Upper Bounds for M/D/1 Queue

We first define $\underline{n}_j^*$ to be the integer batch size that attains the minimum value of $\bar{s}_j$.

Lemma 1: *The expected total service time $\bar{s}_j$ is unimodal and has only one minimum.*

Proof.

$$\bar{s}_j = \frac{\tau_j + n_j\alpha_j}{\left(1-\beta_j^{n_j}\right)}$$

The first derivative of $\bar{s}_j$ with respect to n_j is

$$\frac{d}{dn_j}\bar{s}_j = \frac{\left(\tau_j + n_j\alpha_j\right)\ln(\beta_j)\beta_j^{n_j}}{\left(1-\beta_j^{n_j}\right)^2} + \frac{\alpha_j}{\left(1-\beta_j^{n_j}\right)}$$

To obtain the stationary points, we set the first derivative to 0

$$\frac{\left(\tau_j + n_j\alpha_j\right)\ln(\beta_j)\beta_j^{n_j}}{\left(1-\beta_j^{n_j}\right)^2} + \frac{\alpha_j}{\left(1-\beta_j^{n_j}\right)} = 0$$

$$\left(\frac{\tau_j}{\alpha_j} + n_j\right)\ln(\beta_j)\beta_j^{n_j} + (1-\beta_j^{n_j}) = 0$$

$$\left(\frac{\tau_j}{\alpha_j} + n_j\right) = \frac{1}{\ln(\beta_j)}\left(1 - \frac{1}{\beta_j^{n_j}}\right) \tag{4.4}$$

The LHS of Equation 4.4 has a value of $\frac{\tau_j}{\alpha_j}$ at $n_j = 0$ whereas the RHS has a value of 0 at $n_j = 0$. Looking at the LHS, it increases linearly with respect to n_j. On the other hand, the RHS is an exponentially increasing function. Therefore, they must intersect at only one point, which implies that the function $\bar{s}_j$ must have only one stationary point. An example is shown in Figure 4.3.

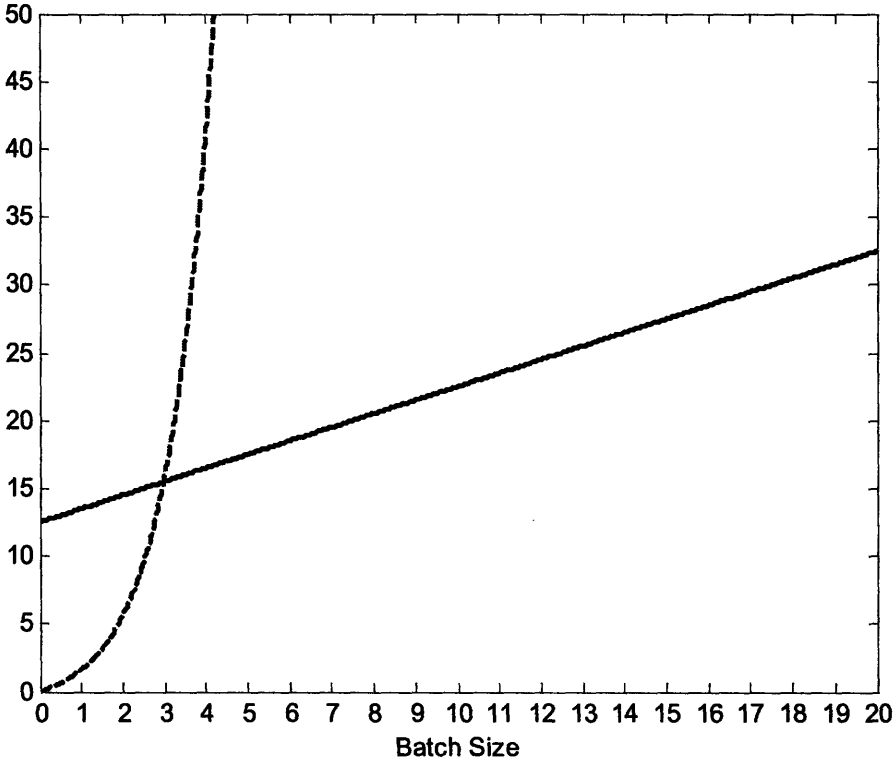

Figure 4.3: Single Stationary Point Exists for $\bar{s}_j$
$\tau = 0.5$, $\alpha = 0.04$, and $\beta = 0.4$

Since $\bar{s}_j\Big|_{n_j=0} = \infty$ and $\bar{s}_j\Big|_{n_j=\infty} = \infty$, then the single stationary point must be a minimum. ■

We can thus characterize the shape of $\bar{s}_j$ as such: The function first decreases monotonically until it attains its minimum, and then increases monotonically.

Proposition 1: *$\underline{n}_j^*$ is a lower bound on the optimal integer batch size (n_j^*) of job type j*

Proof. Let $\bar{s}_j^*$ be the value of $\bar{s}_j$ attained when integer batch size $\underline{n}_j^*$ is used. Alternatively, $\bar{s}_j^*$ is the minimum value of $\bar{s}_j$ that can be attained for all possible integer batch sizes.

By Lemma 1, we see that the value of $\bar{s}_j$ evaluated at any integer batch sizes $n_j < \underline{n}_j^*$ is strictly larger than $\bar{s}_j^*$.

Also, since $\beta_j^{n_j}$ is decreasing with batch size, the value of $\beta_j^{n_j}$ also increases as we decrease the batch size from $\underline{n}_j^*$.

Consider the formula for $E[T]$ given by Equation 4.2.

$$E[T] = \frac{\sum_{j=1}^{J} \lambda_j \bar{s}_j^2 \left(1 + \beta_j^{n_j}\right)}{2\left(1 - \sum_{j=1}^{J} \lambda_j \bar{s}_j\right)} + \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \bar{s}_j$$

Consider also a solution vector $[n_1, n_2, \ldots, \underline{n}_j^*, \ldots, n_J]$ as a candidate batching strategy to minimize $E[T]$. Here, we select the batch size for some job type j to be $\underline{n}_j^*$. The integer batch sizes for the other job types need not be optimal in any way; the only restriction is that this solution vector must not violate the utilization constraint.

Since we have established that $\bar{s}_j$ and $\beta_j^{n_j}$ both increase as we select batch sizes smaller than $\underline{n}_j^*$, then we see that for all batch sizes $n_j < \underline{n}_j^*$, the resulting solution vector $[n_1, n_2, \ldots, n_j, \ldots, n_J]$ will either result in a larger value of $E[T]$, or else be an infeasible solution.

Since this argument holds for all job types, this guarantees us that for any job type j, within the set of integer batch sizes $n_j \leq \underline{n}_j^*$, the minimum value of $E[T]$ is attained when $\underline{n}_j^*$ is used. This completes the proof that $\underline{n}_j^*$ is a lower bound on n_j^*. ■

If we consider batch sizes larger than $\underline{n}_j^*$, then although the value of $\bar{s}_j$ increases, the value of $\beta_j^{n_j}$ will decrease. The only time we would consider choosing a batch size larger than $\underline{n}_j^*$ is if the decrease in $E[T]$ we obtain from the decrease in $\beta_j^{n_j}$ is sufficient enough to offset the increase in $E[T]$ due to the increase in $\bar{s}_j$. In fact, we find that $\underline{n}_j^*$ is very often the optimal integer batch size.

Two factors prevent us from increasing the batch size too much larger than $\underline{n}_j^*$:

1. The increase in $\bar{s}_j$. In $E[T]$, any reduction in $\beta_j^{n_j}$ only affects the job type j, but the increase in $\bar{s}_j$ affects all job types since it directly affects utilization.
2. $\beta_j^{n_j}$ decreases at a decreasing rate with increasing batch size. Thus, once the batch size becomes too large, it will only have a marginal benefit in decreasing $\beta_j^{n_j}$, and we receive diminishing returns.

To find an upper bound on the optimal integer batch size, we first define $\bar{n}_j^*$ to be the integer batch size that attains the minimum value of $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$. We then assert the following:

Lemma 2: *The function* $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$ *is unimodal and has only one minimum.*

Proof. The first derivative of $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$ with respect to n_j is

$$\frac{d}{dn_j}\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)=\bar{s}_j^{\,2}\ln(\beta_j)\beta_j^{n_j}+2\bar{s}_j\left(1+\beta_j^{n_j}\right)\left(\frac{\bar{s}_j\ln(\beta_j)\beta_j^{n_j}}{\left(1-\beta_j^{n_j}\right)^2}+\frac{\alpha_j}{\left(1-\beta_j^{n_j}\right)}\right)$$

To obtain the stationary points, we set the first derivative to 0

$$\bar{s}_j^{\,2} \ln(\beta_j)\beta_j^{n_j} + 2\bar{s}_j\left(1+\beta_j^{n_j}\right)\left(\frac{\bar{s}_j \ln(\beta_j)\beta_j^{n_j}}{\left(1-\beta_j^{n_j}\right)^2} + \frac{\alpha_j}{\left(1-\beta_j^{n_j}\right)}\right) = 0$$

$$\ln(\beta_j)\beta_j^{n_j}\left(1-\beta_j^{n_j}\right)^2 + 2\left(1+\beta_j^{n_j}\right)\ln(\beta_j)\beta_j^{n_j} + 2\left(1+\beta_j^{n_j}\right)\frac{\left(1-\beta_j^{n_j}\right)\alpha_j}{\bar{s}_j} = 0$$

$$\ln(\beta_j)\beta_j^{3n_j} + 3\ln(\beta_j)\beta_j^{n_j} = -\frac{2\alpha_j\left(1+\beta_j^{n_j}\right)\left(1-\beta_j^{n_j}\right)^2}{\tau_j + n_j\alpha_j}$$

$$-\frac{\ln(\beta_j)}{2}\left(\frac{\tau_j}{\alpha_j} + n_j\right) = \frac{\left(1-\beta_j^{2n_j}\right)\left(1-\beta_j^{n_j}\right)}{\left(\beta_j^{3n_j} + 3\beta_j^{n_j}\right)} \qquad (4.5)$$

The intersection of the two sides of Equation 4.5 will determine the stationary points of $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$. The LHS of Equation 4.5 has a value of $-\frac{\ln(\beta_j)\tau_j}{2\alpha_j}$ at $n_j = 0$ whereas the RHS has a value of 0 at $n_j = 0$. Looking at the LHS, it increases linearly with respect to n_j. On the other hand, the RHS is an exponentially increasing function. Therefore, they must intersect at only one point, which implies that the function $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$ must have only one stationary point. An example is shown in Figure 4.4.

Since $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)\Big|_{n_j=0} = \infty$ and $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)\Big|_{n_j=\infty} = \infty$, then the single stationary point must be a minimum. ■

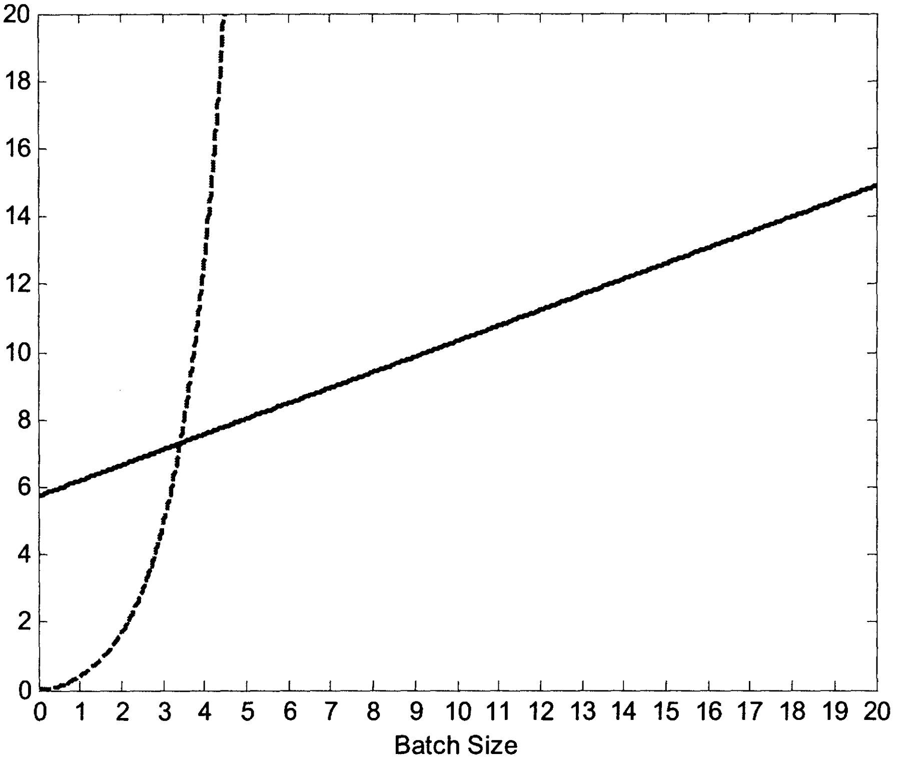

Figure 4.4: Single Stationary Point Exists for $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$
$\tau = 0.5$, $\alpha = 0.04$, and $\beta = 0.4$

Lemma 3: *$\bar{n}_j^*$ is no smaller than $\underline{n}_j^*$*

Proof. From Lemma 1, we have determined the shape of $\bar{s}_j$.

$\bar{s}_j^{\,2}$ will thus have a similar shape as $\bar{s}_j$. More specifically, $\underline{n}_j^*$ that obtains the minimum value of $\bar{s}_j$ also attains the minimum value of $\bar{s}_j^{\,2}$.

For integers smaller than $\underline{n}_j^*$,

1. $\bar{s}_j^{\,2}$ increases
2. $\left(1+\beta_j^{n_j}\right)$ increases

Therefore, the minimum value of $\bar{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$ cannot occur at any integer smaller than $\underline{n}_j^*$.

This implies that the integer batch size $\overline{n}_j^*$ that attains the minimum value of $\overline{s}_j{}^2\left(1+\beta_j^{n_j}\right)$ must be at least as large as $\underline{n}_j^*$. ■

Proposition 2: $\overline{n}_j^*$ *is an upper bound on the optimal integer batch size (*n_j^**) of job type* j

Proof. We have shown from Lemma 2 that $\overline{s}_j{}^2\left(1+\beta_j^{n_j}\right)$ is unimodal with a single minimum; for integers larger than $\overline{n}_j^*$, the value of $\overline{s}_j{}^2\left(1+\beta_j^{n_j}\right)$ is larger.

Also, by Lemma 3, since $\overline{n}_j^*$ is no smaller than $\underline{n}_j^*$, this implies the value of $\overline{s}_j$ increases for integers larger than $\overline{n}_j^*$.

Consider the formula for $E[T]$ given by Equation 4.2.

$$E[T]=\frac{\sum_{j=1}^{J}\lambda_j\overline{s}_j^2\left(1+\beta_j^{n_j}\right)}{2\left(1-\sum_{j=1}^{J}\lambda_j\overline{s}_j\right)}+\sum_{j=1}^{J}\frac{\lambda_j}{\lambda}\overline{s}_j$$

Consider also a solution vector $[n_1,n_2,\ldots,\overline{n}_j^*,\ldots,n_J]$ as a candidate batching strategy to minimize $E[T]$. Here, we select the batch size for some job type j to be $\overline{n}_j^*$. The integer batch sizes for the other job types need not be optimal in any way; the only restriction is that this solution vector must not violate the utilization constraint.

Since we have established that $\overline{s}_j$ and $\overline{s}_j{}^2\left(1+\beta_j^{n_j}\right)$ both increase as we select larger batch sizes than $\overline{n}_j^*$, then we see that for all batch sizes $n_j>\overline{n}_j^*$, the resulting solution vector $[n_1,n_2,\ldots,n_j,\ldots,n_J]$ will either result in a larger value of $E[T]$, or else be an infeasible solution.

Since this argument holds for all job types, this guarantees us that for any job type j, within the set of integer batch sizes $n_j\geq\overline{n}_j^*$, the minimum value of $E[T]$ is attained when $\overline{n}_j^*$ is used. This completes the proof that $\overline{n}_j^*$ is an upper bound on n_j^*. ■

Corollary 1: *If $\underline{n}_j^* = \overline{n}_j^*$, then we have chosen the batch size for job type j optimally.*

We have derived lower and upper bounds on the optimal batch sizes for our M/D/1 queueing model in this section. However, similar bounds can also be found for other models. We demonstrate this for two examples – the M/M/1 model, and a modified M/E$_n$/1 model – in Appendix A.

By examining the lower and upper bounds for the M/D/1 model, we make two very important observations:

- While Propositions 1 and 2 provide bounds on the optimal integer batch size n_j^*, an interesting observation is that the bounds are independent of the job input rate λ_j. Although λ_j must be taken into consideration when determining the optimal batch size n_j^*, the lower and upper bounds remain the same regardless of how λ_j varies.
- Throughout the thesis, we have made the assumption that the single service time of any job is an affine function of the batch size. However, we are able to generalize this assumption when obtaining the lower and upper bounds on n_j^*. For the bounds to remain valid, it is sufficient that $E[S_j]$ is unimodal, and has a single minimum point.

Figures 4.5 and 4.6 illustrate the effect of the lower and upper bounds on the optimal integer batch size for two sample job types with different parameters. Subplots 4.5(a) and 4.6(a) show how $\overline{s}_j$ varies with different choices of integer batch sizes, while Subplots 4.5(b) and 4.6(b) illustrate the relationship for $\overline{s}_j^{\,2}\left(1+\beta_j^{n_j}\right)$.

In the first example (Figure 4.5), we see that the lower bound on the optimal integer batch size n_j^* is 3. The upper bound, as seen from Subplot 4.5(b), is also 3. Therefore, we can conclude that the optimal batch size is 3 (Corollary 1).

For the example illustrated in Figure 4.6, the lower bound as seen from Subplot 4.6(a) is 3, whereas the upper bound is 4. In this case, the optimal batch size is either 3 or 4, and no other batch sizes need to be considered. Regardless of whether this job type is the only job type handled by the machine, or this job type is part of a set of job types, this is always true.

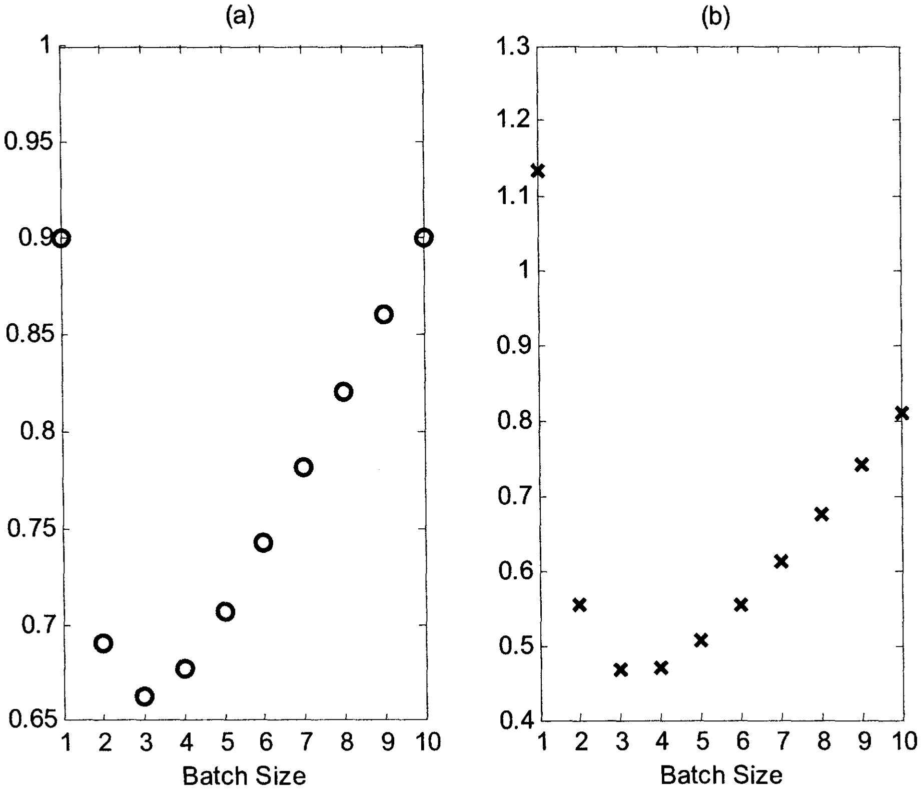

Figure 4.5: Lower and Upper Bounds: $\tau = 0.5$, $\alpha = 0.04$, and $\beta = 0.4$

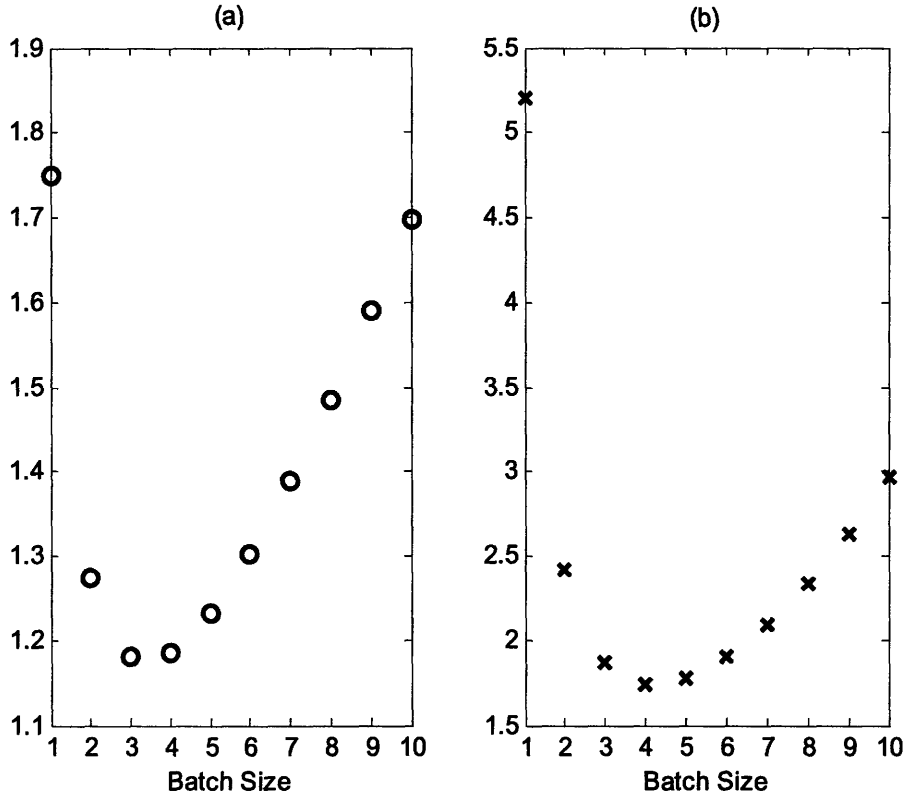

Figure 4.6: Lower and Upper Bounds: $\tau = 0.4$, $\alpha = 0.125$, and $\beta = 0.7$

4.3.2 Observations on Lower and Upper Bounds

Now that we have a method to obtain bounds on the optimal integer batch sizes of each job type, we would like to examine the tightness of these bounds. We computed the bounds on numerous job types, where a job type is a possible combination of the following job parameters:

$$\tau = 0.5, 1, 1.5, \ldots, 3$$

$$\frac{1}{\alpha} = 5, 6, 7, \ldots, 15$$

$$\beta = 0.1, 0.2, 0.3, \ldots, 0.9$$

Also, we calculated the optimal integer batch size assuming that the job type was the only one being serviced by the machine. These results are found in Appendix B.

From the results of the study, we make the following observations:

- The lower and upper bounds are rather tight bounds. From Appendix B, we narrow the search range for the optimal integer batch size significantly.
- Increasing the setup time increases both the upper and lower bounds
- Increasing the service rate increases both the upper and lower bounds
- Increasing the failure probability increases both the upper and lower bounds
- Consider two job types with the same failure probability ($\beta_1 = \beta_2$).
    - o If $\frac{\tau_1}{\alpha_1} = \frac{\tau_2}{\alpha_2}$, then $\underline{n}_1^* = \underline{n}_2^*$, and $\overline{n}_1^* = \overline{n}_2^*$.
    - o If $\frac{\tau_1}{\alpha_1} \geq \frac{\tau_2}{\alpha_2}$, then $\underline{n}_1^* \geq \underline{n}_2^*$, and $\overline{n}_1^* \geq \overline{n}_2^*$.
    - o These observations can be proved by looking at Equations 4.4 and 4.5.
- If the machine handles only 1 type of job, then the optimal integer batch size n_j^* is frequently the lower bound $\underline{n}_j^*$.
- The range between the lower and upper bounds increases with τ , and β.
- The range between the lower and upper bounds decreases with α .

4.3.3 Minimizing Utilization Heuristic

While the optimal solution $[n_1^*, n_2^*, \ldots, n_J^*]$, is obtained by minimizing expected total time, we propose a heuristic that minimizes utilization instead. For the multi-item production system, the utilization is given as $\sum_{j=1}^{J} \lambda_j \overline{s}_j$. To minimize utilization, we can decompose the problem into J smaller subproblems. For each subproblem, the objective is to find the integer batch size that minimizes $\overline{s}_j$. From Section 4.3.1, we had defined this integer batch size to be $\underline{n}_j^*$, which by Proposition 1, is also the lower bound on n_j^*.

Therefore, instead of collectively solving for the optimal integer batch sizes $[n_1^*, n_2^*, \ldots, n_J^*]$, we solve for all the $\underline{n}_j^*$ individually. The collection of $\underline{n}_j^*$ forms our heuristic solution vector $[\underline{n}_1^*, \underline{n}_2^*, \ldots, \underline{n}_J^*]$. The advantage of this approach is that whereas determining the optimal batch sizes n_j^* is difficult due to dependencies among job types, our heuristic allows us to size each job type individually. Obtaining the heuristic batch sizes is also easy, since we have shown that $\bar{s}_j$ is unimodal and thus the minimum $\underline{n}_j^*$ can be found via a straightforward bisection line-search.

4.3.4 Performance of the Minimize Utilization Heuristic

By using the heuristic solution vector, we are essentially trading complexity for optimality. In this section, we measure the performance of the heuristic solution as compared to the optimal solution for the M/D/1 queuing model through a series of tests at various utilization levels. The results are summarized in Table 4.2, and illustrated in Figures 4.7 and 4.8. Figure 4.7 shows, at each utilization level, the average percentage increase in expected total time (over 500 test cases) when we use the heuristic solution instead of the optimal solution. Figure 4.8 provides another measure of performance by plotting the percentages of test cases that showed no increase in expected total time (heuristic solution is exactly the optimal solution), and the percentage of cases that showed increases of less than 1% and 2%.

For each target level of utilization ρ_{test}, we randomly generate 500 test cases.

Test routine for each test case:

1. Each test case comprises 10 different job types, where the input parameters of each job type are randomly chosen as follows:

   τ_j uniformly chosen over [0,5]

   $\dfrac{1}{\alpha_j}$ uniformly chosen over [0,20]

   β_j uniformly chosen over [0,1]

   λ_j uniformly chosen over [0,1]

   Although we don't need λ_j to generate the solution vector for our heuristic, or to obtain the lower and upper bounds on n_j^*, we do need λ_j to compute the actual

n_j^*. We will need the actual optimal batch sizes to compare against the performance of our heuristic batch sizes.

2. The specified level of utilization ρ_{test}, is calculated based on our heuristic solution. Since our heuristic works to minimize utilization and is independent of λ_j, this provides us with a convenient method of controlling ρ. Having obtained the minimum values of $\bar{s}_j$, we multiply each λ_j by a scaling factor, such that $\sum_{j=1}^{J} \lambda_j \bar{s}_j = \rho_{test}$.

3. Obtain $\underline{n}_j^*$ and $\overline{n}_j^*$ for each job type j. By Lemma 1 and 2, this can be done by bisection line search.

4. Using the solution vector $[\underline{n}_1^*, \underline{n}_2^*, \ldots, \underline{n}_J^*]$, we calculate the resultant expected total time $E[T]_{\min\_util}$ using Equation 4.1. This is the solution we would obtain by using the minimized utilization heuristic.

5. We perform an exhaustive search through all the possible combinations of integer batch sizes in order to find $[n_1^*, n_2^*, \ldots, n_J^*]$. The search space is significantly pruned because we know n_j^* must lie between $\underline{n}_j^*$ and $\overline{n}_j^*$. Then, we calculate the minimum expected total time $E[T]^*$.

6. We calculate the percentage increase of $E[T]_{\min\_util}$ from $E[T]^*$ by obtaining the value $\dfrac{E[T]_{\min\_util} - E[T]^*}{E[T]^*}$

For each utilization level, the values of average percentage increase over 500 test cases are recorded in Table 4.2. The detailed test results are found in Appendix C.

Table 4.2: Performance of Minimizing Utilization Heuristic

Test Utilization ρ_{test}	Average percentage increase in $E[T]$	Percentage of cases with no increase in $E[T]$	Percentage of cases with increase less than 1%	Percentage of cases with increase less than 2%
0.1	0.06%	42.2%	99.6%	99.8%
0.2	0.17%	19.6%	97.2%	99.6%
0.3	0.30%	10.2%	96.4%	99.0%
0.4	0.37%	9.2%	93.0%	99.0%
0.5	0.46%	8.4%	89.4%	99.0%
0.55	0.51%	8.0%	86.6%	98.2%
0.6	0.57%	8.0%	83.8%	96.6%
0.65	0.59%	9.6%	82.8%	96.0%
0.66	0.59%	7.2%	84.4%	97.2%
0.67	0.58%	7.6%	82.8%	96.2%
0.68	0.61%	4.8%	82.4%	97.6%
0.69	0.60%	8.2%	84.6%	96.4%
0.7	0.61%	8.0%	85.0%	96.2%
0.71	0.60%	9.8%	82.0%	96.0%
0.72	0.58%	9.4%	85.0%	97.4%
0.73	0.54%	10.6%	83.4%	98.2%
0.74	0.52%	10.8%	85.6%	98.6%
0.75	0.51%	12.0%	87.8%	97.4%
0.8	0.49%	14.2%	88.8%	97.4%
0.85	0.45%	17.0%	91.4%	97.8%
0.9	0.37%	21.2%	93.8%	98.0%
0.91	0.33%	21.6%	94.4%	97.6%
0.92	0.34%	24.6%	94.2%	97.8%
0.93	0.31%	30.4%	94.0%	98.0%
0.94	0.27%	31.0%	96.8%	98.8%
0.95	0.19%	41.0%	96.8%	98.6%
0.96	0.14%	51.6%	98.4%	99.6%
0.97	0.12%	59.4%	98.2%	99.6%
0.98	0.09%	65.8%	98.6%	99.2%
0.99	0.05%	82.2%	99.2%	99.6%

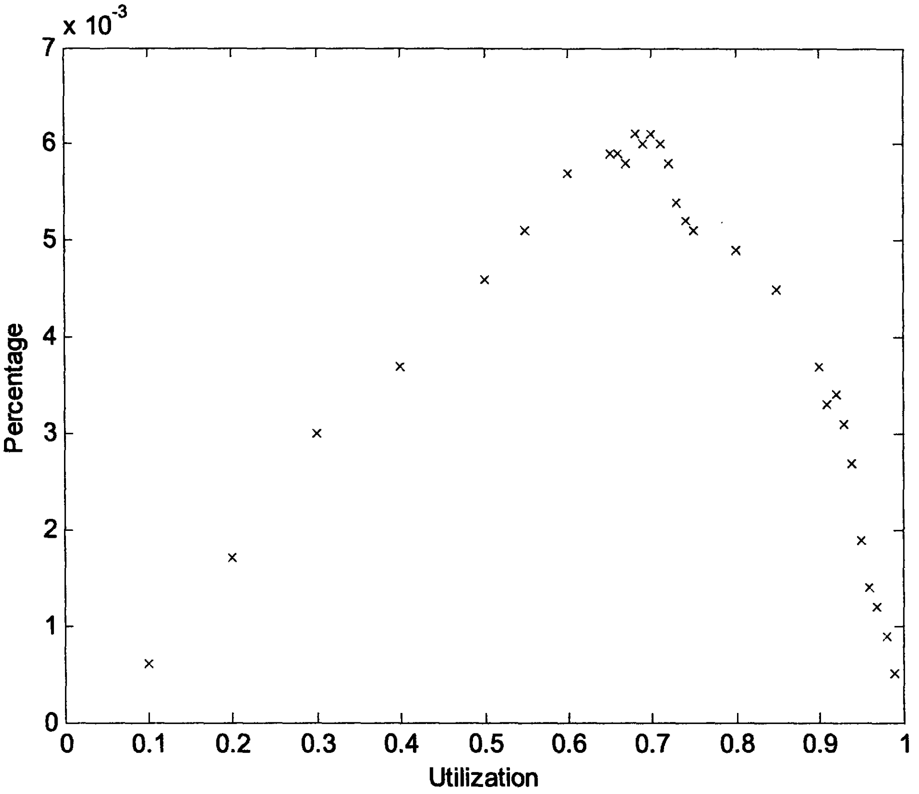

Figure 4.7: Average Increase in Expected Total Time for Various Utilization Levels

From Figure 4.7, we observe that even in the worst case scenario ($\rho \approx 0.7$), the average performance of our heuristic is still within 1% of the optimal. Not only does the heuristic perform well on average, it also attains near-optimal performance for almost all the test cases. From Figure 4.8, we see that in almost all the test cases, our heuristic performs within 2% of the optimal solution. Also, even in the worst case, more than 80% of test cases performed within 1%. This shows that the heuristic performs very well regardless of the system utilization. The actual results of all 500 test cases at every target utilization level can be found in Appendix C.

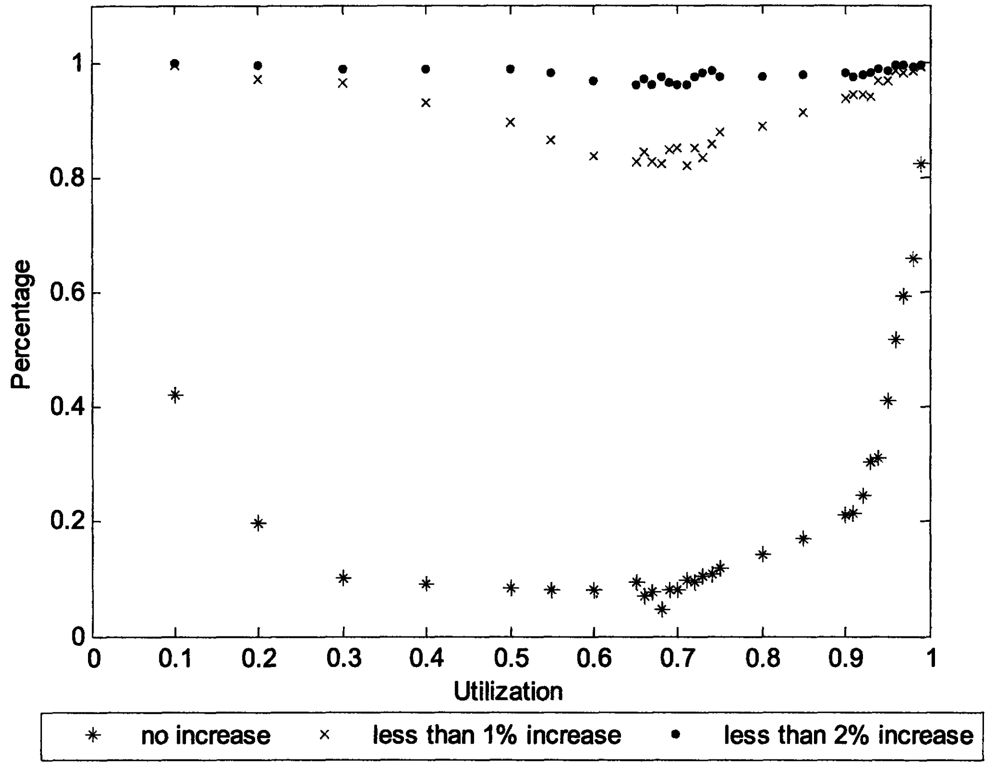

Figure 4.8: Performance of the Heuristic at Various Utilization Levels

An interesting observation is that our heuristic performs very well at low levels of utilization (Figure 4.7, $\rho = 0.1 - 0.4$). This is due to the fact that at these low levels, the system does not experience a large quantity of job traffic. As a result, most jobs that arrive at the system will find the server empty, and thus not experience much waiting time. This means that a large fraction of the expected total time that jobs spend in the system can be attributed to the expected service time. Since our heuristic objective is to minimize the service time of each job type, we expect that our heuristic performs well.

Another observation is that our heuristic performance improves at very high levels of utilization (Figure 4.7, $\rho = 0.8 - 0.99$). High levels of utilization correspond to the server being busy more of the time because job input rates are higher. One can visualize that under these conditions, the number of possible solutions decreases because we have fewer candidate solutions that violate the utilization constraint. In fact, our heuristic solution, which minimizes utilization, will be the last solution to be pruned off as we increase the utilization level. In other words, if the heuristic solution is not feasible, then minimizing expected total time is an infeasible problem. Since the solution space gets smaller with increasing utilization, we expect that our heuristic solution is the optimal solution more frequently. As alluded to by the discussion in Section 4.3.1, the tradeoff to increase the expected total service time in order to reduce the variance of the service time is extremely unfavorable at high utilizations. This explains why our heuristic performs better at very high levels of utilization.

Chapter 5

Batching strategy for multi-item, general demand production systems

Unlike the previous chapters, we are now concerned with batching decisions when we relax the unit demand constraint; that is, we no longer restrict the demand to be for one unit, but it can be any positive integer. As we have seen in Chapter 4, optimal batch sizing is very difficult even when we restrict the demand of each job to be one unit; we had to rely on enumeration to obtain the optimal vector of batch sizes. Even though we managed to reduce the search space by deriving lower and upper bounds on the optimal batch size for every job type, enumeration, in general, is a very time-consuming method. Under general demand conditions, deriving an optimal batching strategy is even more difficult due to the following reasons. First, we do not have a closed-form expression for the expected total job time $E[T]$. Furthermore, because the demand of a batch may only be partially fulfilled after a single service, we are not only concerned with the initial batch size, but also the subsequent batch sizes used for re-service. These reasons, coupled with the fact that there are multiple job types, make both the derivation of an optimal strategy very difficult, and an enumeration approach highly impractical. In this chapter, we thus try to reduce the problem complexity, and propose some simpler, yet effective heuristics. Since the minimize utilization heuristic was shown to perform very well in the unit demand case, it is natural to explore how it will perform in this more general setting.

5.1 Minimize Utilization Heuristic

As we have shown, the minimize utilization heuristic aims to minimize the expected total service time for every job request ($\bar{s}_j$). This means that even with general demand, we can still size the batches for each job type individually. Recall that in the previous chapter, we showed that when demand is restricted to a single unit, a bisection line-search can be employed to obtain the batch size that minimizes $\bar{s}_j$. However, this method will not work with general demands. In this section, we demonstrate how to obtain the batch sizes via dynamic programming (DP). For each of the J job types, we will use the same DP to determine the batch sizes, so we will describe the procedure for the general job type. For ease of notation, we have omitted the subscript j in our discussion. First, we make the following definitions:

D the number of units demanded by the customer
N the *initial* batch size that the machine uses for the first service

We also introduce $T_{D,N}$, defined as the expected total amount of time taken by the machine to meet a job demand of D, given that the initial batch size used for the first service is N. The minimum expected time required is denoted by T_D^*. Mathematically,

$$T_D^* = \min_N T_{D,N} \qquad (5.1)$$

The choice, therefore, is to choose the optimal initial batch size, such that we attain T_D^*. If we define the optimal initial batch size as N^*, then we have,

$$T_{D,N^*} = T_D^* \qquad (5.2)$$

The sequence of events for each job is as follows: the job arrives at the server, and the machine processes a batch of some initial size to meet the job demand. Depending on the outcome of the production, the demand is either fully or partially realized. If re-service is required, the machine then needs to decide on the new batch size. This cycle of re-services continues until the demand is met.

After the initial batch is processed, let the number of good units produced be denoted by the random variable Y. Following our assumption of yield uncertainty, Y is a binomial random variable, with parameters N, and $1-\beta$. After observing the actual yield y, we assume that we know the remaining minimum expected service time to satisfy the outstanding demand. Using our previous definition (Equation 5.1), this quantity is T^*_{D-y}. We can therefore express N^* as follows:

$$N^* = \arg\min_{N \geq D} \{\tau + N\alpha + \sum_{y=0}^{N} \binom{N}{y} \beta^{N-y}(1-\beta)^y T^*_{D-y}\} \tag{5.3}$$

Note that in our formulation, we only need to consider batch sizes $N \geq D$. The reasoning for this is as such: If we choose an initial batch size $N_1 < D$, then we will certainly be unable to satisfy the demand since $Y \leq N_1 < D$. Then, we will definitely require a second batch of units of some size, N_2, to satisfy the $D-Y$ units of unsatisfied demand. In this case, if we had used an original batch size of $N_1 + N_2$, we would have reduced the total time by τ. Since this argument holds for any values of N_1 and N_2, we are always better off selecting our initial batch size to be larger than D.

We also observe that $T^*_{D-y} = 0$ for all $y \geq D$; then we can write Equation 5.3 in a more concise form:

$$N^* = \arg\min_{N \geq D} \{\tau + N\alpha + \sum_{y=0}^{D-1} \binom{N}{y} \beta^{N-y}(1-\beta)^y T^*_{D-y}\} \tag{5.4}$$

All that remains is to determine the values of T^*_{D-y} for $y < D$.

Our base case occurs when the remaining demand is for a single unit. After the initial service time, the resultant demand after this service is either "1 or 0"; thus, we can write the following recursion:

$$\begin{aligned}
T_{1,N} &= \tau + N\alpha + T_{1,N} \times \Pr(Y=0) \\
T_{1,N} &= \tau + N\alpha + T_{1,N} \times \beta^{N} \\
T_{1,N}\left(1-\beta^{N}\right) &= \tau + N\alpha \\
T_{1,N} &= \frac{\tau + N\alpha}{1-\beta^{N}}
\end{aligned} \tag{5.5}$$

Given Equation 5.5, we can evaluate $T_{1,N}$ for $N = 1,2,\ldots,N_{\max}$, where $N_{\max}$ is some user-defined "guess" of the largest possible batch size that would be required. A search can be performed across all the calculated values of $T_{1,N}$ to find T_1^*. From Equation 5.4 when $D = 2$, we see that once T_1^* is found, then we can write a closed-form expression for $T_{2,N}$ similar to Equation 5.5. We can proceed in the same fashion for any amount of demand.

In the general case for D units of demand, we have:

$$\begin{aligned}
T_{D,N} &= \tau + N\alpha + T_{D,N} \times P(Y=0) + T_{D-1}^{*} \times P(Y=1) + \ldots + T_1^{*} \times P(Y=D-1) \\
T_{D,N} &= \tau + N\alpha + T_{D,N} \times \beta^{N} + \sum_{y=1}^{D-1} \binom{N}{y} (1-\beta)^{y} \beta^{N-y} T_{D-y}^{*} \\
T_{D,N}\left(1-\beta^{N}\right) &= \tau + N\alpha + \sum_{y=1}^{D-1} \binom{N}{y} (1-\beta)^{y} \beta^{N-y} T_{D-y}^{*} \\
T_{D,N} &= \frac{\tau + N\alpha + \sum_{y=1}^{D-1} \binom{N}{y} (1-\beta)^{y} \beta^{N-y} T_{D-y}^{*}}{1-\beta^{N}}
\end{aligned} \tag{5.6}$$

To implement the dynamic programming method to find batch sizes for the minimum utilization heuristic, we use the following table:

		Batch Size							
		1	2	$\cdots$	$D-1$	D	$D+1$	$\cdots$	$N_{\max}$
Demand	1	$T_{1,1}$	$T_{1,2}$	$\cdots$	$T_{1,D-1}$	$T_{1,D}$	$T_{1,D+1}$	$\cdots$	$T_{1,N_{\max}}$
	2	–	$T_{2,2}$	$\cdots$	$T_{2,D-1}$	$T_{2,D}$	$T_{2,D+1}$	$\cdots$	$T_{2,N_{\max}}$
	$\vdots$	–	–	$\ddots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\vdots$
	$D-1$	–	–	–	$T_{D-1,D-1}$	$T_{D-1,D}$	$T_{D-1,D+1}$	$\cdots$	$T_{D-1,N_{\max}}$
	D	–	–	–	–	$T_{D,D}$	$T_{D,D+1}$	$\cdots$	$T_{D,N_{\max}}$

The table is filled from the top row downwards. Once all the values in the row have been computed, we search the row for the smallest value. The value of any entry in a particular row is then a function of the optimal values of all the previous rows. We show results for an example ($\tau = 0.5$, $\alpha = 0.126$, $\beta = 0.35$, and $D = 4$) in the following two tables.

Table 5.1: Example of the Dynamic Programming Table

		Batch Size									
		1	2	3	4	5	6	7	8	9	10
Demand	1	0.963	**0.857**	0.917	1.018	1.135	1.257	1.381	1.507	1.632	1.758
	2	-	1.301	1.130	**1.115**	1.177	1.275	1.389	1.510	1.633	1.758
	3	-	-	1.592	1.415	**1.346**	1.362	1.431	1.529	1.642	1.762
	4	-	-	-	1.857	1.694	1.593	**1.568**	1.605	1.681	1.782

Table 5.2: Batch Sizing Results

Demand	Expected total time	Batch size to use
1	0.8565	2
2	1.1154	4
3	1.3455	5
4	1.5683	7

Another advantage of using the table is that we can easily find the batch sizes to use at every stage of re-service. Here, we start with an initial batch of 7 units. After the batch is completed, the user can observe the number of good units produced, and the table immediately allows the user to know what that next batch size should be. If, for example, 2 good units were produced after the first service, then the remaining demand is 2 and the policy selects the next batch to be of size 4.

5.2 Performance of the Heuristic

Since we do not have lower and upper bounds on the optimal batch sizes, searching for the optimal policy to determine the initial batch sizes for each job type is an extremely difficult integer combinatorial problem. Also, because of yield uncertainty, the demand of a batch may not be entirely fulfilled after a single service. As a result, the outstanding demand necessitates yet another batch sizing decision. Immediately, one sees that even for a single job type, there exist an extremely huge number of possible combinations of initial and subsequent batch sizes. The difficulty of the problem is further augmented because we have multiple job types, and the batch sizing of one job type cannot be done independently of others.

In addition, the fact that we do not have a closed-form expression for the expected total time formula means that we have to rely on simulations to compare policies. Clearly, the multitude of possible combinations, coupled with the requirement of simulations, makes optimal batch sizing very computationally intensive, if not infeasible. Nevertheless, we seek to compare the effectiveness of our heuristic against two other batching policies, which at first glance, would be expected to work rather well.

The first is an "expected value" policy. Here, for each batch sizing decision, be it during an initial service or a re-service, we set the batch size such that the expected number of non-defective units produced from the batch is just sufficient to meet the demand. Mathematically, for every job type j, the batch size used is

$$N = \left\lceil \frac{D}{(1-\beta)} \right\rceil \tag{5.7}$$

The second policy sizes the batches such that the probability of satisfying the demand from the batch is at least some target probability $\omega_{threshold}$. We will call this policy the "threshold probability" policy. In other words, N is the minimum batch size such that,

$$\sum_{m=D}^{N} \binom{N}{m} (1-\beta)^m \beta^{N-m} \geq \omega_{threshold} \tag{5.8}$$

When we test this policy, we need to specify the value of $\omega_{threshold}$. It is not immediately clear what the optimal value should be, therefore in our simulations, we use a trial and error approach and test for values of $\omega_{threshold}$ over a certain range.

To compare the minimize utilization heuristic against the two alternatives, we ran 100 simulations each at various test utilization levels.

Test routine for each simulation:

1. Each simulation comprises 10 different job types, where the input parameters of each job type are randomly chosen as such:

   τ_j uniformly chosen over [0,1]

   $\frac{1}{\alpha_j}$ uniformly chosen over [5,25]

   β_j uniformly chosen over [0.1,0.9]

   λ_j uniformly chosen over [0,1]

   D_j uniformly chosen over [1,10]

2. The specified level of utilization ρ_{test}, is calculated based on our minimize utilization heuristic solution. Since our heuristic works to minimize utilization and is independent of λ_j, this provides us with a convenient method of controlling ρ. Using the DP described in the previous section, we are able to calculate the minimum expected service time $\bar{s}_j$ for each job type j. Having obtained the minimum values of $\bar{s}_j$, we multiply each λ_j by a scaling factor, such that $\sum_{j=1}^{J} \lambda_j \bar{s}_j = \rho_{test}$.

3. We generate 500 job arrivals, where each job arrival has a probability of $\frac{\lambda_j}{\lambda}$ of being of type j. The arrival times of each job i ($arrival\_time_i$) are recorded so that we can calculate the total time that a job spends in the system.

4. For each job, we model the yield realization *a priori*, before we simulate any of the policies. We do this so that each policy will experience the same yield outcomes for each job. We create the yield realization via a binary string, which we call a "good-bad" string. This string is essentially a string of '1's and '0's, and is used to model the yield uncertainty associated with the production of items. For a job of particular job type j, we perform a series of coin flip experiments, where we either obtain a non-defective unit with probability $(1-\beta_j)$, or a defective unit with probability β_j. Each trial models a unit that is produced by the machine, and we perform sufficient trials until the demand is fully satisfied.

   The reason why we need to decide on the realized yield before testing any of the policies is to eliminate the randomness associated with the binomial yield distribution. If this was not done, then a less optimal policy could in fact perform better in the simulation simply due to the unpredictability of the yield.

5. Given the pre-determined "good-bad" binary strings for all 500 jobs, we then test the various policies to determine their performance. We record the completion times for each job i under each policy k ($completion\_time_{i,k}$) are also recorded.

6. We calculate the total time that each job spends in the system.

$$total\_time_{i,k} = (completion\_time_{i,k} - arrival\_time_i)$$

7. Then, we calculate the average time that a job spends in the system for policy k.

$$average\_time_k = \frac{1}{450}\left(\sum_{i=51}^{500} total\_time_{i,k}\right)$$

   Note that here, we use the first 50 jobs to allow the system to attain steady-state, and so have neglected the times for these jobs in the calculation.

8. To further eliminate any randomness due to the yield realization, for each simulation of 500 job arrivals, we repeat Steps 3-6 an additional 49 times. Essentially, this generates 50 sets of "good-bad" strings per simulation so that we have a good mix of possible yield realizations to test the policies. Each set consists 500 strings, with each string corresponding to the yield realization of that particular job. The mean of the average job times over the 50 sets is then considered to be the average job time of the simulation.

The test routine we have just described is for a single simulation. To fully test the policies under a variety of job parameters, we perform 100 simulations, which would give a better indication of policy efficiency. In other words, for a value of test utilization ρ_{test}, we repeat the test routine described above 100 times. For each simulation (test routine), we calculate $average\_time_k$ for the minimize utilization heuristic, the "expected value" policy, as well as the “threshold probability” policy. We compared the performance of the “expected value” policy and the “threshold probability” policy against that of the minimize utilization heuristic at several test utilization levels, and the increase in average job times are recorded in Table 5.3. The detailed test results at each test utilization level can be found in Appendix D.

Table 5.3: Average Percentage Increase in Expected Job Time

	Test Utilization Levels ρ_{test}				
	0.5	0.6	0.7	0.8	0.9
Policy 1	5.32%	7.08%	8.13%	12.95%	22.39%
Policy 2(a) $\omega_{threshold} = 0.6$	8.49%	11.54%	14.12%	21.61%	37.17%
Policy 2(b) $\omega_{threshold} = 0.65$	7.07%	9.47%	11.71%	18.50%	33.13%
Policy 2(c) $\omega_{threshold} = 0.7$	7.08%	8.64%	11.20%	18.54%	33.00%
Policy 2(d) $\omega_{threshold} = 0.75$	8.03%	9.10%	13.03%	22.88%	40.62%
Policy 2(e) $\omega_{threshold} = 0.8$	10.73%	11.49%	17.39%	31.67%	58.20%
Policy 2(f) $\omega_{threshold} = 0.85$	15.53%	16.84%	26.19%	49.21%	92.66%
Policy 2(g) $\omega_{threshold} = 0.9$	24.88%	26.85%	44.20%	91.04%	169.66%
Policy 1 – “Expected value” policy Policy 2 – “Threshold probability” policy					

From the results, we also see that if the batch size has to be selected according to the “threshold probability” policy, then the optimal value of $\omega_{threshold}$ is around 70% (Policy 2(c)). However, one should note that the performance of the “threshold probability” policy is extremely sensitive to the selection of $\omega_{threshold}$. Given the difficulty in selecting the threshold probability, as well as the poor performance of this policy as seen from Table 5.3, we would not advocate the adoption of the “threshold probability” policy. The increase in performance of Policy 1 and 2(c) as compared to the minimize utilization heuristic are plotted in Figure 5.1.

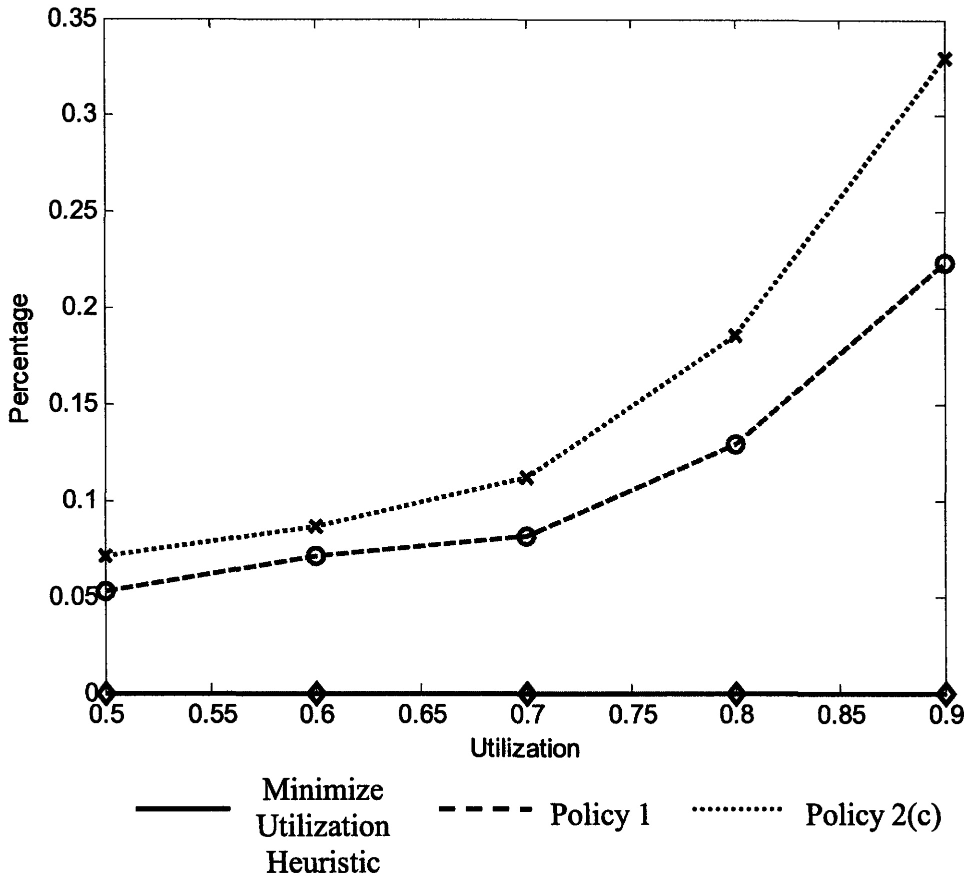

Figure 5.1: Average Increase in Expected Job Time for Policies 1 and 2(c)

From Figure 5.1, it is evident that the minimize utilization heuristic performs better than the alternative policies. The reason for this is because both the "expected value" and "threshold probability" policies do not make use of all the job input parameters in determining the batch size. While these alternative policies only require the demand and failure probability information, the minimize utilization heuristic uses all the job parameters to size the batches. Consequently, the batch sizes from the alternative policies perform less well than those obtained via the minimum utilization heuristic.

Another observation is that production systems are more "forgiving" at low levels of utilization, and non-optimal policies suffer a smaller increase in expected job time. This is because at high utilization levels, jobs spend more time waiting for service, hence any delay in service to the jobs earlier in the queue will cause a greater propagation of delay up the queue. Therefore, the performance of non-optimal policies gets increasingly worse as utilization goes up.

5.3 Modified Minimize Utilization Heuristic

In this section, we start with the solution obtained from our heuristic, and attempt to find better policies by modifying the approach. The minimize utilization heuristic solves for the batch sizes to J individual problems. For every job type j, we solve the following minimization problem:

$$\min_{n_j} \bar{s}_j$$

Instead of using the expected service time as the objective, we propose a modification which, in addition, attempts to reduce the number of services of a batch. This revised objective is driven by the aim to reduce the amount of setups required. Following the notation in Chapter 2, we let the number of services required by a batch until its demand is completely satisfied be denoted by the random variable M . Therefore, the new optimization problem for each job type j is instead,

$$\min_{n_j} \bar{s}_j + \hat{w}_j E[M]$$

where $\hat{w}_j$ is a weight factor.

Alternatively, since each round of service incurs a setup time, then the problem can be rewritten as

$$\min_{n_j} \bar{s}_j + w_j E[\mathit{total\ setup\ time}]$$

where the new weight factor $w_j = \hat{w}_j / \tau$.

To solve this optimization problem, observe that it is possible to leverage the dynamic programming formulation in Section 5.1. Instead of using the setup time τ_j for the job type j, we now use the weighted setup time $(1+w_j)\tau_j$.

Immediately, we see that the minimize utilization heuristic that is detailed in Section 5.1 is a special case, where we set $w_j = 0$ for all j. Also if we set $w_j >> 1$, then re-services are extremely undesirable, since we incur very large setup time penalties. This will then have the effect of selecting extremely large batch sizes in order to minimize the probability of re-service. Therefore, the challenge is to select values of w_j that will give us better policies. However, the difficulty lies in selecting the appropriate weights, and this is further complicated by the fact that w_j need not be the same for every job type.

In the first round of analysis, we make the assumption that w_j is the same across all the job types. Therefore, we are searching for a single weight factor w. While we would like to make the weight factor large, so as to drive down the number of services required, this in turn will cause the system utilization of the system to increase, and lead to an increase in the average job time. To see how the system utilization changes with respect to the weight factor, we plot in Figure 5.2 an example in which 10 job types were used. The job parameters for the 10 job types were randomly generated, and are recorded in Table 5.4.

Table 5.4: Input Parameters for Ten Job Types

	τ_j	α_j	β_j	λ_j	D_j
Job Type 1	0.9501	0.0577	0.1463	0.0021	9
Job Type 2	0.2311	0.0480	0.3823	0.1010	1
Job Type 3	0.6068	0.0427	0.7505	0.0602	7
Job Type 4	0.4860	0.0506	0.1079	0.1261	4
Job Type 5	0.8913	0.1173	0.2111	0.0631	9
Job Type 6	0.7621	0.0763	0.2622	0.0566	6
Job Type 7	0.4565	0.0422	0.2590	0.1145	8
Job Type 8	0.0185	0.0428	0.5830	0.0711	5
Job Type 9	0.8214	0.0757	0.3178	0.0274	4
Job Type 10	0.4447	0.0437	0.2591	0.0909	2

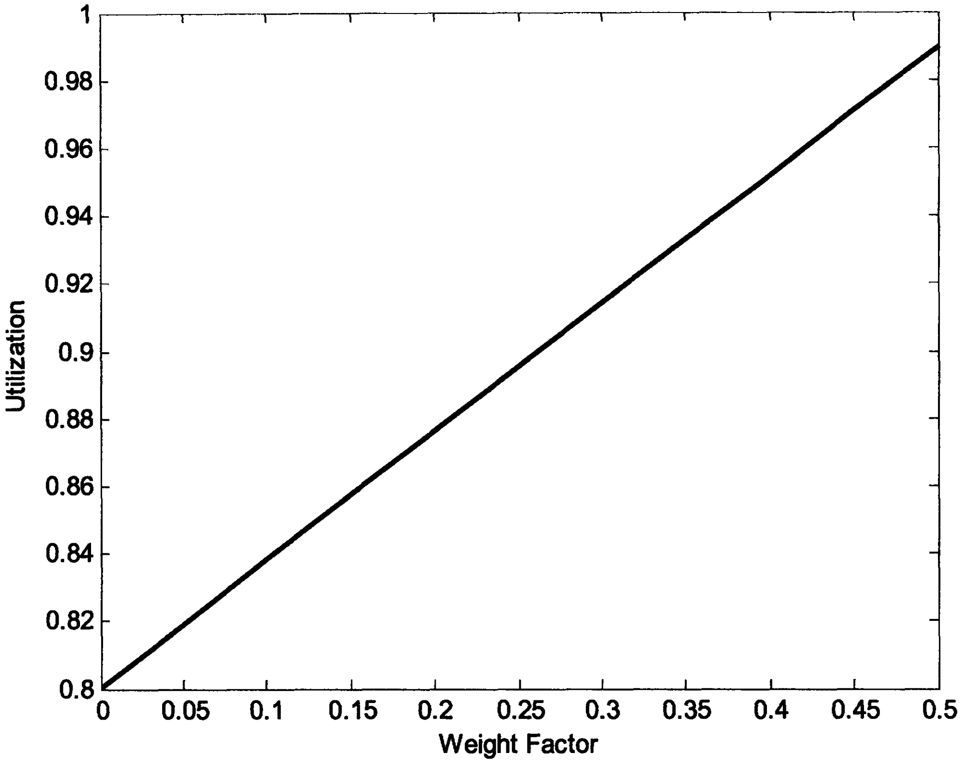

Figure 5.2: System Utilization as a Function of Weight Factor

From Figure 5.2, it would seem that utilization is linear with respect to the weight factor; nevertheless, the relationship is not linear. We elaborate on this below.

From Equation 5.6, we have derived for all demand D, and batch sizes $N \geq D$

$$T_{D,N} = \frac{\tau + N\alpha + \sum_{y=1}^{D-1} \binom{N}{y} (1-\beta)^y \beta^{N-y} T^*_{D-y}}{1-\beta^N}$$

If N^* is indeed invariant of τ, then the following argument holds:

1. We see that T_1^* is an affine function of τ.
2. Observing $T_{D,N}$ for any demand D, we can see that if T^*_{D-m} are affine functions of τ, for all $m = 1,2,\ldots,D-1$, then in fact $T_{D,N}$ is also an affine function of τ. T_D^*

is simply equal to $T_{D,N}$ evaluated at the optimal batch size, so it will also be an affine function of τ.

3. By induction, since T_1^* is an affine function of τ, then so is T_D^* for any value of D.
4. T_D^* being an affine function of τ implies that it is an affine function of weight factor also.
5. By repeating this induction argument for all job types, we then have that utilization is perfectly linear with respect to the weight factor. (i.e. Figure 5.2 above will be a perfectly straight line)

In reality, N^* is not invariant to τ. This was observed in Chapter 3.4.1, where we discovered that as τ increases, N^* increases so as to reduce the number of services. This implies that the variable time component of $T_{D,N}$ (the time components that are affected by the batch size) makes up a larger percentage of $T_{D,N}$. At larger values of τ, the same weight factor w results in a smaller increase in $T_{D,N}$ (and therefore also T_D^*) as compared to at smaller values of τ. Therefore, Figure 5.2 is not perfectly linear, but is slightly concave downward (the rate of change, or the second derivative is negative). However, what we observe is that the curvature is very slight, and a linear approximation is actually very accurate. It is this linear approximation that we will exploit when choosing the weight factor w.

Our method of choosing w is as such: Given the test utilization level ρ_{test}, let our guess of the value of utilization of the system when the optimal policy is implemented be ρ_{opt_guess}. We emphasize that ρ_{opt_guess} is simply a guess, and may not be accurate. However, we are able to make reasonably good guesses from the results obtained in Chapter 4, where we have seen that the optimal utilization is usually not much larger than the minimum utilization. Using the straight-line approximation to the relationship between the system utilization and the weight factor, we then calculate the corresponding value of w.

For each test case we have two utilization levels – the test utilization level ρ_{test}, and our guess of the optimal utilization level ρ_{opt_guess}. For each test combination of $(\rho_{test}, \rho_{opt\_guess})$, we run 100 simulations like those described in Section 5.2. We calculate and record the average percentage change in expected job time with weighted setup times compared to the minimize utilization heuristic. These results are reported in Table 5.5, and can be interpreted as follows: a positive entry in the table indicates that the average performance using weighted setup times is worse than when the original setup times (minimize utilization heuristic) was used. Conversely, a negative entry indicates that the average performance with weighted setup times is worse than that of the minimize utilization heuristic. For example, at $\rho_{test} = 0.6$, if we estimate the optimal system utilization ρ_{opt_guess} to be 0.64, we achieve an expected job time that is 0.13% smaller on average. The actual simulation results can be found in Appendix E.

Table 5.5: Average Percentage Change in Expected Job Time

	$\rho_{opt\_guess} - \rho_{test}$				
	0.01	0.02	0.03	0.04	0.05
$\rho_{test} = 0.1$	**+0.16%**	+0.64%	+1.18%	+1.80%	+2.45%
$\rho_{test} = 0.2$	**0.00%**	+0.13%	+0.33%	+0.56%	+0.84%
$\rho_{test} = 0.3$	**-0.05%**	0.00%	+0.06%	+0.25%	+0.36%
$\rho_{test} = 0.4$	-0.06%	**-0.08%**	-0.03%	+0.05%	+0.11%
$\rho_{test} = 0.5$	-0.06%	**-0.09%**	-0.06%	-0.07%	-0.02%
$\rho_{test} = 0.6$	-0.09%	-0.08%	-0.10%	**-0.13%**	-0.10%
$\rho_{test} = 0.7$	-0.06%	-0.11%	**-0.14%**	-0.09%	-0.01%
$\rho_{test} = 0.8$	-0.06%	**-0.11%**	-0.08%	-0.09%	+0.04%
$\rho_{test} = 0.9$	-0.08%	-0.10%	**-0.13%**	0.00%	+0.11%

From Table 5.5, we see that at low values of system utilization ($\rho = 0.1 - 0.4$), as well as high values ($\rho > 0.8$), the best choice of utilization is frequently around the minimum utilization. The reasons for this are very intuitive, and were already highlighted in Section 4.3.4. At low levels of utilization, the server is idle most of the time, so most jobs do not spend much time waiting in the queue. Therefore, minimizing the service time, which is equivalent to minimizing the utilization, is actually a very good heuristic. In fact, such an approach seems to be optimal most of the time. At high utilizations, a slight increase in utilization results in a very large increase in expected job time. Hence, the optimal utilization level tends towards the minimum utilization.

It is at intermediate system utilization levels that we see a greater deviation from the minimum utilization level. However, the key observation is that even in these cases where there is more "freedom" in batch sizing, the performance of the DP policies that use weighted setup times is not significantly better than the original minimize utilization heuristic. From the simulation results, the largest improvement in policy performance that we found is not even 1%; indeed it is much smaller. In addition, because we do not know the true optimal utilization, the value of ρ_{opt_guess}, and therefore the weight factor w, that we use are at best, a good estimate. Given the difficulty in estimating the optimal utilization, we conclude that we are not much better off by using weighted setup times in the DP with a single weight factor w.

Alternatively, we consider the case where we can assign different weight factors w_j to the different job types. Similar to the case when we use the same weight factor w for all the job types, we guess the optimal utilization level ρ_{opt_guess}. Therefore, the difference in utilizations $(\rho_{opt\_guess} - \rho_{test})$, has to be divided among the J job types. We calculate the increase in utilization attributed to job type j using the following formula:

$$(\rho_{opt\_guess} - \rho_{test})\frac{\lambda_j \bar{s}_j}{\sum_{i=1}^{J} \lambda_i \bar{s}_i} \tag{5.9}$$

The intuition is that if, compared to other job types, job type j has a larger job arrival rate, or a longer expected service time, then the system spends more time servicing these type of jobs. Correspondingly, we should place more priority on reducing the number of setups for these jobs. We then solve for the weight factors w_j using Equation 5.9 and the linear approximation relationship between utilization and w_j. However, from our simulations, we find that the performance when w_j is different is even worse than when we restrict w_j to be the same for all job types. With this analysis, we conclude that our minimize utilization heuristic works very well, and any marginal improvement that we may obtain from modifying the solution is so insignificant that it is not worth the computational effort and policy complexity.

(This Page Intentionally Left Blank)

Chapter 6

Conclusion

In this thesis, we have explored the batch sizing problem for a multi-item, general demand production system with uncertain yield, where the objective is to minimize the expected total time that a job spends in the system. To come up with an effective strategy, we first analyzed the scenarios when the system only processes jobs that require a single unit of demand. For these systems, we found that a bisection line-search algorithm was effective in obtaining solutions. Such solutions are proven to be optimal for the single-item, unit demand case, but can only be hypothesized to be optimal in the multi-item case. In addition, we observed that while minimizing the system utilization may not be a provably optimal batching strategy, such an approach performs very well with regard to the multi-item batching problem. The main advantage of this minimum utilization heuristic is that it decouples the interdependent batch sizing decisions such that we can size each job type individually. Our simulation results have also shown that although we are forsaking optimality for simplicity, the penalty (increase in expected job time) that is incurred is very small.

As an extension to this work, one might consider the cost of overproduction in determining a batching strategy. This may be an important concern, especially in the custom-product microfabrication industry, since these hi-tech products may be very expensive. Each excess item that is produced thus leads to significant loss in profit. For instance, we might desire to minimize the expected overproduction subject to a constraint on the expected job time. Here, we would model the time constraint by stating that the expected job time $E[T]$ must not exceed some threshold C. We discuss briefly two approaches that could be adopted for the multi-item, unit demand case.

The first method that could be considered is a greedy heuristic. Here, we start out at a feasible solution that satisfies the time constraint, and at every step, proceed to an alternative feasible solution (if it exists) that yields the best local reduction in expected overproduction. We outline the algorithm below:

Greedy Heuristic Algorithm:

1. Solve for the batch sizes either by using the bisection line-search algorithm, or the minimize utilization heuristic presented in Chapter 4.

2. For each job type j, we calculate the improvement I_j obtained, given that we decrease its batch size by 1.

$$I_j = \frac{\textit{decrease in } E[\textit{overproduction}]}{\textit{increase in } E[T]} \lambda_j$$

3. Select the job type that gives us the largest value of I_j, and decrease its batch size by 1. Obviously, two constraints must be satisfied at all times:

a) The batch size of every job type must be at least 1

b) By selecting the best job and decreasing its batch size, we must not violate the time constraint ($E[T] \leq C$)

Repeat Steps 2-3 until no feasible job type remains (algorithm terminates).

There are several advantages in using this greedy heuristic. First, it is easy to implement. Also, the resulting batch sizes obtained when the algorithm terminates are already integer-valued, and can be used immediately as a feasible policy without the need for branch-and-bound. However, the main disadvantage is that it is not guaranteed that the algorithm will terminate at an optimal batching policy.

On the other hand, we can choose to solve a new optimization problem. With the objective to minimize expected overproduction ($E[\textit{overproduction}]$), our integer optimization problem is then:

$$\begin{aligned} \min \quad & E[\mathit{overproduction}] \\ s.t. \quad & E[T] \le C \\ & n_j^* \ge 1 \end{aligned}$$

where we have denoted our variables as n_j^* as a reminder that they must be integers.

To solve this problem, we first relax the integrality constraints on the variables to obtain the following relaxed problem:

$$\begin{aligned} \min \quad & E[\mathit{overproduction}] \\ s.t. \quad & E[T] \le C \\ & n_j \ge 1 \end{aligned}$$

We can show that for the unit demand case, for each job type j, its expected overproduction is a convex function of n_j. Therefore, since the total expected overproduction is the sum of the J individual job type expected overproductions, it is convex with respect to the batch sizes.

In order to solve this optimization problem, we will make the assumption that the function $E[T]$ is convex with respect to the batch sizes. In Chapters 2 and 3, we have observed that the graphical plot of $E[T]$ appears convex, although we were not able to prove it mathematically.

By allowing the assumption that $E[T]$ is convex, we then see that the optimization problem is a constrained convex optimization problem. To solve this, we can take the Lagrangian dual of the LP relaxed problem, and then implement a gradient method to solve the dual. Alternatively, we can solve the relaxed problem directly by solving the Karush-Kuhn-Tucker (KKT) conditions.

The advantage of this second approach is that we may obtain better batching policies than compared to the greedy heuristic. However, there are several disadvantages to this approach. The main disadvantage is that by relaxing the integrality constraints on the batch sizes, we then need to use some rounding heuristic, or a branch-and-bound algorithm to obtain feasible integer batch sizes. Another issue that has to be considered is that we made the assumption that $E[T]$ is indeed convex with respect to the batch sizes.

Finally, we wish to highlight the following concern that may limit the usefulness of the two methods we have proposed. Although these two ideas can be adopted when we wish to incorporate overproduction into batching strategies for the multi-item, unit demand case, it is not immediately clear how to extend these methods to the cases when demand is allowed to be arbitrary. This is because in both methods, it is necessary to know the exact expression for the expected job time $E[T]$. As we have discussed in Chapter 2, this is mathematically tedious to compute when faced with general demands.

Appendix A

Lower and Upper Bounds on the Optimal Batch Size for Alternative Queueing Models

1. M/M/1

In this queueing model, the single service time X_j is exponentially distributed with mean $\bar{x}_j$. The variance of X_j is thus $\bar{x}_j^2$ and Equation 2.20 can be written as

$$E[T] = \frac{\sum_{j=1}^{J} \lambda_j \left(\frac{\bar{x}_j^2}{1-\beta_j^n} + \left(\frac{\bar{x}_j}{(1-\beta_j^n)} \right)^2 \left(1+\beta_j^n\right) \right)}{2\left(1-\sum_{j=1}^{J} \lambda_j \frac{\bar{x}_j}{1-\beta_j^n}\right)} + \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \frac{\bar{x}_j}{1-\beta_j^n}$$

$$E[T] = \frac{\sum_{j=1}^{J} \lambda_j \left(\bar{s}_j^2 \left(1-\beta_j^n\right) + \bar{s}_j^2 \left(1+\beta_j^n\right) \right)}{2\left(1-\sum_{j=1}^{J} \lambda_j \bar{s}_j\right)} + \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \bar{s}_j$$

$$E[T] = \frac{\sum_{j=1}^{J} \lambda_j \bar{s}_j^2}{2\left(1-\sum_{j=1}^{J} \lambda_j \bar{s}_j\right)} + \sum_{j=1}^{J} \frac{\lambda_j}{\lambda} \bar{s}_j \qquad \text{(A.1)}$$

Proposition 3: *The lower and upper bounds on the optimal integer batch size (n_j^*) of job type j are exactly the same.*

Proof. From Equation A.1, we see that to minimize $E[T]$, we simply have to minimize $\bar{s}_j$. In terms of lower and upper bounds, this implies that they are exactly the same. ■

An interesting point is that the mean service time $\bar{x}_j$ is not required to be an affine function of the batch size.

2. Modified M/E$_n$/1 queue

In this model, the service time of each individual unit in a batch is exponentially distributed with mean α_j. Also, we associate a fixed service time τ_j with every batch of type j. Said differently, X_j is a sum of n_j exponential random variables (n_j^{th}-order Erlang) that is shifted by a positive constant τ_j. The mean and variance of X_j are then,

$$\bar{x}_j = \tau_j + n_j\alpha_j$$

$$\sigma_{Xj}^2 = n_j\alpha_j^2$$

The $E[T]$ equation is thus expressed as

$$E[T] = \frac{\sum_{j=1}^{n}\lambda_j\left(\left(\frac{n_j\alpha_j^2}{\left(1-\beta_j^n\right)}\right)+\bar{s}_j^2\left(1+\beta_j^n\right)\right)}{2\left(1-\sum_{j=1}^{n}\lambda_j\bar{s}_j\right)}+\sum_{j=1}^{n}\frac{\lambda_j}{\lambda}\bar{s}_j \qquad \text{(A.2)}$$

Let n_1 be the integer batch size that obtains minimizes $\bar{s}_j$

Let n_2 be the integer batch size that minimizes $\left(\frac{n_j\alpha_j^2}{\left(1-\beta_j^n\right)}\right)+\bar{s}_j^2\left(1+\beta_j^n\right)$

Proposition 4: *The lower bound on the optimal integer batch size (n_j^*) is given by* $\min(n_1, n_2)$. *The upper bound is given by* $\max(n_1, n_2)$

Proof. By Lemma 2, we have established that the function $\bar{s}_j^2\left(1+\beta_j^n\right)$ is unimodal, and has only 1 minimum point.

We now show that the function $\frac{n_j\alpha_j^2}{\left(1-\beta_j^n\right)}$ is strictly increasing with batch size.

The first derivative is

$$\frac{d}{dn_j}\frac{n_j\alpha_j^2}{\left(1-\beta_j^n\right)} = \alpha_j^2\left(\frac{1}{\left(1-\beta_j^n\right)}+\frac{n_j\ln(\beta_j)\beta_j^n}{\left(1-\beta_j^n\right)^2}\right) \qquad \text{(A.3)}$$

It suffices to show that Equation A.3 is strictly positive for all positive batch sizes.

$$\alpha_j^2\left(\frac{1}{\left(1-\beta_j^n\right)}+\frac{n_j\ln(\beta_j)\beta_j^n}{\left(1-\beta_j^n\right)^2}\right)>0$$

$$(1-\beta_j^n)+n_j\ln(\beta_j)\beta_j^n>0$$

$$1>\beta_j^n-n_j\ln(\beta_j)\beta_j^n$$

$$1>\beta_j^n(1-n_j\ln(\beta_j)) \tag{A.4}$$

When $n_j=0$, then the RHS of Equation A.3 is exactly equal to 1. For positive batch sizes, the RHS is strictly less than 1. This can be proved by showing that the derivative of the RHS is strictly negative.

So, we know that $\left(\frac{n_j\alpha_j^2}{\left(1-\beta_j^n\right)}\right)+\bar{s}_j^2\left(1+\beta_j^n\right)$ is also unimodal, and has 1 minimum point.

Assume now that $n_1\le n_2$

For any integers smaller than n_1, $\bar{s}_j$ increases since n_1 is the integer that achieves the minimum value by definition.

Similarly, for any integers smaller than n_1, $\left(\frac{n_j\alpha_j^2}{\left(1-\beta_j^n\right)}\right)+\bar{s}_j^2\left(1+\beta_j^n\right)$ increases since the function is unimodal, and $n_1\le n_2$ by assumption.

Thus, we see that the value of $E[T]$ (as given by Equation A.2), evaluated at any batch size $n<n_1$ is strictly smaller than the expected total time if we use n_1. In this case, n_1 is a lower bound on the optimal integer batch size.

In the event that $n_1\ge n_2$, it is easy to see by symmetry arguments that n_2 is a lower bound.

Therefore $\min(n_1,n_2)$ is a lower bound on the optimal integer batch size.

The proof for the upper bound is very similar to the lower bound proof that we have just presented. ■

(This Page Intentionally Left Blank)

Appendix B

Lower and Upper Bounds, and Optimal Batch Sizes for Various Job Parameters

τ	Q	β	lower	upper	optimal
			bound	bound	(min E[T])
0.5	5	0.1	1	1	1
0.5	5	0.2	1	2	1
0.5	5	0.3	2	2	2
0.5	5	0.4	2	2	2
0.5	5	0.5	2	3	2
0.5	5	0.6	3	3	3
0.5	5	0.7	3	4	3
0.5	5	0.8	4	5	4
0.5	5	0.9	6	8	6
0.5	6	0.1	1	1	1
0.5	6	0.2	1	2	1
0.5	6	0.3	2	2	2
0.5	6	0.4	2	2	2
0.5	6	0.5	2	3	2
0.5	6	0.6	3	3	3
0.5	6	0.7	3	4	3
0.5	6	0.8	4	6	4
0.5	6	0.9	7	9	7
0.5	7	0.1	1	1	1
0.5	7	0.2	1	2	1
0.5	7	0.3	2	2	2
0.5	7	0.4	2	2	2
0.5	7	0.5	2	3	2
0.5	7	0.6	3	3	3
0.5	7	0.7	4	4	4
0.5	7	0.8	5	6	5
0.5	7	0.9	7	9	7
0.5	8	0.1	1	1	1
0.5	8	0.2	1	2	2
0.5	8	0.3	2	2	2
0.5	8	0.4	2	2	2
0.5	8	0.5	2	3	3
0.5	8	0.6	3	4	3
0.5	8	0.7	4	5	4
0.5	8	0.8	5	6	5
0.5	8	0.9	8	10	8

0.5	9	0.1	1	1	1
0.5	9	0.2	2	2	2
0.5	9	0.3	2	2	2
0.5	9	0.4	2	3	2
0.5	9	0.5	3	3	3
0.5	9	0.6	3	4	3
0.5	9	0.7	4	5	4
0.5	9	0.8	5	6	5
0.5	9	0.9	8	10	8
0.5	10	0.1	1	1	1
0.5	10	0.2	2	2	2
0.5	10	0.3	2	2	2
0.5	10	0.4	2	3	2
0.5	10	0.5	3	3	3
0.5	10	0.6	3	4	3
0.5	10	0.7	4	5	4
0.5	10	0.8	5	7	5
0.5	10	0.9	8	11	8
0.5	11	0.1	1	1	1
0.5	11	0.2	2	2	2
0.5	11	0.3	2	2	2
0.5	11	0.4	2	3	2
0.5	11	0.5	3	3	3
0.5	11	0.6	3	4	3
0.5	11	0.7	4	5	4
0.5	11	0.8	6	7	6
0.5	11	0.9	9	11	9
0.5	12	0.1	1	2	1
0.5	12	0.2	2	2	2
0.5	12	0.3	2	2	2
0.5	12	0.4	2	3	2
0.5	12	0.5	3	3	3
0.5	12	0.6	3	4	3
0.5	12	0.7	4	5	4
0.5	12	0.8	6	7	6
0.5	12	0.9	9	11	9
0.5	13	0.1	1	2	1
0.5	13	0.2	2	2	2
0.5	13	0.3	2	2	2
0.5	13	0.4	2	3	2
0.5	13	0.5	3	3	3
0.5	13	0.6	4	4	4
0.5	13	0.7	4	5	4
0.5	13	0.8	6	7	6
0.5	13	0.9	9	12	9
0.5	14	0.1	1	2	1
0.5	14	0.2	2	2	2
0.5	14	0.3	2	2	2
0.5	14	0.4	3	3	3
0.5	14	0.5	3	4	3
0.5	14	0.6	4	4	4
0.5	14	0.7	5	6	5
0.5	14	0.8	6	8	6
0.5	14	0.9	10	12	10
0.5	15	0.1	1	2	1
0.5	15	0.2	2	2	2

0.5	15	0.3	2	2	2
0.5	15	0.4	3	3	3
0.5	15	0.5	3	4	3
0.5	15	0.6	4	4	4
0.5	15	0.7	5	6	5
0.5	15	0.8	6	8	6
0.5	15	0.9	10	12	10
1	5	0.1	1	1	1
1	5	0.2	2	2	2
1	5	0.3	2	2	2
1	5	0.4	2	3	2
1	5	0.5	3	3	3
1	5	0.6	3	4	3
1	5	0.7	4	5	4
1	5	0.8	5	7	5
1	5	0.9	8	11	8
1	6	0.1	1	2	1
1	6	0.2	2	2	2
1	6	0.3	2	2	2
1	6	0.4	2	3	2
1	6	0.5	3	3	3
1	6	0.6	3	4	4
1	6	0.7	4	5	4
1	6	0.8	6	7	6
1	6	0.9	9	11	9
1	7	0.1	1	2	1
1	7	0.2	2	2	2
1	7	0.3	2	2	2
1	7	0.4	3	3	3
1	7	0.5	3	4	3
1	7	0.6	4	4	4
1	7	0.7	5	6	5
1	7	0.8	6	8	6
1	7	0.9	10	12	10
1	8	0.1	1	2	1
1	8	0.2	2	2	2
1	8	0.3	2	3	2
1	8	0.4	3	3	3
1	8	0.5	3	4	3
1	8	0.6	4	5	4
1	8	0.7	5	6	5
1	8	0.8	6	8	7
1	8	0.9	10	13	10
1	9	0.1	1	2	2
1	9	0.2	2	2	2
1	9	0.3	2	3	2
1	9	0.4	3	3	3
1	9	0.5	3	4	3
1	9	0.6	4	5	4
1	9	0.7	5	6	5
1	9	0.8	7	8	7
1	9	0.9	11	13	11
1	10	0.1	2	2	2
1	10	0.2	2	2	2
1	10	0.3	2	3	2
1	10	0.4	3	3	3

1	10	0.5	3	4	3
1	10	0.6	4	5	4
1	10	0.7	5	6	5
1	10	0.8	7	9	7
1	10	0.9	11	14	11
1	11	0.1	2	2	2
1	11	0.2	2	2	2
1	11	0.3	2	3	2
1	11	0.4	3	3	3
1	11	0.5	3	4	4
1	11	0.6	4	5	4
1	11	0.7	5	6	5
1	11	0.8	7	9	7
1	11	0.9	12	14	12
1	12	0.1	2	2	2
1	12	0.2	2	2	2
1	12	0.3	2	3	2
1	12	0.4	3	3	3
1	12	0.5	4	4	4
1	12	0.6	4	5	4
1	12	0.7	6	7	6
1	12	0.8	8	9	8
1	12	0.9	12	15	12
1	13	0.1	2	2	2
1	13	0.2	2	2	2
1	13	0.3	3	3	3
1	13	0.4	3	3	3
1	13	0.5	4	4	4
1	13	0.6	5	5	5
1	13	0.7	6	7	6
1	13	0.8	8	9	8
1	13	0.9	12	15	12
1	14	0.1	2	2	2
1	14	0.2	2	2	2
1	14	0.3	3	3	3
1	14	0.4	3	4	3
1	14	0.5	4	4	4
1	14	0.6	5	5	5
1	14	0.7	6	7	6
1	14	0.8	8	9	8
1	14	0.9	13	16	13
1	15	0.1	2	2	2
1	15	0.2	2	2	2
1	15	0.3	3	3	3
1	15	0.4	3	4	3
1	15	0.5	4	4	4
1	15	0.6	5	5	5
1	15	0.7	6	7	6
1	15	0.8	8	10	8
1	15	0.9	13	16	13
1.5	5	0.1	1	2	1
1.5	5	0.2	2	2	2
1.5	5	0.3	2	2	2
1.5	5	0.4	3	3	3
1.5	5	0.5	3	4	3
1.5	5	0.6	4	4	4

1.5	5	0.7	5	6	5
1.5	5	0.8	6	8	6
1.5	5	0.9	10	12	10
1.5	6	0.1	1	2	2
1.5	6	0.2	2	2	2
1.5	6	0.3	2	3	2
1.5	6	0.4	3	3	3
1.5	6	0.5	3	4	3
1.5	6	0.6	4	5	4
1.5	6	0.7	5	6	5
1.5	6	0.8	7	8	7
1.5	6	0.9	11	13	11
1.5	7	0.1	2	2	2
1.5	7	0.2	2	2	2
1.5	7	0.3	2	3	2
1.5	7	0.4	3	3	3
1.5	7	0.5	3	4	3
1.5	7	0.6	4	5	4
1.5	7	0.7	5	6	5
1.5	7	0.8	7	9	7
1.5	7	0.9	11	14	11
1.5	8	0.1	2	2	2
1.5	8	0.2	2	2	2
1.5	8	0.3	2	3	2
1.5	8	0.4	3	3	3
1.5	8	0.5	4	4	4
1.5	8	0.6	4	5	4
1.5	8	0.7	6	7	6
1.5	8	0.8	8	9	8
1.5	8	0.9	12	15	12
1.5	9	0.1	2	2	2
1.5	9	0.2	2	2	2
1.5	9	0.3	3	3	3
1.5	9	0.4	3	3	3
1.5	9	0.5	4	4	4
1.5	9	0.6	5	5	5
1.5	9	0.7	6	7	6
1.5	9	0.8	8	9	8
1.5	9	0.9	13	15	13
1.5	10	0.1	2	2	2
1.5	10	0.2	2	2	2
1.5	10	0.3	3	3	3
1.5	10	0.4	3	4	3
1.5	10	0.5	4	4	4
1.5	10	0.6	5	5	5
1.5	10	0.7	6	7	6
1.5	10	0.8	8	10	8
1.5	10	0.9	13	16	13
1.5	11	0.1	2	2	2
1.5	11	0.2	2	2	2
1.5	11	0.3	3	3	3
1.5	11	0.4	3	4	3
1.5	11	0.5	4	5	4
1.5	11	0.6	5	6	5
1.5	11	0.7	6	7	6
1.5	11	0.8	8	10	8

1.5	11	0.9	14	17	14
1.5	12	0.1	2	2	2
1.5	12	0.2	2	2	2
1.5	12	0.3	3	3	3
1.5	12	0.4	3	4	3
1.5	12	0.5	4	5	4
1.5	12	0.6	5	6	5
1.5	12	0.7	6	7	6
1.5	12	0.8	9	10	9
1.5	12	0.9	14	17	14
1.5	13	0.1	2	2	2
1.5	13	0.2	2	3	2
1.5	13	0.3	3	3	3
1.5	13	0.4	3	4	3
1.5	13	0.5	4	5	4
1.5	13	0.6	5	6	5
1.5	13	0.7	7	8	7
1.5	13	0.8	9	11	9
1.5	13	0.9	14	17	15
1.5	14	0.1	2	2	2
1.5	14	0.2	2	3	2
1.5	14	0.3	3	3	3
1.5	14	0.4	3	4	3
1.5	14	0.5	4	5	4
1.5	14	0.6	5	6	5
1.5	14	0.7	7	8	7
1.5	14	0.8	9	11	9
1.5	14	0.9	15	18	15
1.5	15	0.1	2	2	2
1.5	15	0.2	2	3	2
1.5	15	0.3	3	3	3
1.5	15	0.4	4	4	4
1.5	15	0.5	4	5	4
1.5	15	0.6	5	6	5
1.5	15	0.7	7	8	7
1.5	15	0.8	9	11	9
1.5	15	0.9	15	18	15
2	5	0.1	2	2	2
2	5	0.2	2	2	2
2	5	0.3	2	3	2
2	5	0.4	3	3	3
2	5	0.5	3	4	3
2	5	0.6	4	5	4
2	5	0.7	5	6	5
2	5	0.8	7	9	7
2	5	0.9	11	14	11
2	6	0.1	2	2	2
2	6	0.2	2	2	2
2	6	0.3	2	3	2
2	6	0.4	3	3	3
2	6	0.5	4	4	4
2	6	0.6	4	5	4
2	6	0.7	6	7	6
2	6	0.8	8	9	8
2	6	0.9	12	15	12
2	7	0.1	2	2	2

2	7	0.2	2	2	2
2	7	0.3	3	3	3
2	7	0.4	3	4	3
2	7	0.5	4	4	4
2	7	0.6	5	5	5
2	7	0.7	6	7	6
2	7	0.8	8	9	8
2	7	0.9	13	16	13
2	8	0.1	2	2	2
2	8	0.2	2	2	2
2	8	0.3	3	3	3
2	8	0.4	3	4	3
2	8	0.5	4	4	4
2	8	0.6	5	6	5
2	8	0.7	6	7	6
2	8	0.8	8	10	8
2	8	0.9	13	16	14
2	9	0.1	2	2	2
2	9	0.2	2	2	2
2	9	0.3	3	3	3
2	9	0.4	3	4	3
2	9	0.5	4	5	4
2	9	0.6	5	6	5
2	9	0.7	6	7	6
2	9	0.8	9	10	9
2	9	0.9	14	17	14
2	10	0.1	2	2	2
2	10	0.2	2	3	2
2	10	0.3	3	3	3
2	10	0.4	3	4	3
2	10	0.5	4	5	4
2	10	0.6	5	6	5
2	10	0.7	7	8	7
2	10	0.8	9	11	9
2	10	0.9	15	18	15
2	11	0.1	2	2	2
2	11	0.2	2	3	2
2	11	0.3	3	3	3
2	11	0.4	4	4	4
2	11	0.5	4	5	4
2	11	0.6	5	6	5
2	11	0.7	7	8	7
2	11	0.8	9	11	9
2	11	0.9	15	18	15
2	12	0.1	2	2	2
2	12	0.2	2	3	2
2	12	0.3	3	3	3
2	12	0.4	4	4	4
2	12	0.5	4	5	4
2	12	0.6	5	6	5
2	12	0.7	7	8	7
2	12	0.8	10	11	10
2	12	0.9	16	19	16
2	13	0.1	2	2	2
2	13	0.2	2	3	2
2	13	0.3	3	3	3

2	13	0.4	4	4	4
2	13	0.5	4	5	5
2	13	0.6	6	6	6
2	13	0.7	7	8	7
2	13	0.8	10	11	10
2	13	0.9	16	19	16
2	14	0.1	2	2	2
2	14	0.2	2	3	3
2	14	0.3	3	3	3
2	14	0.4	4	4	4
2	14	0.5	5	5	5
2	14	0.6	6	6	6
2	14	0.7	7	8	7
2	14	0.8	10	12	10
2	14	0.9	17	20	17
2	15	0.1	2	2	2
2	15	0.2	3	3	3
2	15	0.3	3	3	3
2	15	0.4	4	4	4
2	15	0.5	5	5	5
2	15	0.6	6	7	6
2	15	0.7	7	9	8
2	15	0.8	10	12	10
2	15	0.9	17	20	17
2.5	5	0.1	2	2	2
2.5	5	0.2	2	2	2
2.5	5	0.3	3	3	3
2.5	5	0.4	3	3	3
2.5	5	0.5	4	4	4
2.5	5	0.6	4	5	4
2.5	5	0.7	6	7	6
2.5	5	0.8	8	9	8
2.5	5	0.9	12	15	12
2.5	6	0.1	2	2	2
2.5	6	0.2	2	2	2
2.5	6	0.3	3	3	3
2.5	6	0.4	3	4	3
2.5	6	0.5	4	4	4
2.5	6	0.6	5	5	5
2.5	6	0.7	6	7	6
2.5	6	0.8	8	10	8
2.5	6	0.9	13	16	13
2.5	7	0.1	2	2	2
2.5	7	0.2	2	2	2
2.5	7	0.3	3	3	3
2.5	7	0.4	3	4	3
2.5	7	0.5	4	5	4
2.5	7	0.6	5	6	5
2.5	7	0.7	6	7	6
2.5	7	0.8	9	10	9
2.5	7	0.9	14	17	14
2.5	8	0.1	2	2	2
2.5	8	0.2	2	3	2
2.5	8	0.3	3	3	3
2.5	8	0.4	3	4	3
2.5	8	0.5	4	5	4

2.5	8	0.6	5	6	5
2.5	8	0.7	7	8	7
2.5	8	0.8	9	11	9
2.5	8	0.9	15	18	15
2.5	9	0.1	2	2	2
2.5	9	0.2	2	3	2
2.5	9	0.3	3	3	3
2.5	9	0.4	4	4	4
2.5	9	0.5	4	5	4
2.5	9	0.6	5	6	5
2.5	9	0.7	7	8	7
2.5	9	0.8	9	11	9
2.5	9	0.9	15	18	15
2.5	10	0.1	2	2	2
2.5	10	0.2	2	3	2
2.5	10	0.3	3	3	3
2.5	10	0.4	4	4	4
2.5	10	0.5	4	5	4
2.5	10	0.6	6	6	6
2.5	10	0.7	7	8	7
2.5	10	0.8	10	11	10
2.5	10	0.9	16	19	16
2.5	11	0.1	2	2	2
2.5	11	0.2	2	3	2
2.5	11	0.3	3	3	3
2.5	11	0.4	4	4	4
2.5	11	0.5	5	5	5
2.5	11	0.6	6	6	6
2.5	11	0.7	7	8	7
2.5	11	0.8	10	12	10
2.5	11	0.9	16	20	17
2.5	12	0.1	2	2	2
2.5	12	0.2	3	3	3
2.5	12	0.3	3	3	3
2.5	12	0.4	4	4	4
2.5	12	0.5	5	5	5
2.5	12	0.6	6	7	6
2.5	12	0.7	7	9	8
2.5	12	0.8	10	12	10
2.5	12	0.9	17	20	17
2.5	13	0.1	2	2	2
2.5	13	0.2	3	3	3
2.5	13	0.3	3	4	3
2.5	13	0.4	4	4	4
2.5	13	0.5	5	5	5
2.5	13	0.6	6	7	6
2.5	13	0.7	8	9	8
2.5	13	0.8	11	12	11
2.5	13	0.9	17	21	18
2.5	14	0.1	2	2	2
2.5	14	0.2	3	3	3
2.5	14	0.3	3	4	3
2.5	14	0.4	4	4	4
2.5	14	0.5	5	5	5
2.5	14	0.6	6	7	6
2.5	14	0.7	8	9	8

2.5	14	0.8	11	13	11
2.5	14	0.9	18	21	18
2.5	15	0.1	2	2	2
2.5	15	0.2	3	3	3
2.5	15	0.3	3	4	3
2.5	15	0.4	4	4	4
2.5	15	0.5	5	6	5
2.5	15	0.6	6	7	6
2.5	15	0.7	8	9	8
2.5	15	0.8	11	13	11
2.5	15	0.9	18	22	18
3	5	0.1	2	2	2
3	5	0.2	2	2	2
3	5	0.3	3	3	3
3	5	0.4	3	4	3
3	5	0.5	4	4	4
3	5	0.6	5	5	5
3	5	0.7	6	7	6
3	5	0.8	8	10	8
3	5	0.9	13	16	13
3	6	0.1	2	2	2
3	6	0.2	2	2	2
3	6	0.3	3	3	3
3	6	0.4	3	4	3
3	6	0.5	4	5	4
3	6	0.6	5	6	5
3	6	0.7	6	7	6
3	6	0.8	9	10	9
3	6	0.9	14	17	14
3	7	0.1	2	2	2
3	7	0.2	2	3	2
3	7	0.3	3	3	3
3	7	0.4	3	4	4
3	7	0.5	4	5	4
3	7	0.6	5	6	5
3	7	0.7	7	8	7
3	7	0.8	9	11	9
3	7	0.9	15	18	15
3	8	0.1	2	2	2
3	8	0.2	2	3	2
3	8	0.3	3	3	3
3	8	0.4	4	4	4
3	8	0.5	4	5	4
3	8	0.6	5	6	5
3	8	0.7	7	8	7
3	8	0.8	10	11	10
3	8	0.9	16	19	16
3	9	0.1	2	2	2
3	9	0.2	2	3	2
3	9	0.3	3	3	3
3	9	0.4	4	4	4
3	9	0.5	5	5	5
3	9	0.6	6	6	6
3	9	0.7	7	8	7
3	9	0.8	10	12	10
3	9	0.9	16	19	17

3	10	0.1	2	2	2
3	10	0.2	3	3	3
3	10	0.3	3	3	3
3	10	0.4	4	4	4
3	10	0.5	5	5	5
3	10	0.6	6	7	6
3	10	0.7	7	9	8
3	10	0.8	10	12	10
3	10	0.9	17	20	17
3	11	0.1	2	2	2
3	11	0.2	3	3	3
3	11	0.3	3	4	3
3	11	0.4	4	4	4
3	11	0.5	5	5	5
3	11	0.6	6	7	6
3	11	0.7	8	9	8
3	11	0.8	11	12	11
3	11	0.9	18	21	18
3	12	0.1	2	2	2
3	12	0.2	3	3	3
3	12	0.3	3	4	3
3	12	0.4	4	4	4
3	12	0.5	5	5	5
3	12	0.6	6	7	6
3	12	0.7	8	9	8
3	12	0.8	11	13	11
3	12	0.9	18	21	18
3	13	0.1	2	2	2
3	13	0.2	3	3	3
3	13	0.3	3	4	3
3	13	0.4	4	5	4
3	13	0.5	5	6	5
3	13	0.6	6	7	6
3	13	0.7	8	9	8
3	13	0.8	11	13	11
3	13	0.9	19	22	19
3	14	0.1	2	2	2
3	14	0.2	3	3	3
3	14	0.3	3	4	3
3	14	0.4	4	5	4
3	14	0.5	5	6	5
3	14	0.6	6	7	6
3	14	0.7	8	9	8
3	14	0.8	11	13	12
3	14	0.9	19	22	19
3	15	0.1	2	2	2
3	15	0.2	3	3	3
3	15	0.3	3	4	3
3	15	0.4	4	5	4
3	15	0.5	5	6	5
3	15	0.6	6	7	7
3	15	0.7	8	10	8
3	15	0.8	12	13	12
3	15	0.9	19	23	20

(This Page Intentionally Left Blank)

Appendix C

Test Results of Minimize Utilization Heuristic for Unit Demand

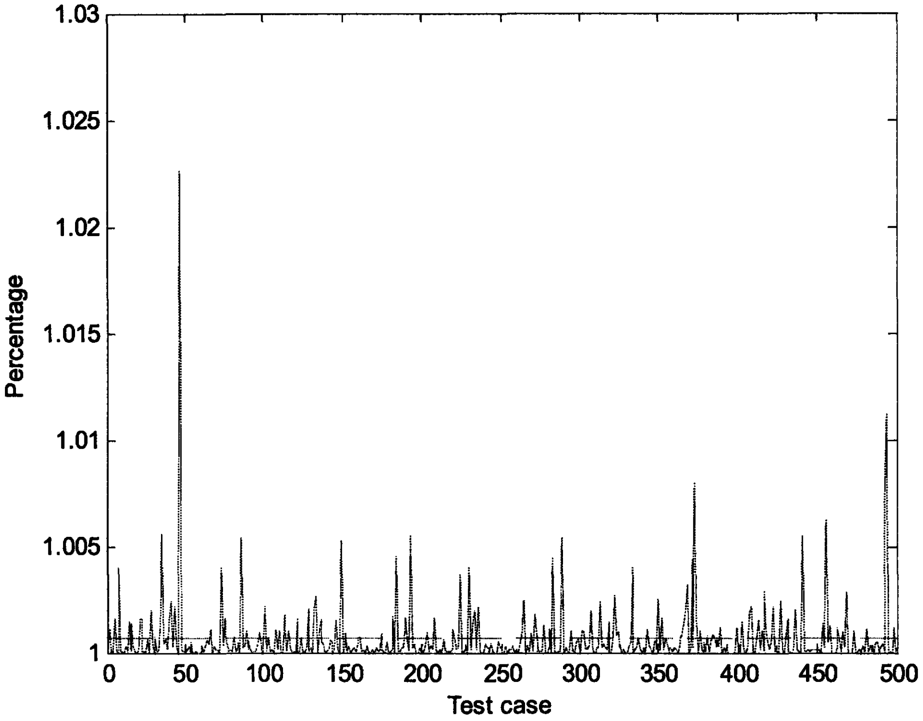

Figure C.1: Test Results at $\rho = 0.1$

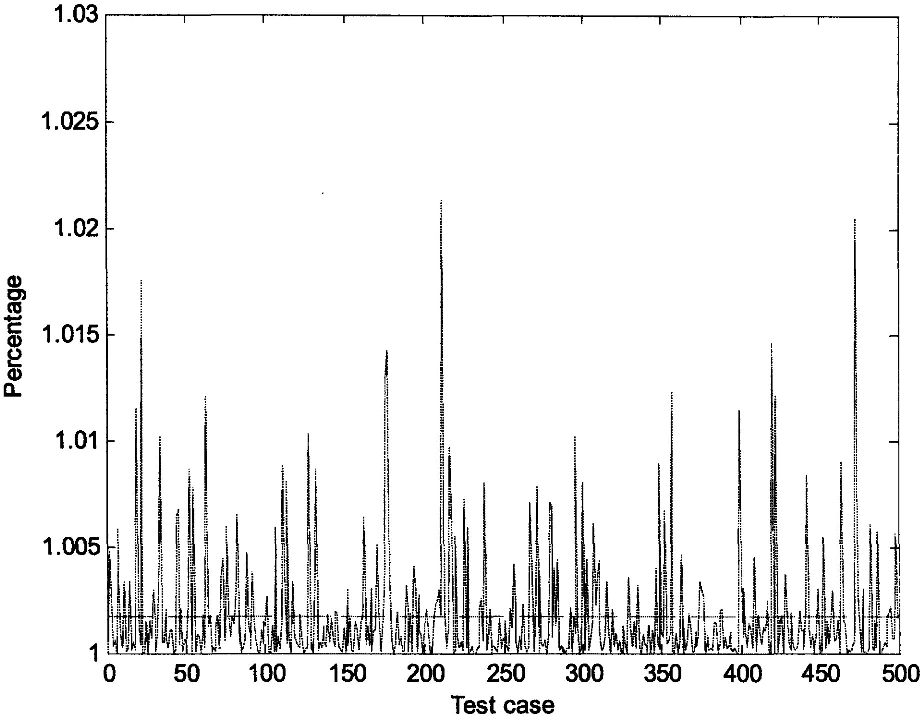

Figure C.2: Test Results at $\rho = 0.2$

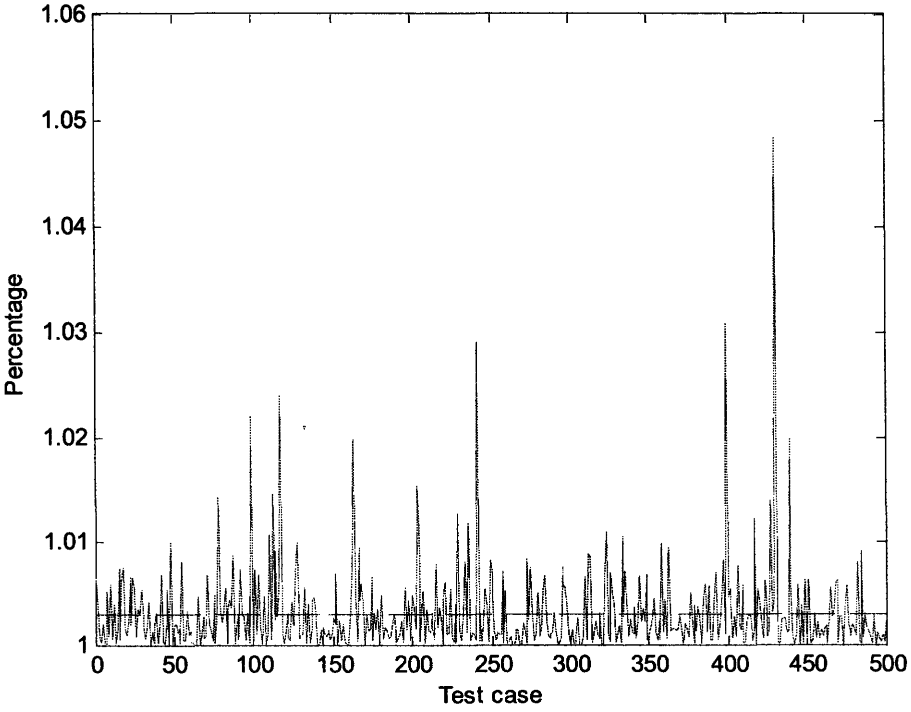

Figure C.3: Test Results at $\rho = 0.3$

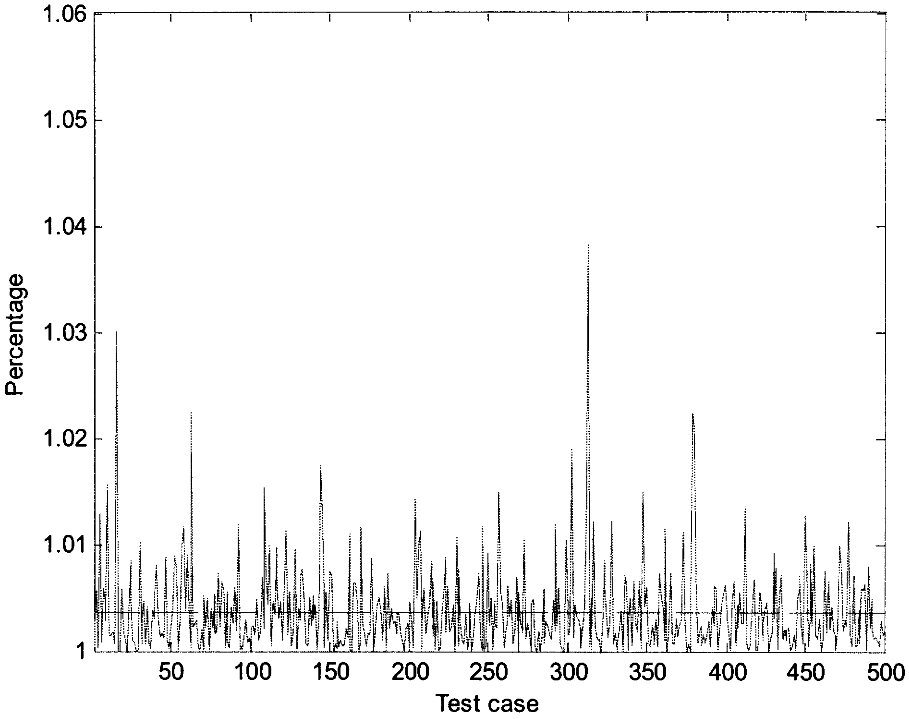

Figure C.4: Test Results at $\rho = 0.4$

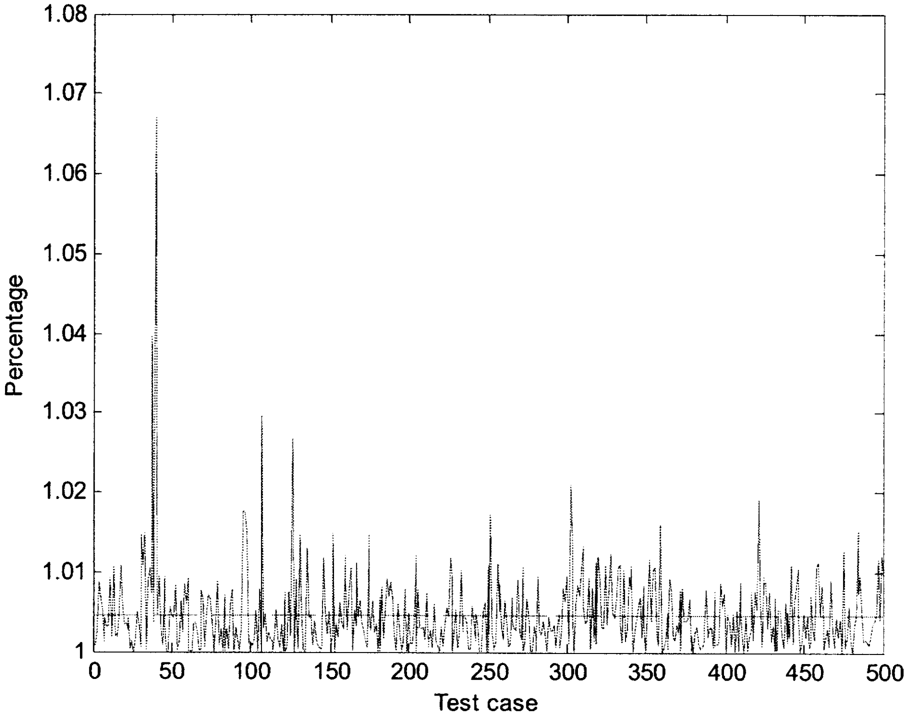

Figure C.5: Test Results at $\rho = 0.5$

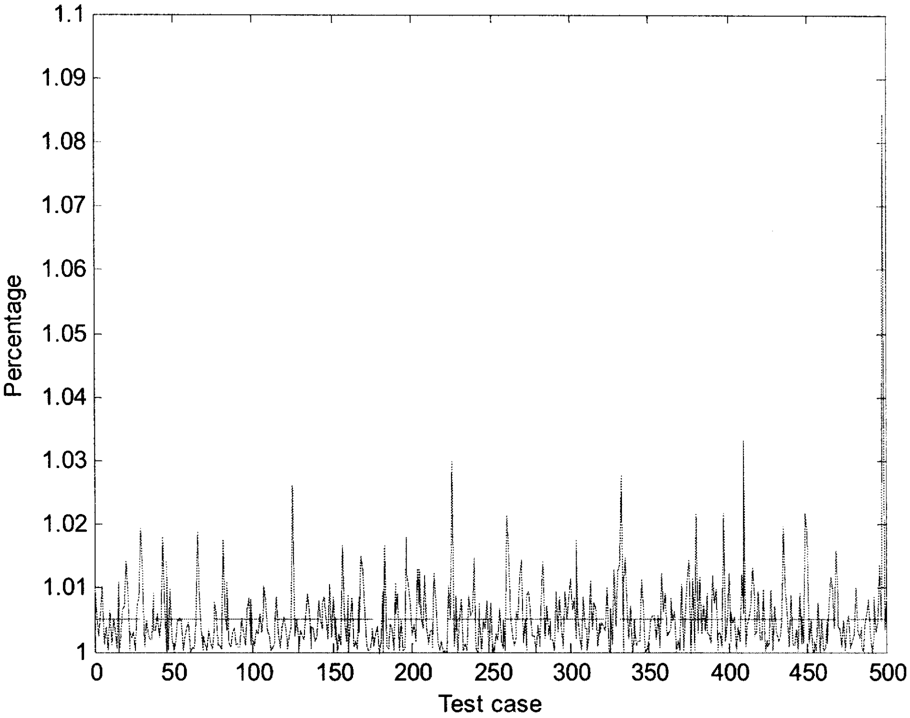

Figure C.6: Test Results at $\rho = 0.55$

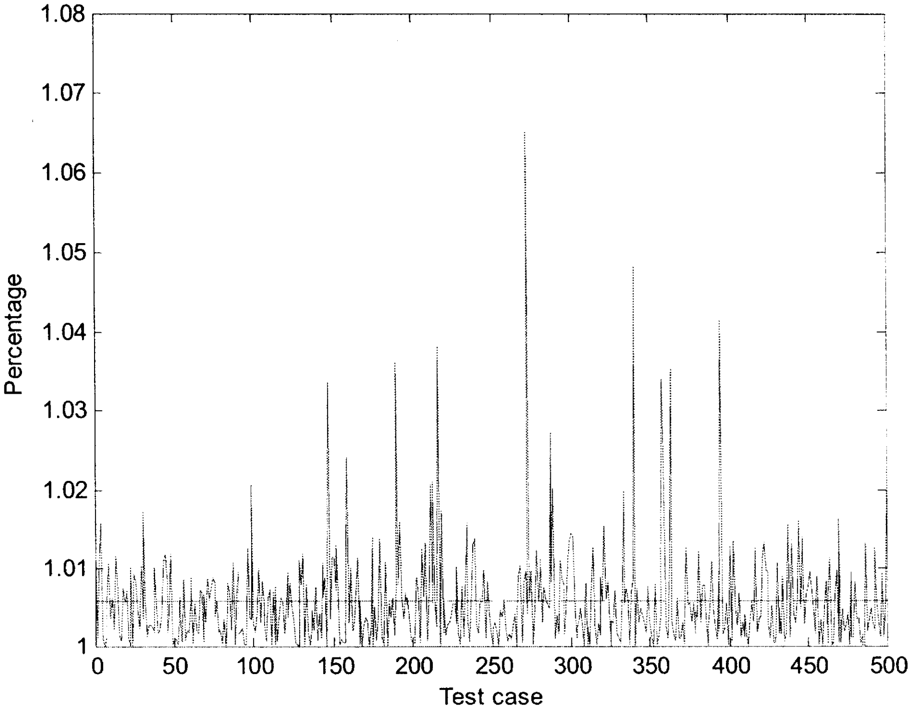

Figure C.7: Test Results at $\rho = 0.6$

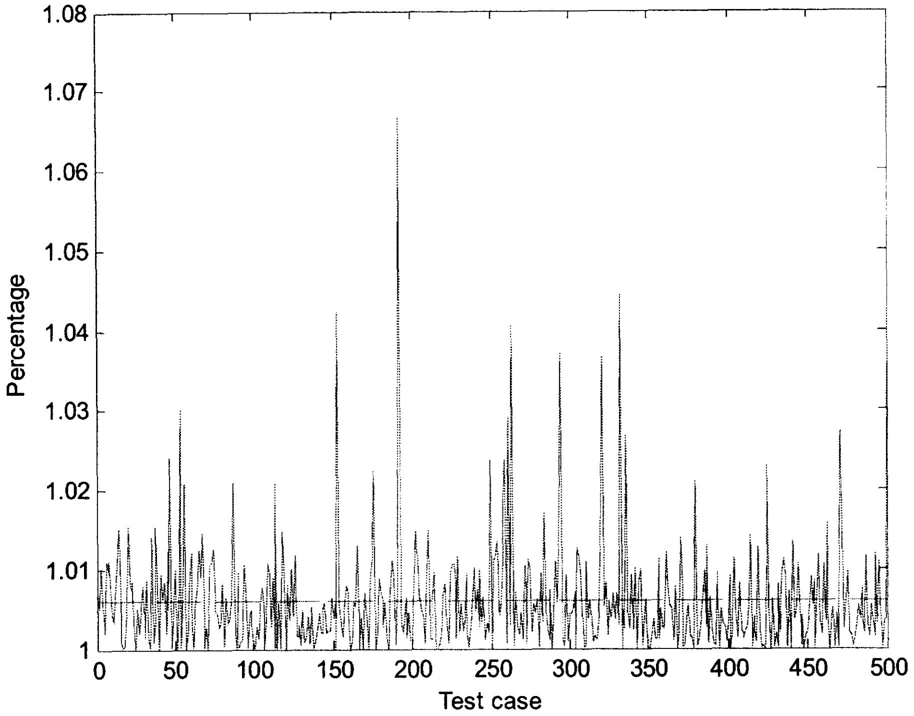

Figure C.8: Test Results at $\rho = 0.65$

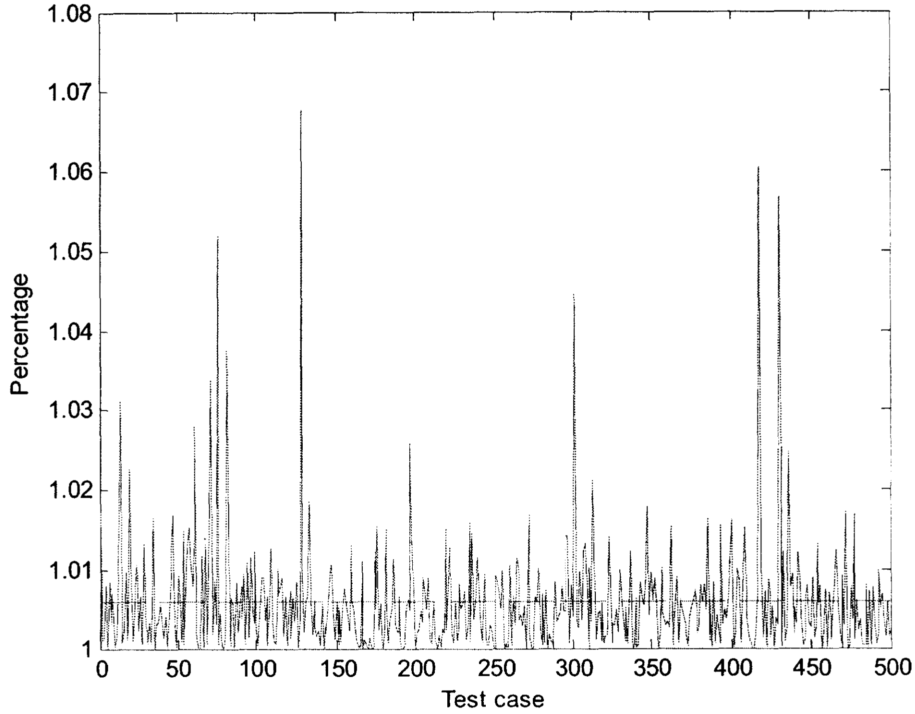

Figure C.9: Test Results at $\rho = 0.66$

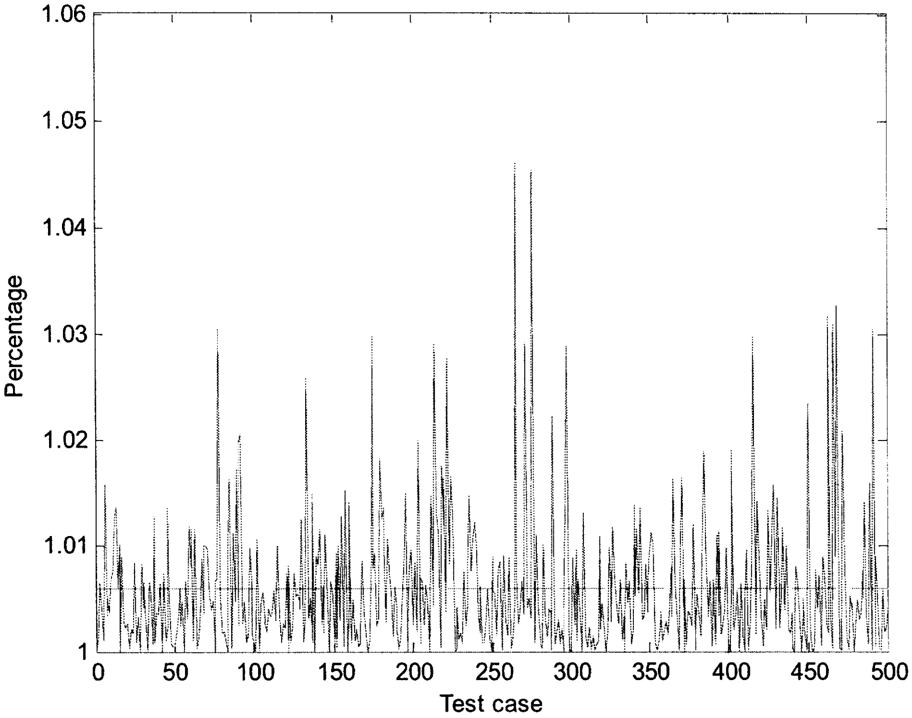

Figure C.10: Test Results at $\rho = 0.67$

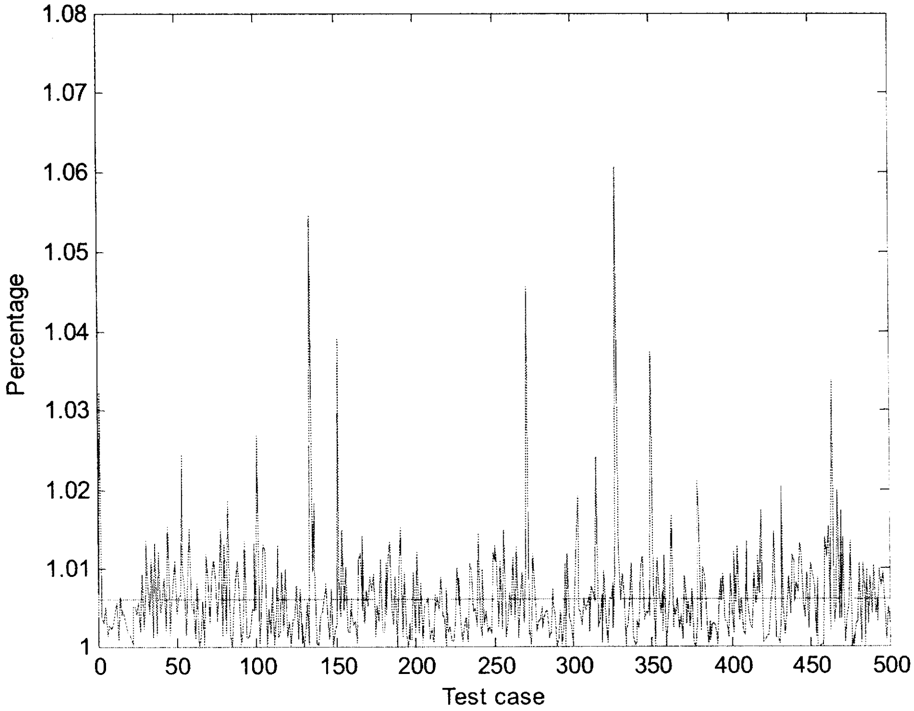

Figure C.11: Test Results at $\rho = 0.68$

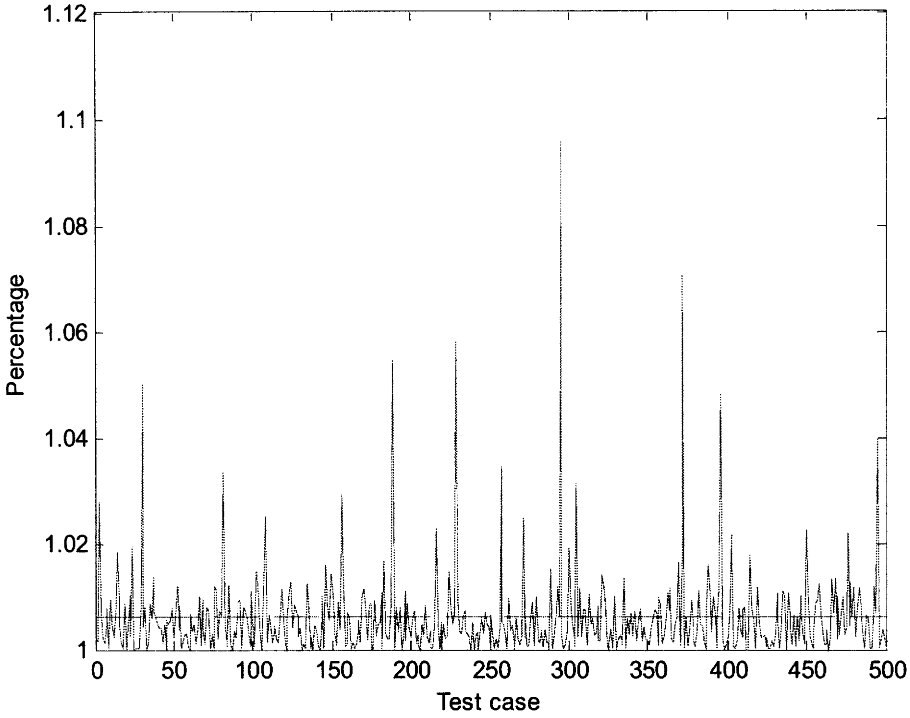

Figure C.12: Test Results at $\rho = 0.69$

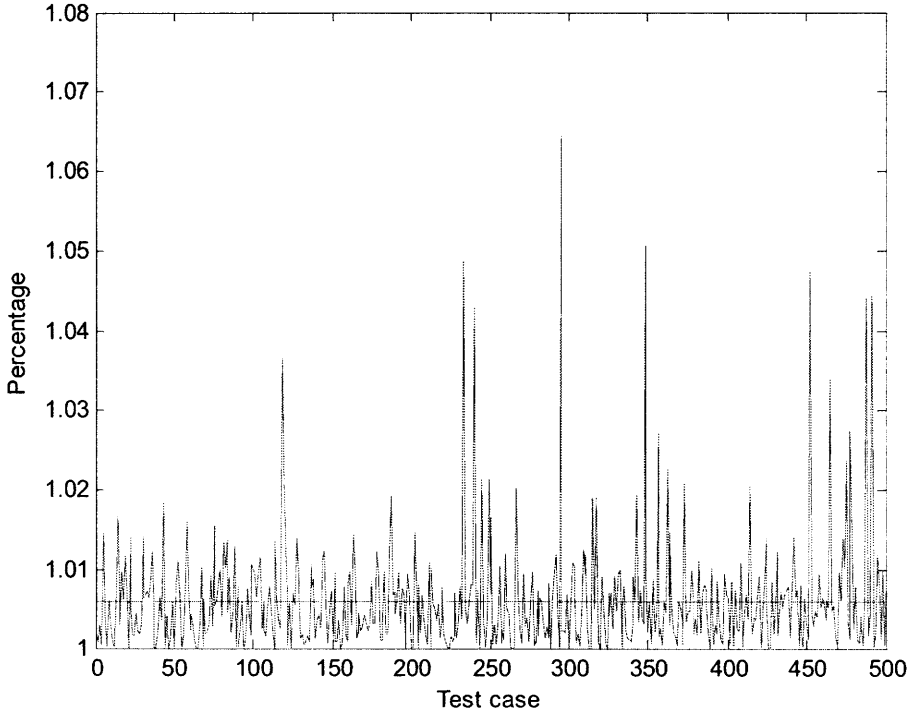

Figure C.13: Test Results at $\rho = 0.7$

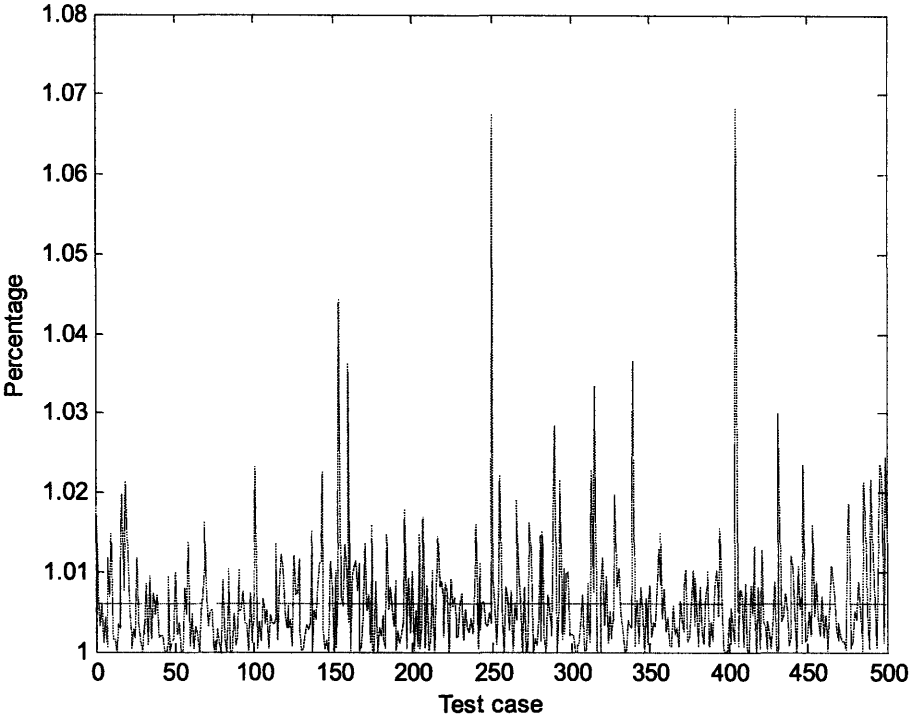

Figure C.14: Test Results at $\rho = 0.71$

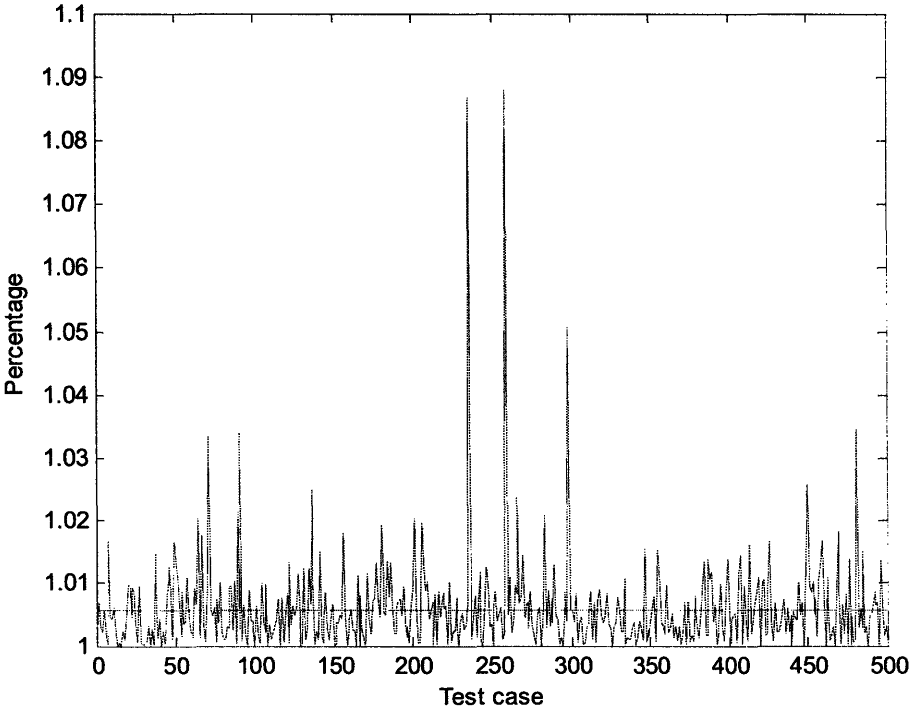

Figure C.15: Test Results at $\rho = 0.72$

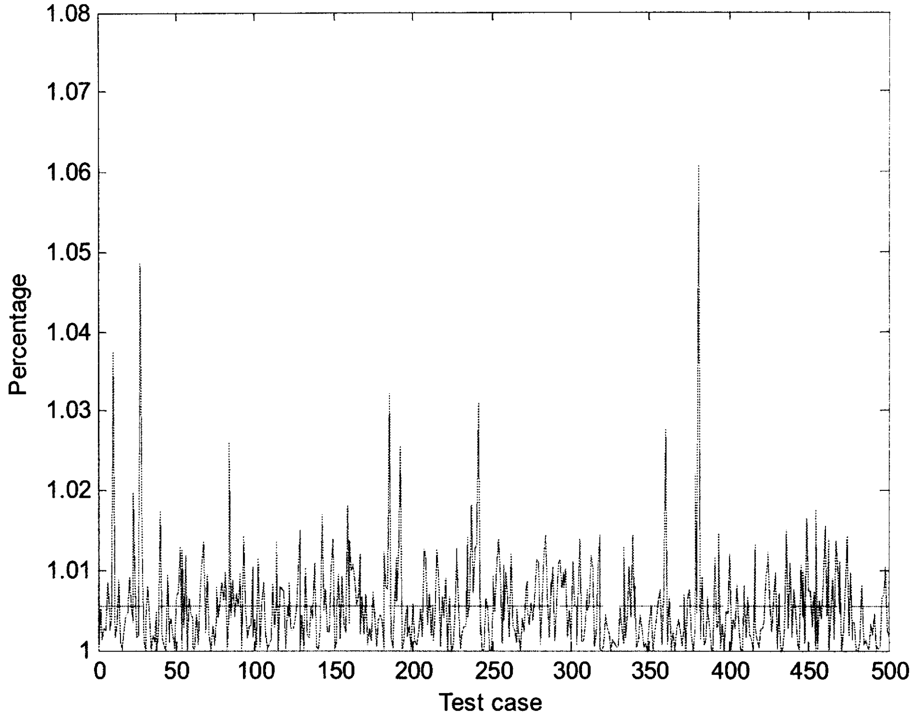

Figure C.16: Test Results at $\rho = 0.73$

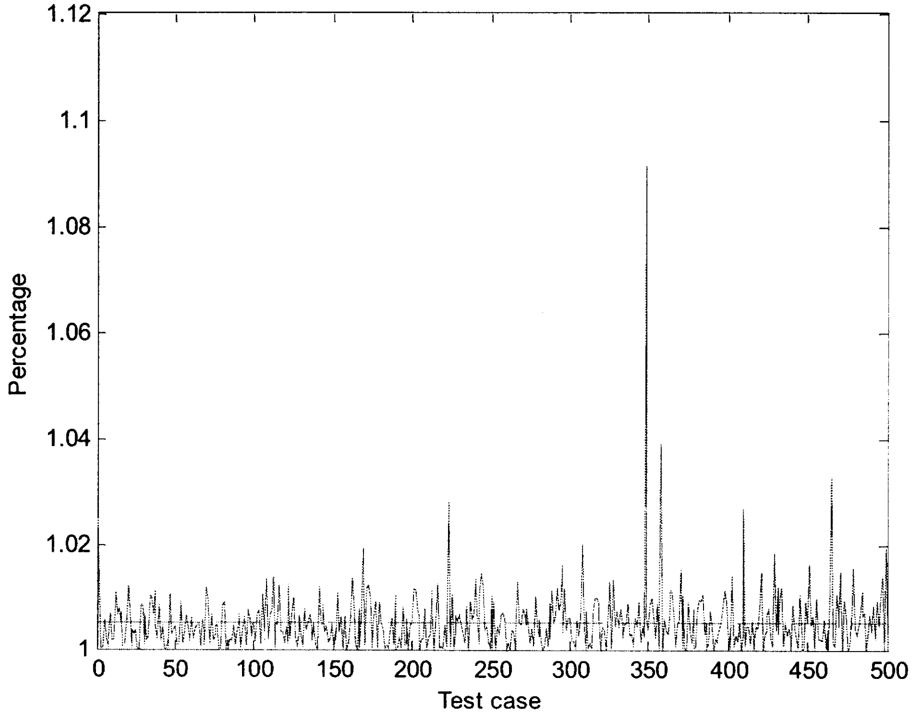

Figure C.17: Test Results at $\rho = 0.74$

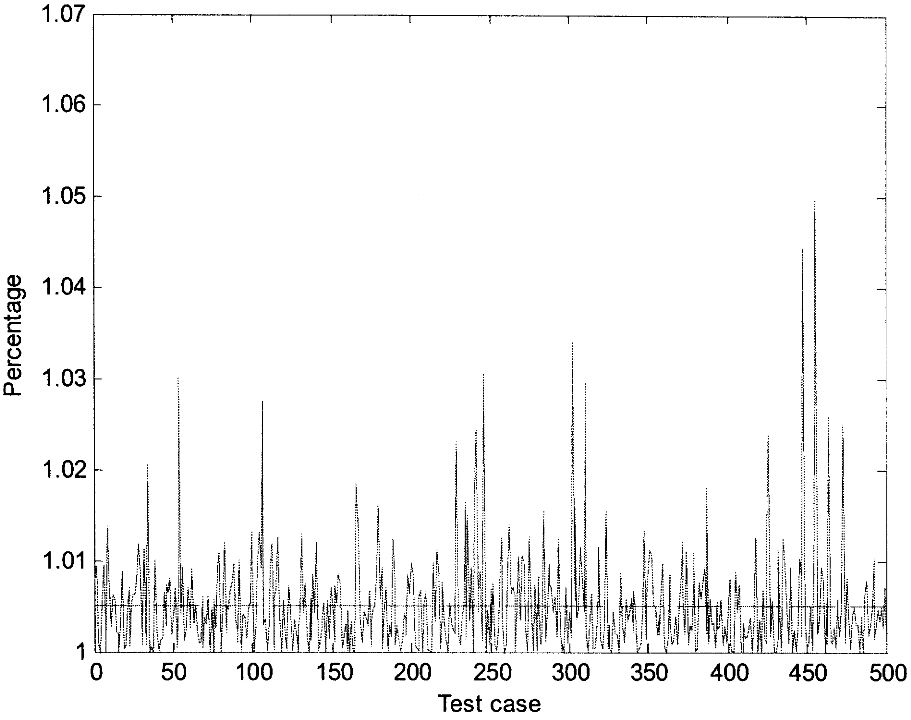

Figure C.18: Test Results at $\rho = 0.75$

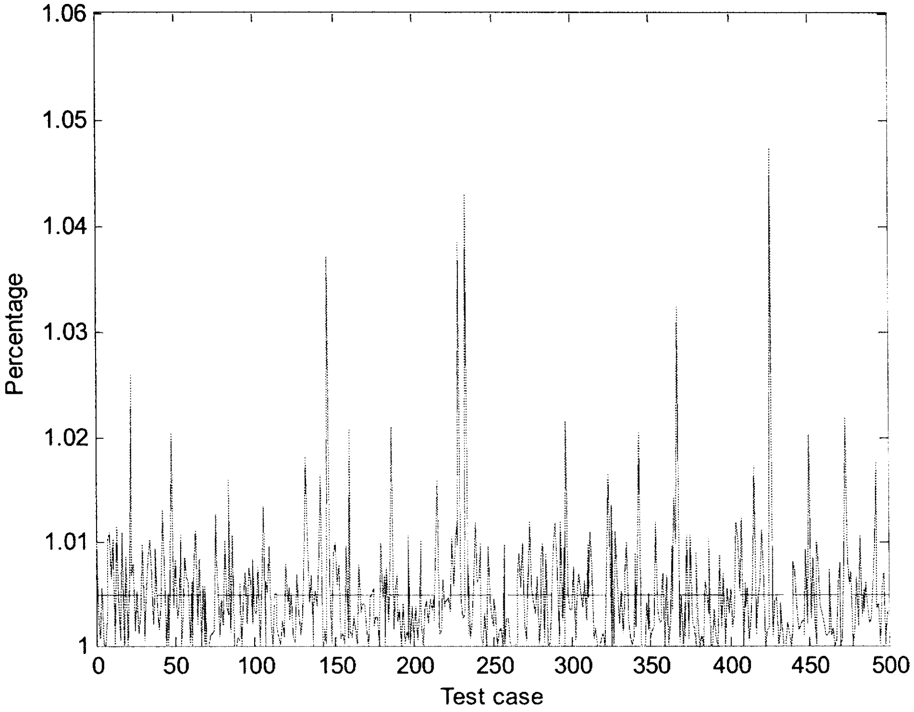

Figure C.19: Test Results at $\rho = 0.8$

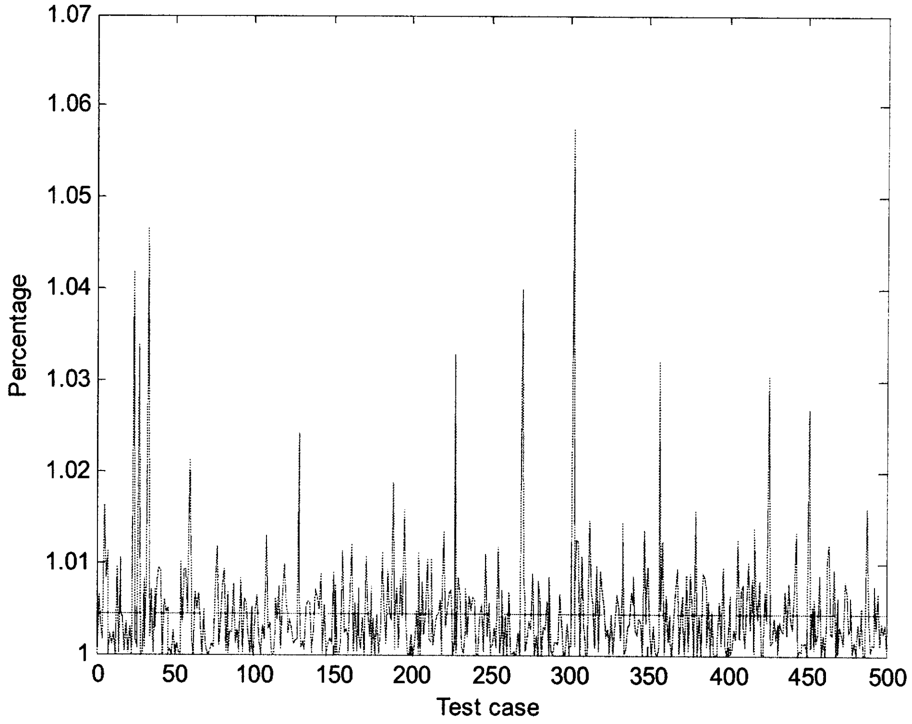

Figure C.20: Test Results at $\rho = 0.85$

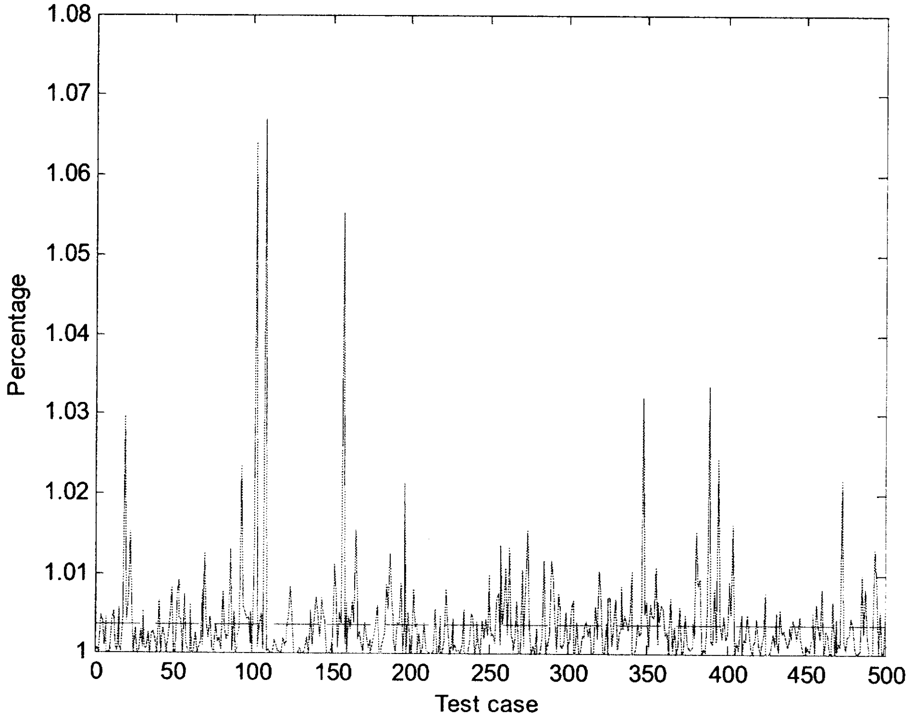

Figure C.21: Test Results at $\rho = 0.9$

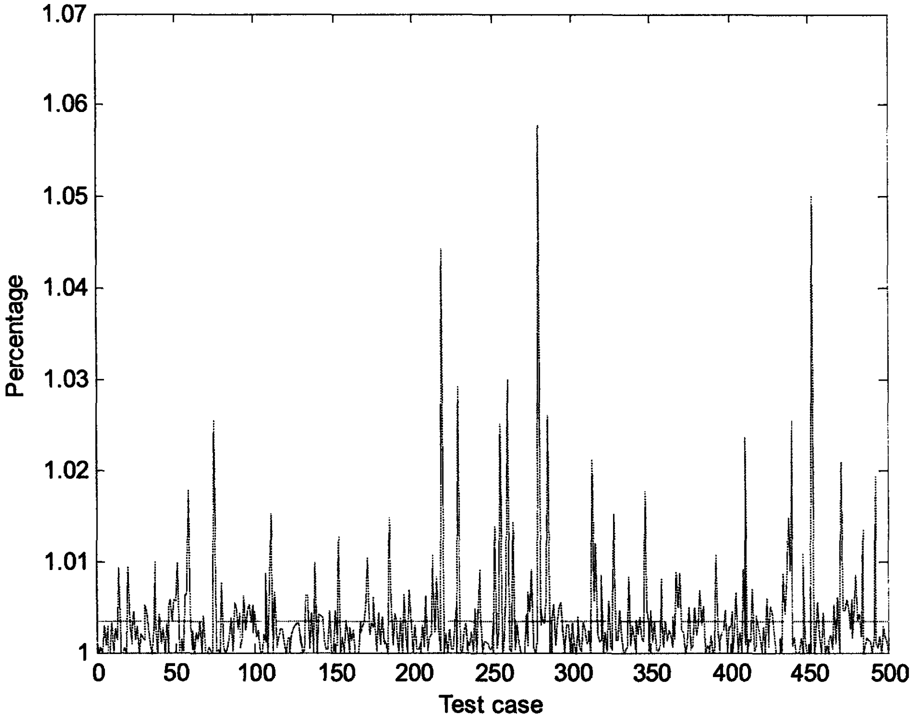

Figure C.22: Test Results at $\rho = 0.91$

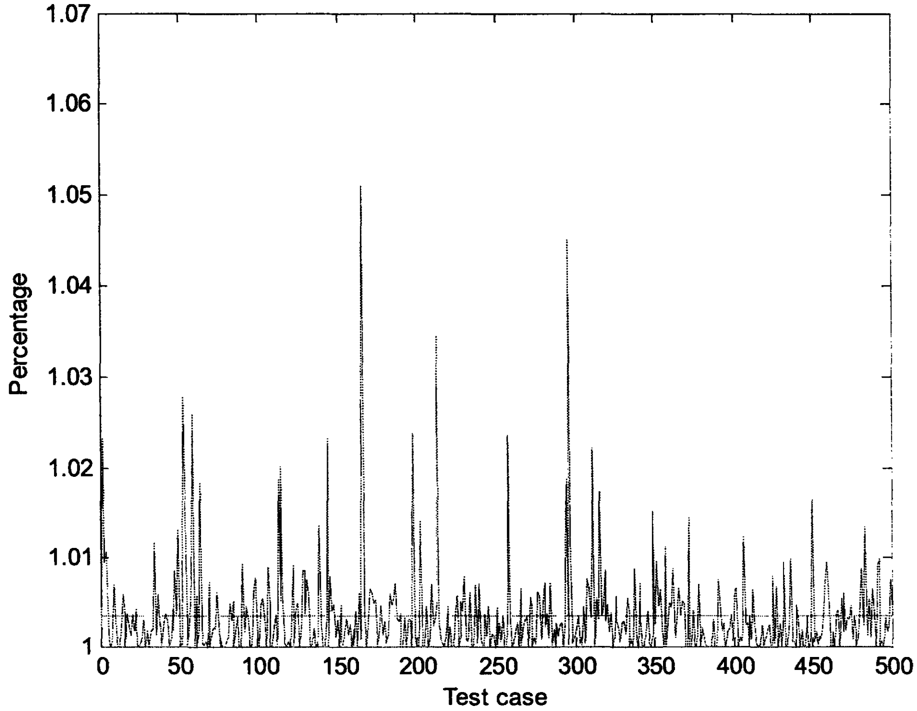

Figure C.23: Test Results at $\rho = 0.92$

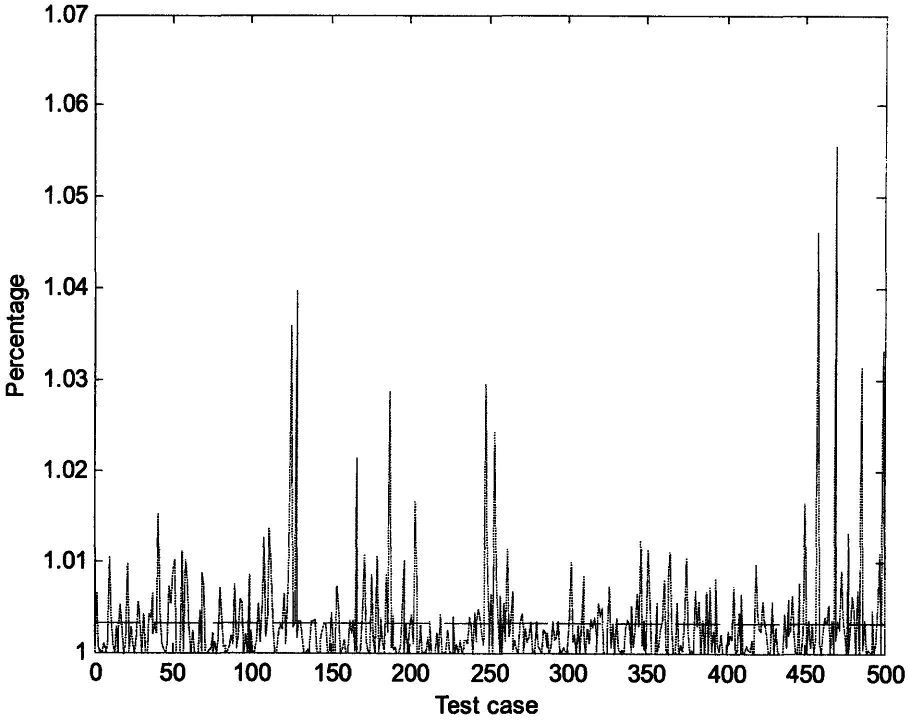

Figure C.24: Test Results at $\rho = 0.93$

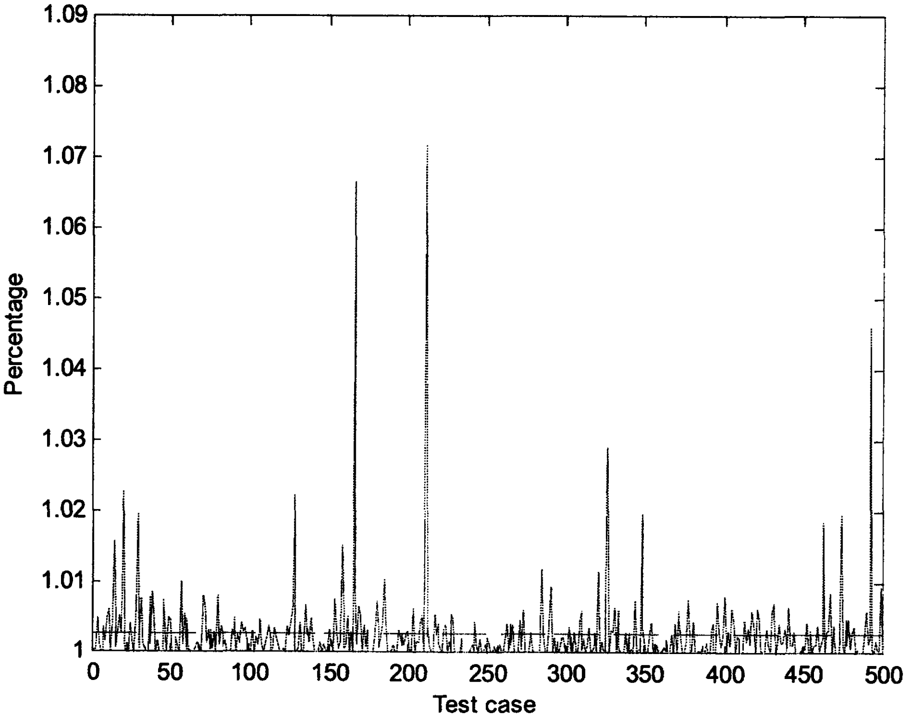

Figure C.25: Test Results at $\rho = 0.94$

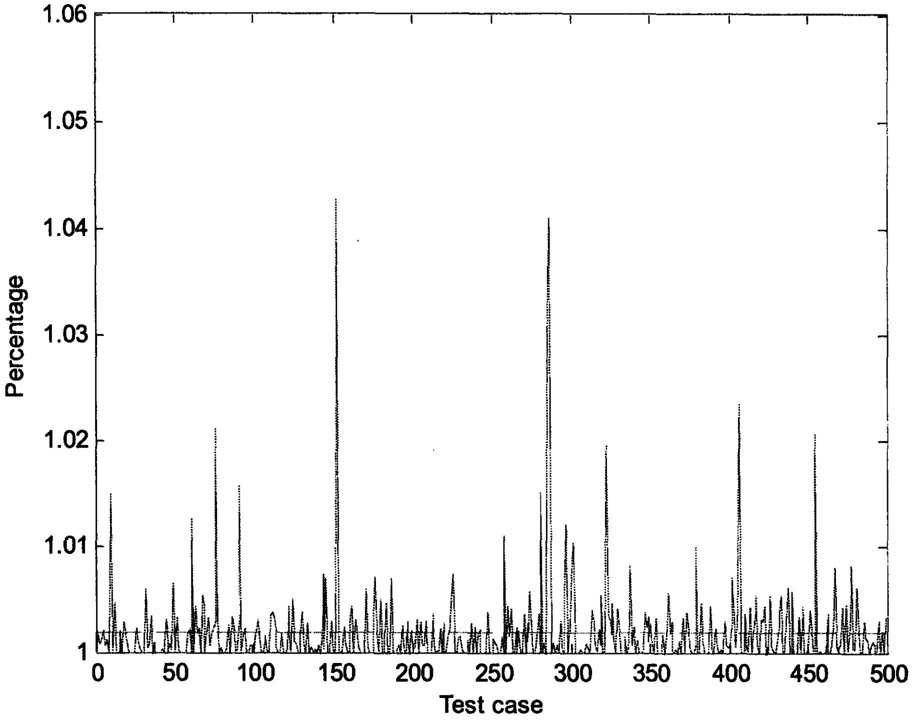

Figure C.26: Test Results at $\rho = 0.95$

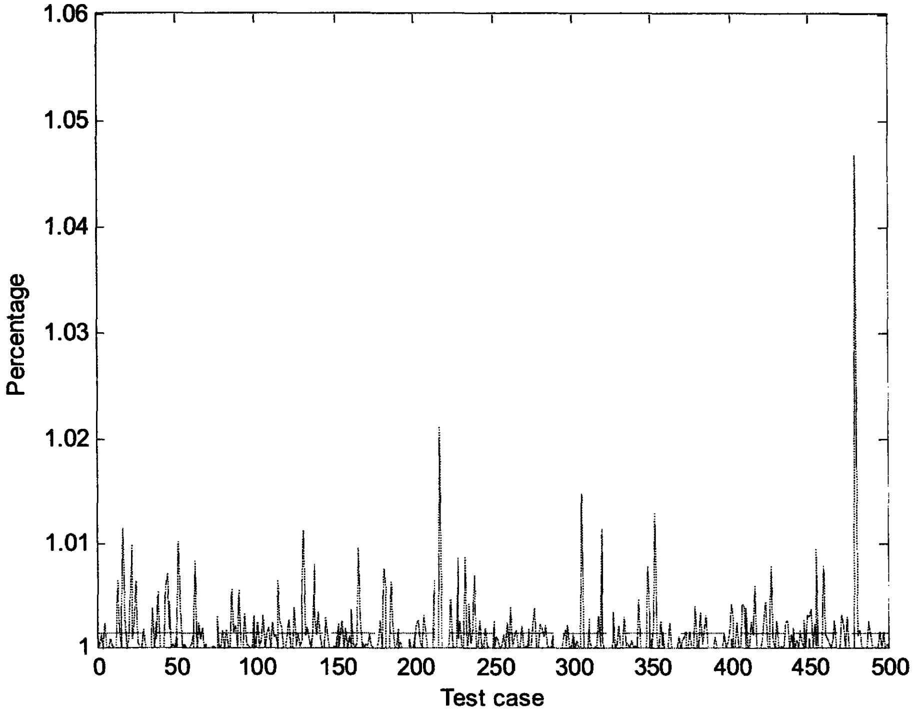

Figure C.27: Test Results at $\rho = 0.96$

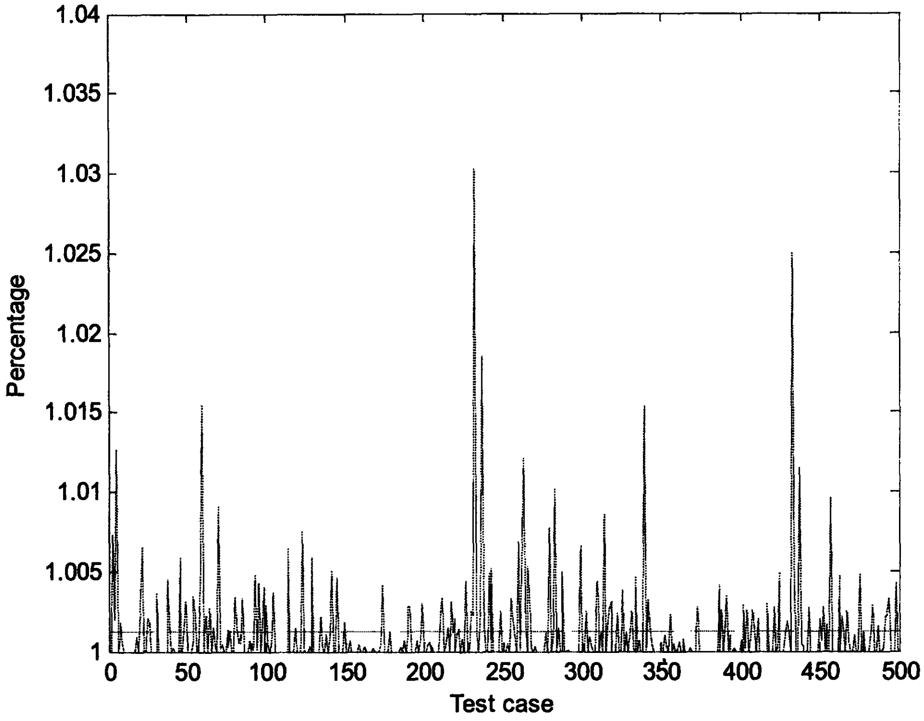

Figure C.28: Test Results at $\rho = 0.97$

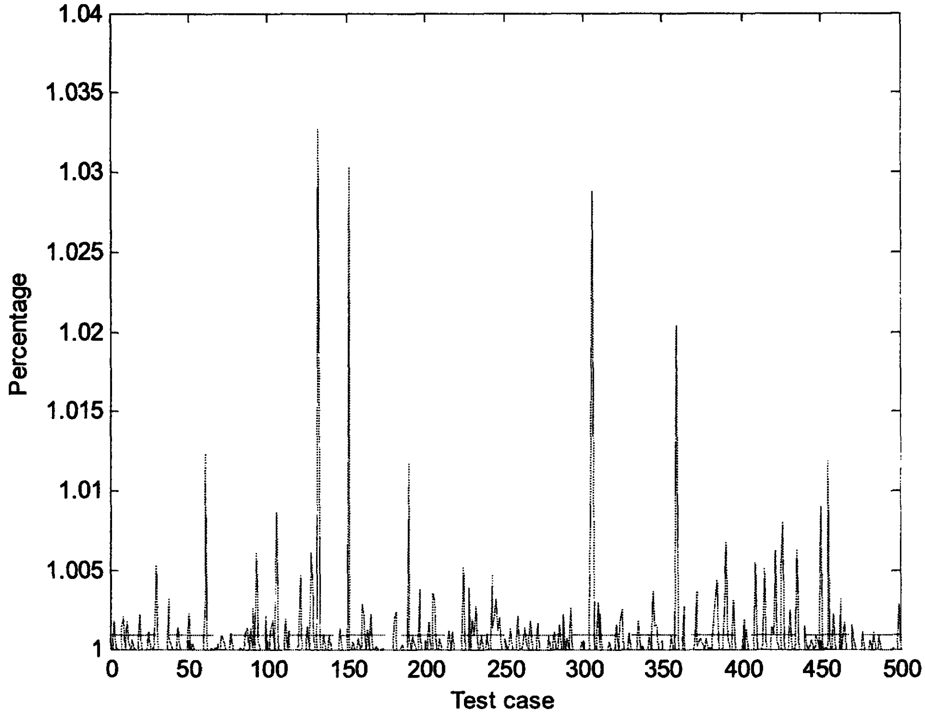

Figure C.29: Test Results at $\rho = 0.98$

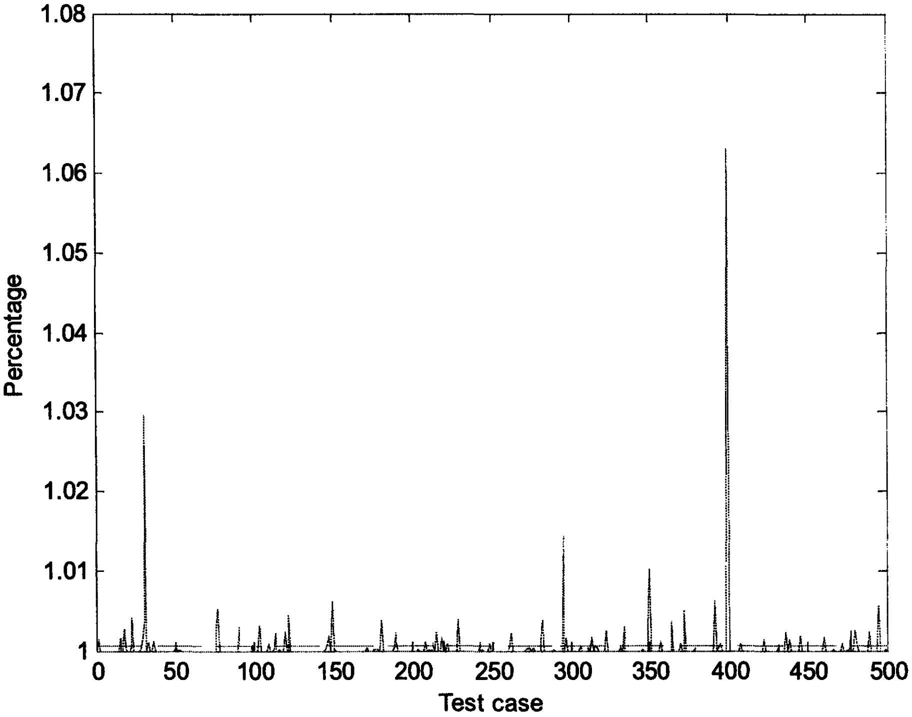

Figure C.30: Test Results at $\rho = 0.99$

Appendix D

D.1 Test Results at $\rho_{test} = 0.5$

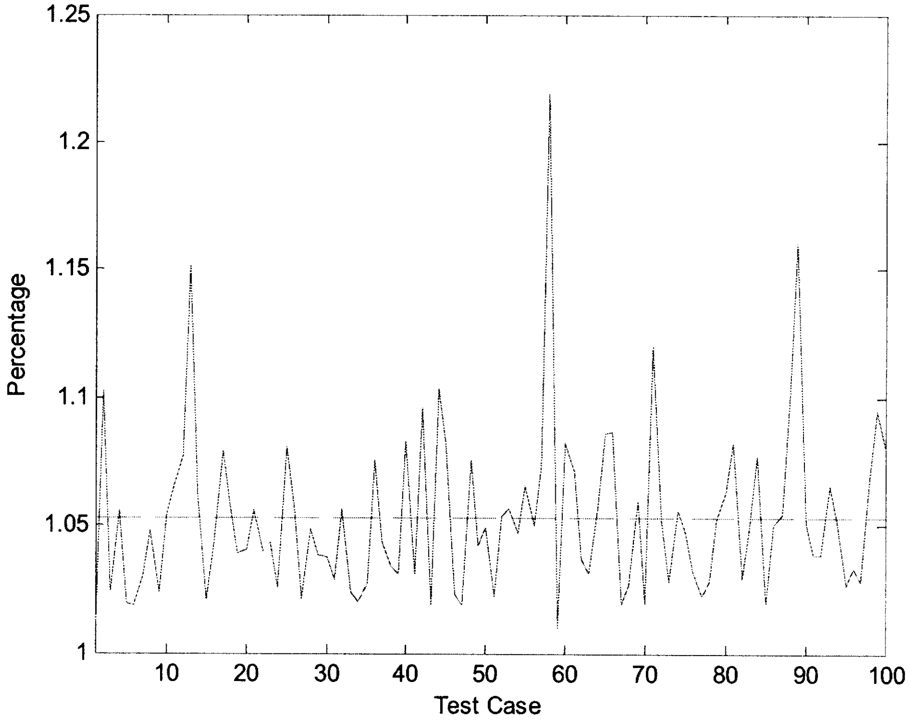

Figure D.1.1: Performance of Policy 1 against Minimize Utilization Heuristic

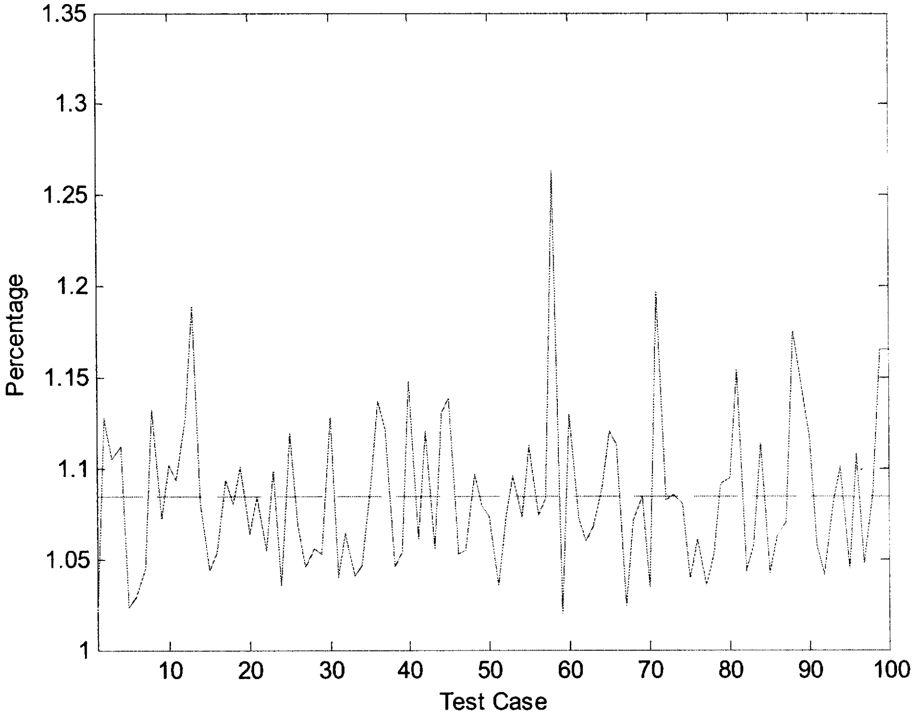

Figure D.1.2: Performance of Policy 2(a) against Minimize Utilization Heuristic

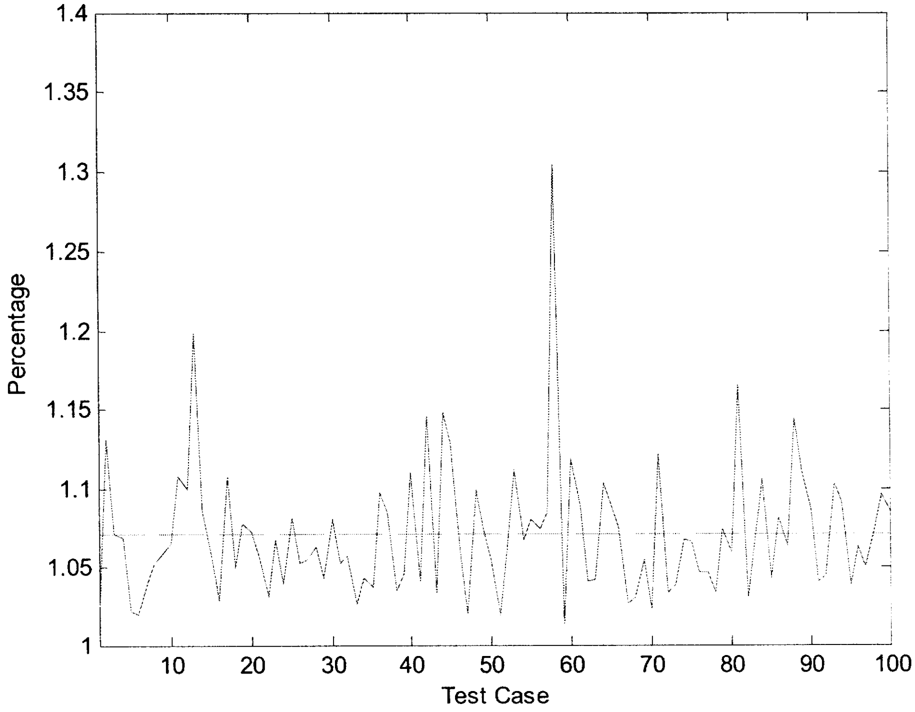

Figure D.1.3: Performance of Policy 2(b) against Minimize Utilization Heuristic

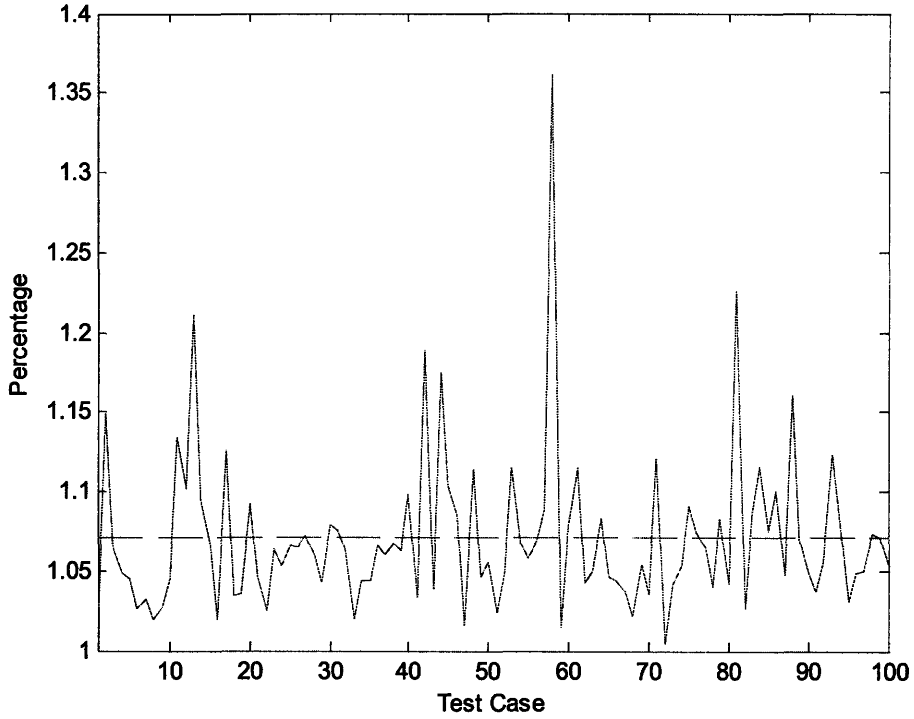

Figure D.1.4: Performance of Policy 2(c) against Minimize Utilization Heuristic

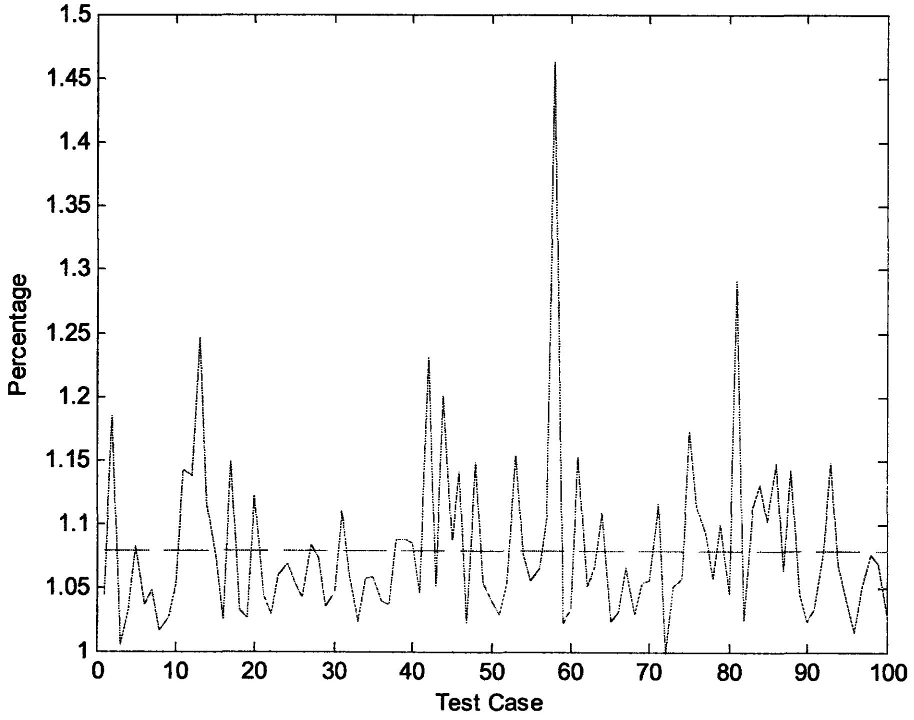

Figure D.1.5: Performance of Policy 2(d) against Minimize Utilization Heuristic

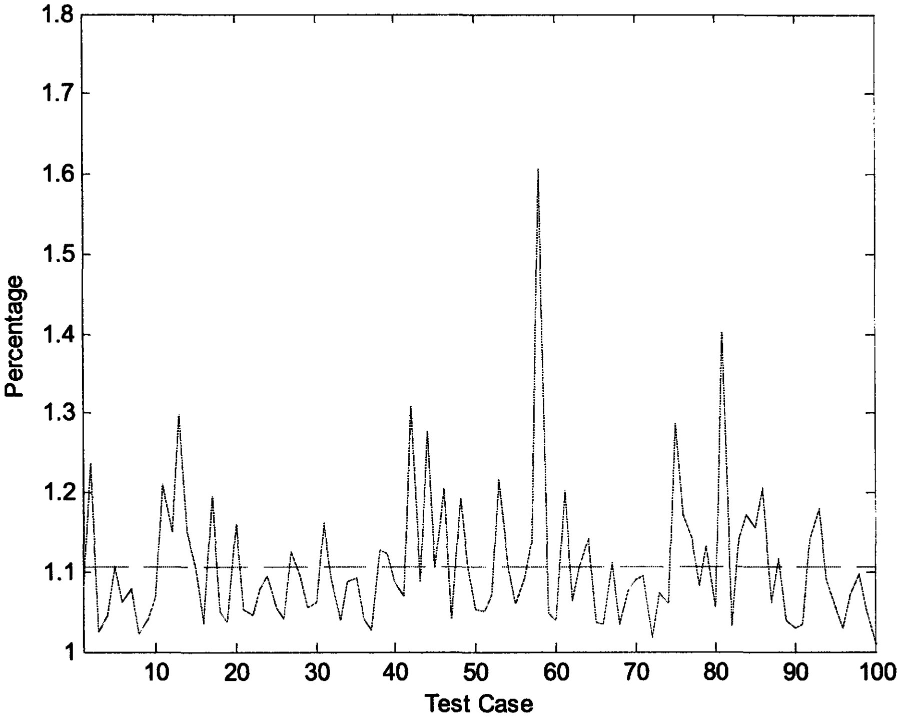

Figure D.1.6: Performance of Policy 2(e) against Minimize Utilization Heuristic

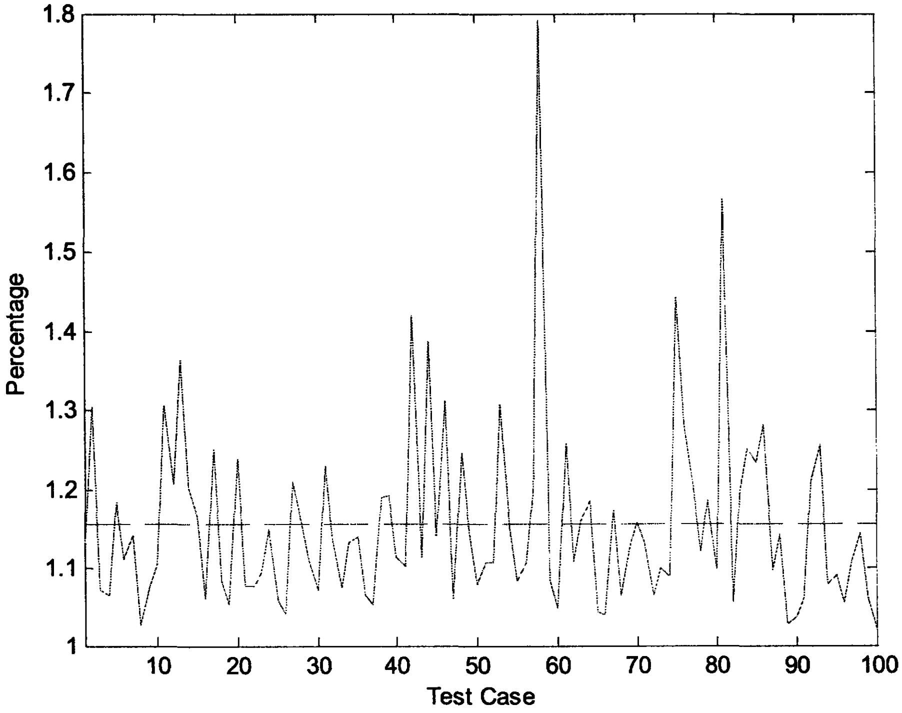

Figure D.1.7: Performance of Policy 2(f) against Minimize Utilization Heuristic

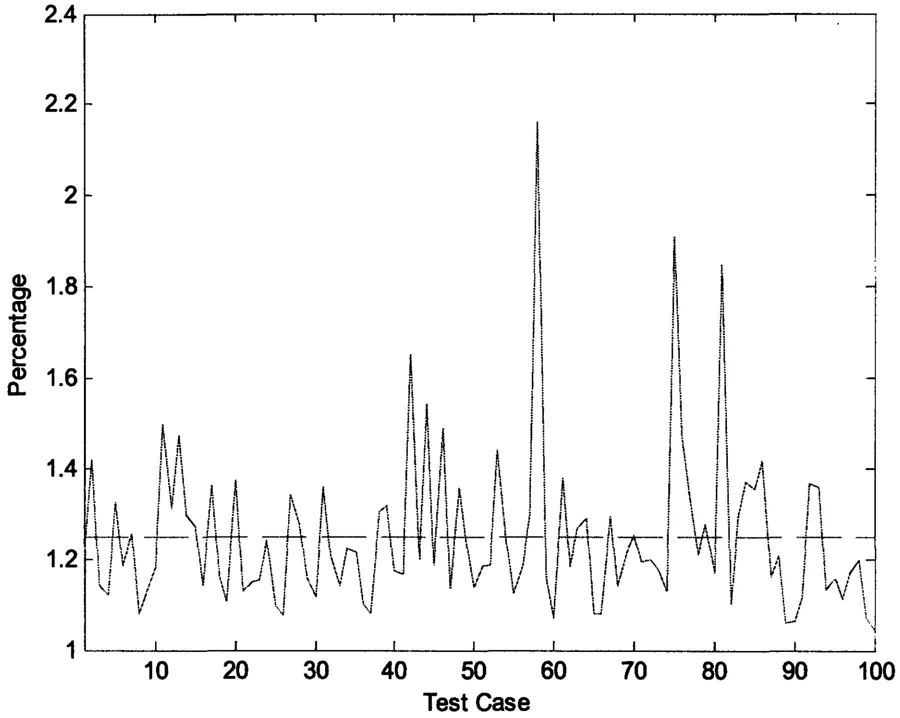

Figure D.1.8: Performance of Policy 2(g) against Minimize Utilization Heuristic

D.2 Test Results at $\rho_{test} = 0.6$

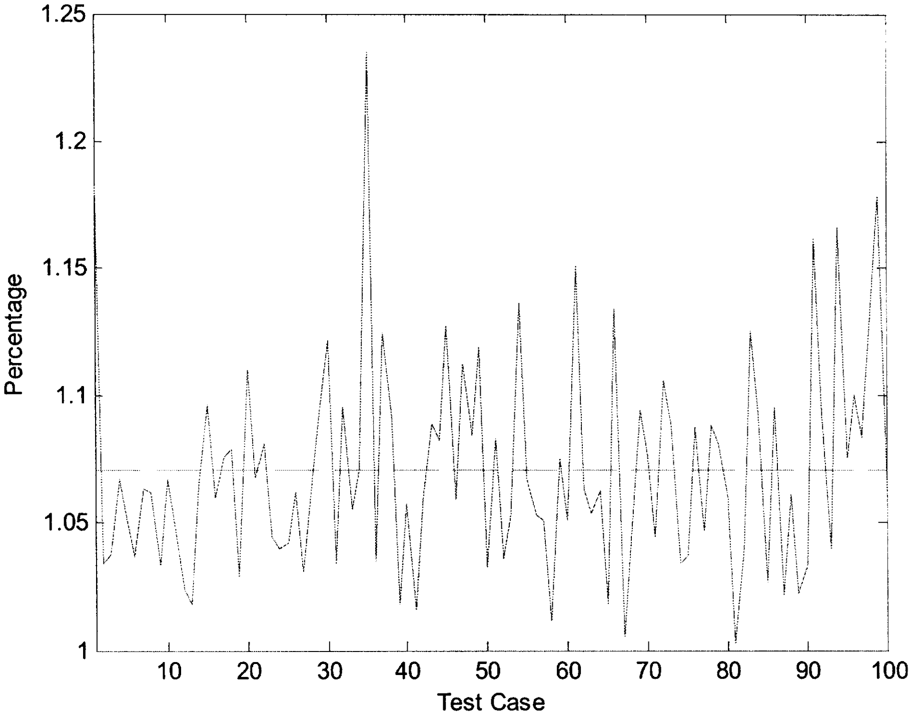

Figure D.2.1: Performance of Policy 1 against Minimize Utilization Heuristic

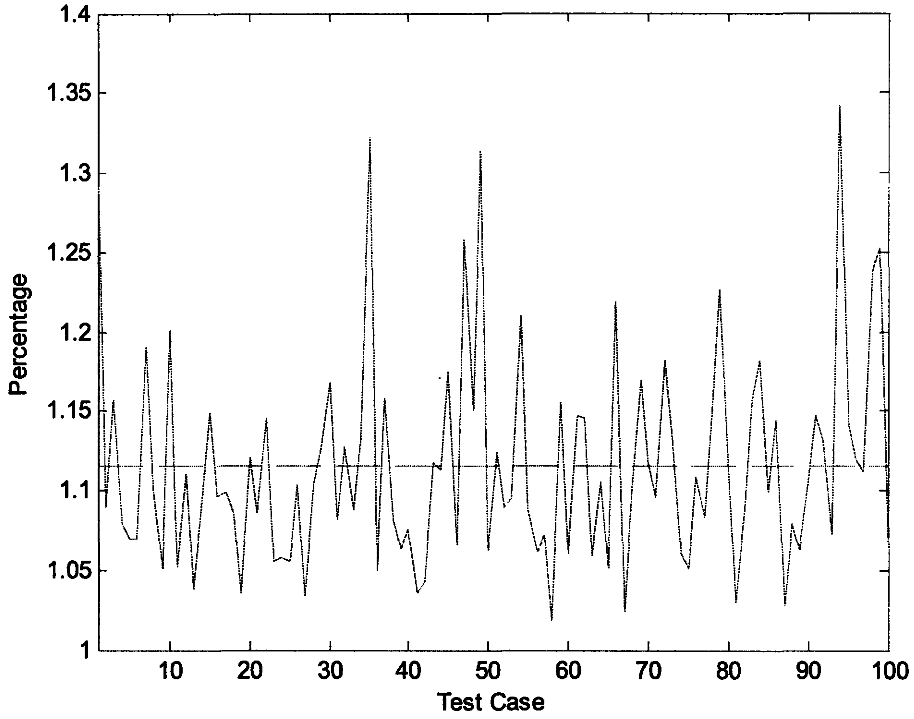

Figure D.2.2: Performance of Policy 2(a) against Minimize Utilization Heuristic

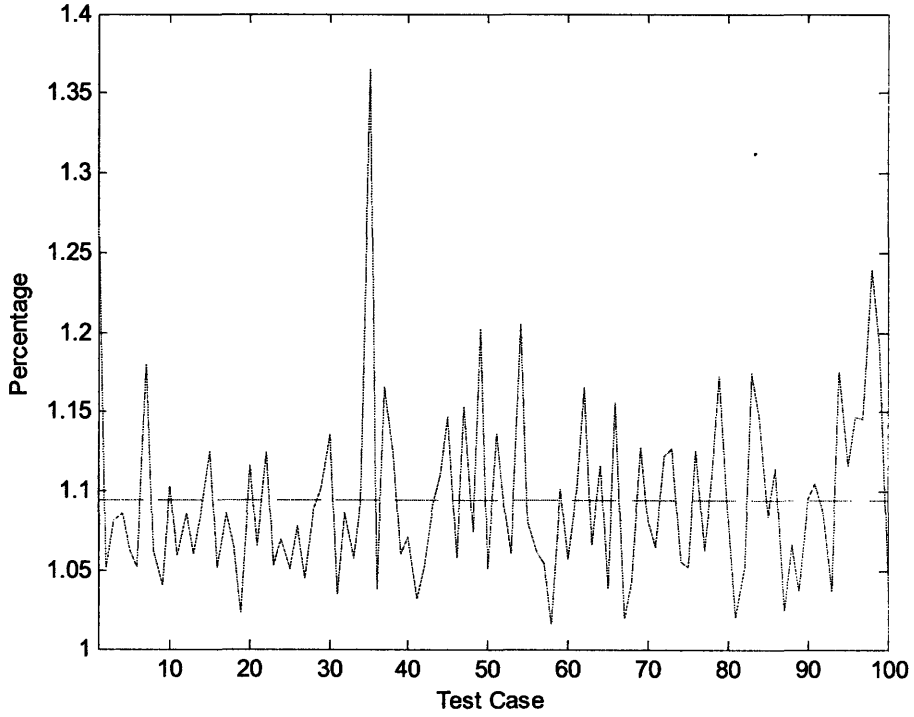

Figure D.2.3: Performance of Policy 2(b) against Minimize Utilization Heuristic

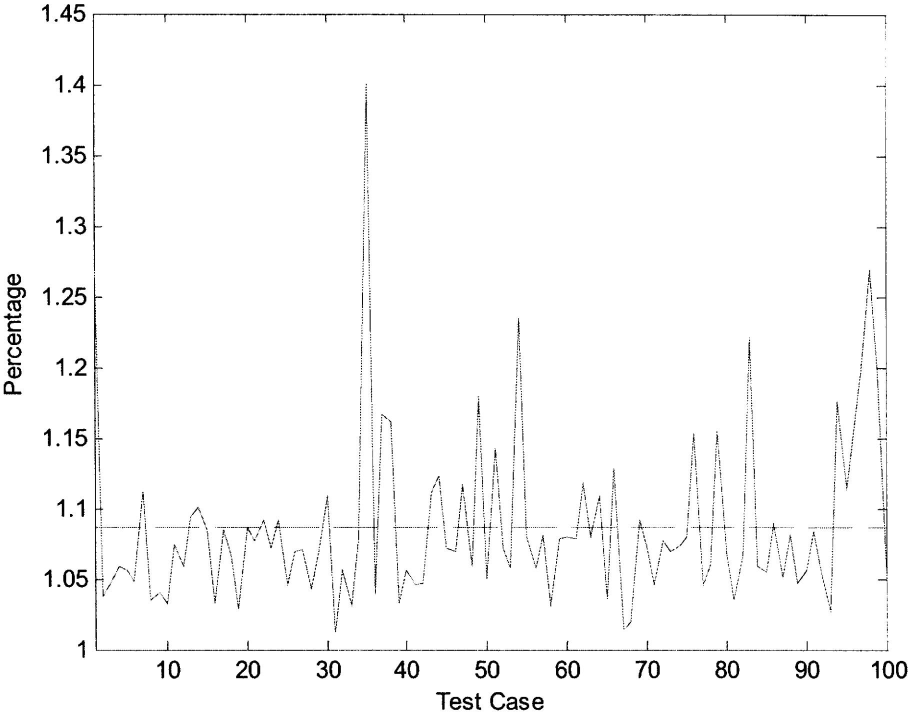

Figure D.2.4: Performance of Policy 2(c) against Minimize Utilization Heuristic

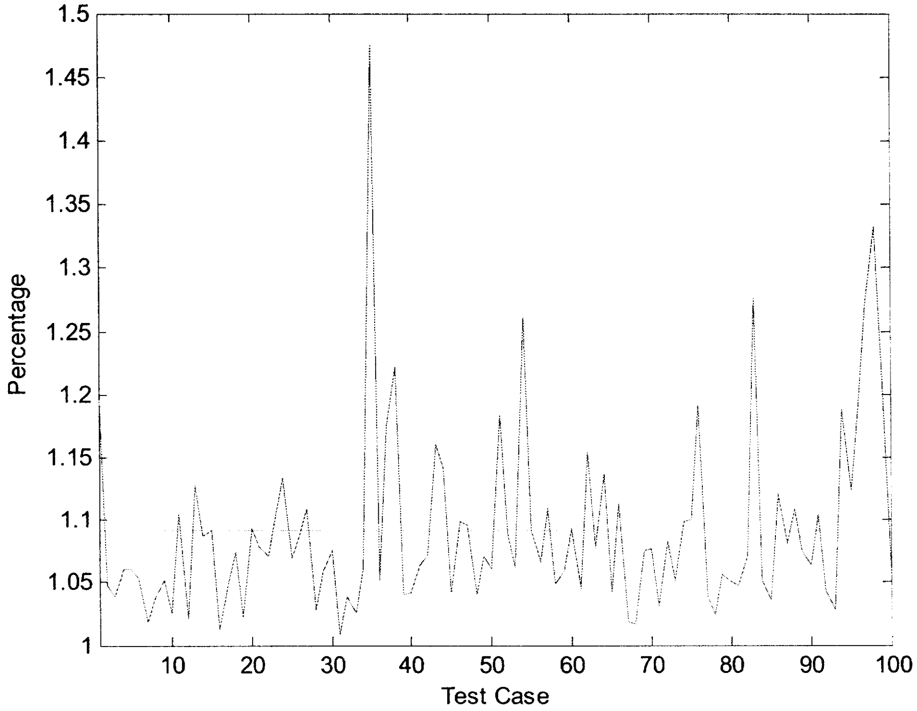

Figure D.2.5: Performance of Policy 2(d) against Minimize Utilization Heuristic

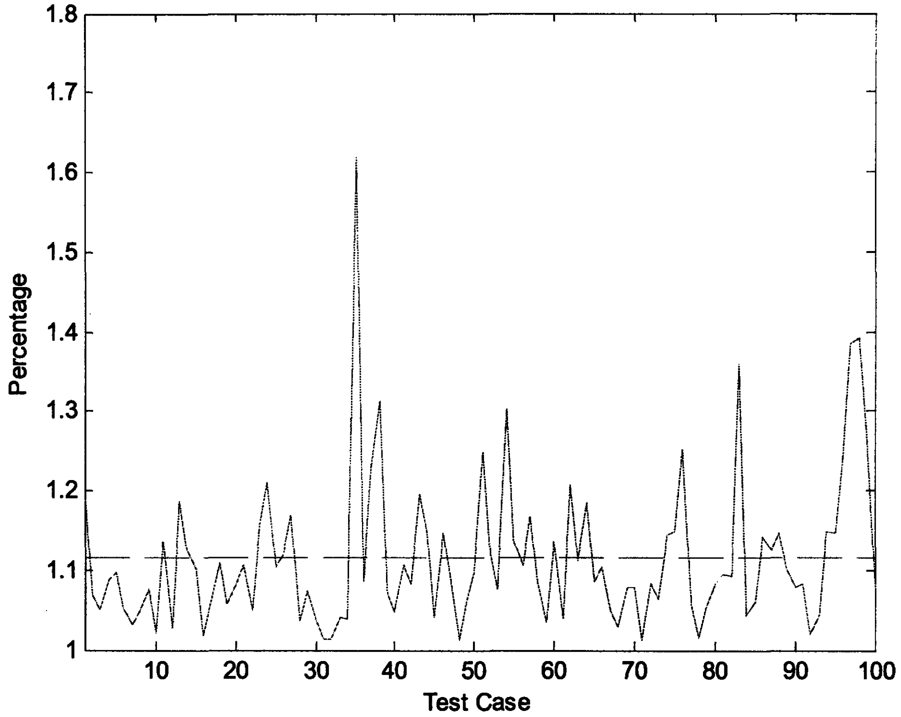

Figure D.2.6: Performance of Policy 2(e) against Minimize Utilization Heuristic

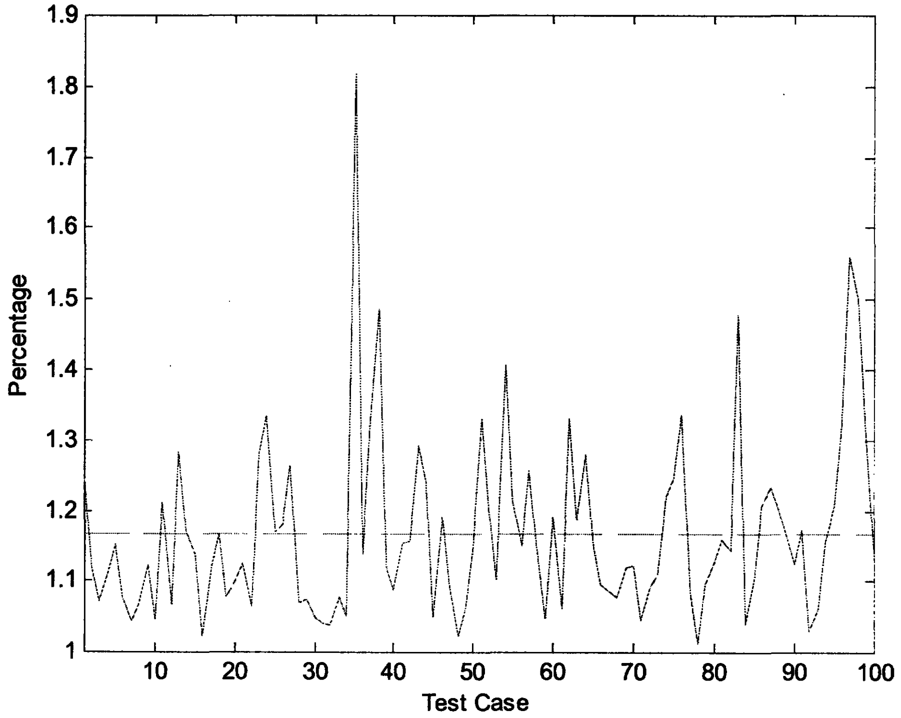

Figure D.2.7: Performance of Policy 2(f) against Minimize Utilization Heuristic

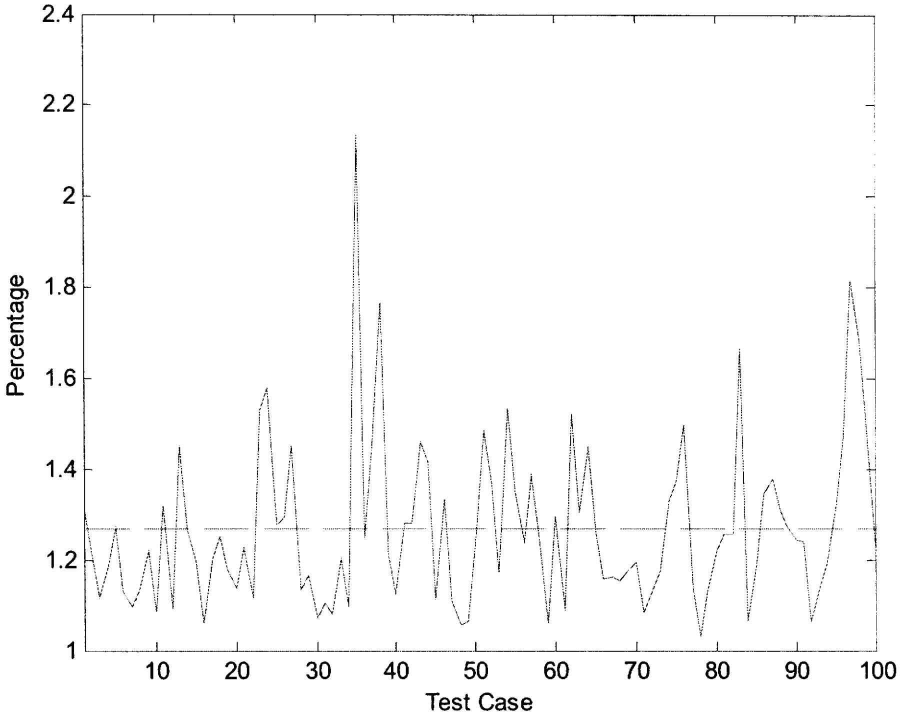

Figure D.2.8: Performance of Policy 2(g) against Minimize Utilization Heuristic

D.3 Test Results at $\rho_{test} = 0.7$

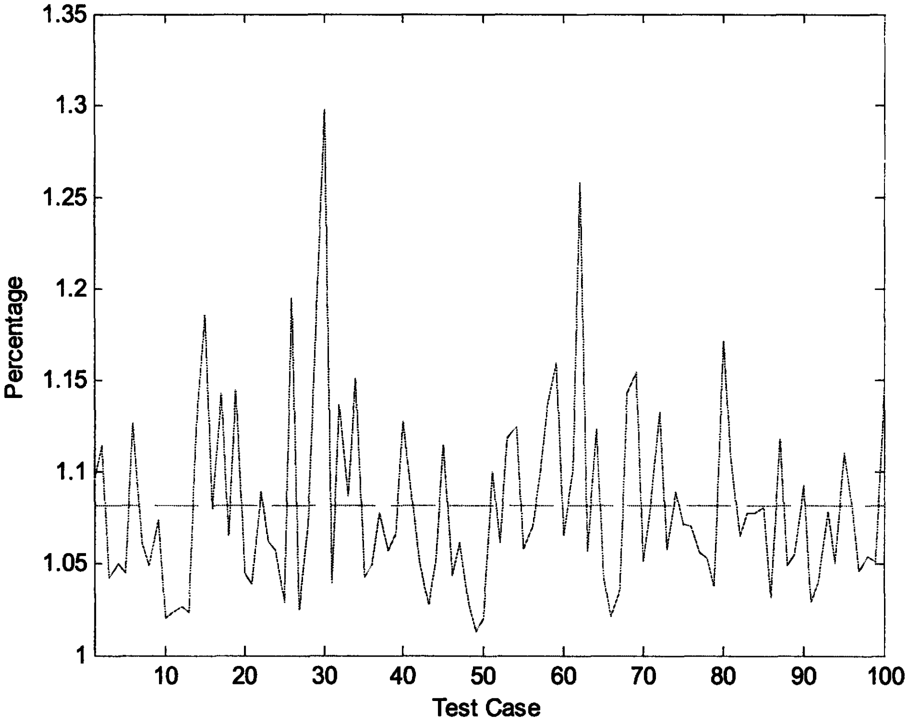

Figure D.3.1: Performance of Policy 1 against Minimize Utilization Heuristic

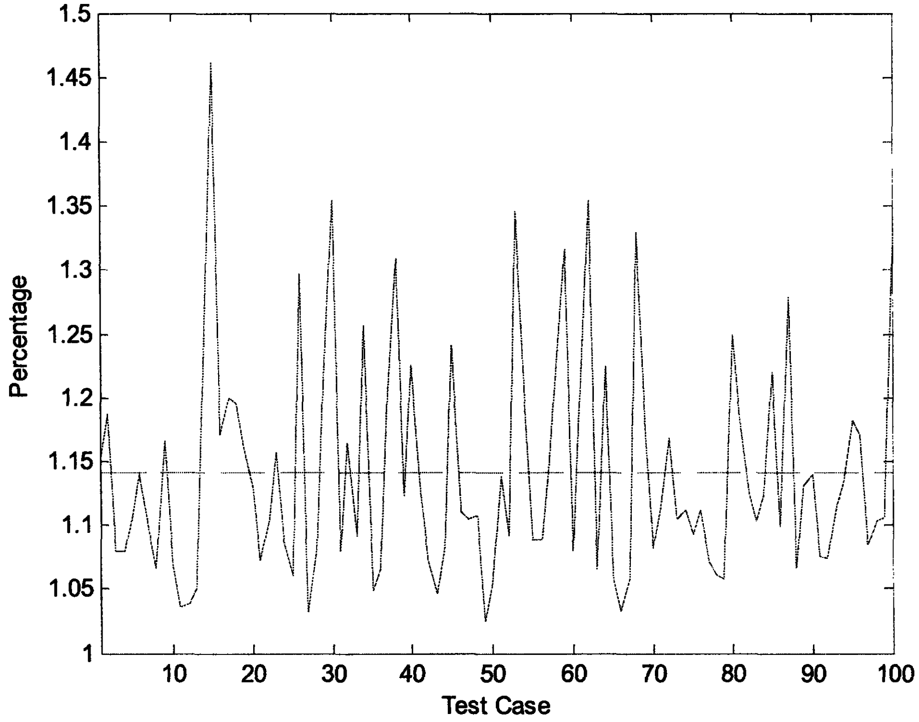

Figure D.3.2: Performance of Policy 2(a) against Minimize Utilization Heuristic

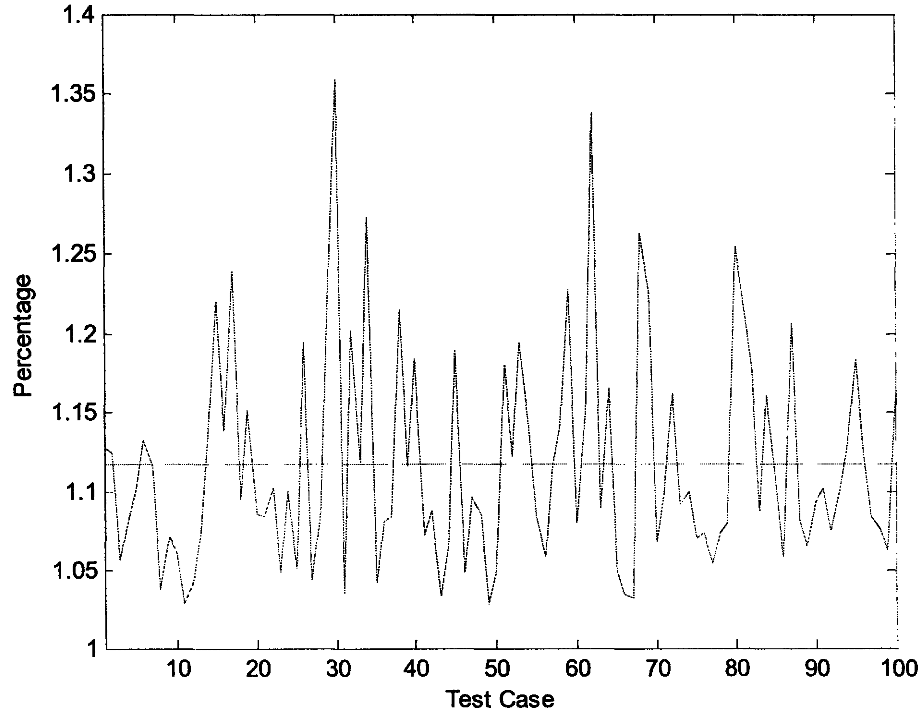

Figure D.3.3: Performance of Policy 2(b) against Minimize Utilization Heuristic

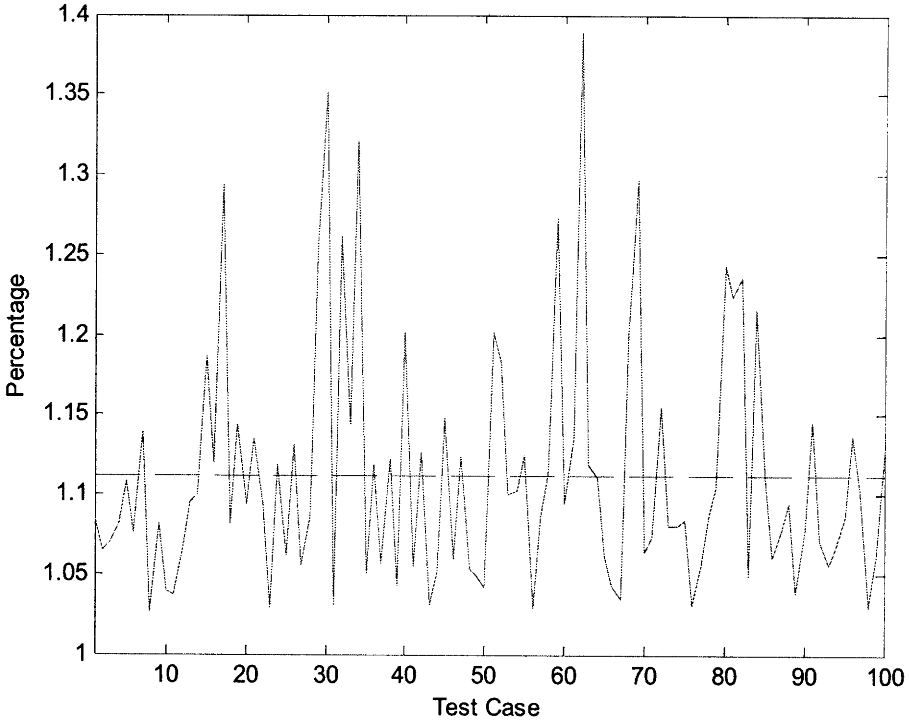

Figure D.3.4: Performance of Policy 2(c) against Minimize Utilization Heuristic

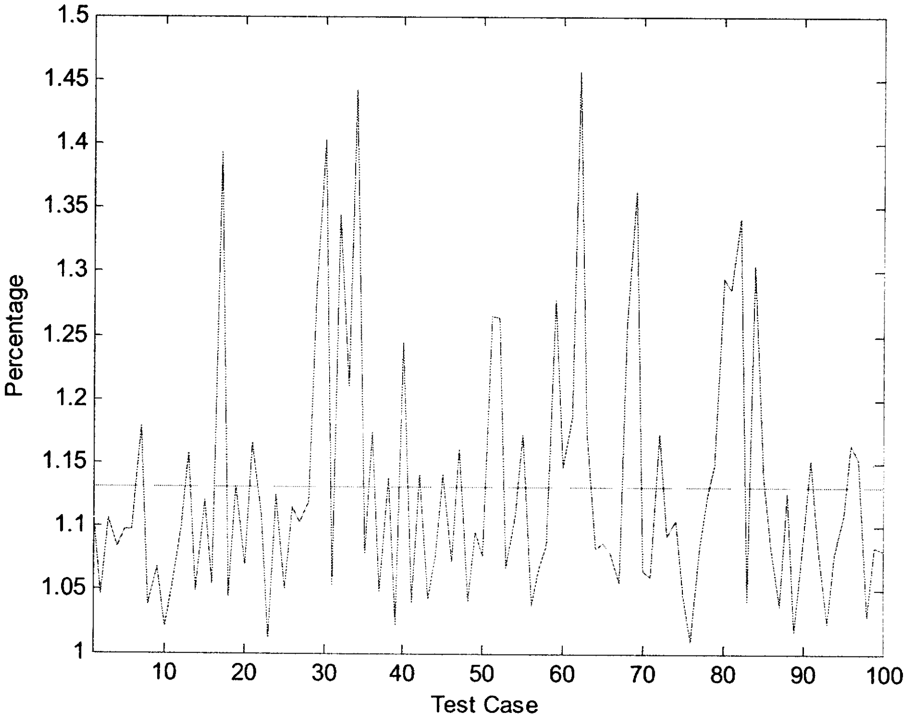

Figure D.3.5: Performance of Policy 2(d) against Minimize Utilization Heuristic

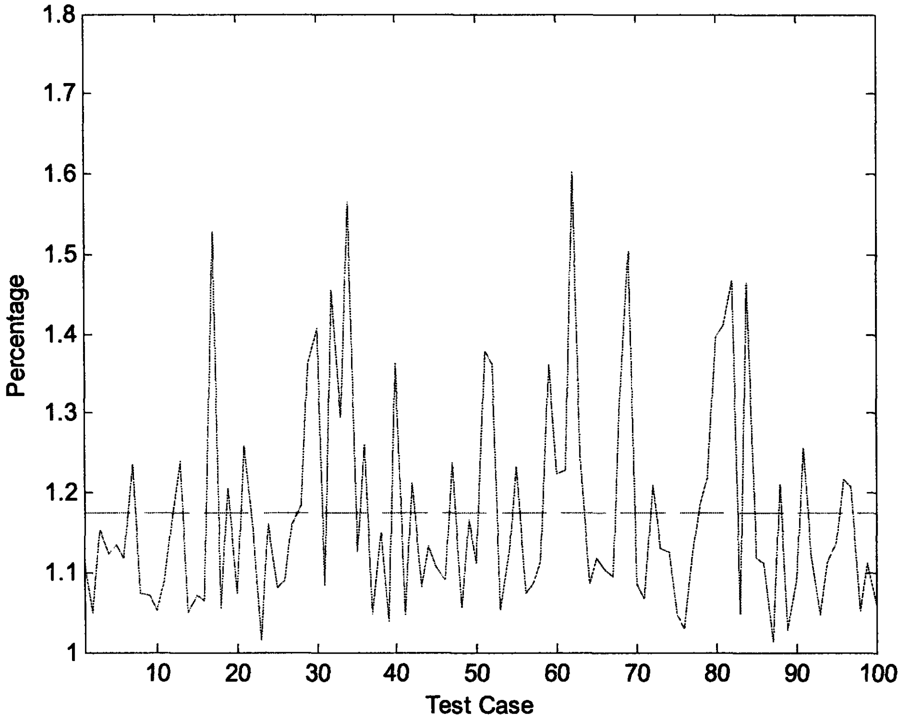

Figure D.3.6: Performance of Policy 2(e) against Minimize Utilization Heuristic

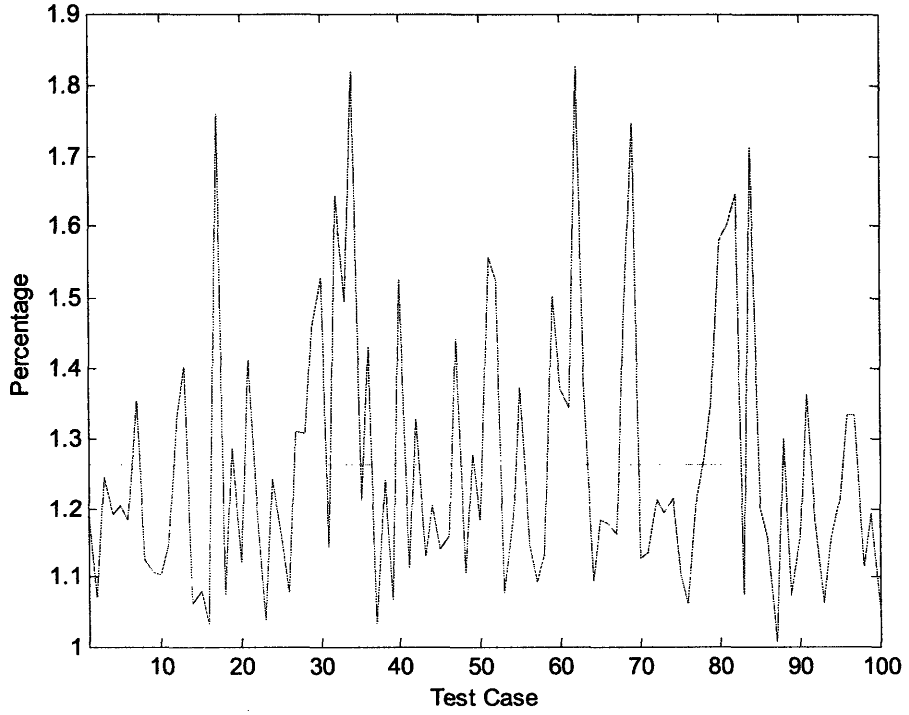

Figure D.3.7: Performance of Policy 2(f) against Minimize Utilization Heuristic

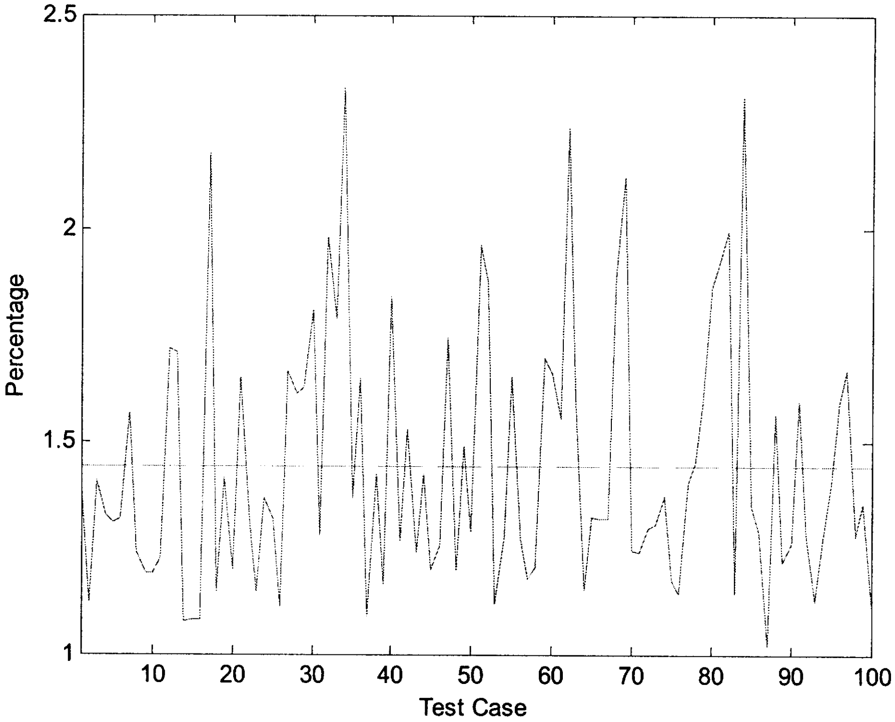

Figure D.3.8: Performance of Policy 2(g) against Minimize Utilization Heuristic

D.4 Test Results at $\rho_{test} = 0.8$

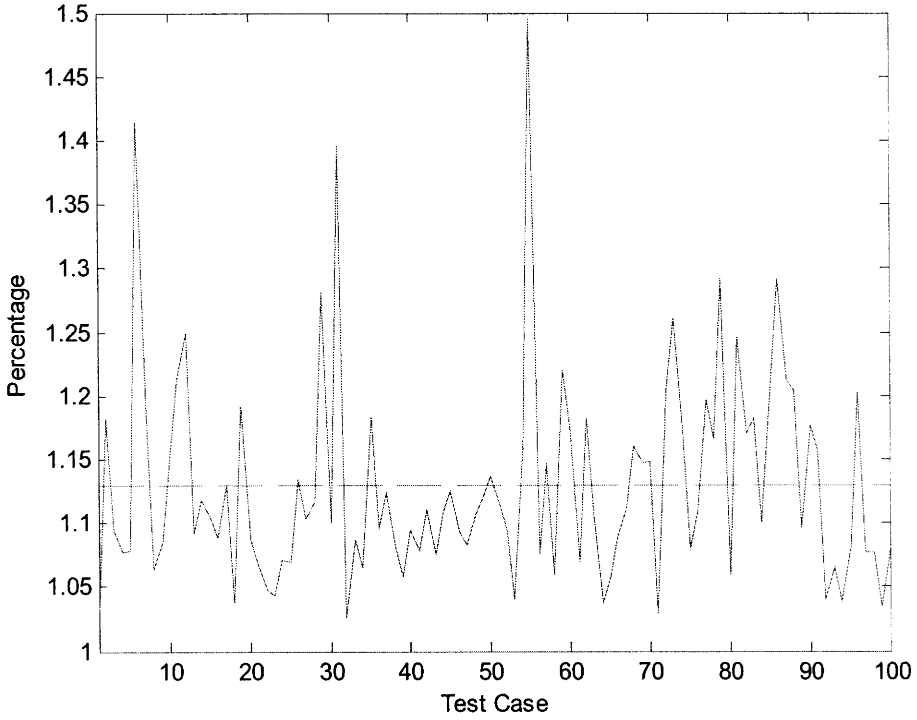

Figure D.4.1: Performance of Policy 1 against Minimize Utilization Heuristic

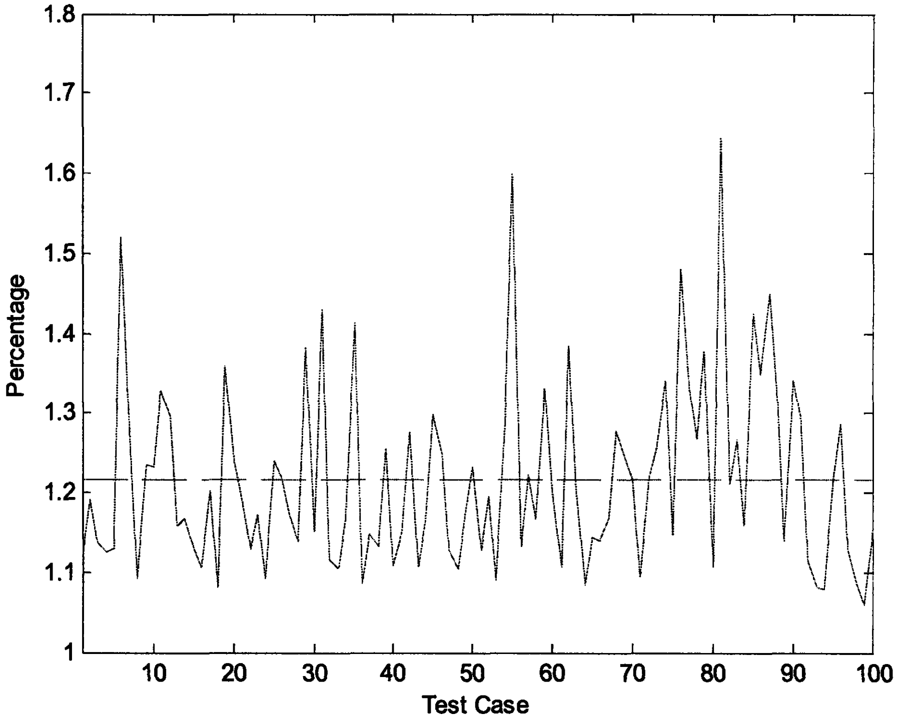

Figure D.4.2: Performance of Policy 2(a) against Minimize Utilization Heuristic

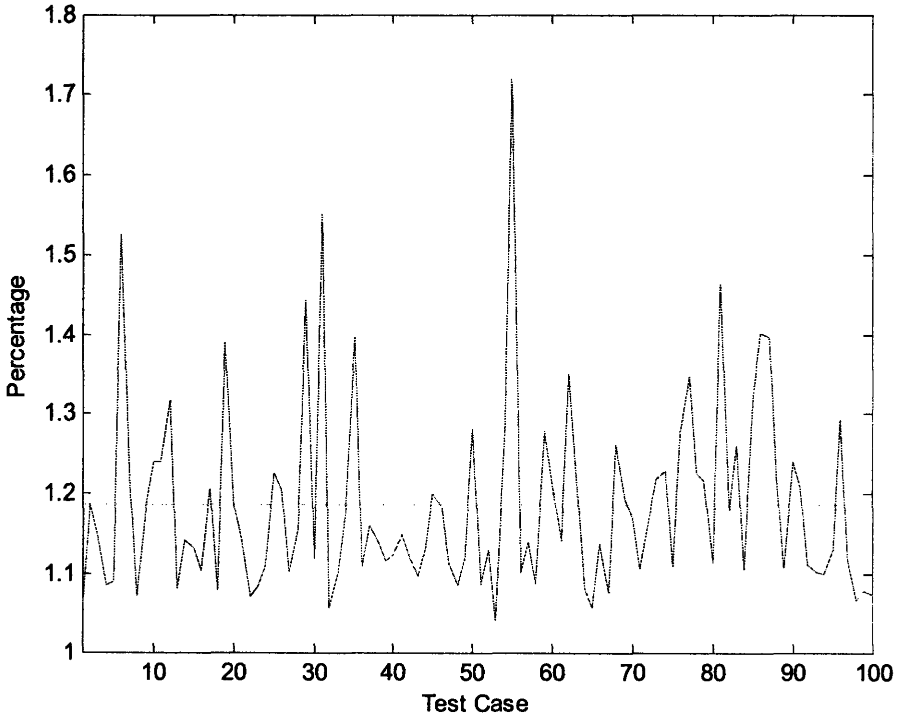

Figure D.4.3: Performance of Policy 2(b) against Minimize Utilization Heuristic

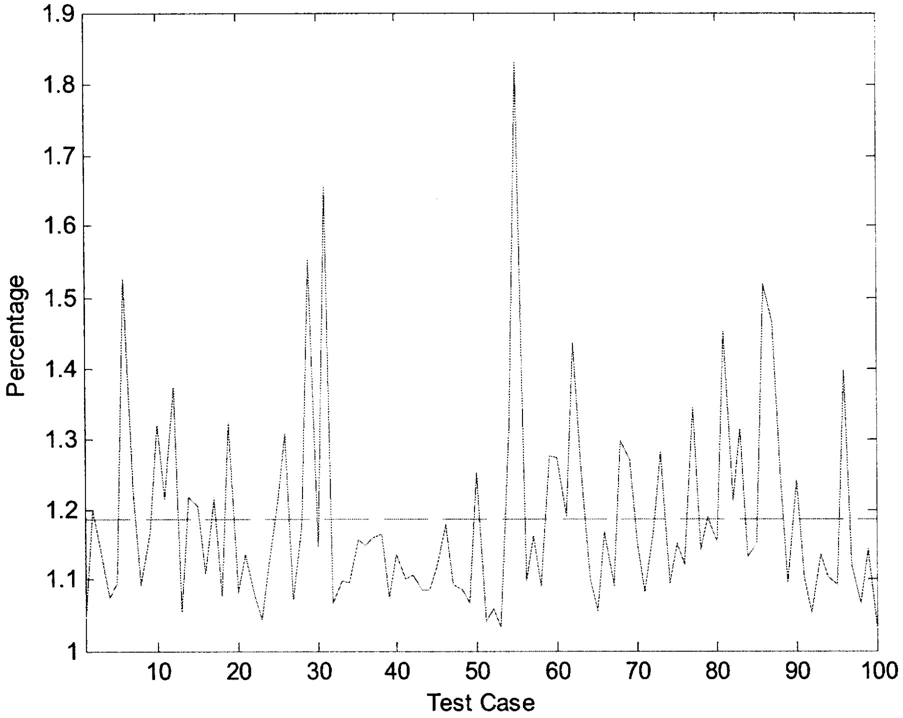

Figure D.4.4: Performance of Policy 2(c) against Minimize Utilization Heuristic

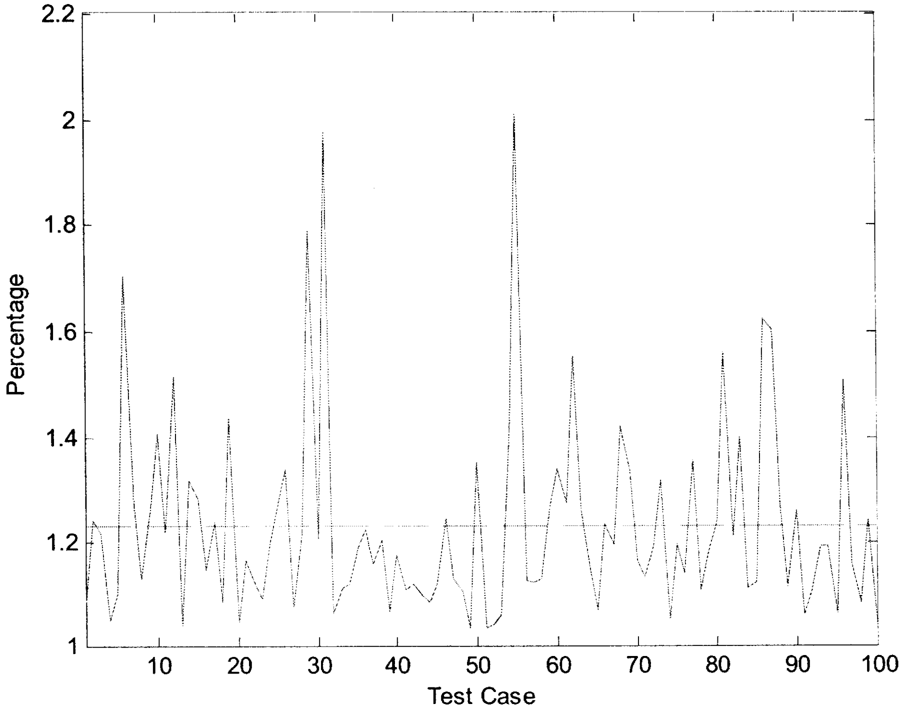

Figure D.4.5: Performance of Policy 2(d) against Minimize Utilization Heuristic

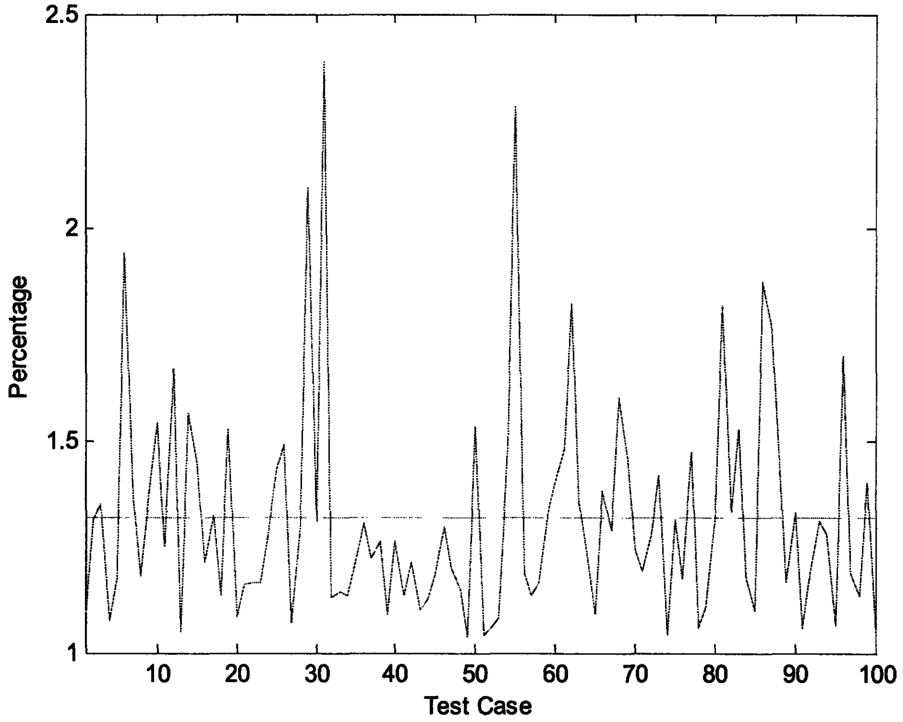

Figure D.4.6: Performance of Policy 2(e) against Minimize Utilization Heuristic

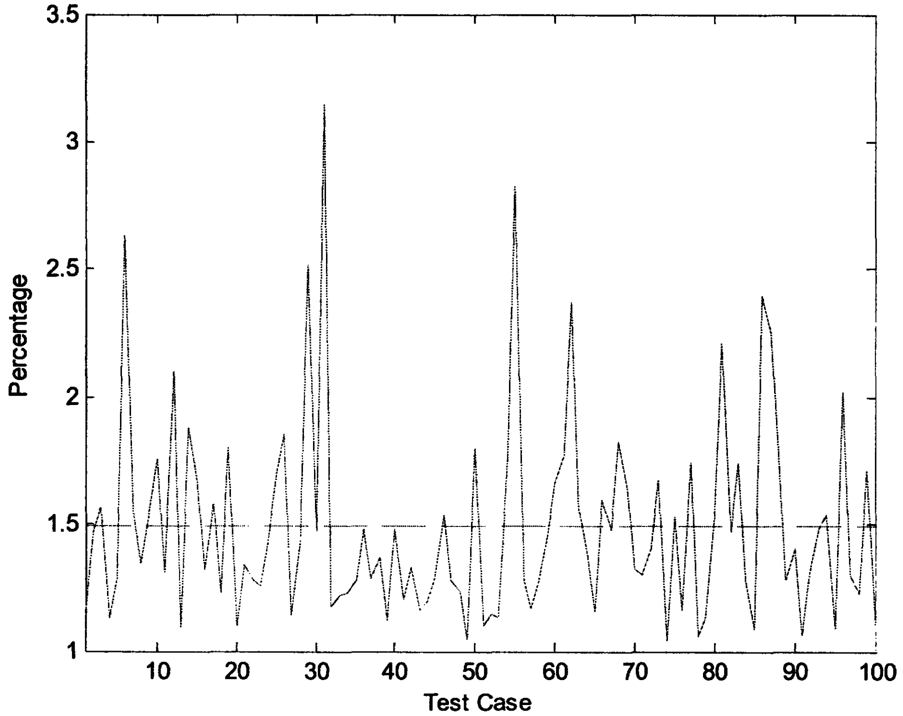

Figure D.4.7: Performance of Policy 2(f) against Minimize Utilization Heuristic

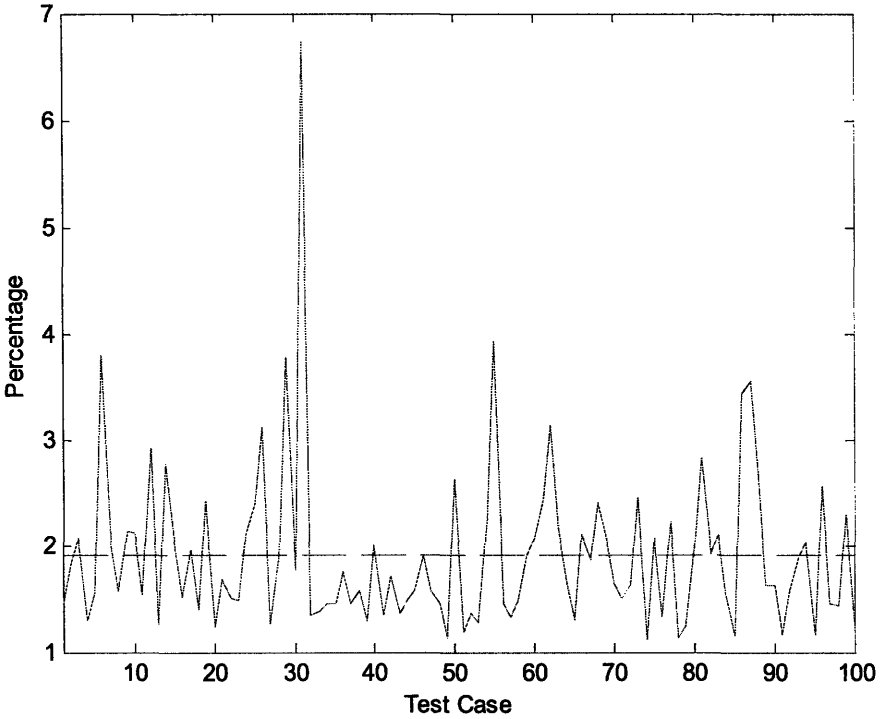

Figure D.4.8: Performance of Policy 2(g) against Minimize Utilization Heuristic

D.5 Test Results at $\rho_{test} = 0.9$

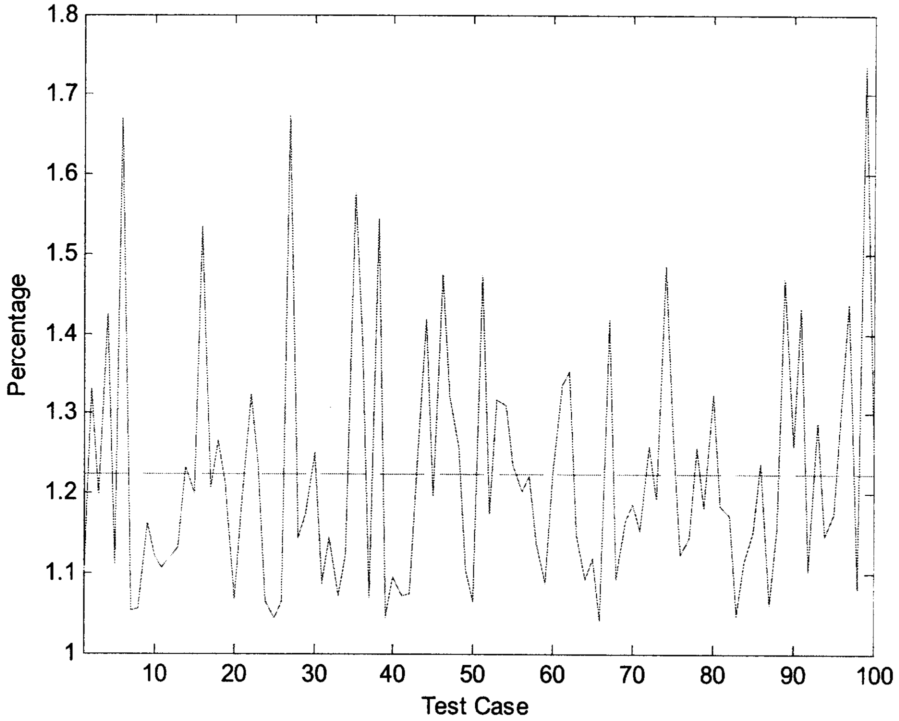

Figure D.5.1: Performance of Policy 1 against Minimize Utilization Heuristic

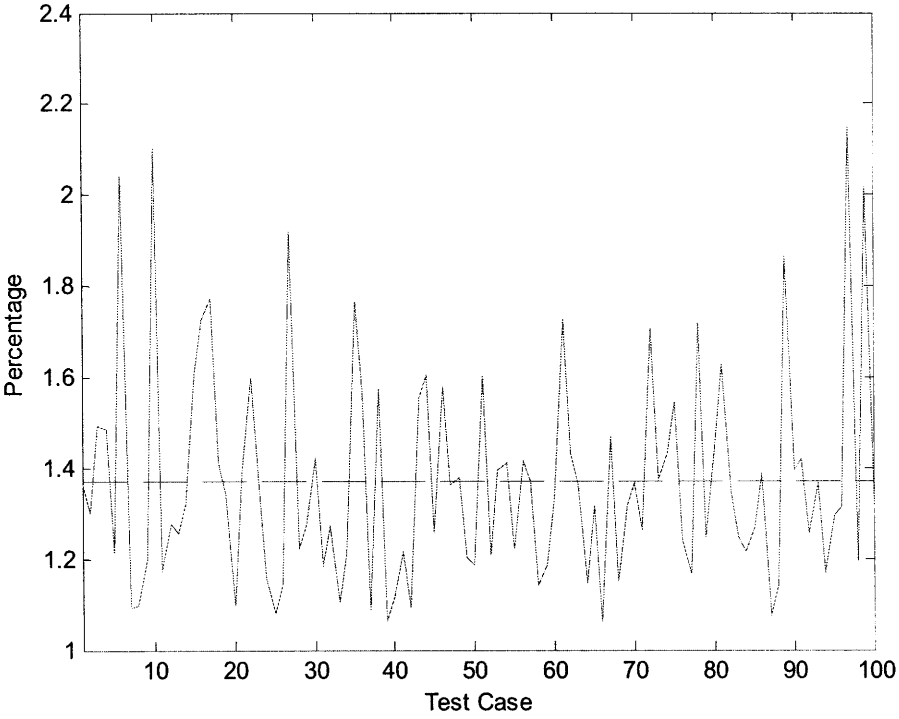

Figure D.5.2: Performance of Policy 2(a) against Minimize Utilization Heuristic

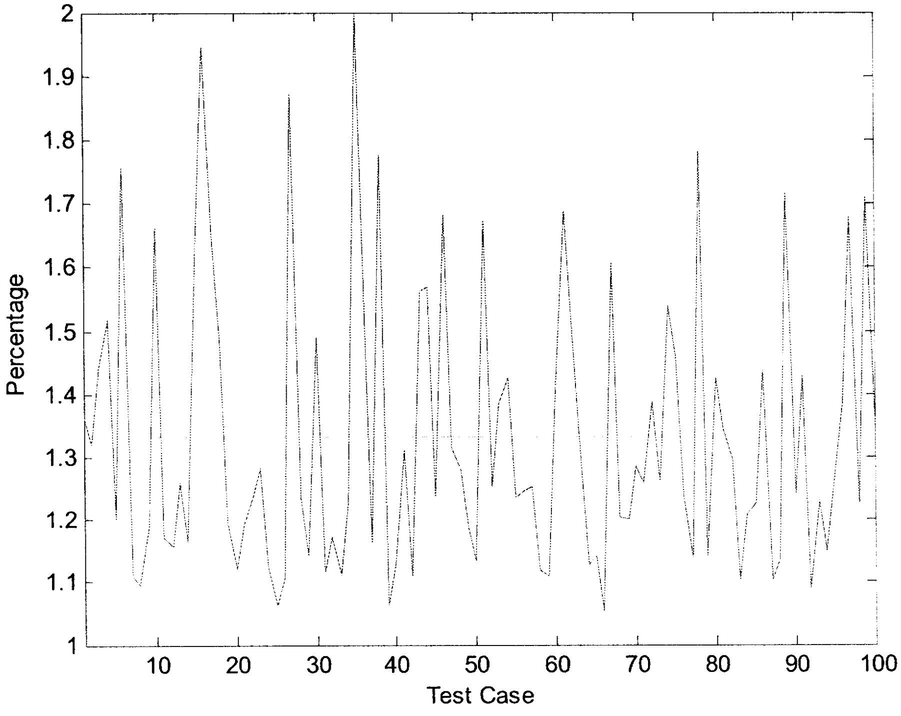

Figure D.5.3: Performance of Policy 2(b) against Minimize Utilization Heuristic

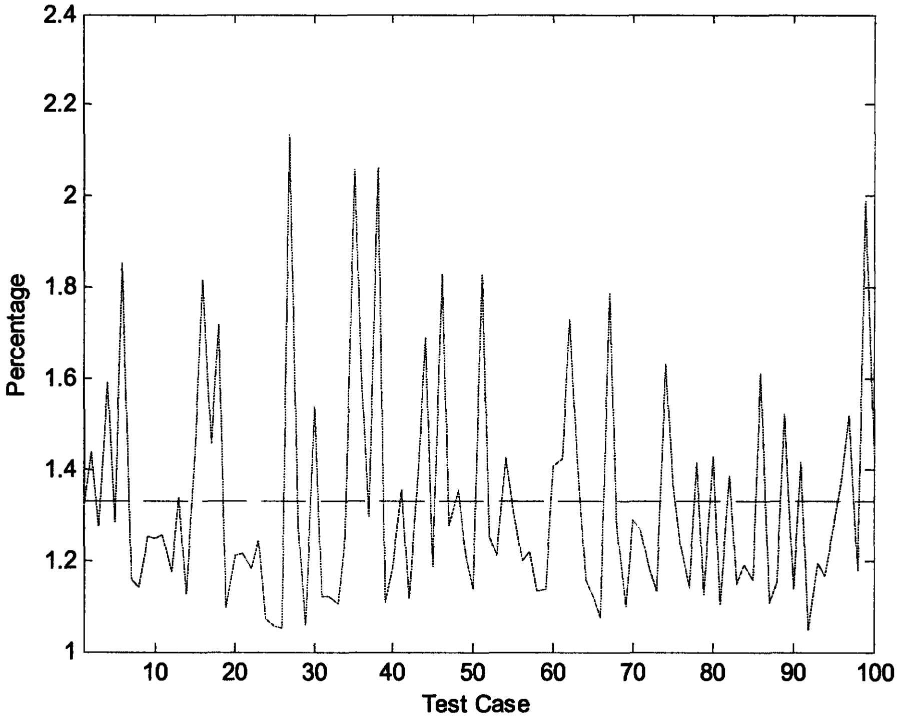

Figure D.5.4: Performance of Policy 2(c) against Minimize Utilization Heuristic

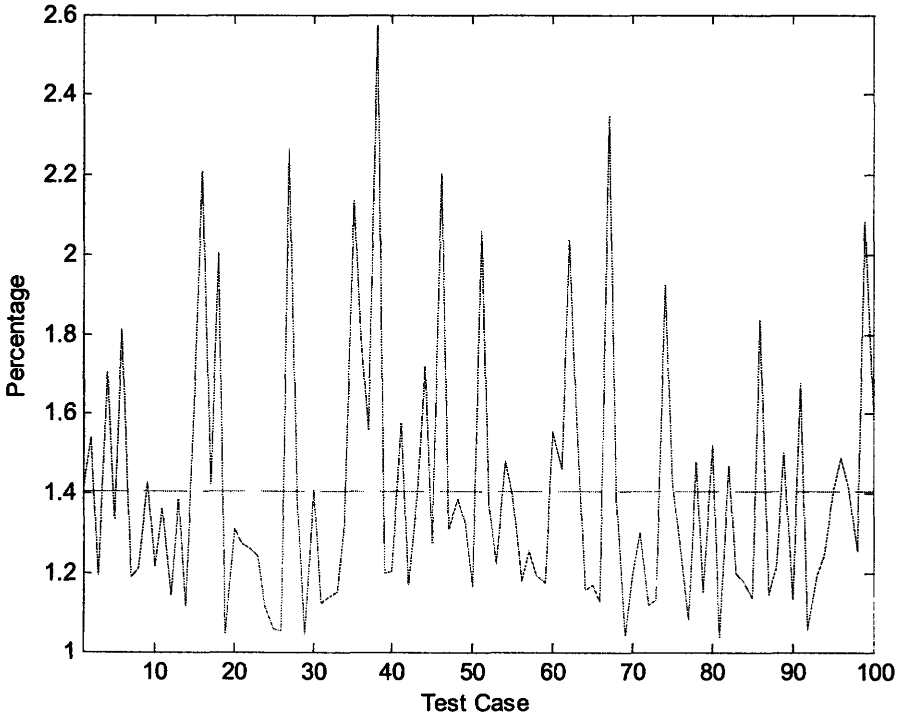

Figure D.5.5: Performance of Policy 2(d) against Minimize Utilization Heuristic

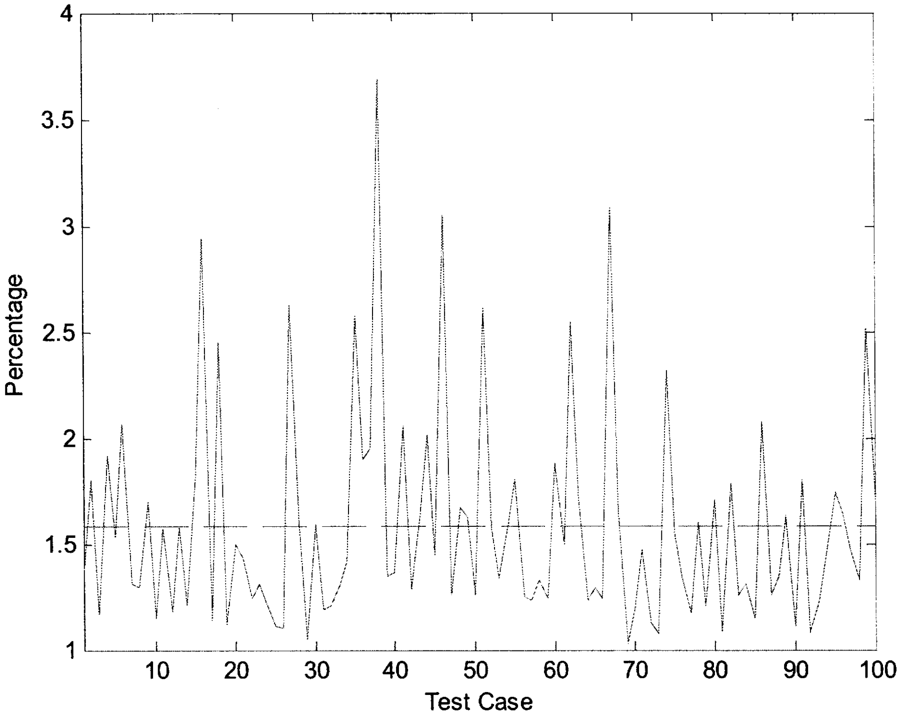

Figure D.5.6: Performance of Policy 2(e) against Minimize Utilization Heuristic

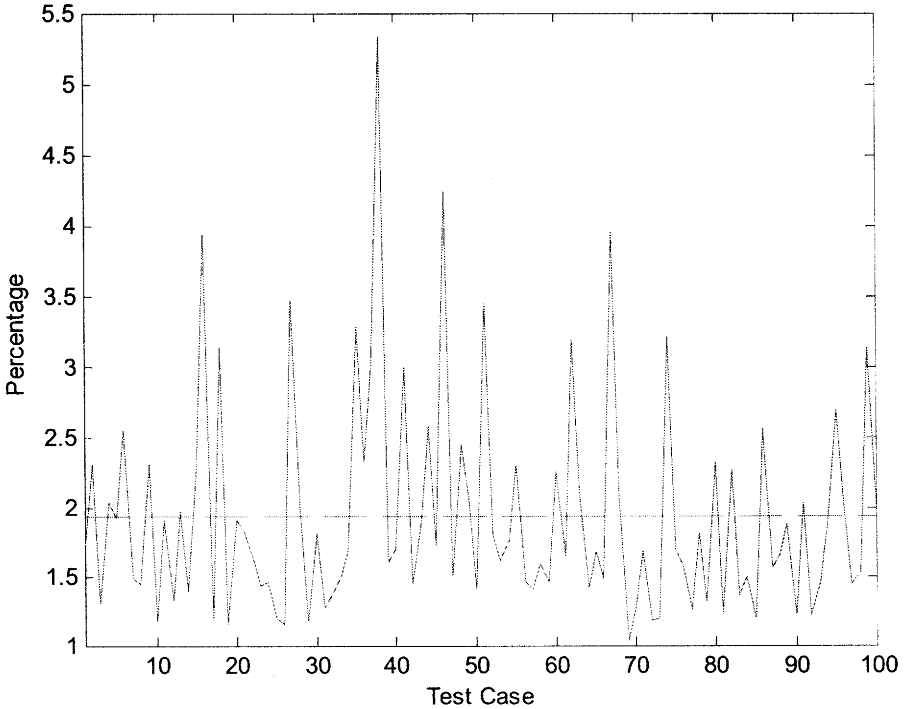

Figure D.5.7: Performance of Policy 2(f) against Minimize Utilization Heuristic

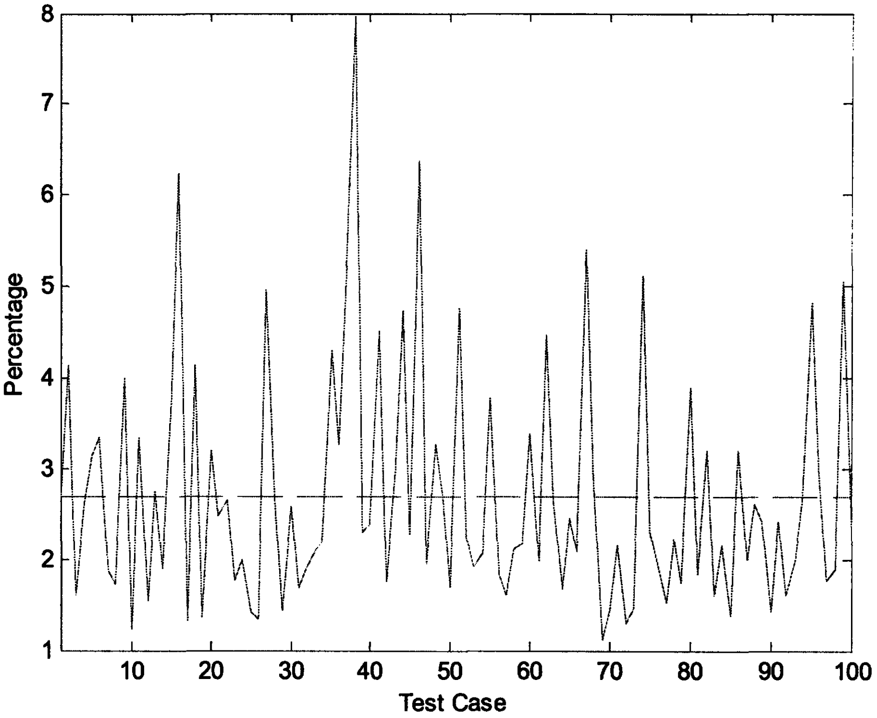

Figure D.5.8: Performance of Policy 2(g) against Minimize Utilization Heuristic

(This Page Intentionally Left Blank)

Appendix E

E.1 Test Results at $\rho_{test} = 0.1$

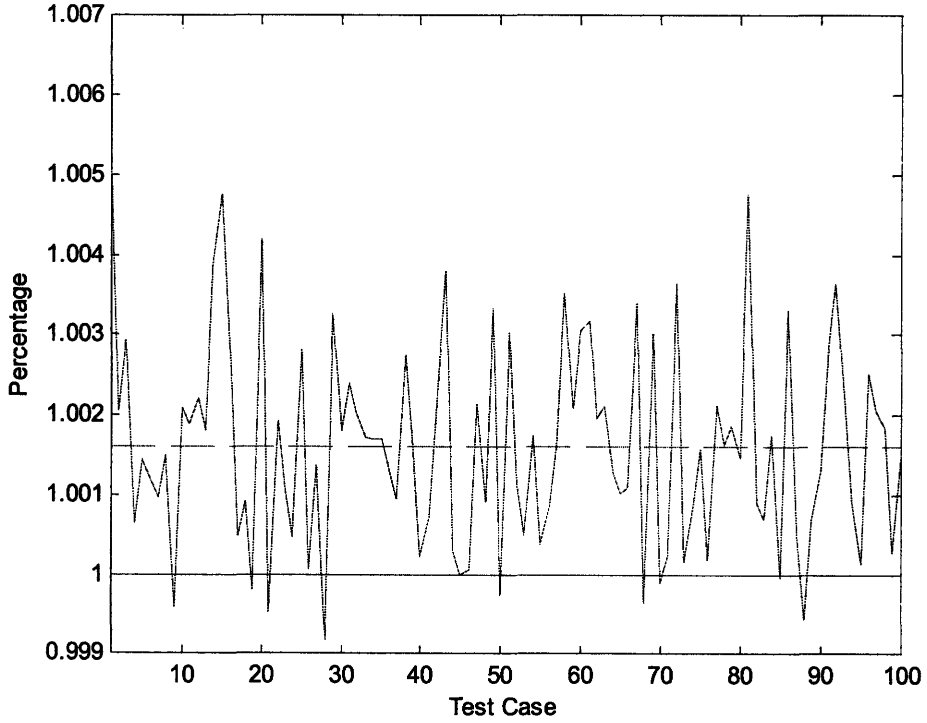

Figure E.1.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.11$

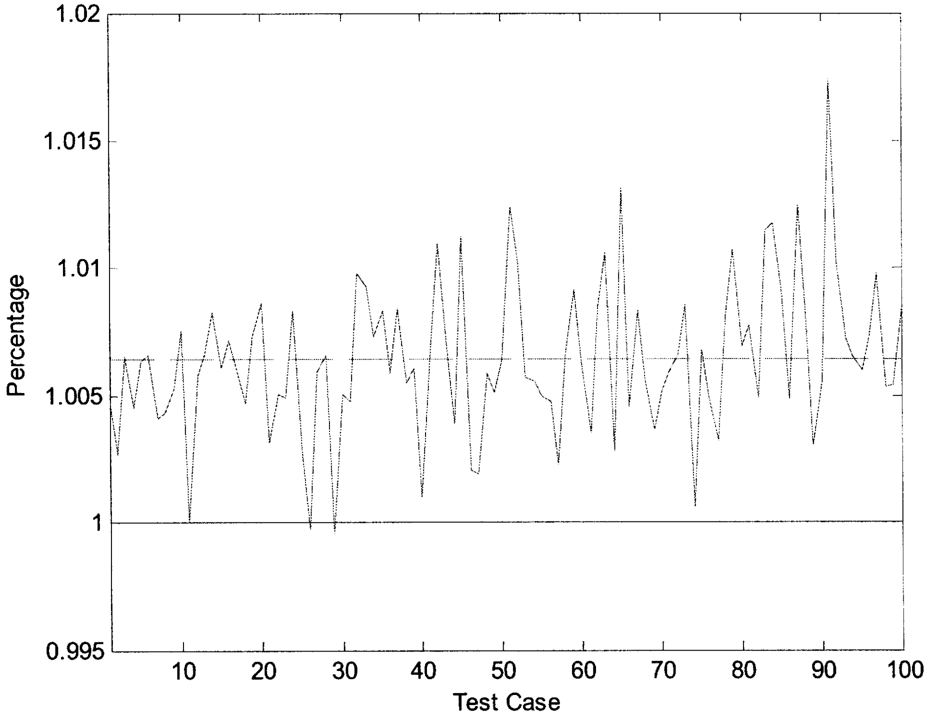

Figure E.1.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.12$

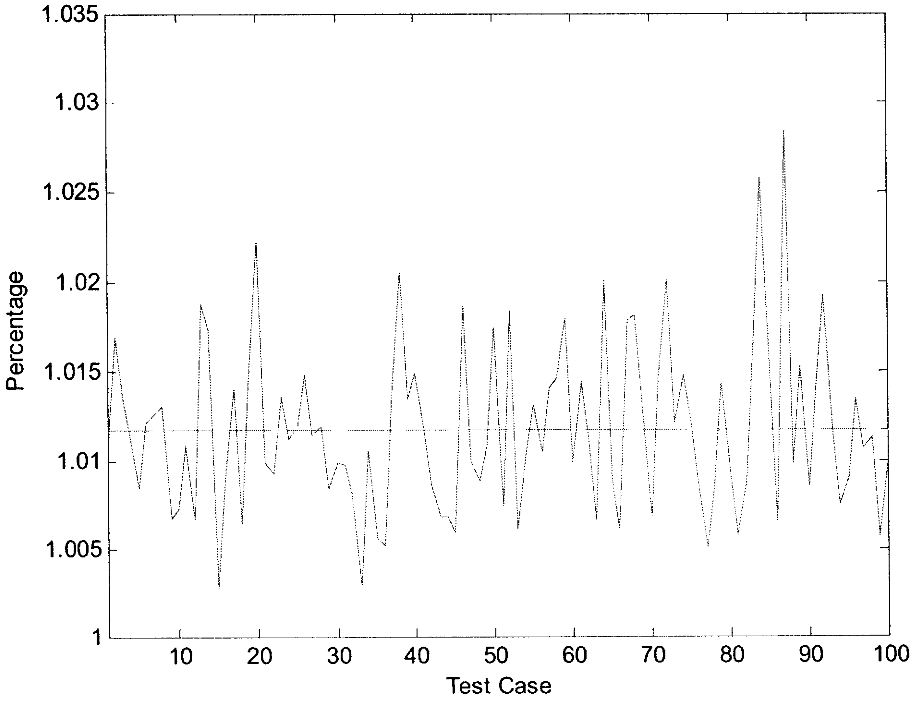

Figure E.1.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.13$

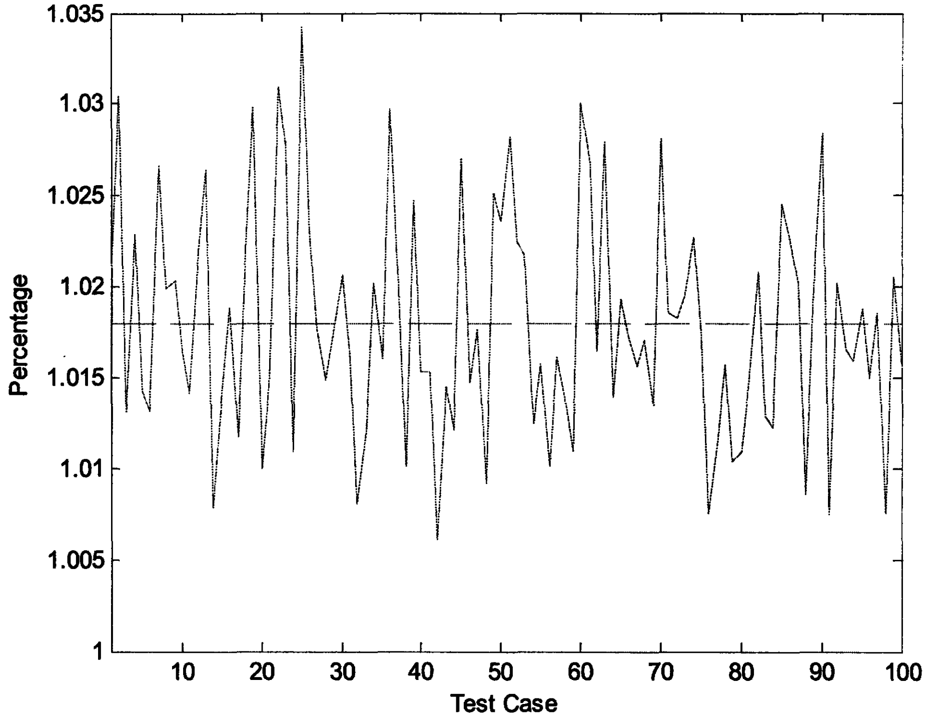

Figure E.1.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.14$

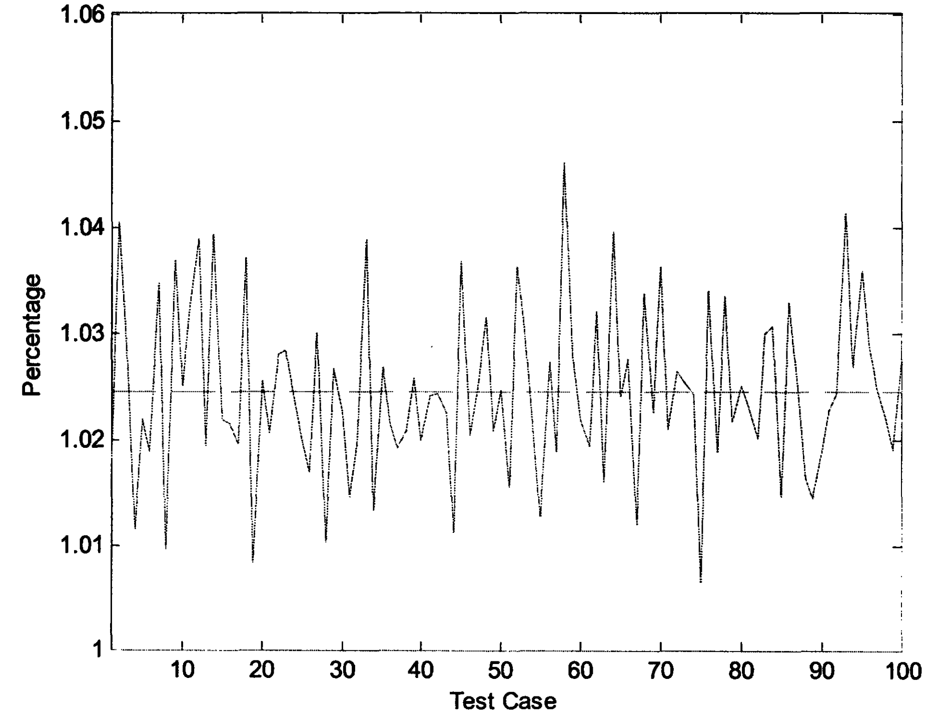

Figure E.1.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.15$

E.2 Test Results at $\rho_{test} = 0.2$

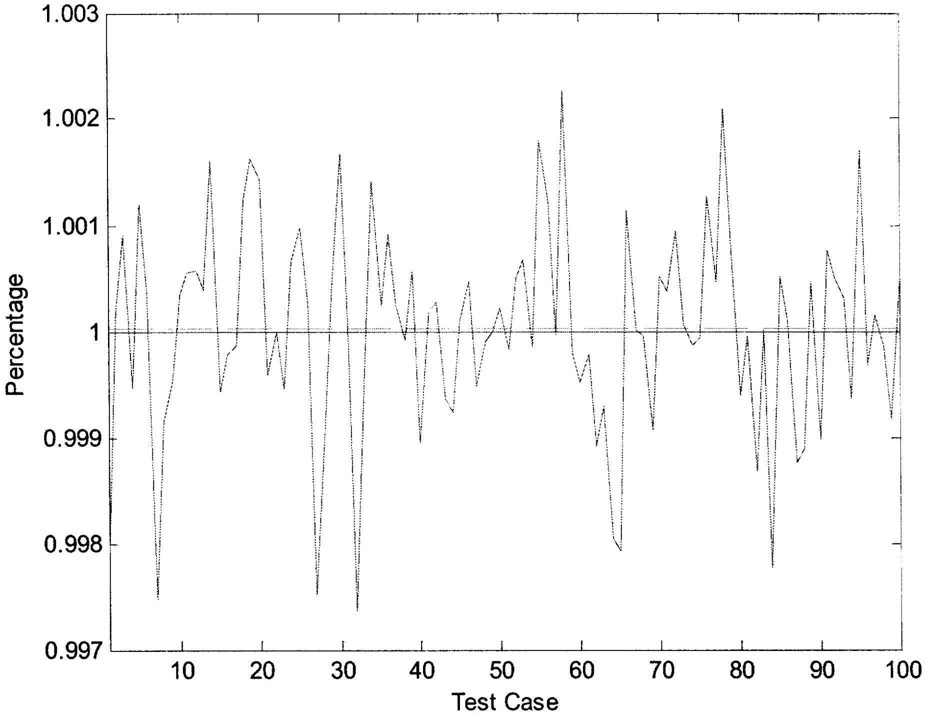

Figure E.2.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.21$

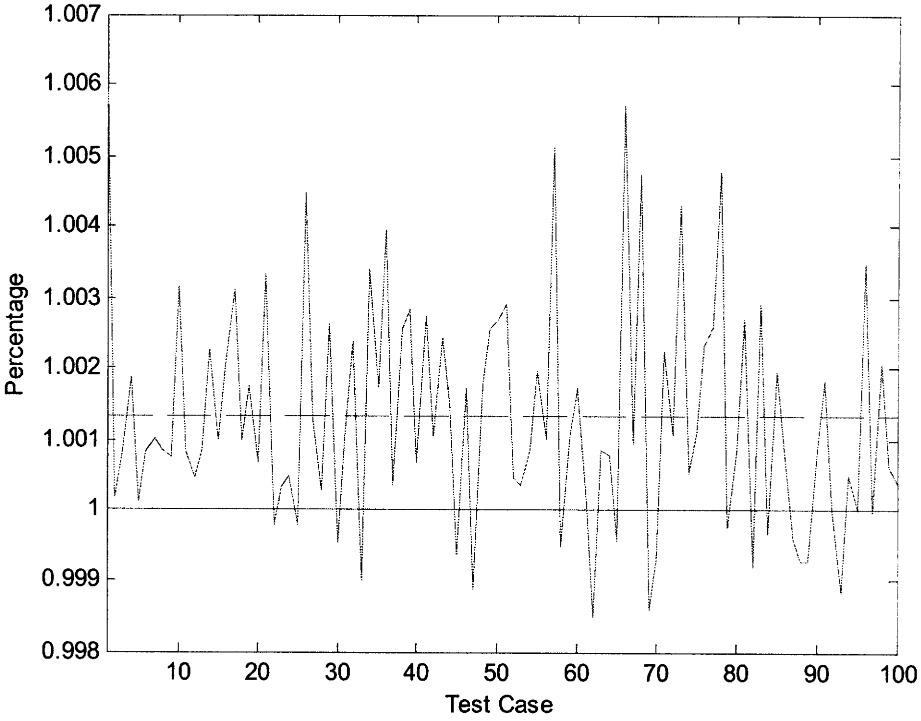

Figure E.2.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.22$

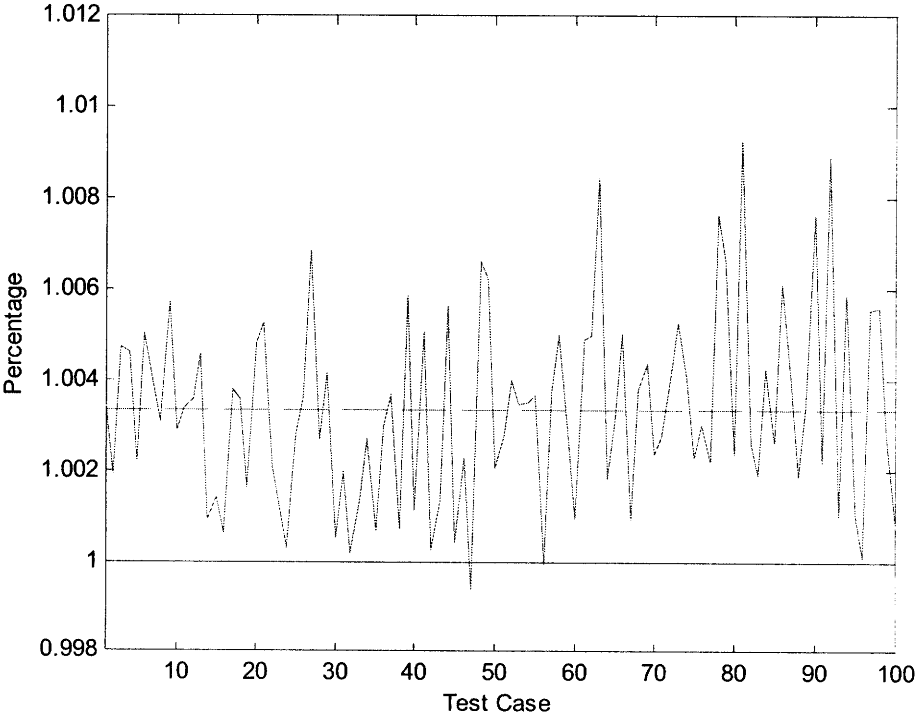

Figure E.2.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.23$

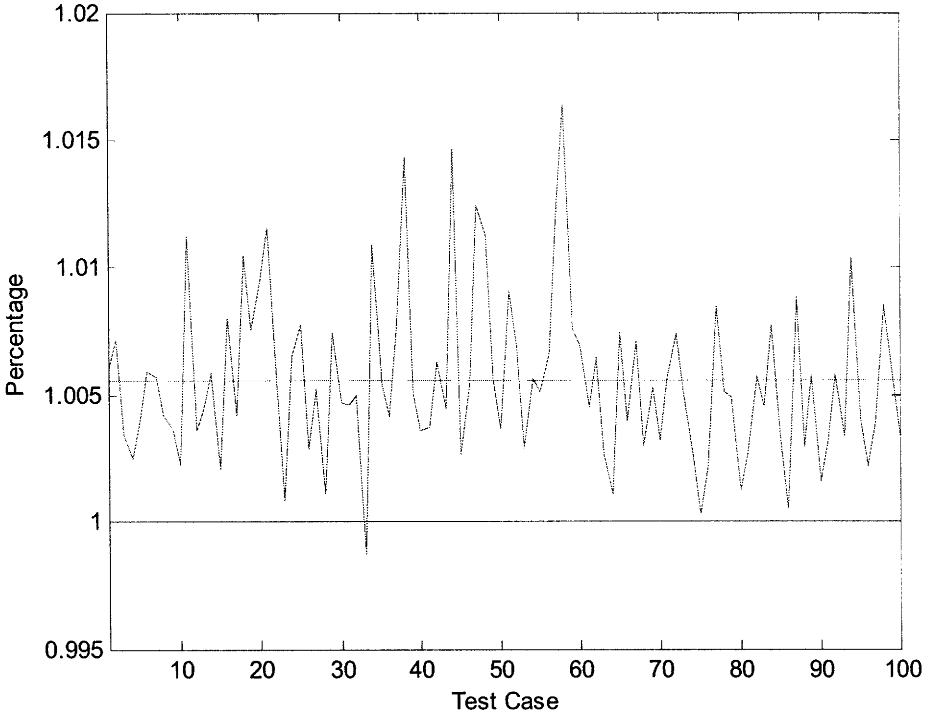

Figure E.2.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.24$

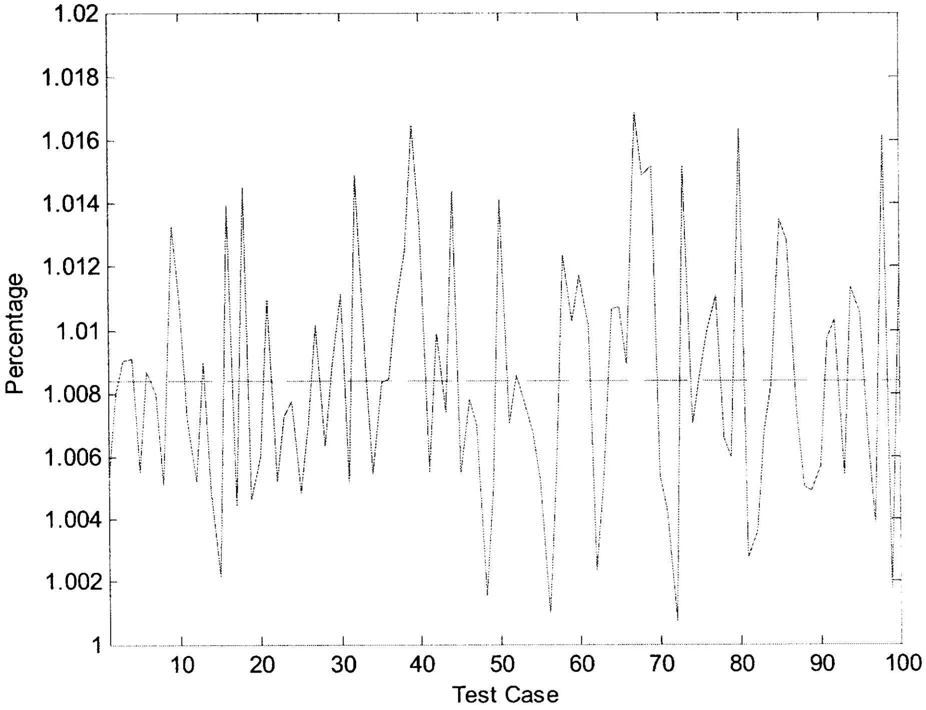

Figure E.2.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.25$

E.3 Test Results at $\rho_{test} = 0.3$

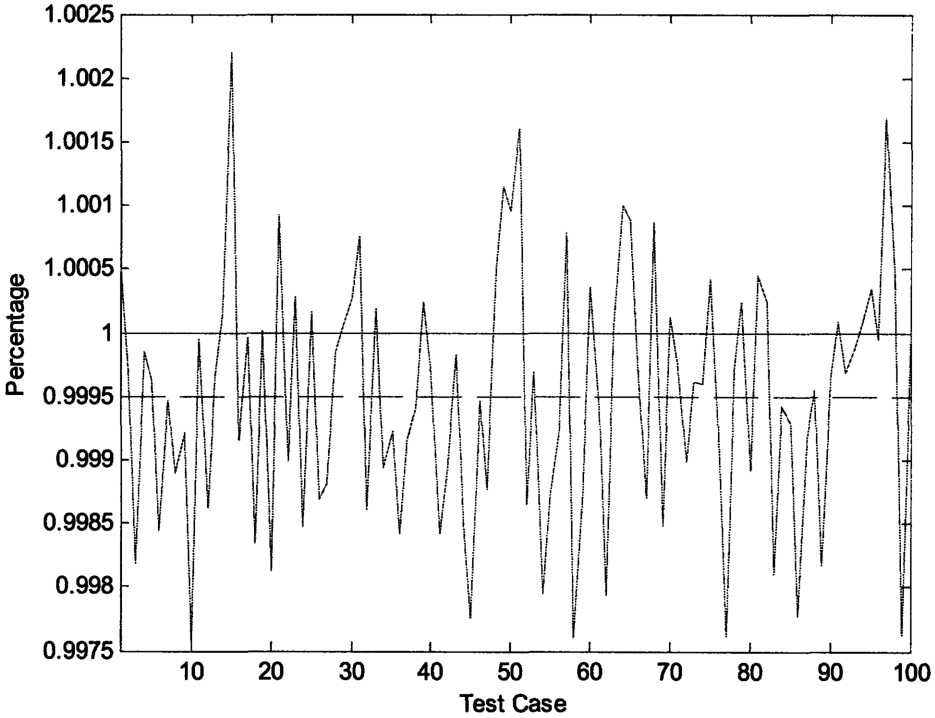

Figure E.3.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.31$

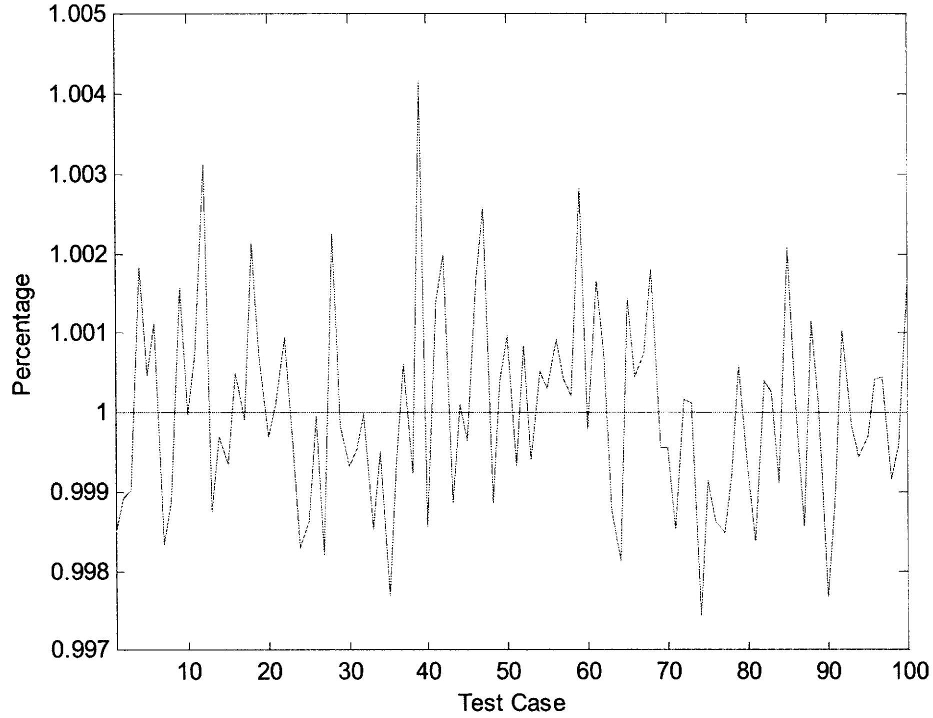

Figure E.3.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.32$

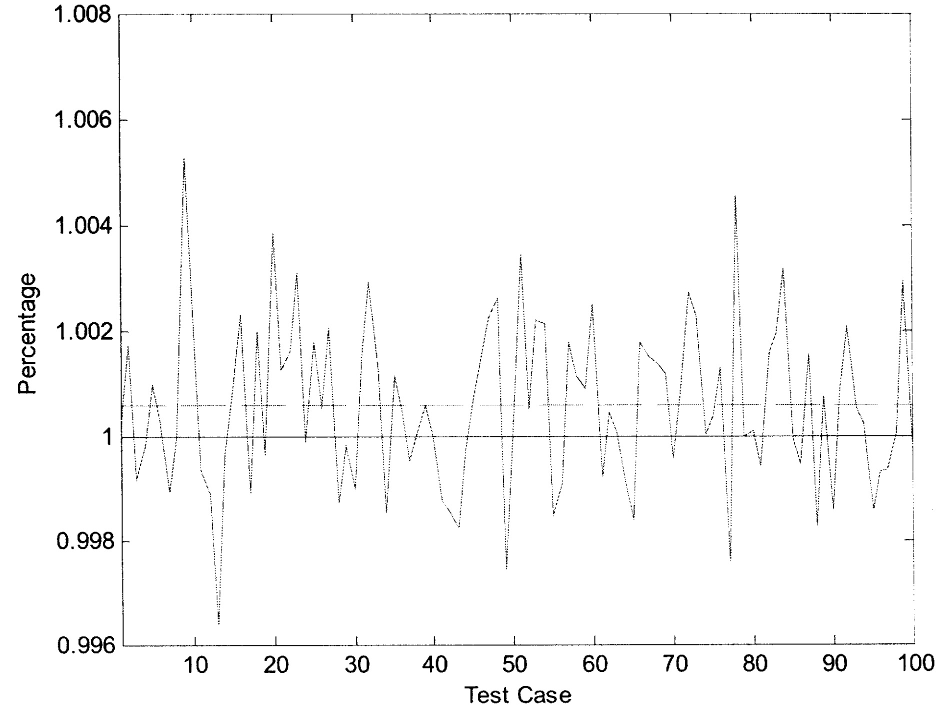

Figure E.3.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.33$

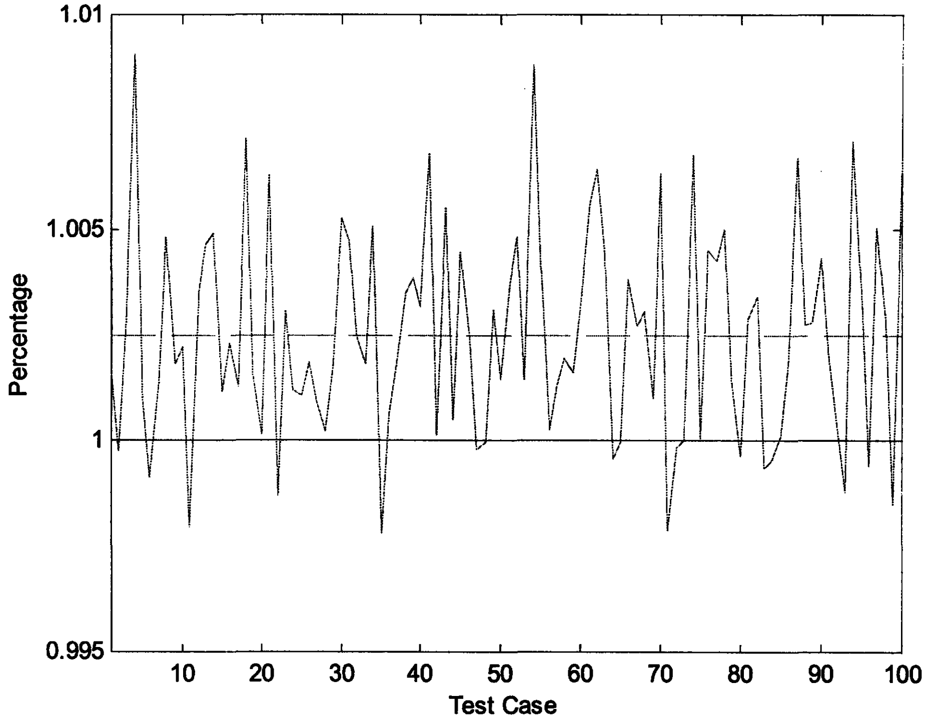

Figure E.3.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.34$

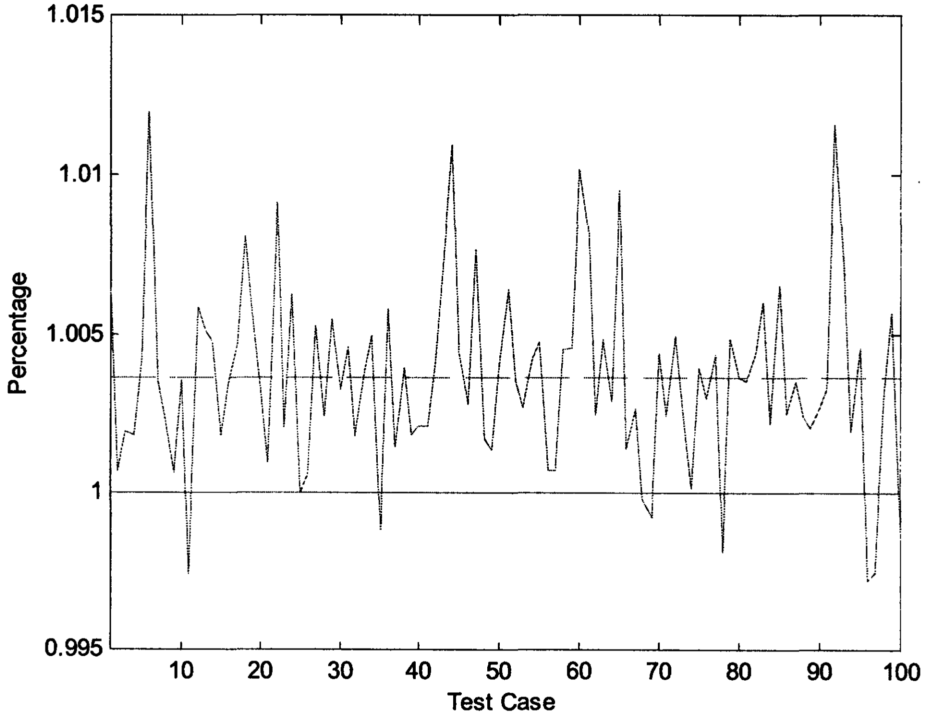

Figure E.3.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.35$

E.4 Test Results at $\rho_{test} = 0.4$

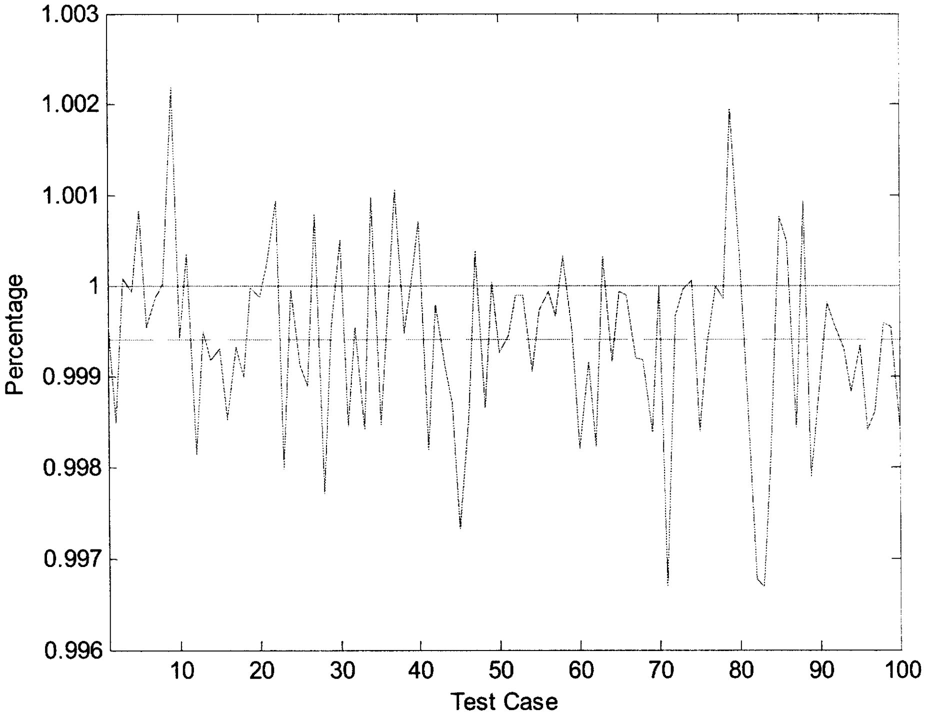

Figure E.4.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.41$

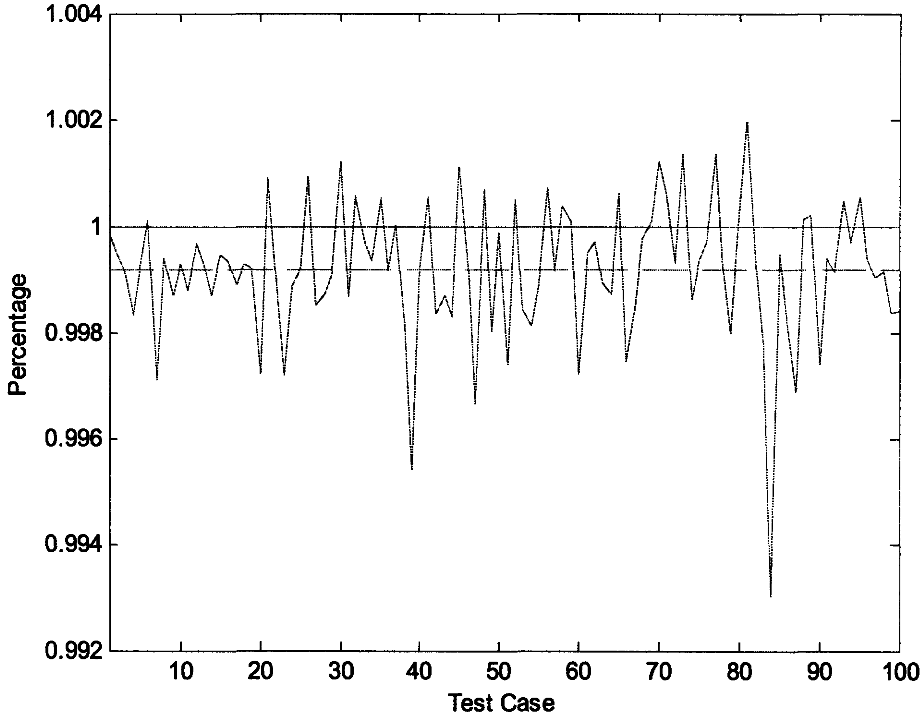

Figure E.4.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.42$

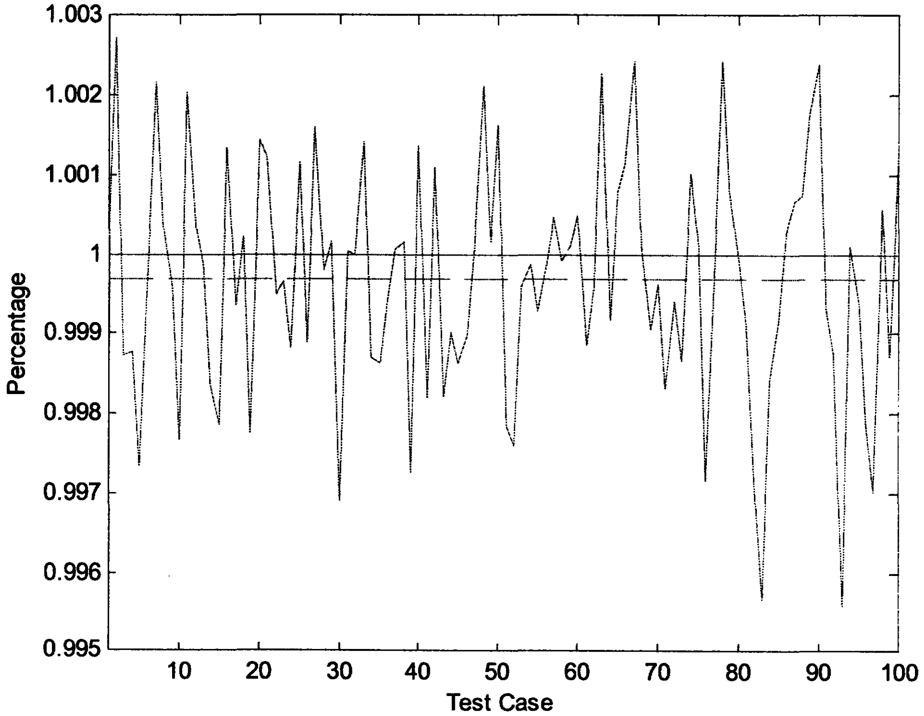

Figure E.4.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.43$

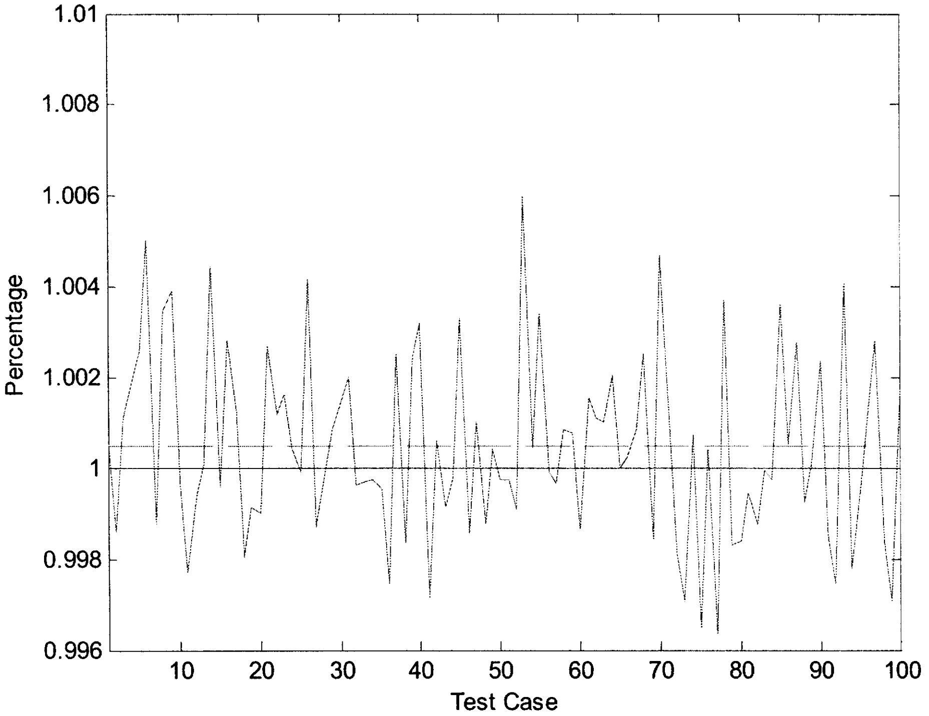

Figure E.4.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.44$

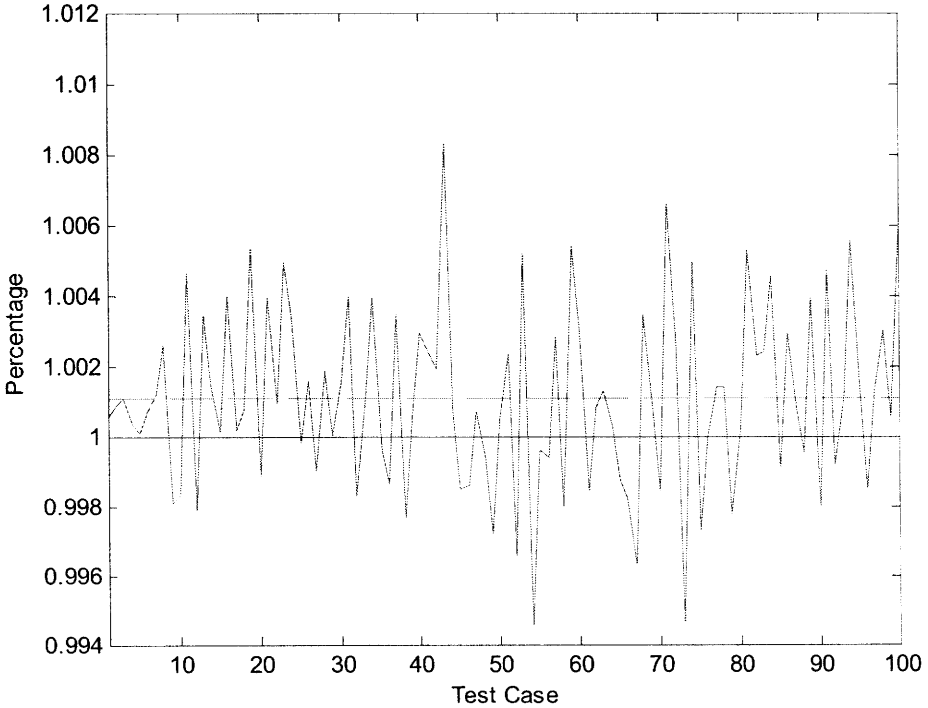

Figure E.4.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.45$

E.5 Test Results at $\rho_{test} = 0.5$

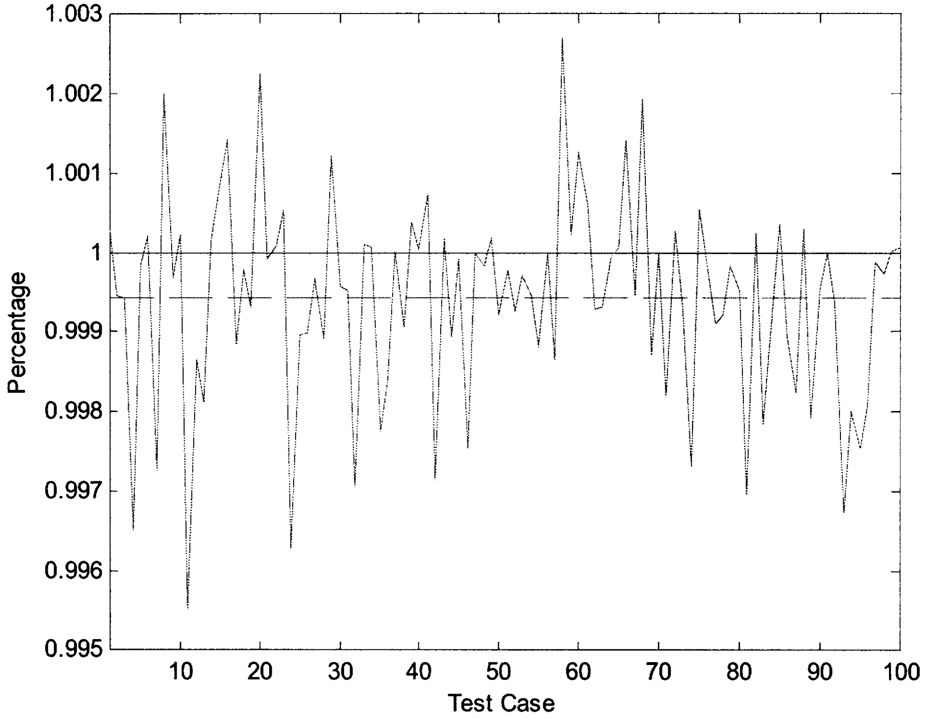

Figure E.5.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.51$

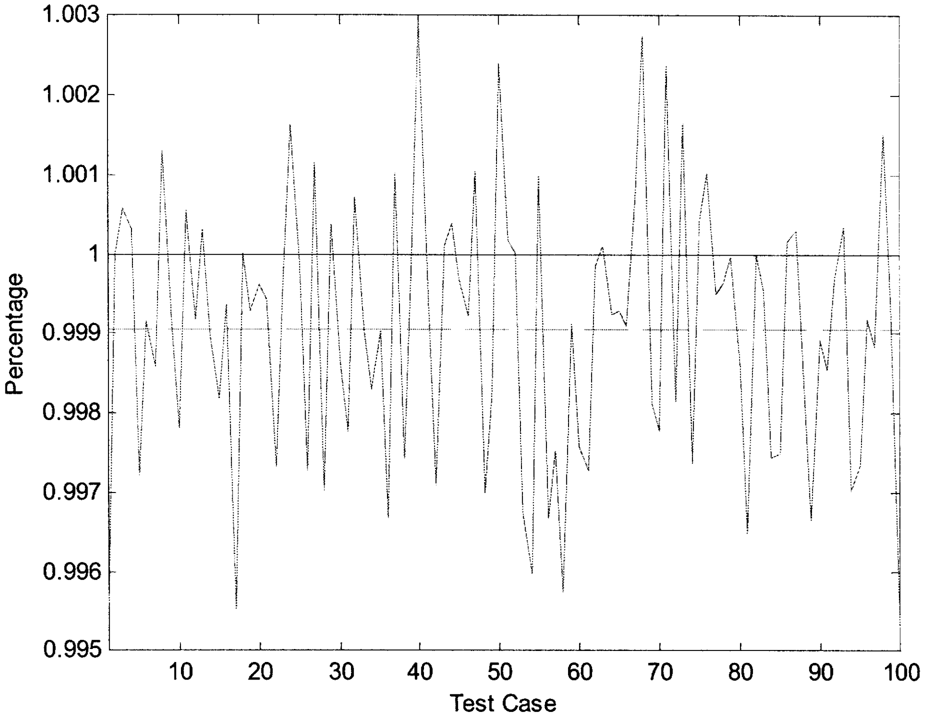

Figure E.5.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.52$

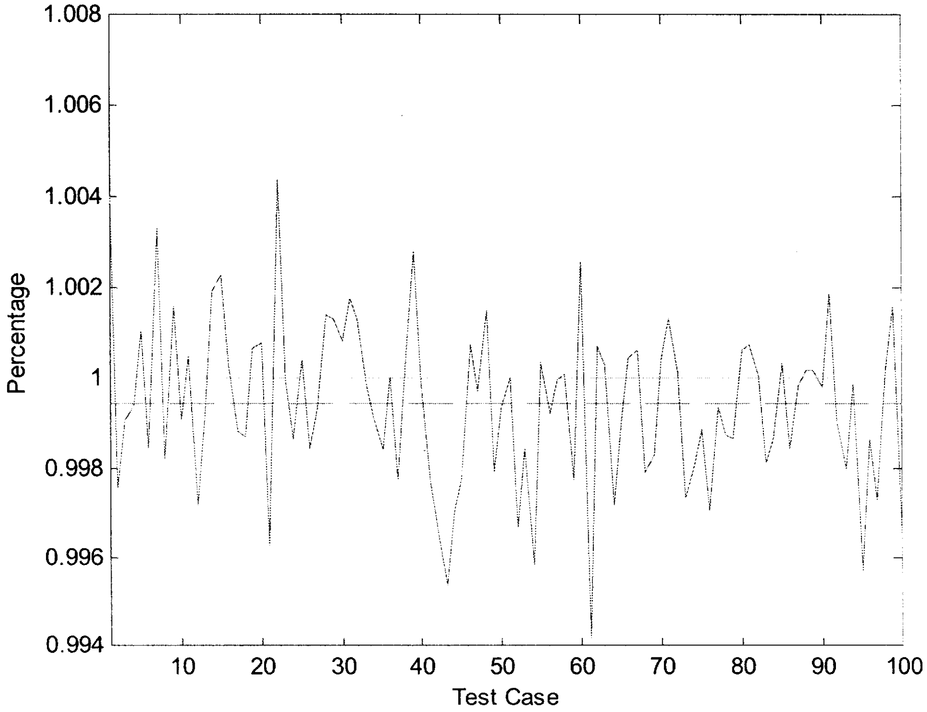

Figure E.5.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.53$

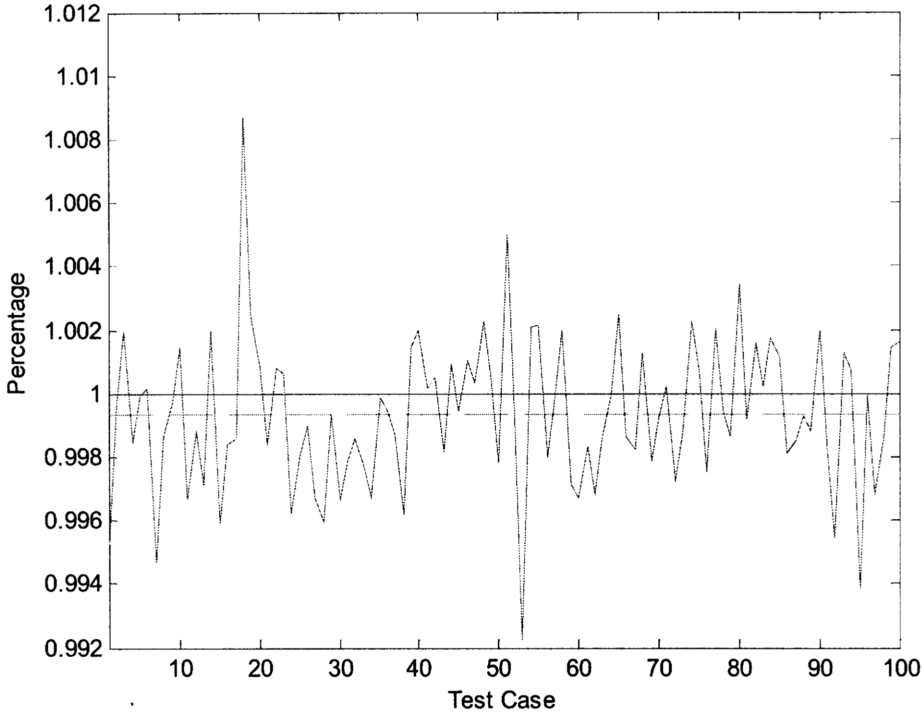

Figure E.5.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.54$

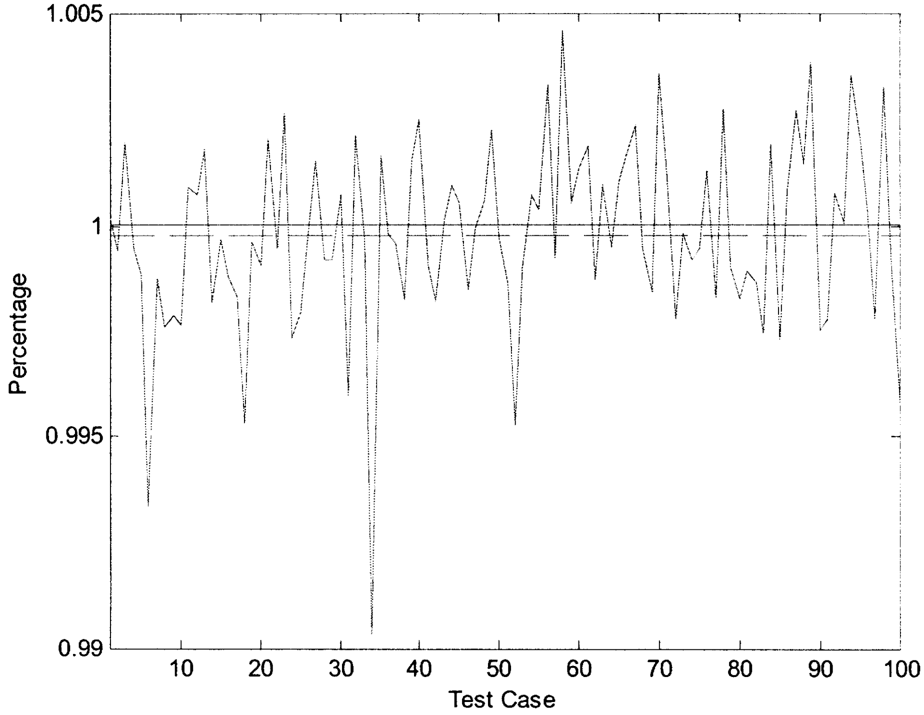

Figure E.5.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.55$

E.6 Test Results at $\rho_{test} = 0.6$

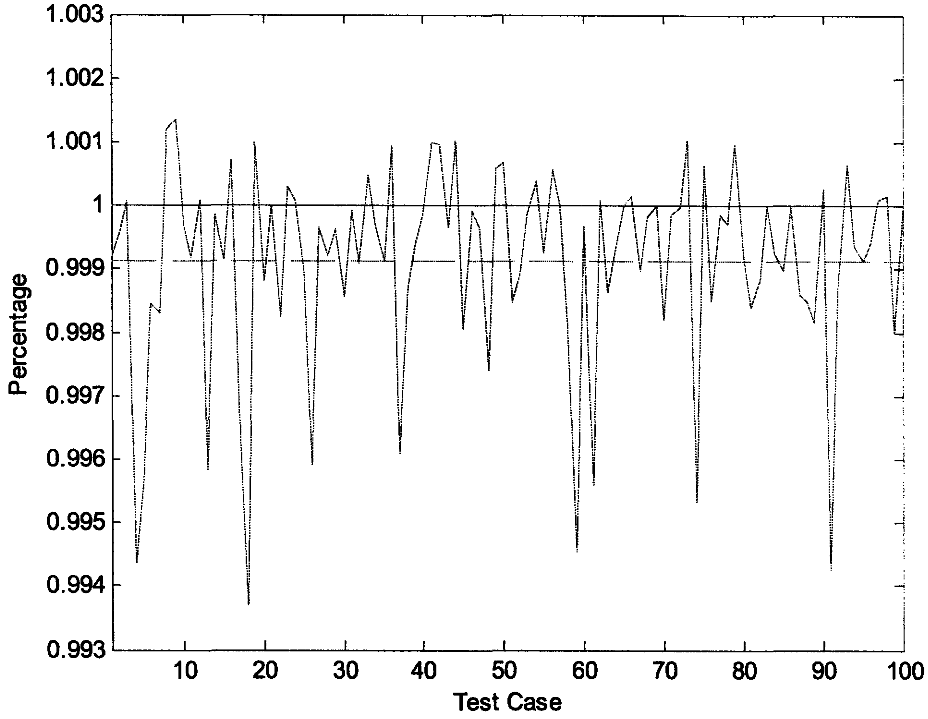

Figure E.6.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.61$

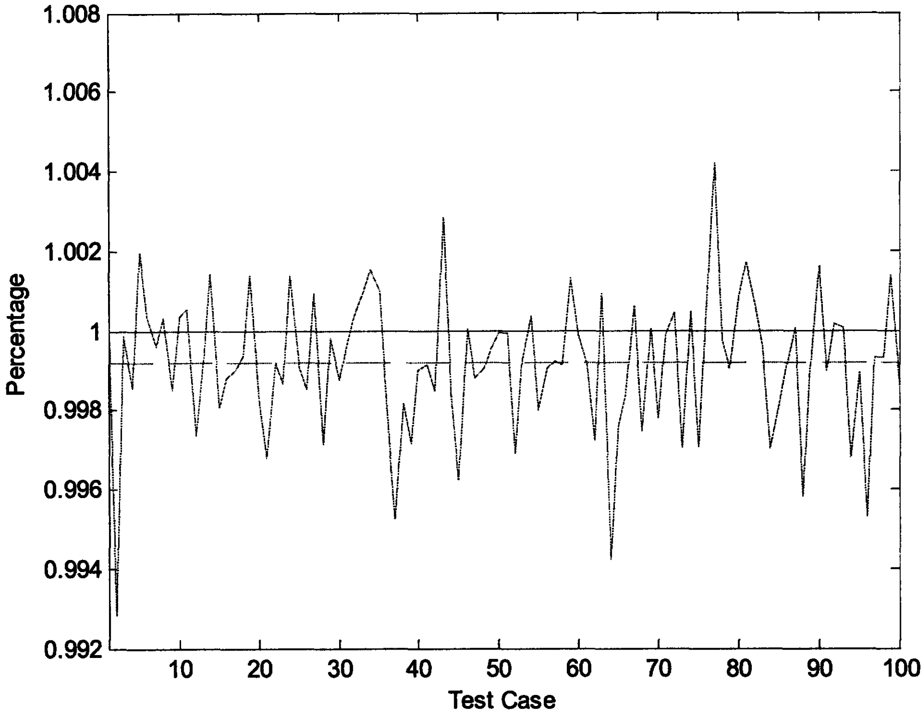

Figure E.6.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.62$

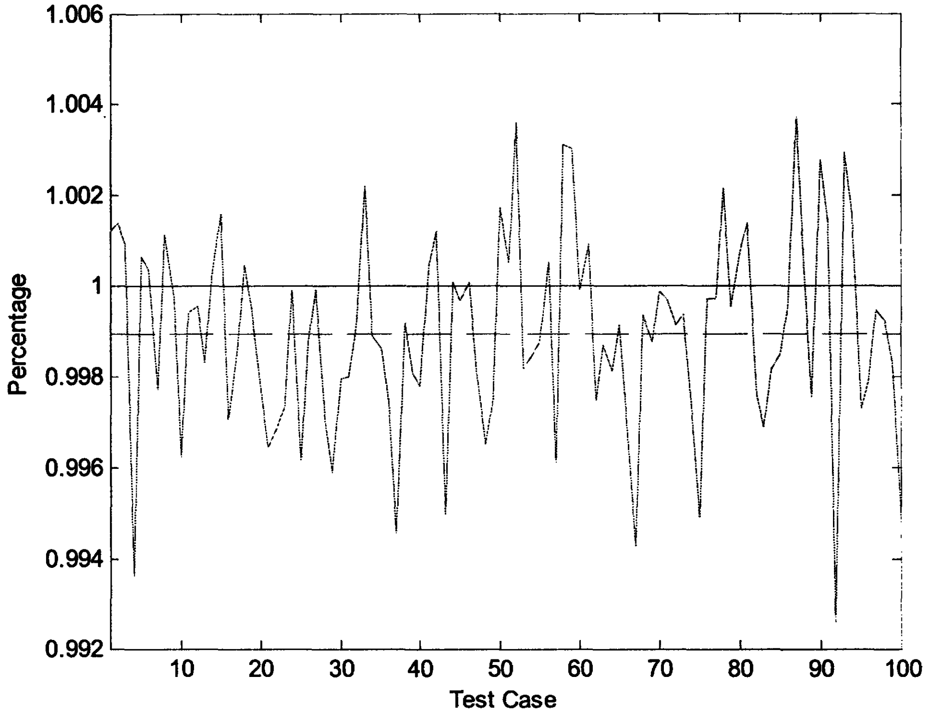

Figure E.6.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.63$

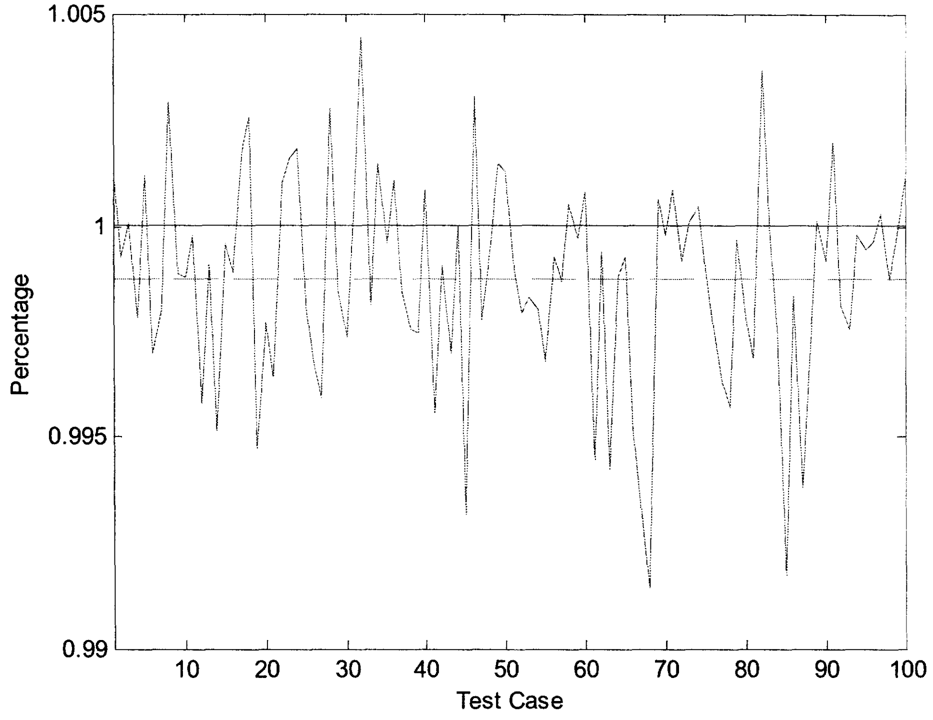

Figure E.6.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.64$

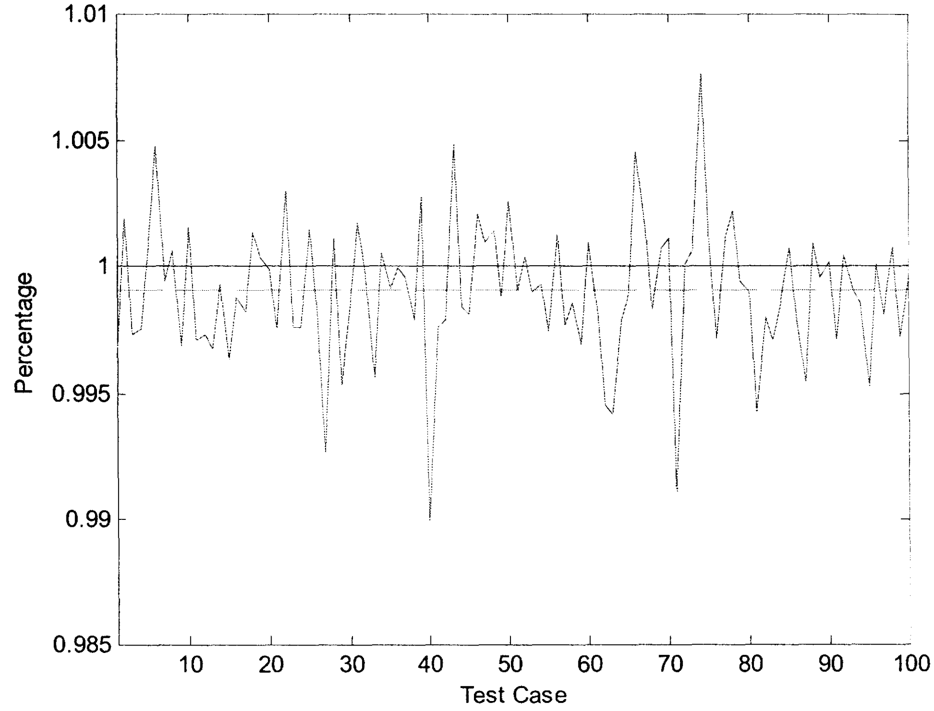

Figure E.6.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.65$

E.7 Test Results at $\rho_{test} = 0.7$

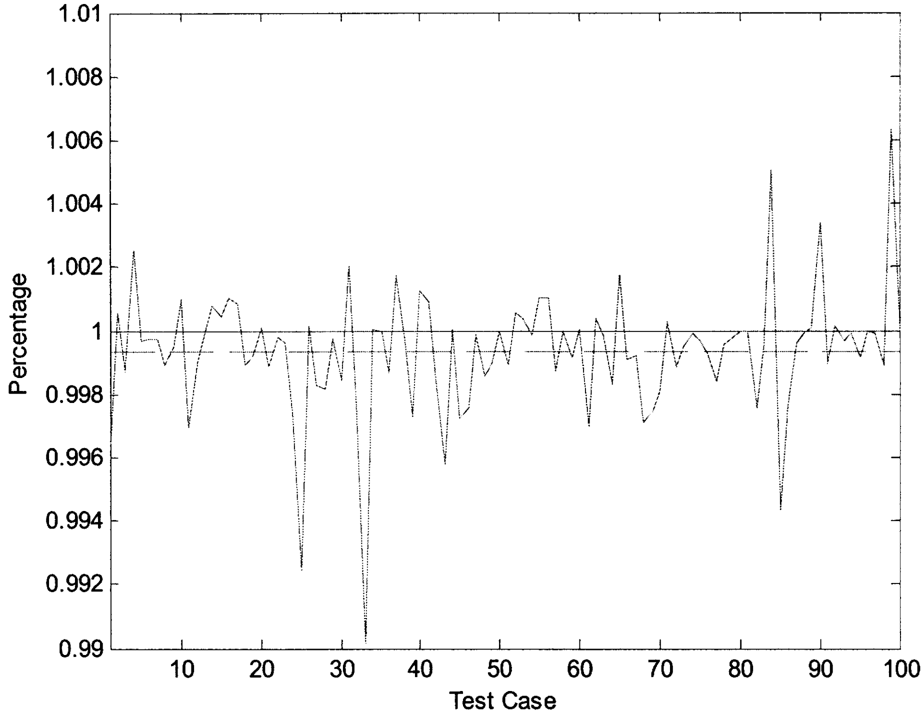

Figure E.7.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.71$

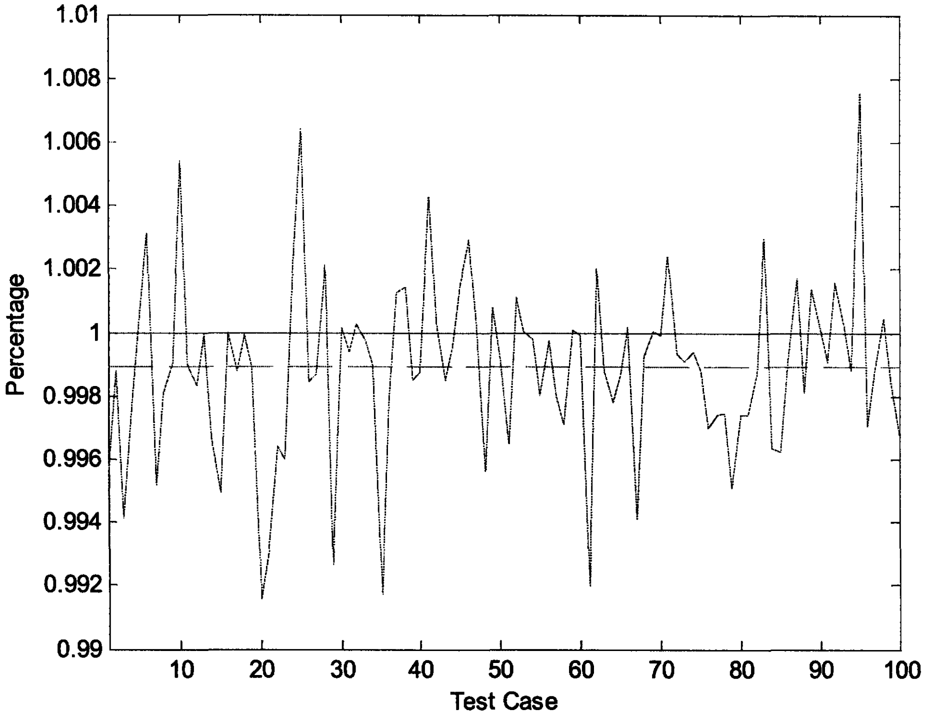

Figure E.7.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.72$

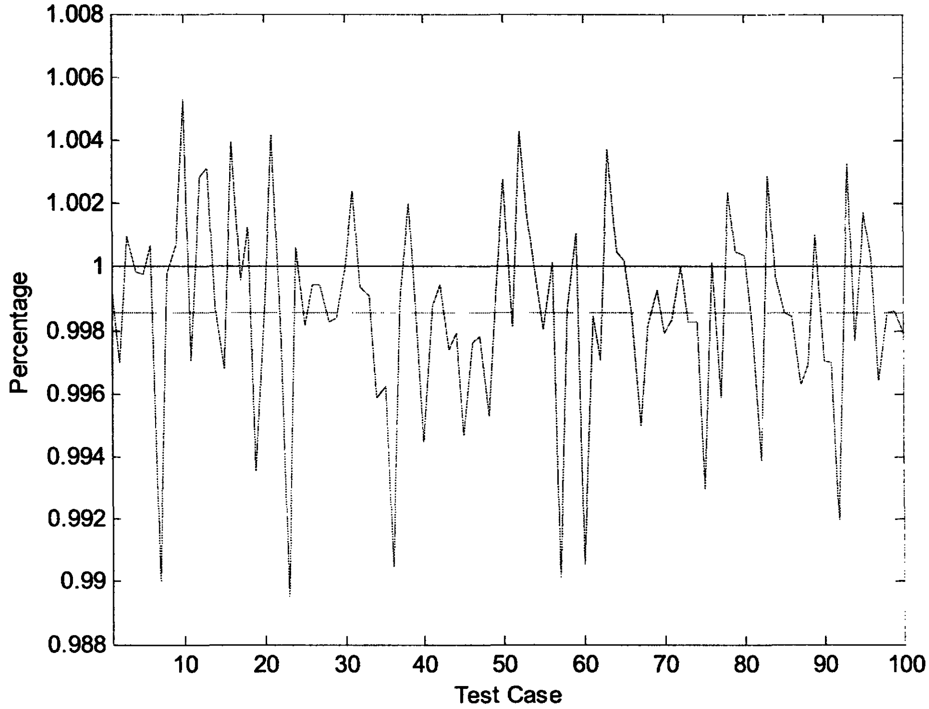

Figure E.7.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.73$

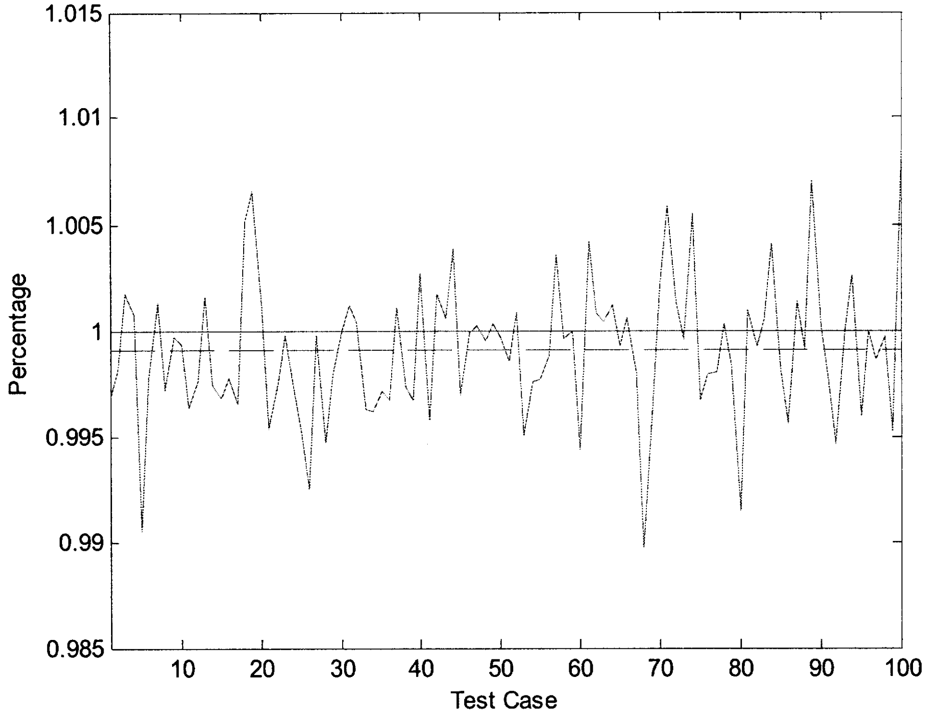

Figure E.7.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.74$

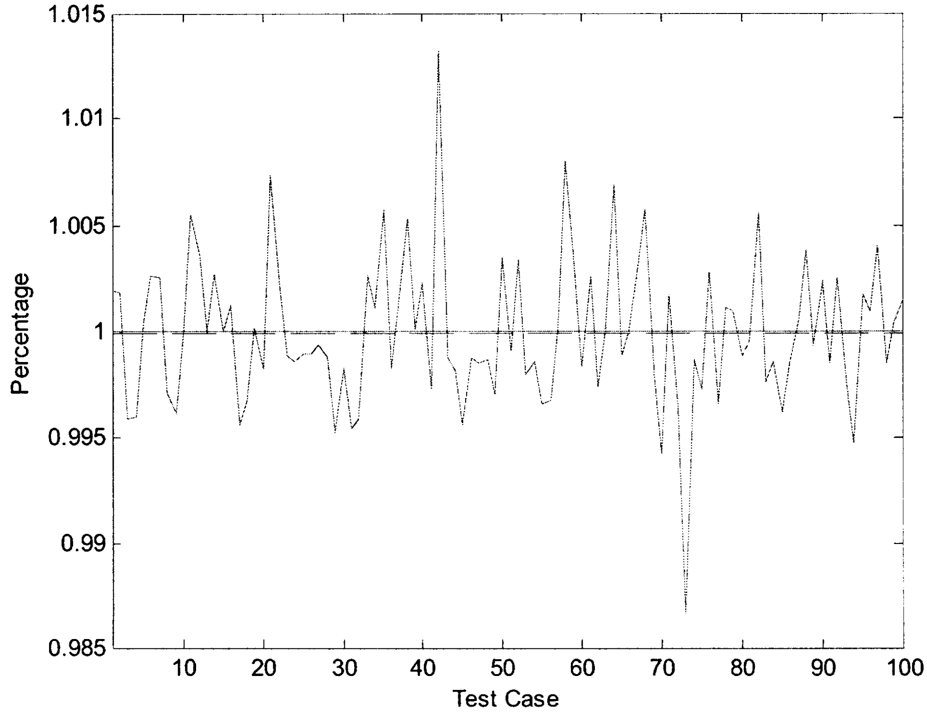

Figure E.7.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.75$

E.8 Test Results at $\rho_{test} = 0.8$

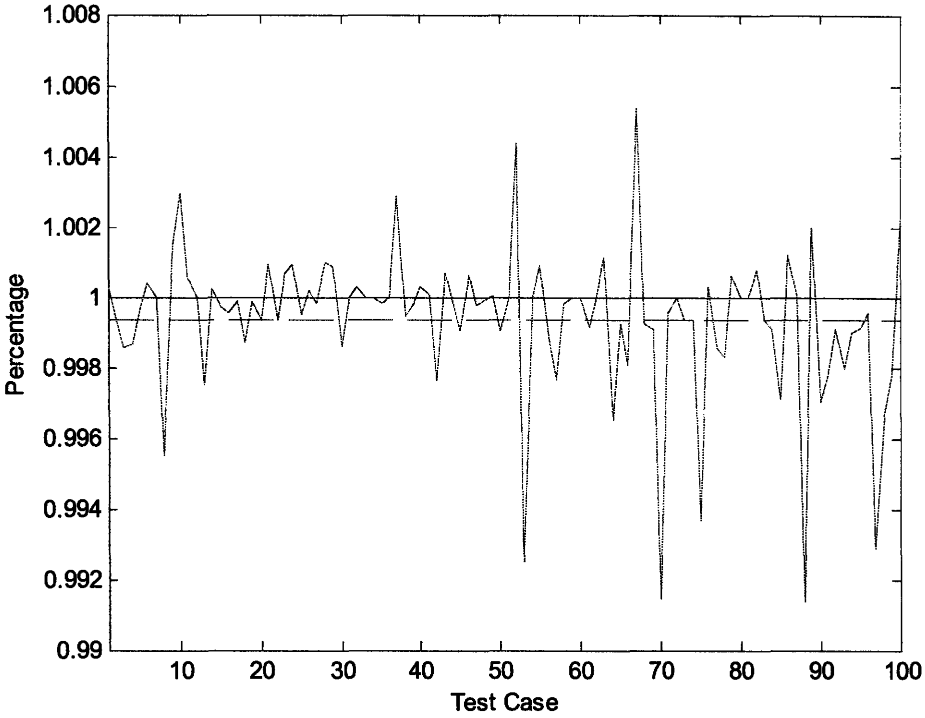

Figure E.8.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.81$

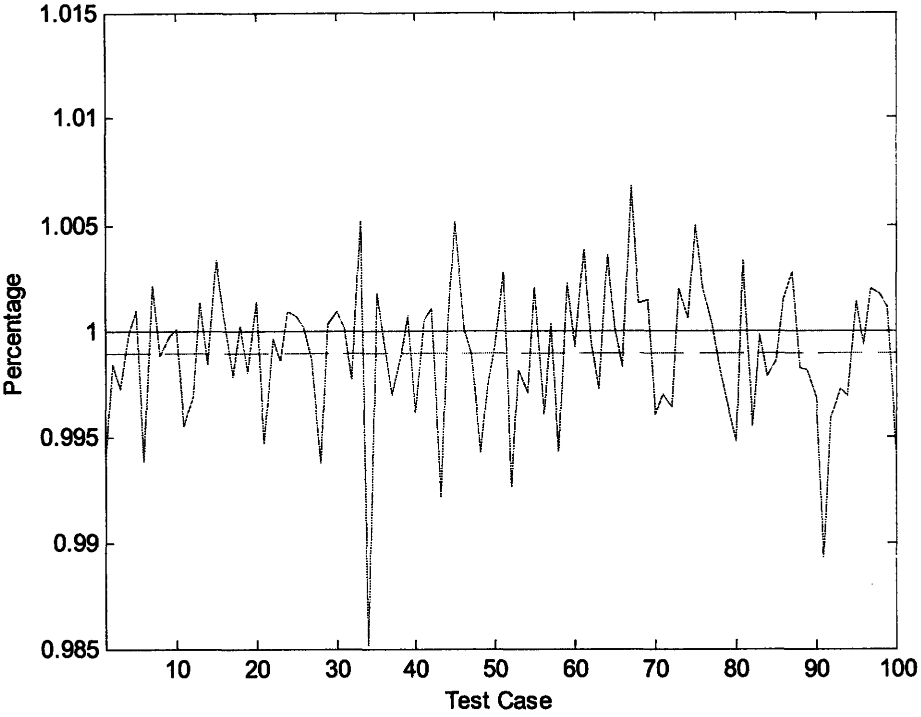

Figure E.8.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.82$

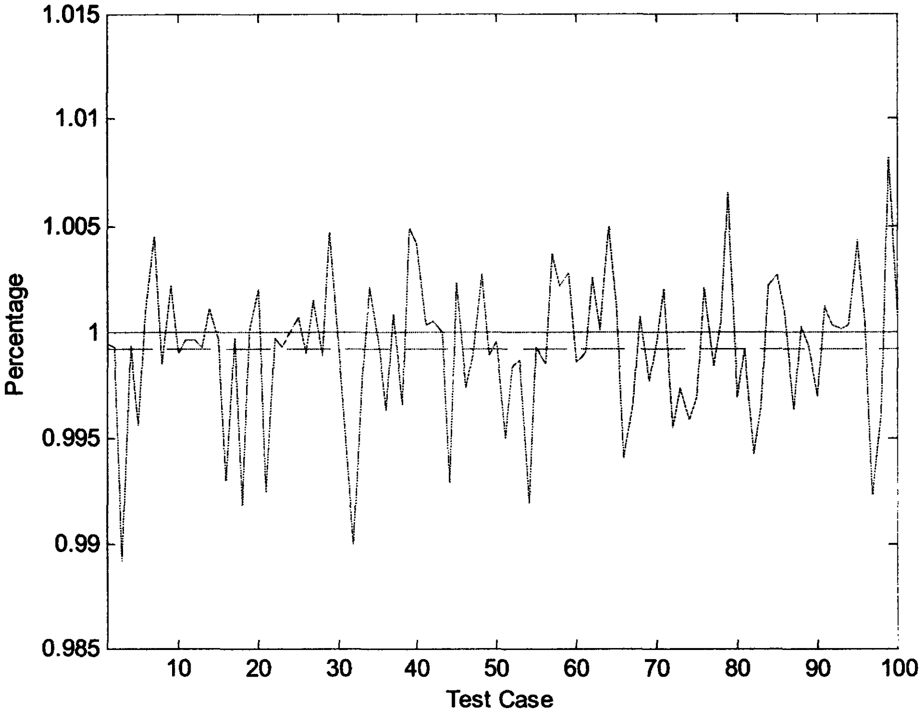

Figure E.8.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.83$

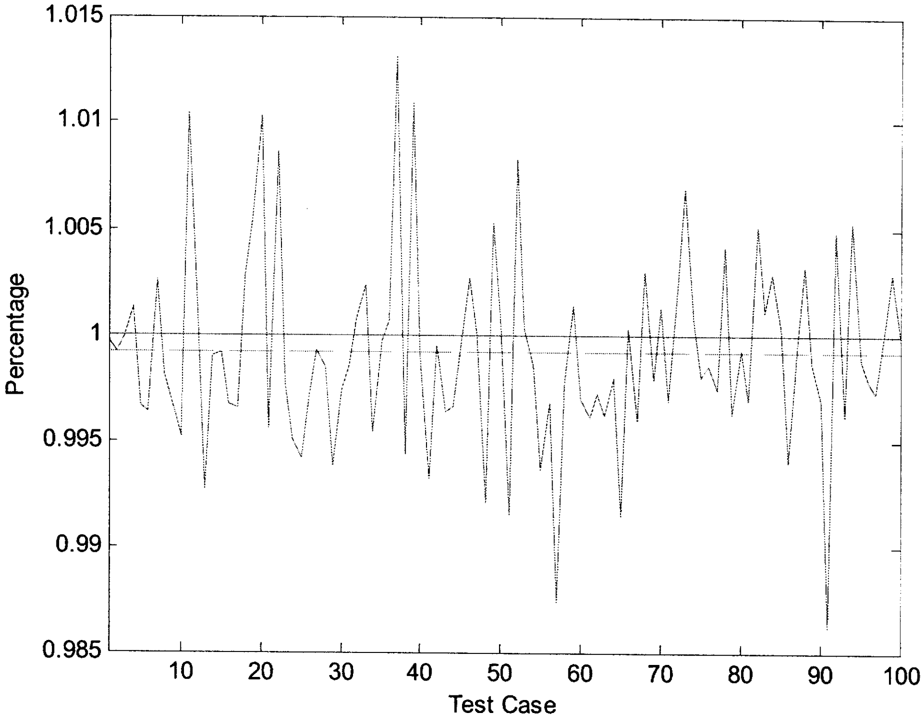

Figure E.8.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.84$

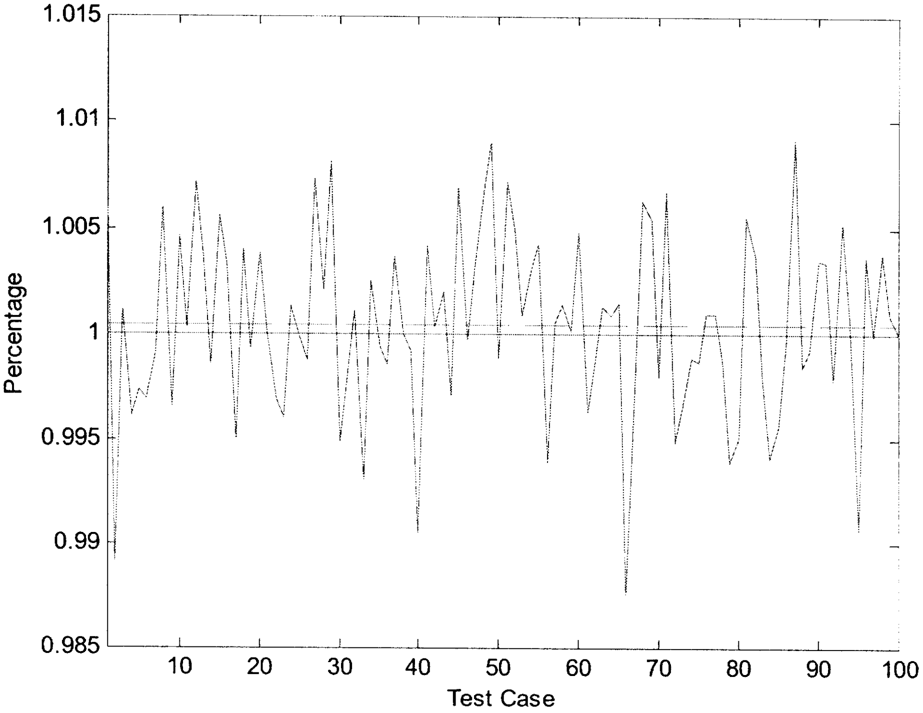

Figure E.8.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.85$

E.9 Test Results at $\rho_{test} = 0.9$

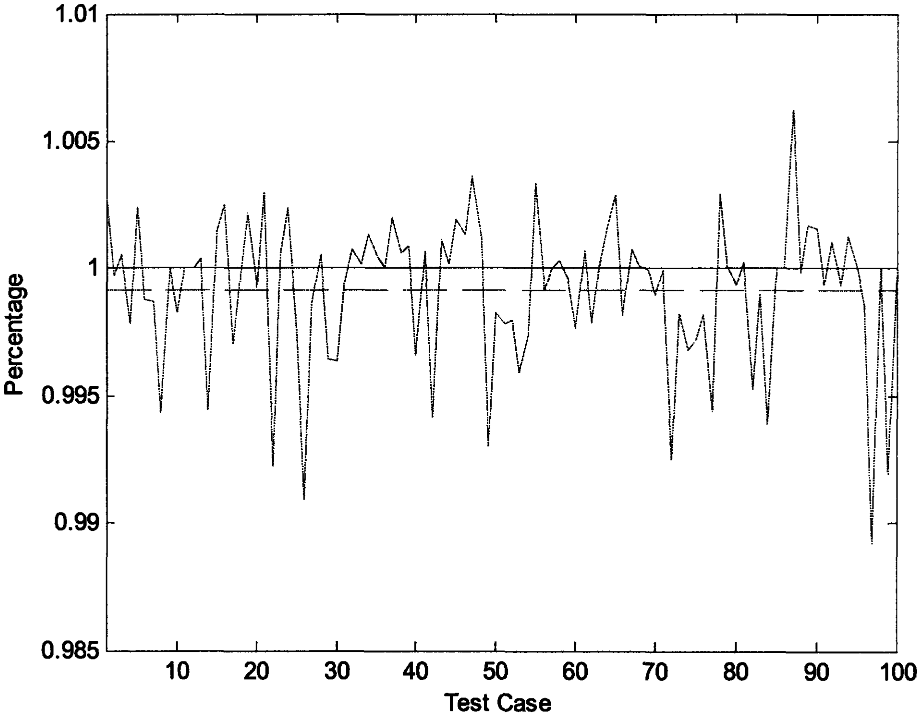

Figure E.9.1: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.91$

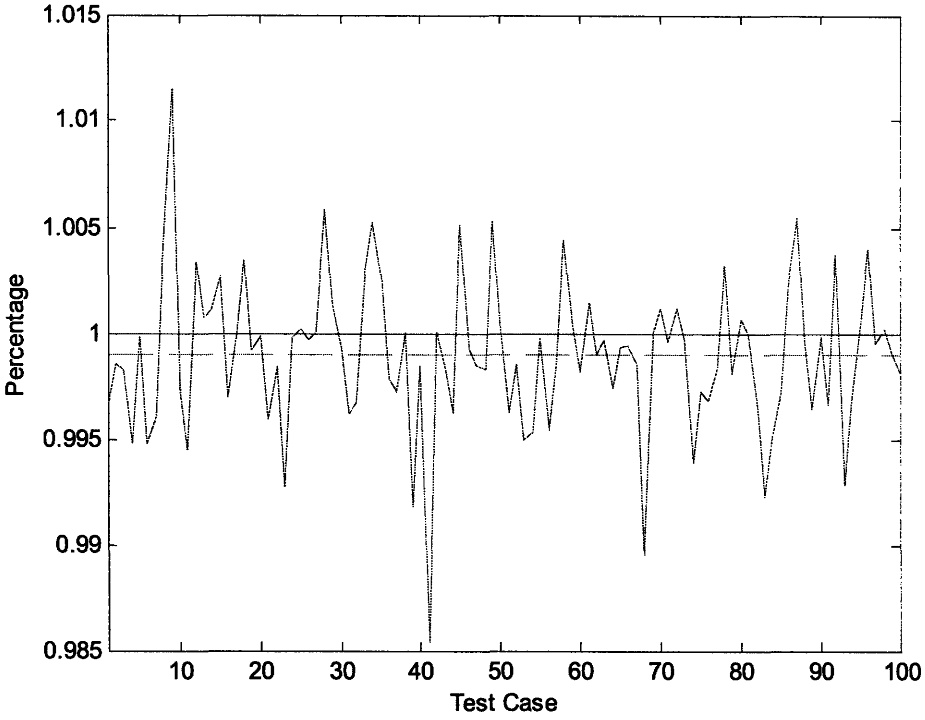

Figure E.9.2: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.92$

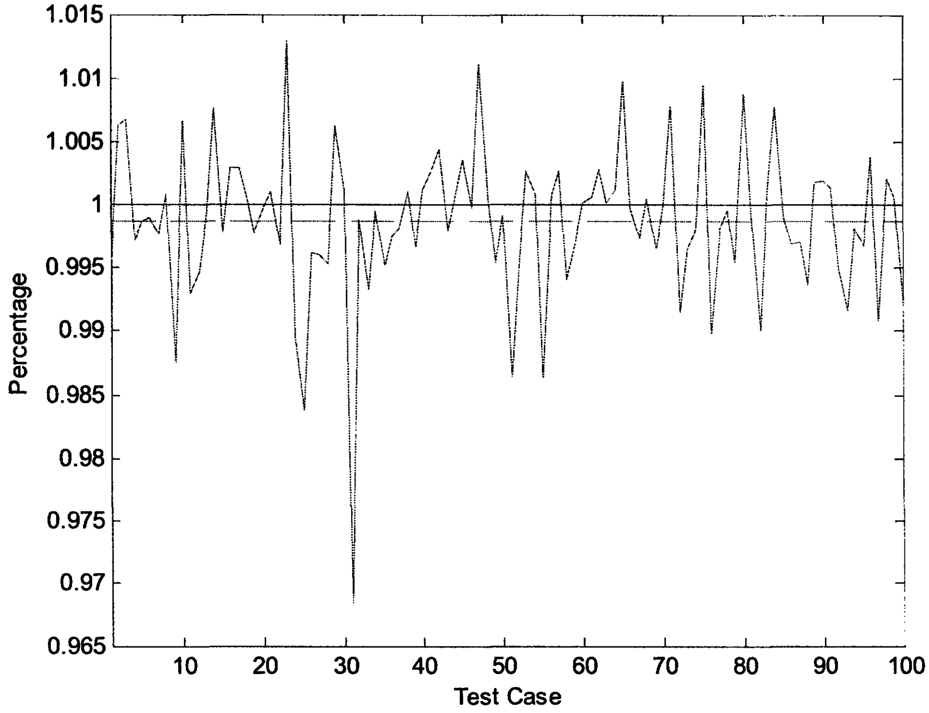

Figure E.9.3: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.93$

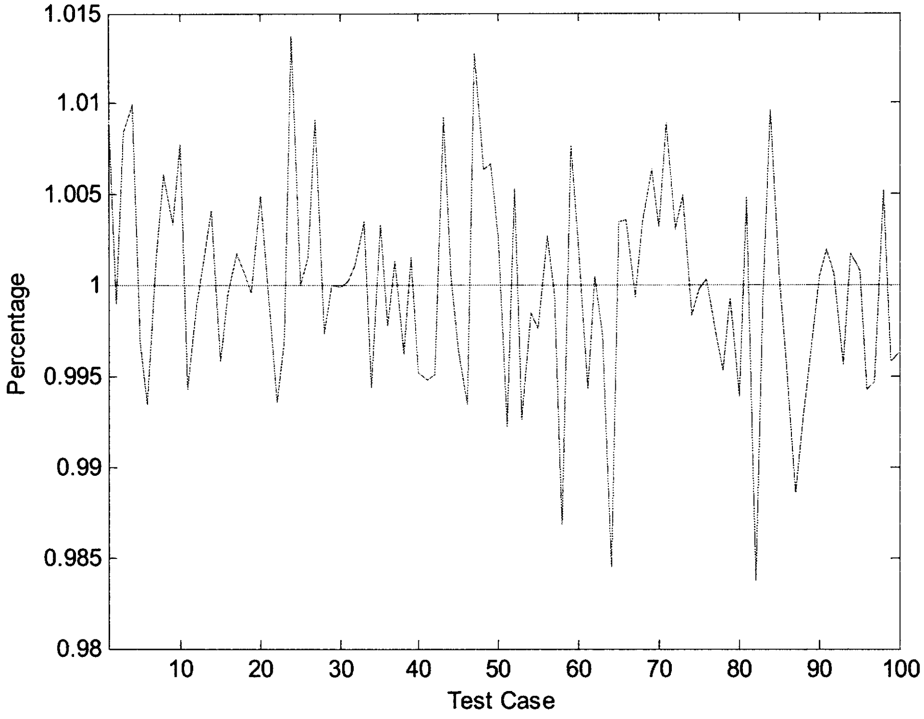

Figure E.9.4: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.94$

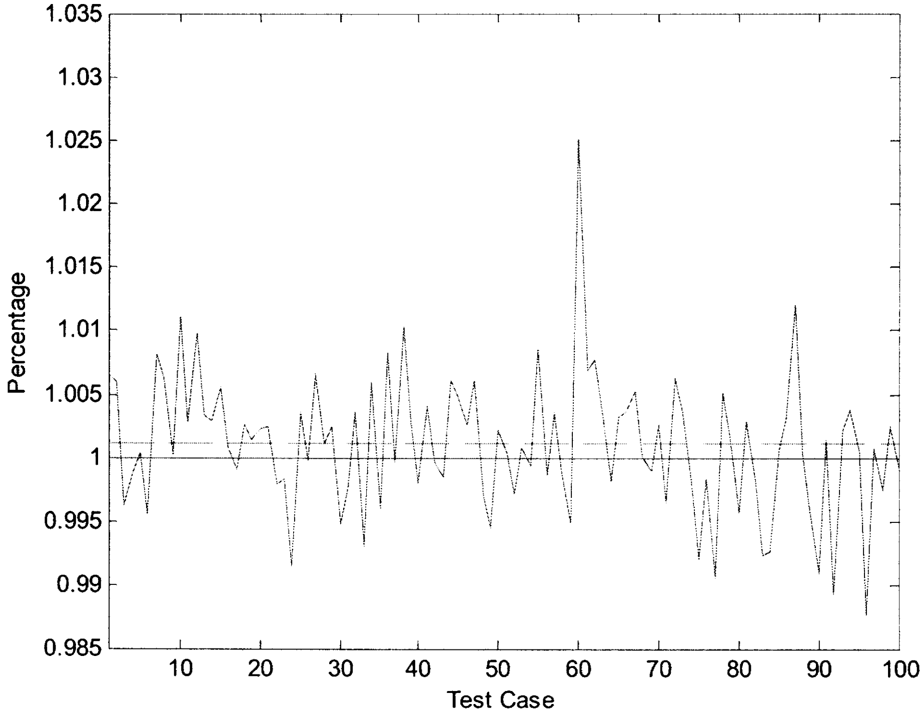

Figure E.9.5: Performance of Modified Heuristic at $\rho_{opt\_guess} = 0.95$

(This Page Intentionally Left Blank)

Bibliography

[1] Bowman, E. H., 1955. Using Statistical Tools to Set a Reject Allowance. *NACA Bulletin* 36(10), 1334-1342.

[2] Bowman, E. H., and Fetter, R. B., 1957. *Analysis for Production Management.* Richard D. Irwin, Inc., Homewood, Illinois.

[3] Buzacott, J. A., and Shanthikumar, J. G., 1993. *Stochastic Models of Manufacturing Systems*. Prentice Hall, Englewood Cliffs, NJ.

[4] Delfausse, J., and Stalzman, S., 1966. Values for Optimum Reject Allowances. *Naval Research Logistics Quarterly* 13(2), 147-157.

[5] Giffler, B., 1960. Determining an Optimal Reject Allowance. *Naval Research Logistics Quarterly* 7(2), 201-206.

[6] Goode, H. P., and Staltzman, S., 1961. Computing Optimum Shrinkage Allowances of Small Order Sizes. *Journal of Industrial Engineering* 12(1), 57-61.

[7] Jönsson, H., and Silver, E. A., 1985. Impact of Processing and Queueing Times on Order Quantities. *Material Flow* 2, 221-230.

[8] Karmarkar, U. S., 1987. Lot Sizes, Lead Times and In-Process Inventories. *Management Science* 33(3). 409-418.

[9] Karmarkar, U. S., Kekre, S., and Kekre, S., 1983. Multi-item Lot Sizing and Lead Times. Graduate School of Management Working Paper No. QM8325. University of Rochester, 1983.

[10] Kuik, R., and Tielemans, P., 1998. Analysis of Expected Queueing Delays for Decision Making in Production Planning. *European Journal of Operational Research* 110, 658-681.

[11] Larson, R. C., and Odoni, A. R., 1981. *Logistical and Transportation Planning Methods*. Prentice-Hall, NJ.

[12] Levitan, R. E., 1960. The Optimum Reject Allowance Problem. *Management Science* 6, 172-186.

[13] Little, J. D. C., 1960. A Proof for the Queuing Formula: $L = \lambda W$. *Operations Research* 9, 383-387.

[14] Llewellyn, R. W., 1959. Order Sizes of Job Lot Manufacturing. *Journal of Industrial Engineering* 10(3), 176-180.

[15] Stalk, G., 1988. Time – The Next Source of Competitive Advantage. *Harvard Business Review* 66(4), 41-51.

[16] White, D. J., 1965. Dynamic Programming and Systems of Uncertain Duration. *Management Science* 12, 37-67.

[17] Yano, C. A., and Lee, H. L., 1995. Lot Sizing with Random Yields: A Review. *Operations Research* 43(2), 311-334.