AFIT/GSO/ENS/93D-16

THE APPLICATION OF SIMULATED ANNEALING TO STOCHASTIC SYSTEMS

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Space Operations

Charles B. Warrender
Captain, USAF

Approved for public release; distribution unlimited

THESIS APPROVAL

STUDENT: Captain Charles Bret Warrender CLASS: GSO-93D

THESIS TITLE: The Application of Simulated Annealing to Stochastic Systems

DEFENSE DATE: 18 November 1993

COMMITTEE:

Advisor: James S Shedden

JAMES S. SHEDDEN, Lt Col, USAF
Assistant Professor of Operation Research,
Department of Operational Sciences, AFIT/ENS

Reader: Dennis C Dietz

DENNIS C. DIETZ, Lt Col, USAF
Assistant Professor of Operation Research,
Department of Operational Sciences, AFIT/ENS

Acknowledgments

Lt Col James Shedden proposed this project. I am indebted to him not only for suggesting the topic, but also for giving me his time and patience as an advisor. I know that I have gained more objectivity towards my writing and a clearer understanding of the scientific process as a result of his interest. I hope that my effort in conducting this research reflects well on him.

I must thank my mother- and father-in-law for helping with the purchase of a computer used in doing the research. I would not have been able to conduct so many experiments with such ease without sole access to this computer.

Conducting some research and writing a report has very little meaning if some objective reader cannot understand the results. I thank Lt Col Dietz for making sure that the results do make some sense. I must extend to him additional thanks for the use of his computer code; code that eased the evaluation of some of the results.

I must apologize to those closest to me whom I may have offended or ignored while I pursued this research. Knowing that a thesis effort demands an extraordinary amount of time and concentration gives little comfort to those for whom the research is meaningless, but for whom my affection is not. I hope that my son, Scott, and my daughter, Diane, will forgive me the time I took away from them. Most of all, I thank my wife, Christy, for making it easier on everyone, for being there when I was not, and for being there when I was, too.

Charles B. Warrender

Table of Contents

List of Figures

List of Tables

Abstract

Simulated Annealing was used to optimize three constrained simulation models. For each of these models, seven different acceptance functions were evaluated and compared against the performance of Local Search. These comparisons demonstrated the affect that different acceptance functions have on the performance of the algorithm. The performance was measured by the average solution quality and average efficiency obtained from several runs.

The first model facilitated the implementation of Simulated Annealing using the SLAM simulation language. The configuration space was small, described by only two decision variables. It demonstrated the viability of using Simulated Annealing to optimize the variable settings in a simulation model. The second model, with six decision variables, provided greater insight to the advantages and limitations of Simulated Annealing. This model was implemented as an open queuing network. The third model, similar to the second, was implemented as a closed queuing network. The results from this variation were completely unexpected. They showed a wide performance separation among the different acceptance functions that was not present in the first two models.

No attempt was made to justify the use of Simulated Annealing from a theoretical perspective. Rather, empirical results from the three models were used to infer the practical utility of the algorithm.

THE APPLICATION OF SIMULATED ANNEALING TO STOCHASTIC SYSTEMS

I. Introduction

Simulated Annealing provides a means for optimizing the inputs to a simulation model. Simulated Annealing optimizes an unknown function by mimicking the physical process of annealing. The function being modeled typically represents a *system*. In this research, Simulated Annealing is applied tu stochastic models.

A simulation model can capture much of the randomness found in a system that a functional model cannot. Most systems yield variable outcomes under the same conditions due to some element of uncertainty. The response describes not a single outcome, but a sample from a population of outcomes. The following research examines Simulated Annealing as a way to find a desired, near-optimal configuration of a stochastic system.

1.1. Background

Traditionally, a problem is modeled as a "black box" requiring a variety of inputs and generating some output. The problem solver must model the "black box" process, decide which inputs to include, and determine an appropriate range of values for each input. Each problem instance forms relationships among the inputs, or *decision variables*, and the output, or *response variable*. For example, assume that you want to design a satellite for space exploration: the decision variables are the spacecraft bus and the sensors placed on the satellite; the response variable is the mission value of the satellite. The spacecraft bus inherently constrains the total power, volume and mass available. Each sensor uniquely determines a power requirement, unit volume, mass, and mission-value. A "black box" formulation would use these relationships to determine the overall mission-value of a given satellite design, or *configuration* of decision variables. Each *combination* of decision variables describes an *alternative* satellite design.

Optimization of a problem tries to find a configuration of decision variables that results in the "best" response. Typically, a response variable can be given a numerical value either as the result of a direct numerical measurement, like the mission-value of a satellite design, or as the result of a transformation from a qualitative measurement to a numerical scale. In either case, the possible responses can be ordered in some way from smallest, or *minimum* valued response, to the largest, or *maximum* valued response. In most problems, the "best" response corresponds to either the minimum or maximum depending on the problem formulation and the goal, or *objective*, of the problem solver.

There are many different ways to find an optimum configuration. The preferred model depends on the nature of the "black box" process. If a set of analytic expressions adequately represents the process, then an analytic technique can often be used to find an optimum. Computer algorithms based on analytic techniques solve large problems in reasonable amounts of computing time (CPU time).

In many cases however, the number of possible configurations, the nature of the objective, or the complex nature of the process prohibits the use of analytic algorithms to find an optimum (20:33-4). In these cases, the configurations can be searched empirically, using a heuristic technique to guide the search to an approximate optimum. A heuristic algorithm is considered valid if it can produce an acceptable configuration within the limits of computing time (20:36). Local Search and a variant, Simulated Annealing, are heuristic algorithms that have been used to find near-optimal configurations to combinatorial problems.

1.2. Problem

Local Search and Simulated Annealing have been applied to *deterministic* problems, where the responses are modeled by analytic functions. This research addresses the adaptation of Simulated Annealing to *stochastic* problems, where the responses are modeled more appropriately by simulations. Although Local Search and Simulated Annealing will be discussed at length in Chapter 2, some brief remarks are needed to put the problem in context.

Local Search compares alternative configurations using their response values. The method retains the configuration resulting in a more optimal value in each successive comparison. Simulated Annealing compares the responses of alternative configurations using an *acceptance test*. The test accepts alternative configurations based on the difference in response value. It may, according to a probability density function, retain less optimal alternatives. This ability to occasionally degrade the solution allows the algorithm to escape poor local optima; an ability missing in the Local Search method.

Theoretical arguments prove that Simulated Annealing will converge to the true optimum given an infinite run length and using a specific acceptance test. In practice, the algorithm runs for a finite length and results in a near-optimum solution using various acceptance tests. These results pertain solely to deterministic problems. What happens when the problems are stochastic? Some research has been done showing the promise of adapting the algorithm for these problems, but much has yet to be explored (13:395). The existing literature does not address the relative performance of differing acceptance tests applied to stochastic problems. This research will investigate the performance of seven such alternatives.

1.3. Scope

Three representative problems are modeled using the SLAM II simulation language. The performance of Local Search and Simulated Annealing with seven alternative acceptance tests are compared for quality and efficiency in each of these problem instances. The estimated optimal response measures solution quality. The actual computer time used is the chosen measure of computational efficiency. Comparisons between the two measures can then be used to infer which of the algorithms perform better than the others for the models investigated.

1.4. General Approach

Three separate simulation models are used to compare the different algorithms. The first model, the timing of traffic lights, is a simple model with only two decision variables. The simplicity of this model enables the development of an overall approach. The approach itself concerns four major components:

1. Formulation of the simulation model,
2. Formulation of the heuristic algorithm,
3. Selection of the acceptance tests, and
4. Development of the comparison tests.

The second model, the configuration of job-shop machines, is a more complicated model with six decision variables. The real-world complexity of this model and large variations in response help substantiate the general methodology for optimizing stochastic problems. The machines are configured so that parts can enter and leave the system freely. This is an open-queuing network. The third simulation model alters the flow of parts by fixing the number in the system and never letting them leave. This is a closed-queuing network.

1.5. Sequence of Presentation

A review of the current literature provides a foundation for the methodology. The review begins with the methods of heuristic programming and Simulated Annealing and then proceeds to the results of theory and practice. A review of stochastic optimization bridges the conceptual gap between analytic and stochastic models. A methodology and several alternative acceptance tests is developed for Simulated Annealing. The three problems discussed test the methodology by generating empirical data and inferring relationships among the alternatives. These results are used to motivate additional Simulated Annealing research.

II. Literature Review

The majority of the literature concerning Simulated Annealing deals with combinatorial problems whose response to a given set of inputs is constant. Although simulation problems differ in the nature of the response, the results outlined in the literature provide a starting point for adapting the algorithm. The results detail both the theoretical performance and the actual empirical performance of the algorithm for a number of problem classes. These results guide the methodology developed in a later section. First, however, a discussion of heuristic algorithms introduce the basic concepts needed.

2.1. Heuristic Algorithms

A heuristic algorithm compares two responses, decides which one is more optimal, and concludes that the corresponding configuration is better (20:35). The simplest way to find an optimum using this strategy is to compare the responses from all possible configurations and pick the best one, a technique called *exhaustive enumeration*. Unfortunately, most problems have far too many possible configurations to apply this technique within reasonable time limits. For instance, a problem with m decision variables, each with n possible values, has n^m possible configurations (19:73). If $n=m=6$ then there would be 46,656 configurations. Assuming it took one second to evaluate each response, it would take thirteen hours to complete the algorithm. If the number of possible values increased by just one, $n=7$, the time required to complete the algorithm would increase to seventy-eight hours. When the solution time increases exponentially as the problem size increases linearly, the problem is classified as *NP-complete* (non deterministic polynomial time complete) (18:671). Clearly, exhaustive enumeration is impractical for many real-life problems.

Although real-life problems may become very large, they are nevertheless *bounded* in size and *constrained* in time (20:35). There are always a finite number of possible configurations to examine and there is always a finite time within which the problem must be resolved. These restrictions allow us to observe that:

> Problems having the same mathematical model, but different bounds on either the size of their input domain or the computing time, must be considered as different problems and hence may require very different heuristic algorithms to solve.(20:35)

Certainly exhaustive enumeration guarantees that the optimum will eventually be found. In many cases, however, the best configuration obtained in a given number of hours, minutes, or even seconds must be used instead of the true optimum. The objective of an algorithm tailored to execute withir specified time limit must replace optimality with *acceptability* (20:40). For instance, NP-complete problems often require too much computing time to solve for the exact optimum, so a near-optimal solution must be accepted. The only justification for the claim that a given solution is acceptable comes from the empirical evidence obtained from sample problems and the reliability implied by that evidence.(20:42)

Assuming that several algorithms can be formulated to execute within a specified time limit for a given problem instance, the question remains as to which algorithm is best. S. Lin proposed a method to compare different heuristic algorithms by focusing first on solution quality and then on efficiency using these qualitative definitions:

DEFINITION 2.1: *DOMINATE*
If the solution produced by algorithm A has a high probability of being closer to the optimum than the solution produced by algorithm B, then A dominates B.

DEFINITION 2.2: *COMPETE*
If neither algorithm A nor algorithm B dominates, then A and B compete.

DEFINITION 2.3: *EFFICIENT*
If algorithm A produces solutions substantially faster than algorithm B, then A is more efficient than B.

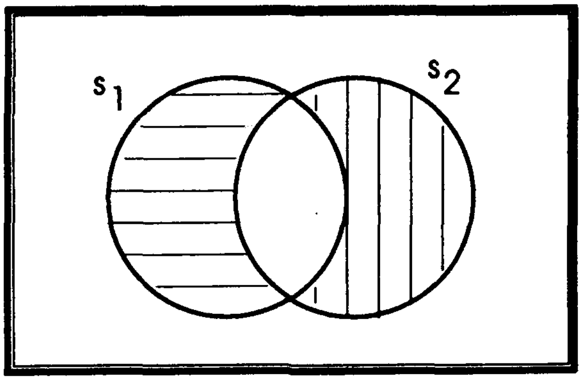
DEFINITION 2.4: *BETTER*
If algorithm A dominates algorithm B or if algorithm A is more efficient than algorithm B, then A is better than B (20:40-41).

For a given problem instance with alternative algorithms, the performance of each algorithm can only be determined through experimentation, consensus, or structural analysis (20:41). Each person can evaluate the alternative algorithms using their own judgment, but an accepted method for comparison lends more credibility to the results. Kirkpatrick *et al.* suggest that the *average performance* of competing algorithms provides the most practical basis for comparison (18:672). Measures of solution quality and efficiency constitute the performance measures used for comparison in this thesis. The alternative algorithms include Local Search and variations of Simulated Annealing.

2.1.1. Local Search

Local Search, the simplest alternative algorithm, guarantees that a locally optimal configuration will be found. To understand why this is so, some nomenclature must be presented. Let S represent the set of all possible configurations of decision variables and s represent a single configuration, $s \in S$. Define the *distance* to be minimized between configurations s_1 and s_2 (20:38):

$$d(s_1,s_2) = \tfrac{1}{2}\{\text{the number of objects in } s_1 \text{ not in } s_2 + \text{the number of objects in } s_2 \text{ not in } s_1\} \quad (2.1)$$



Where a decision variable is the number of objects of a certain type, a subset of the configuration of objects. Define the ***neighborhood*** of a configuration s as the subset of configurations in S that lie within a specified distance, designated by λ (20:37-8):

$$N(s) = \{s'\colon d(s,s') \leq \lambda \text{ and } s' \in S\} \quad (2.2)$$

Local Search proceeds from these definitions. First, an initial configuration is selected at random from S and denoted the *incumbent* optimal configuration, s_i, and its response y_i determined. Next, a neighboring configuration is selected at random and denoted the *challenger*, s'_i,and its response y'_i determined. If the challenger's response is closer to the objective than the incumbent's response, then it replaces the incumbent, otherwise it is rejected. Challengers are tried successively until an incumbent is found such that none of its neighbors' responses are closer to the objective. The final incumbent is, by definition, a local optimum (20:37-8).

Solution quality depends heavily on the starting location in the configuration space *S*. Once the method reaches a local a optimum, it cannot escape and the result may be a poor solution. To overcome this limitation, an analyst typically applies the Local Search algorithm from various initial configurations and chooses the best solution as the final optimal configuration. When the configuration space contains many poor local optima, Local Search yields poor results.

2.1.2. Simulated Annealing

The Simulated Annealing algorithm is a variant of Local Search that overcomes this problem. In Local Search, only challenger configurations that improve the objective are allowed to replace the incumbent. Simulated Annealing relaxes the acceptance of challengers to include "some" that are worse alternatives, enabling the algorithm to escape poor local optima.

The method determines when a less optimal configuration should be retained by emulating the physical process of annealing. It simulates a physical process, which is the reason for naming the technique "Simulated Annealing." A group of researchers led by Kirkpatrick is credited for originally developing the algorithm and Cerny is credited for establishing its usefulness in solving combinatorial problems (18: 7).

2.1.2.1. Physical Annealing.

Annealing improves the pliability of a metal or alloy by minimizing internal stress. The metal is heated to a specified temperature to relieve the stress and then cooled slowly to room temperature to keep the stress small. Fast cooling, known as quenching, induces large amounts of internal stress which causes brittleness. A metal can be shaped by using the annealing process to relieve the stress of working it (2:429).

Internal stress results from the build-up of potential energy stored in the configuration of atoms. Heating the metal allows the atoms to move randomly into many configurations. Slowly cooling the metal establishes and maintains the atoms near thermal equilibrium.

2.1.2.2. Thermal Equilibrium.

When a metal is in thermal equilibrium, the probability of a configuration of atoms having a given potential energy is determined by the specified temperature. In equilibrium, at temperature T, the probability of atoms being in a state with a potential energy E_1 is given by the Boltzmann distribution (21:1088):

$$P(Energy = E_1) = \frac{1}{Z(T)} e^{-E_1/k_B T} \quad (2.3)$$

Where k_B is the Boltzmann constant, a constant of proportionality which converts units of temperature into units of energy (6:83). This conversion makes the exponent dimensionless. When applied to combinatorial problems, Simulated Annealing drops this particular constant since the response value does not measure energy.

As time progresses, a given configuration of atoms may transition to a neighboring configuration with potential energy E_2. The likelihood of transitioning depends upon the change in potential energy E_2- E_1 and the temperature T. If the transition would result in a lower potential energy, then the transition occurs. If the change in energy would increase the system's potential, then the transition occurs with probability (29:7):

$$P(Transition) = e^{-(E_2 - E_1)/k_B T} \quad (2.4)$$

This equation represents the *a priori* probability that the system will transition to the new configuration (21:1089). It provides the basis for determining the likelihood of a transition for any process in equilibrium. For any system in equilibrium at a given temperature, only two elements are needed calculate the probability: a configuration and a value for the resultant energy (7:49).

2.1.2.3. Analogy.

Simulated Annealing uses this basis in determining the acceptance of a detrimental move. Combinatorial problems have easily recognizable configurations that represent solutions. These solutions do not, in general, result in a measurable energy. The energy term is meaningless. Each configuration results, instead, in a response value measured in units that are problem specific. If the response value is used in place of the energy, then the constant of proportionality in Equation 2.4 relates different quantities. This constant is dropped and the temperature is replaced with a "temperature" function, *T*.

In physical annealing, the temperature determines the average energy of the metal. When the system is in equilibrium, the actual energy is near this average energy. As the temperature is lowered, the average energy is lowered until the metal reaches a ground state. Simulated Annealing mimicks the physical process. The atoms in the metal are equivalent to the decision variables in a combinatorial problem. The atomic structure is equivalent to a configuration of decision variables. The energy is equivalent to the response value. The temperature is equivalent to a temperature function, a control parameter that determines an average response value when the combinatorial system is in equilibrium. As the control parameter is lowered, the average value of the response is lowered. Eventually, the process settles on a value at or near the global minimum.

Let the objective value for an incumbent's response be given by y_i, and a challenger's response by y_c. For any given problem in which the objective is to minimize the response, the probability of accepting a challenger configuration is given by:

$$P(Transition) = \begin{cases} 1 & ; \Delta \geq 0 \\ e^{\Delta/T} & ; \Delta < 0 \end{cases} \quad \text{where} \quad \Delta = y_i - y_c \quad (2.5)$$

And in order to maximize the response:

$$P(Transition) = \begin{cases} 1 & ; \Delta \geq 0 \\ e^{\Delta/T} & ; \Delta < 0 \end{cases} \quad \text{where} \quad \Delta = y_c - y_i \quad (2.6)$$

In either case, the *temperature T* represents a decreasing function with the number of challenger configurations. Initially, the *a priori* probability of accepting detrimental moves is high, and gradually the probability decreases until none of the detrimental moves are accepted.

Many different functional forms have been proposed for decrementing the temperature, for controlling *the cooling rate* (9:7). Ideally, the temperature will maintain equilibrium and converge towards the optimum within a reasonable number of iterations. Although theory may guide the selection of a temperature function, empirical results often prove more useful when choosing among several alternatives. The average performance of an algorithm using a given temperature function can be measured empirically once the constants have been tuned.

2.2. Theoretical Results

The role of theory in heuristic methods is to provide some reassurance that the method converges to the exact optimum in an infinite limit (1:vii). Certainly if the method does ı.ot converge when taken to an infinite limit, then any candidate obtained in a finite limit will not be close to the true solution. At present, Simulated Annealing theory provides this reassurance through convergence proofs for two general classes of cooling schedules. Simulated Annealing can be formulated in two fundamental ways and the proof for convergence depends upon which formulation is used.

In either case, however, the theory of Markov processes must be used. Consider the way in which Simulated Annealing progresses from one incumbent solution to the next. There is some probability that a given neighbor will be accepted as the incumbent. The same relationship holds for a Markov Chain. Technically, a Markov chain relates any pair of outcomes by a conditional probability. The probability of going to state *j* on the *k*-th trial is conditioned on being in state *i* on the (*k*-1)-th trial, $p_{ij}(k-1, k)$. The cumulative probability of being in state *j* on the *k*-th trial is given by (29:13):

$$P_j(k) = \sum_i P_i(k-1) p_{ij}(k-1,k) \quad (2.7)$$

This essentially states that the probability of being in state *j* on the *k*-th trial depends on a relationship that involves the likelihood of being in any given state on the previous trial times the probability of transitioning from that state to the new state in one step. A Markov chain is *homogenous* if $p_{ij}(k-1,k)$ does not depend on *k* (the one-step transition probabilities remain constant). Otherwise the Markov chain is *nonhomogeneous* (29:12-13). These are the two classes of cooling schedules for which existing theory proves convergence. The homogeneous case is generally easier to prove.

2.2.1. Homogenous Theory

The proof for convergence of Simulated Annealing using a sequence of homogeneous Markov chains requires enough runs at each temperature setting to ensure homogeneity. As a consequence of constant p_{ij}, the cumulative probabilities approach stationary values, π_j, regardless of the original starting point (27:160):

$$\pi_j = \sum_l \pi_l p_{lj}, \quad \text{for all } j; \quad \sum_l \pi_l = 1 \quad (2.8)$$

This requires that each Markov chain be irreducible (each configuration is reachable from every other configuration in random sequences of trials), a requirement that Metropolis demonstrated for a process in equilibrium (21:1088). As a result of these conditions, there is a non zero probability of reaching a given configuration from any other configuration in an infinite number of trials.

The proof, stated in simple terms is this: An infinite sequence of Markov Chains, each of infinite length, will converge exactly to the optimum (26:326-327). The principal use of this theorem is not so much to prove that a homogenous algorithm converges to the exact optimum as it is to qualify the characteristics of the algorithm. In the case of homogeneous algorithms, a sufficient condition for convergence is to make the length of each Markov chain extremely long. In practice, simulated annealing cannot reach equilibrium within a finite length Markov Chain (26:313). The best, then, that homogenous algorithms can do is to converge as rapidly as possible to the stationary probability for a fixed temperature. The acceptance function for a minimization problem with the fastest rate of convergence is given by Equation 2.5. Unfortunately, homogenous theory does not indicate a way to compromise the infinite length Markov chain within practical limits. The length of each chain as well as the length of the sequence of Markov chains depend on trial and error experience.

2.2.2. Nonhomogeneous Theory.

The nonhomogeneous theory takes a somewhat more practical approach to qualifying algorithms. The requirement that each Markov chain reach equilibrium at each temperature is dropped (26:316). Rather than having a stationary transition probability that is independent of the initial configuration, nonhomogeneous theory simply requires that the transition probability remains independent of the initial configuration. It has been proven that an algorithm with the following temperature update function will converge to the exact optimum in an infinite sequence of trials (26:303)

$$T(t) = {}^{\gamma}\!\!\Big/_{\log(t+t_0)} \quad (2.9)$$

where t is the number of the current trial, t_0 is a constant that adjusts the probability of acceptance (which must be at least 1 to prevent a divide-by-zero error), and γ is another constant which must be *tuned* for a particular problem instance. Note that each Markov chain in the sequence is of length one, a single trial. Unfortunately, the sequence generated by this update function converges much too slowly to be of any practical use (26:327). Table 2.1 shows how slowly the algorithm converges for the parameter settings $t_0 = 1$, and $\Delta \quad = -1$.

Ultimately, it is the actual performance of an algorithm, whether it mimics the homogeneous or the nonhomogeneous theory, and the artful tuning of that algorithm to a given problem instance that matters. After all, the objective of this heuristic algorithm is to find an acceptable solution within the limits of computing time.

Trial Number	$T(t) = 10/\log(t+1)$	$P(\text{Transition}) = e^{-1/T(t)}$
100,000	2.00	0.61
200,000	1.89	0.59
300,000	1.83	0.58
400,000	1.78	0.57

TABLE 2.1

2.3. Empirical Results

Johnson *et al.* evaluated Simulated Annealing formulations and compared their performance with competing algorithms for several problem classes. Both the considerations of problem formulation and the results of these analyses provide an empirical foundation for the Simulated Annealing methodology to follow.

2.3.1. Problem Formulation

Kirkpatrick *et al.* suggest that there are four components needed to implement a Simulated Annealing algorithm in practice (18:779):

1) A description of a system's configuration,
2) A generator of trials,
3) A quantitative objective function,
4) A schedule of "temperatures" and "times" over which the process is to be annealed.

These constitute the problem formulation. Initially, a concise description of the system is required--which is the same as finding the input variables affecting the process and determining the relationships among those variables. The description requires both defining the decision variables and limiting the ranges those variables can assume.

The set of decision variables also implies a neighborhood structure. Given a configuration of decision variables, neighboring configurations can be generated by changing the values of those decision variables within a specified "distance" (Equation 2.1). For example, the configuration of a satellite may consist of five sensors, three of type A and two of type B. A neighboring configuration can be generated by using two sensors of type A, s_1, and three sensors of type B, s_2. The "distance" would be computed by:

$$\begin{aligned} d(s_1,s_2) &= \tfrac{1}{2}\{\text{the number of objects in } s_1 \text{ not in } s_2 \; + \\ &\qquad \text{the number of objects in } s_2 \text{ not in } s_1\} \\ &= \tfrac{1}{2}\{1 \text{ Type A Sensor } + \; 1 \text{ Type B Sensor}\} \\ &= 1 \end{aligned}$$

Another configuration, s_3, may lie a greater distance away, $d(s_1,s_3) = 3$. If the neighborhood size is specified to *lie within a distance* $\lambda = 2$, then s_2 would be a neighbor of s_1, but s_3 would not be a neighbor.

The literature disagrees on the best specification for this neighborhood size. An empirical study suggests that smaller values perform better than larger values (8:546). A theoretical study suggests that a dynamic value performs better than a constant value (initially large but decreasing as time progresses) (33:183). The neighborhood size plays a large part in defining the total number of neighbors, but it is not the only consideration. Typically, with a fixed value for λ , the number of neighbors increases with problem size. Clearly, the selection of a neighborhood size for a given problem rests with the problem solver and is usually determined through trial and error.

Once the neighborhood size has been selected, a method for generating trials from a random starting configuration can be developed. The original Metropolis procedure used a random generator of trials to simulate the randomness of physical processes. The generator required each decision variable to be perturbed some random distance from its incumbent state but within the limiting distance of the neighborhood (21:1088). The problem with randomly generating trials is that the same configuration of decision variables may be tried several times before exhausting all possible configurations within the same neighborhood--thereby wasting computing time. Why retry a configuration once it has already been rejected while there are still configurations which have yet to be tried?

The theory supporting homogeneous algorithms requires that each Markov chain be irreducible and aperiodic. This means that the one-step transition probabilities must remain constant at each temperature setting. In order to maintain this requirement, the algorithm must select neighbors at random.

Fortunately, nonhomogeneous algorithms do not need to meet this strict requirement because the supporting theory does not require the one-step probabilities to remain constant. These algorithms can take advantage of more *efficient trial generators* (28:396). Imagine that you are in a labyrinth and you must decide how to proceed at each intersection--you prejudice your turns so that you spend less time retracing your steps. Likewise, more efficient trial generators bias the search pattern within a neighborhood to minimize computer time wasted on repeated configurations. Johnson *et al.* found that by using a structured trial generator they were able to improve the solutions, "almost as much as one would obtain by doubling the running time and staying with the standard method (15:885)." The nature of the bias can affect the performance of the algorithm for a given problem and must be determined by experience and intuition.

Once the trial generator has been established, a method for calculating the value of a configuration must be devised. The form of this function depends on the measurable effects from the system, the availability of actual data, and the objectives to be achieved.

There may be various output measures and several objectives that need to be addressed. The response function can combine these factors by means of a weighted sum. The sign and coefficient of a factor would reflect its relative importance in the overall goal.

After calculating a number representing the true value of a given configuration, the algorithm can proceed with its fundamental comparison of the incumbent and challenger configurations. The acceptance function makes this comparison by determining the overall likelihood of accepting a challenger at the given temperature with a given response value difference. Recall that the temperature determines an equilibrium distribution of response values. A sample taken from a uniform [0,1] distribution represents the likelihood of a specific challenger, a random draw from the equilibrium distribution. If the sampled value is less than the overall likelihood, then the challenger is accepted. Otherwise it is rejected and the incumbent remains the same. The temperature function is defined as a decreasing function cf *t* and bounded so that the algorithm terminates within a finite number of trials.

All four of these inter-related components work together to specify a Simulated Annealing algorithm: a configuration of decision variables that describe the system; a generator of trials that permutes the incumbent configuration; a quantitative objective function that measures the system's response; and an annealing schedule to search the configuration space. Figure 2.1 shows these relationships.

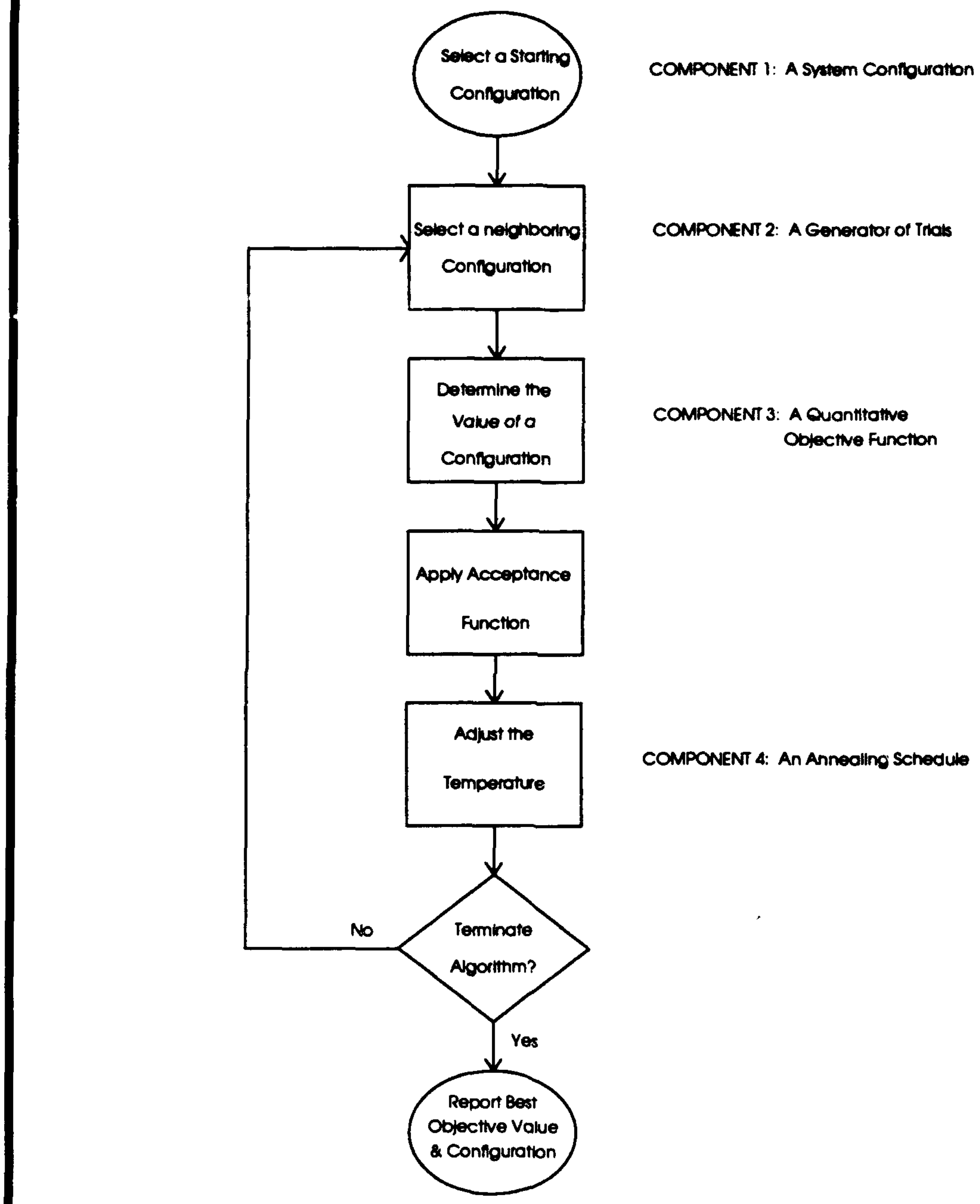

Select a Starting Configuration
COMPONENT 1: A System Configuration
Select a neighboring Configuration
COMPONENT 2: A Generator of Trials
Determine the Value of a Configuration
COMPONENT 3: A Quantitative Objective Function
Apply Acceptance Function
Adjust the Temperature
COMPONENT 4: An Annealing Schedule
No
Terminate Algorithm?
Yes
Report Best Objective Value & Configuration

FIGURE 2.1: PROBLEM FORMULATION RELATIONSHIPS

2.3.2. Analyses

Johnson *et al.* formulated Simulated Annealing algorithms for several problem classes including Graph Partitioning. Graph Partitioning is NP-complete -- the computing time required for an exact optimum increases exponentially as the problem size increases linearly. Reviewing this formulation provides insight regarding the actual specification of the Simulated Annealing Algorithm. Before discussing the problem in detail, some graph theory nomenclature must be reviewed:

DEFINITION 2.5 (14:7): *Graph G*
A pair of sets (V,E) where V is nonempty, and E is a (possibly empty) set of unordered pairs of elements of V. At most one edge may join two vertices.

DEFINITION 2.6 (14:7): *Vertices V(G)*
The elements $v \in \quad V$. Let p denote the number of elements in V.

DEFINITION 2.7 (14:7): *Edges E(G)*
The elements $e \in \quad E$. Let q denote the number of elements in E.

DEFINITION 2.8 (14:8): *Adjacent/Neighbor*
If the vertices v_1 and v_2 are incident with the same edge e, then v_1 and v_2 are adjacent, or neighbor each other.

DEFINITION 2.9 (14:8): *Incident*
If a vertex v_1 is an endpoint of an edge e, then v_1 is incident with e.

Graphically:

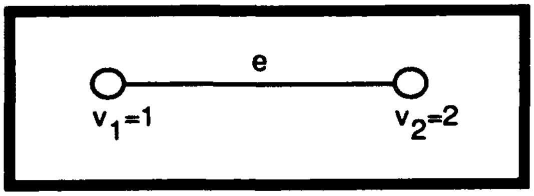


FIGURE 2.2: A GRAPH

The Graph-Partitioning problem can be stated as follows: Given a Graph G, partition the vertices of G into two sets of equal size. The objective of the partition is to minimize the number of edges in G that have vertices in both of these sets. The response function, y, is simply the number of edges with this property. Figure 2.3 shows one instance of the problem:

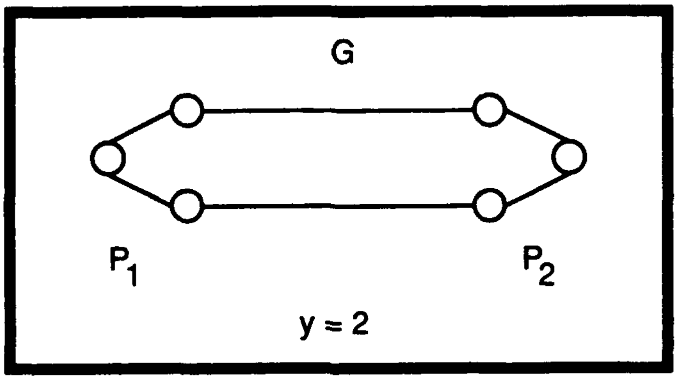


FIGURE 2.3: GRAPH PARTITIONING

The problem formulation used by Johnson *et al.* is just one way to apply Simulated Annealing and may not be the best implementation for the Graph Partitioning problem. Nevertheless, it illustrates the relationships among the four components of problem formulation. Their formulation is as follows:

1) System Configuration: The decision variables are the assignments of vertices $v_1,...,v_p$ to partitions P_1 and P_2. Decision variables can be assigned values of 1 or 2, representing P_1 and P_2 respectively. The neighborhood size is one. This means that a neighbor of the incumbent configuration can be obtained by simply changing the assignment of a single vertex. This scheme occasionally unbalances the partitions so that one set contains more vertices than the other. The algorithm accounts for the imbalance through a penalty function. The penalty function adds to the original response function in

order to bias challenger configurations. When the partition is out of balance, the penalty function forces alternatives towards equal partitions (15:867,870).

2) Generator of trials: Select a vertex at random and change its current assignment (15:870).

3) A quantitative objective function: The number of edges with vertices in both sets P_1 and P_2 plus an imbalance factor times the number of vertices out of balance (15:871):

$$y(s) = \left|\{E = \{v_1, v_2\} : v_1 = 1 \wedge v_2 = 2\}\right| + 0.05\left|\{v \in P_1\} - \{v \in P_2\}\right| \quad (2.10)$$

4) Schedule: Establish an initial "temperature" so that 40% of less optimal trials are accepted, a value determined through experimentation. Set the length of each Markov chain to sixteen times the number of neighbors within a neighborhood size of one. At the end of each chain, reduce the temperature by five percent. Terminate the algorithm when five concurrent Markov *chains accept fewer than 2% of the* trials. All of these tuning parameters were selected through a trial and error process and these values were deemed best (15:872). Whether or not this is a good algorithm, in the sense of definition 2.5, cannot be decided for any implementation of Simulated Annealing (15:869).

Johnson *et al.* compared the performance of this algorithm against the performance of Local Search and the Kernighan-Lin algorithm, the recognized benchmark for the Graph Partitioning problem (15:870). They performed 1000 runs of each of these algorithms on random graphs formed with 500 vertices and averaging 1196 edges. The results were:

Algorithm (1000 Cases Each)	Average Objective Value	Average CPU Time (sec)
Local Search	280	1.0
Kernighan-Lin	235	3.7
Simulated Annealing	215	360.0

TABLE 2.2

Are these competing algorithms? Clearly if the computing time is so limited that Simulated Annealing cannot complete a single case, then it cannot compete with the other two algorithms. On the other hand, if there is enough time available to run one case through Simulated Annealing, there is enough time to run 360 cases of Local Search or 100 cases of Kernighan-Lin. To compare the algorithms in the latter case, Johnson *et al.* equalized the running time over the algorithms given five cases of Simulated Annealing. Their results showed that both Simulated Annealing and Kernighan-Lin dominate Local Search and that Simulated Annealing and Kernighan-Lin compete with each other.

This study illustrated the general approach to formulating the Simulated Annealing algorithm, highlighting the fact that any implementation is highly dependent on the judgment of the problem solver. It also validated the comparison of two algorithms by adjusting the computing time first. In any realistic problem, the available computing time will dictate which algorithms to implement. Once the algorithms have been adjusted to the time constraint, then the solution quality can be measured and the best competing algorithms determined. This approach is be used for this thesis.

How should Simulated Annealing deal with the effects of a stochastic response? The following section addresses these considerations.

2.4. Modeling a Random Process

If a system is stochastic, then the model should respond like a random variable rather than like a deterministic function. In this case, the "value" of a given configuration of decision variables cannot be measured with a function, but must be estimated with a random sample. Two fundamental concepts must be examined to determine how a problem solver can 1) model a random process, and 2) compare two configurations using random samples of the system's response.

Generally, deterministic models represent processes on a gross scale while stochastic models represent processes on a small scale (21:14). For example, a set of differential equations can be used to model analytically the mixing of two fluids. Although the model describes the general behavior of the fluids, it would not describe the minute behavior--the Brownian motion in the process. A simulation model could account for the Brownian motion and result in a more detailed model (21:11).

Often the goal of a stochastic model is to capture the minute behavior of a process. The problem solver may need to employ a simulation model to do this. Simulation, however, requires much more computing time to solve a given problem than a conventional technique would require (12:53). The reason for the increase in computing time is this: simulation recreates the main elements of a process within the controlled environment of a computer and uses pseudo-random representations for the truly random factors (21:12). This means that a single run may not adequately represent the configuration. The more a process varies, the more inadequate a single run becomes. To overcome this limitation, a random sample based on batched observations can be used

(25:724-725). Each function evaluation in the deterministic case corresponds to batched observations in the stochastic case.

Within the system there may be any number of random factors. Data must be collected for each factor and a probability distribution fitted to the observations. A uniform pseudo-random number seeded into an inverse form of a given distribution generates one "observation" of that random factor (25:708). Matching probability distributions to random factors, only one part of *validation*, requires a great deal of work to ensure that the overall process is accurately modeled.

Assuming that a suitable simulation model has been developed and validated, an assumption that this thesis makes for the problems it considers, then the problem of optimizing the response must be addressed. The most straightforward, although naive approach, would be to substitute an estimate of the response for the constant response in an optimization technique designed for deterministic models (32:595). Jorge Haddock and John Mittenthal successfully performed one such study using Simulated Annealing and a steady-state mean from a simulation for each estimate (13:389). The problem with using this naive approach is that of picking an adequate sample size to ensure that the estimate is meaningful. The need to take a random sample rather than a single outcome implies that the computing time increases with the size of the sample (4:199). Therefore, the sample size must be large enough to provide a meaningful estimate of the response but small enough to keep the algorithm within computing limits.

The sample size depends naturally enough on the process itself. If the process varies a great deal, a larger sample would be needed. If the process did not vary, it would be constant -- requiring only one outcome. Recent Simulated Annealing theory states that a sufficient control on the variability of the response requires that the variance of the estimate go to zero faster than the temperature function (26:329). The estimate's variance is given by (17:2-37):

$$s^2 = \frac{1}{m-1}\sum_{i=1}^{m}(y_i - \bar{y}) \quad (2.11)$$

Where m is the number of observations in the estimate, y_i is the value of each observation, and $\bar{y}$ is the mean of the observations. Clearly, one way to decrease the variance of an estimate is to take more observations. The number of observations must approach infinity as the variance approaches zero.

As with previous theoretical results, this result may not give any practical guidance except to provide some reassurance that Simulated Annealing is tolerant of some "noise" in the estimate. In practice, the available amount of time will dictate how small the variance *can* be made. Furthermore, there is a trade-off between the accuracy of each estimate and the number of trials in each annealing run. A trade-off that can only be resolved by trial and error. The usefulness of Simulated Annealing for stochastic models must be shown empirically and through practical application.

2.5. Summary.

This literature review provides the foundation for a Simulated Annealing method that is applied to three simulation models. The empirical studies reviewed show how to formulate a Simulated Annealing algorithm, giving the insight needed to devise alternative algorithms. The experimental results substantiate which alternative algorithms dominate and which ones compete with each other. The comparison techniques reviewed provide the measurement needed: the average performance as measured by solution quality and efficiency. Existing Simulated Annealing theory gives the reassurance that such algorithms converge to the exact optimum, at least when the problems are deterministic.

As yet, there has been little research into the adaptation of Simulated Annealing for optimizing simulation models. What little that does exist from the fields of simulation and Simulated Annealing theory suggests nothing to prevent such an adaptation. The purpose of this thesis is to explore alternative ways to make the adaptation effective.

III. Methodology

The methodology consists of four fundamental parts: the simulation model, the Simulated Annealing algorithm, the alternative acceptance tests, and the companson between those alternatives. The simulation model captures the key elements in the process to be optimized so that trial configurations can be evaluated. The Simulated Annealing algorithm incorporates the simulation model into a common programming structure so that the decision variables can be optimized. This common structure not only permits alternative acceptance tests to be tested, but also enables Local Search to be implemented. The alternative acceptance tests explore the behavior of differing probability distribution functions. This behavior affects the solution quality and computing efficiency of the algorithm. The performance of these alternatives compared with the performance of Local Search provides some justification for using Simulated Annealing. Each of these parts is presented in more detail.

3.1. The Simulation model

The simulation model is at the heart of the methodology because it represents the process itself. Although a simulation model can be stated in general terms, tne Simulated Annealing algorithm depends on the specific simulation tool used by the problem solver. The fist step is to select a simulation language that can be tailored for optimization -- such as SLAM II (25:759). The next step is to develop each simuiation model using SLAM II in such a way that the input variables can be modified between simulation runs. The

SLAM language uses FORTRAN subroutines external to the simulation model to control these modifications. The subroutines are called *user inserts*.

SLAM II scenarios consist of four pieces: the scenario definition, the control statements, the network model, and the user inserts. The scenario definition simply links all of these pieces together. The control statements specify certain initialization conditions and executive constraints. The network model describes the process in terms of the input variables and generates the response value. The appendices contain the details for each of the simulation models presented.

3.1.1. The Timing of Traffic Lights.

The first of the three simulation models, the timing of traffic lights, was selected for two reasons: 1) it is simple, and 2) it has already been used for a case study in simulation optimization. The model is simple because it requires only two decision variables, the "green" times for each traffic light. The objective v alue is the average time a car must wait at either traffic light. The approach used in the SLAM study conducted by Pegden and Gately gives additional background needed to develop the Simulated Annealing algorithm (24:18).

The traffic light prcblem originated in A. Alan Pritsker's text on SLAM (24:22). The system modeled is a two lane road which is undergoing repairs along a 500 meter stretch, shown in Figure 3.1. Traffic must be regulated along this stretch since only one

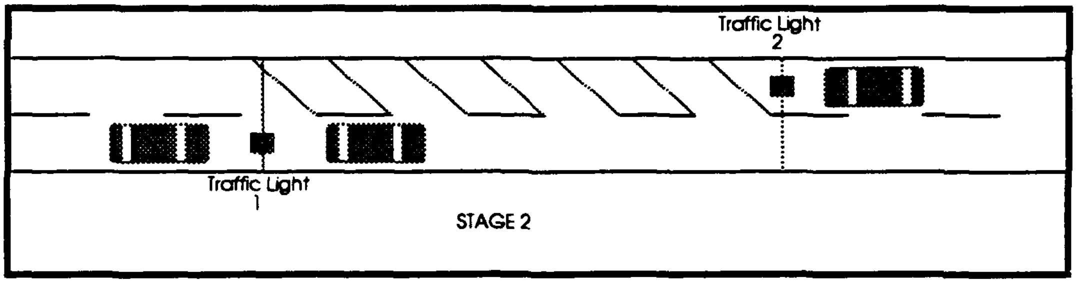


FIGURE 3.1: REGULATING TRAFFIC

lane is available. In order to regulate traffic, two traffic lights have been placed so that one is at either end of this one-lane stretch of road. A cycle must be designed for these lights to minimize the average time a car must wait before proceeding across the stretch. Each cycle consists of four stages: 1) both lights are red, 2) light 1 is green while light 2 is red, 3) both lights are red, and 4) light 1 is red while light 2 is green. The timing for stages (1) and (3) are fixed at 55 seconds while the timing for stages (2) and (4) can vary.

3.1.2. The Allocation of Machines in a Job-Shop -- Open Queuing Network.

The second simulation model was selected because it represented a complex problem and implemented easily. The complexity comes from the size of the problem: it requires six decision variables. These variables represent the number of machines available at each of six machine stations. The model was easy to implement because of the similarity between these stations. The objective value is the average total time that a part must wait at the various machine stations while being processed. The objective is to minimize this value. The fewer machines there are available at each station, the longer a part must wait on average. The total number of machines available to all stations is constrained to less than 25. The problem is to optimally allocate these machines to the six machine stations illustrated in Figure 3.2.

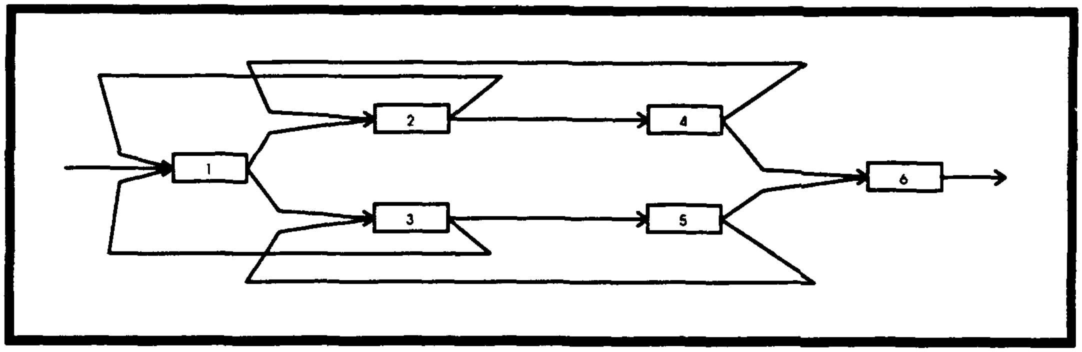


FIGURE 3.2 OPEN QUEUING NETWORK

The figure shows the flow of parts through each of the machine stations. A part enters the system at station 1 and then proceeds to either station 2 or station 3. From station 2 (3), the part goes on to station 4 (5) or, if it needs more work, it returns to station 1. The part proceeds to station 6 or returns to station 2 (3) for additional work after completing the activity at station 4 (5). After the part completes the activity at station 6 it leaves the system. Once the part leaves the system, the total time spent waiting for service can be calculated.

3.1.3. The Allocation of Machines in a Job-Shop -- Closed Queuing Network.

The third simulation model is no more than a variation on the second model. Instead of allowing the parts to enter and exit the system freely, the model contains a fixed number of parts that cycle through the stations. Figure 3.3 shows the altered flow of parts.

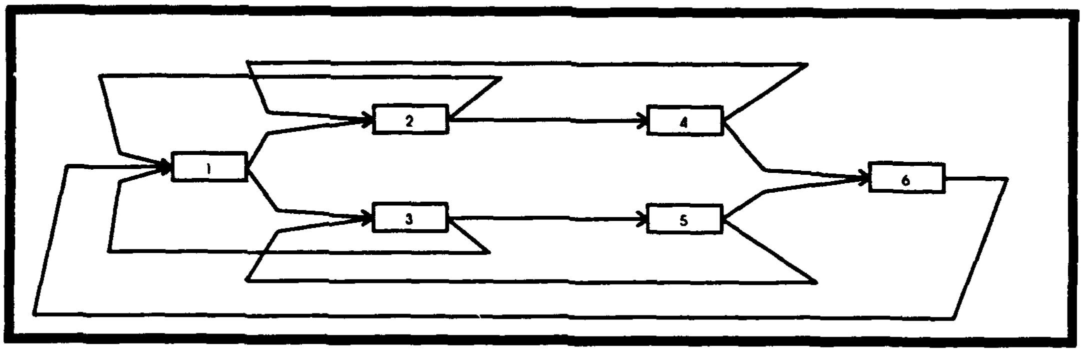


FIGURE 3.3 CLOSED QUEUING NETWORK

The model contains 30 parts that begin processing at station 1 and finish at station 6, where the waiting time for one cycle is measured. The part proceeds through the machines stations as before. The objective value is the total waiting time accrued by a part from station 1 through station 6. After completing the activity at station 6, the waiting time for a given part is reset to zero and it re-enters the system at station 1.

3.2. The Simulated Annealing Algorithm

The Simulated Annealing algorithm optimizes the decision variables for a given simulation model. Its structure provides a common platform for testing alternative acceptance functions. The generic framework built into the network models enables the user inserts to be developed as modules. The global variables can be passed back and forth between the user inserts through the SLAM executive. The algorithm itself consists of four such modules: the executive module, the initialization module, the network module, and the acceptance module. Figure 3.7 shows the program's flow through each of these.

The executive module consists of the SLAM executive and its extensions. The SLAM executive is a FORTRAN based program that defines a common block of variable names as well as user-callable functions and subroutines.(25:389) The structure of the MAIN program allows the user to specify input conditions, network control, and output processing through external subroutines.

The initialization module consists of the INTLC subroutine and its subordinates. "INTLC" is the name reserved by the SLAM executive for the initialization of a simulation. Before executing a network model, the SLAM executive carries out the instructions in this user-written subroutine. For the Simulated Annealing algorithm, these instructions perform two main functions: (1) to determine a starting configuration by calling the FIRST subroutine, and (2) to select a neighboring configuration by calling the NEXT subroutine. The FIRST subroutine selects a starting configuration using a uniform[0,1] random number from the RAN subroutine to permute a base configuration.

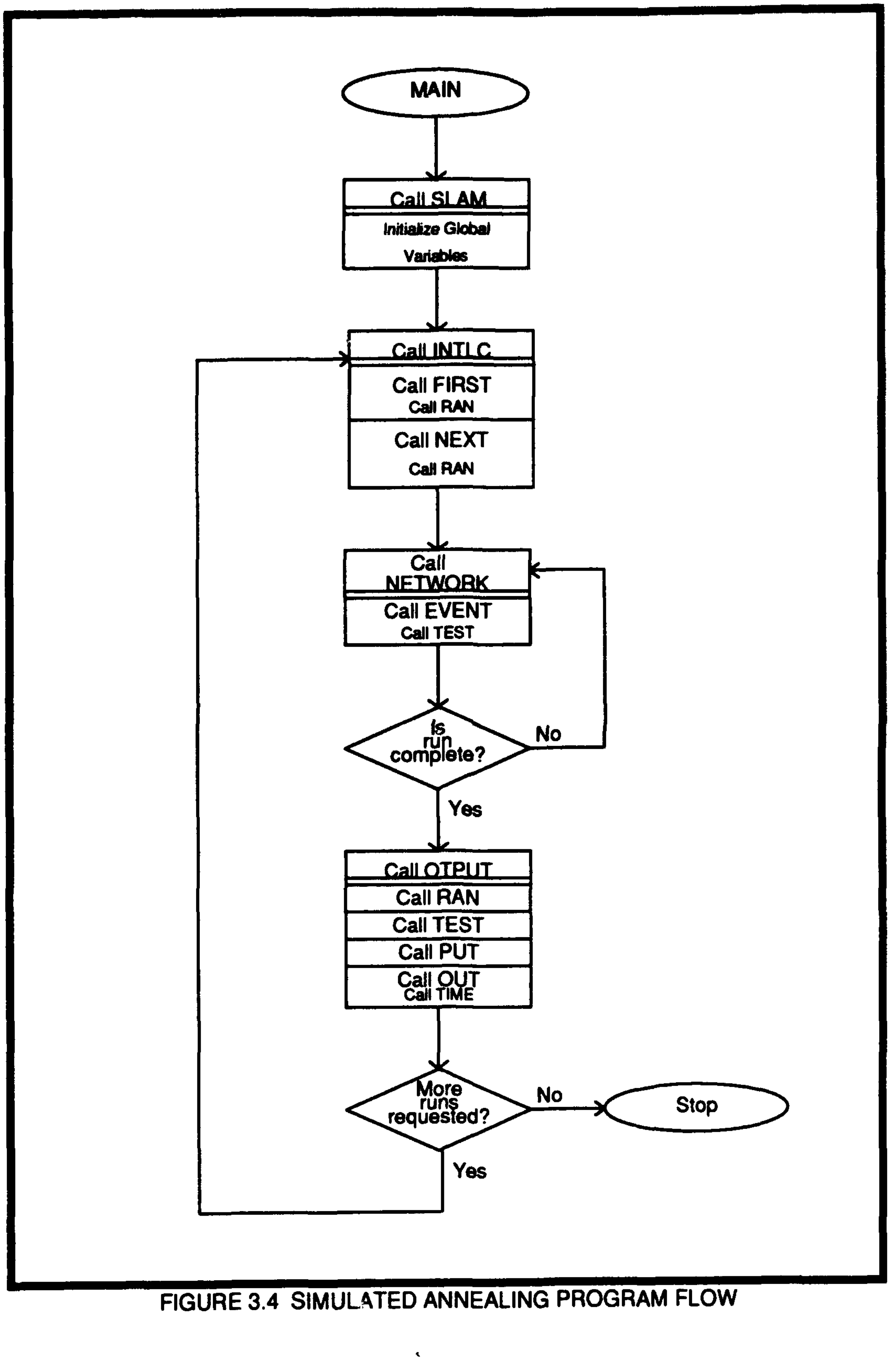


FIGURE 3.4 SIMULATED ANNEALING PROGRAM FLOW

The NEXT subroutine may or may not use a random number from the RAN subroutine, depending on the permutation scheme used. A fixed permutation scheme is used on the first and third simulation models and a random permutation scheme is used on the second model. The INTLC subroutine cycles through the alternative acceptance tests. For each acceptance test, INTLC calls the FIRST subroutine to determine a starting configuration and successively calls the NEXT subroutine to try neighboring configurations. INTLC reinitializes the random number seeds used by the RAN subroutine between successive tests. This ensures that all of the acceptance tests use the same set of starting configurations. This reduces the noise between alternative acceptance tests so that true performance differences can predominate.

The network module consists of the network model and the EVENT subroutine with its subordinate, the TEST subroutine. The network model generates an estimate of the response value for a given configuration. The EVENT subroutine is used to collect observations on the performance measure using many of the techniques reported by Pegden and Gately: collecting observations after the simulation reaches steady-state, batching observations to minimize variance, and varying batch sizes to reduce computer time (24:19-22). Existing studies referenced by Pegden, Gately, and Pritsker lend credibility to the values used to implement these techniques (24:21; 25:735).

The network model calls the EVENT subroutine when an entity passes through an EVENT node (i.e., when a car passes through a green light or when a part completes processing). As in Pegden's study, the EVENT subroutine is designed to collect observations only after the first 500 seconds in order to achieve steady-state. Fifty samples are collected for determining each batch mean. Ten batch means are collected before the estimated change in response is determined. After the first ten batch means are collected, the probability of accepting the challenger is calculated using the TEST subroutine. This is a preliminary test used to determine the need for collecting more batch means. If the probability of acceptance obtained from the TEST subroutine falls below a predetermined threshold probability, then EVENT stops the trial. Otherwise the network model continues to generate more observations until a total of 30 batch means are collected or the probability of acceptance falls below the threshold. The threshold probability is 20% for the Timing of Traffic Lights and Closed Queuing Network models and 100% for the Open Queuing model. The threshold trades some efficiency (by requiring more batches) for more accuracy in the estimates of challenger responses with acceptance probabilities above the threshold. Therefore, the algorithm used on the Open Queuing model is more efficient but less accurate than the algorithms used on the other two models. The more accurate algorithm would have been used on the Open Queuing model, but the lack of time made this impossible. By varying the number of batch means used to estimate a challenger's response, the algorithm uses less time estimating unlikely moves. This makes the algorithm generally more efficient. Since the actual efficiency depends on the probability returned to EVENT from TEST, there may be a difference in efficiency among the alternative acceptance tests.

The acceptance module consists of the OTPUT subroutine and its subordinates. "OTPUT" is a reserved name like INTLC and EVENT. Once a simulation run completes, SLAM automatically executes OTPUT's instructions. The Simulated Annealing algorithm uses this link both to compare estimated responses and to adjust the annealing schedule. OTPUT compares the estimated response by using the RAN and TEST subroutines and adjusts the annealing schedule by updating the temperature and other control parameters. The RAN subroutine supplies a uniform[0,1] random number and the TEST subroutine determines the probability of accepting the challenger. These values are used in the final acceptance test. An acceptance probability equal to 1 indicates that the challenger is more desirable than the incumbent and must be accepted. Otherwise, the challenger is accepted only when the random number drawn from RAN is less than the probability calculated by TEST. If the challenger is accepted, then the PUT subroutine records the configuration and response statistics.

The PUT subroutine not only maintains these values for the incumbent, but also for the best-to-date. The incumbent may have a less optimal response value than the best value obtained from all the previous trials. The best-to-date records the most optimal response encountered. Once each annealing run completes, the OUT subroutine writes the performance statistics, including the CPU time used, and the decision variable settings for the best-to-date to an output file. The TIME subroutine determines the CPU time used. The statistics recorded in the output file are used to compute the average solution quality and the average efficiency. These performance measures are used to compare Local Search and Simulated Annealing with seven different acceptance tests.

3.3. The Acceptance Tests

Aside from validating the general methodology of Simulated Annealing, the purpose of this thesis is to investigate alternative acceptance functions. The first step in formulating these variations is to code the algorithm crudely and observe the annealing behavior. The next step is to characterize both the observed and the desired behavior. The final step is to devise alternative acceptance tests to achieve the desired behavior. Each of these steps are discussed in more detail.

3.3.1. Annealing Behavior Observed.

The crude behavior of Simulated Annealing is observed by coding a selected technique for a given problem and tracking the value of the incumbent's response over the annealing period. Specifically, the annealing algorithm designed by Johnson *et al.* is used to optimize the timing of traffic lights. Figure 3.5 shows the result of one such annealing run.[1]

The crude annealing code used in Figure 3.5 implements an homogeneous algorithm that lowers the temperature gradually in stages. A lower temperature causes a decrease in the probability of accepting a fixed change in the response. Since the value for a good starting temperature is unknown, the value is set arbitrarily to one. Using the same approach that Johnson's group used, the temperature is reduced every ten trials and the annealing run is stopped when the ratio of acceptances to rejections is less than a threshold value. While Figure 3.5 shows an outcome of one annealing run using this algorithm, it does not illustrate how the probability of acceptance changes over time.

It seems that a more systematic, logical approach for characterizing the acceptance test is needed. One approach is to plot the equilibrium probability used to determine the likelihood of acceptance. In order to picture how the acceptance probability decreases over time, the trial number is plotted against the corresponding probability of acceptance for several response value differences. The plot is termed a convergence plot since the acceptance probability tends towards zero as time progresses (as the trials number gets larger). Figure 3.6 shows the convergence plot for the algorithm used in Figure 3.5.

[1] The "Response Value" used was not the average waiting time, but a value contrived to account for variance. Although this response measure was not adopted later, the results in Figure 3.8 illustrate the characteristic annealing behavior.

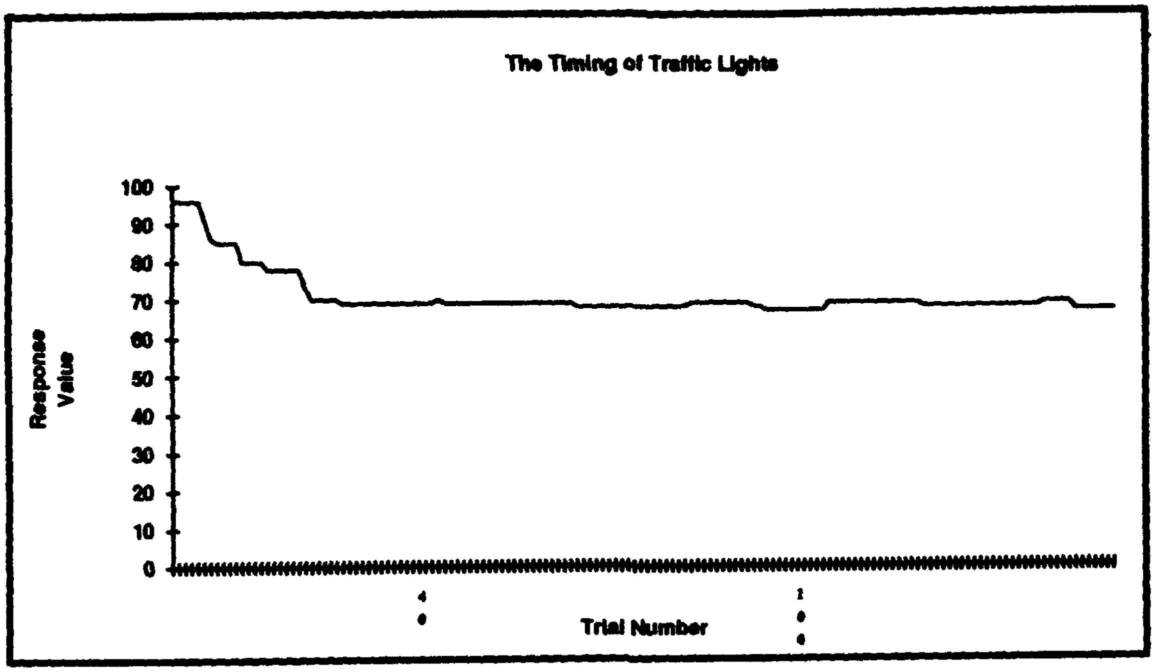


FIGURE 3.5 ANNEALING BEHAVIOR

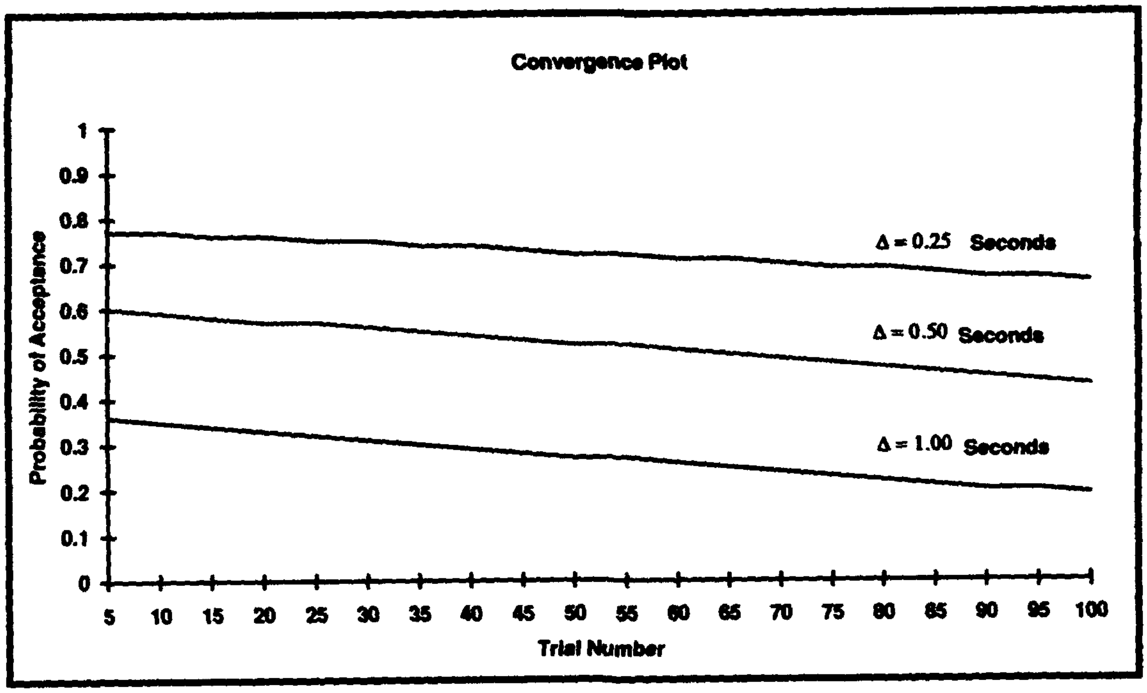


FIGURE 3.6 JOHNSON'S ALGORITHM

3.3.2. Annealing Behavior Desired.

Does the annealing behavior shown in Figure 3.6 coincide with the desired annealing behavior? The Simulated Annealing algorithm used by Johnson's group is designed for deterministic problems. The algorithm uses thousands of trial configurations and convergence upon the optimum occurs very slowly. An acceptance plot for an algorithm of this sort is shown in Figure 3.7. Note that the probability of acceptance is high during the early trials and approaches zero in the latter trials. The convergence plot shown in Figure 3.6 is does not appear to behave in quite the same way. The problem lies in the fact that only 100 trials are shown. Why? The amount of CPU time available to find a good solution limits the number of trials that can be made. With simulation models, the time used in estimating each trial's response reduces the total number of trials.

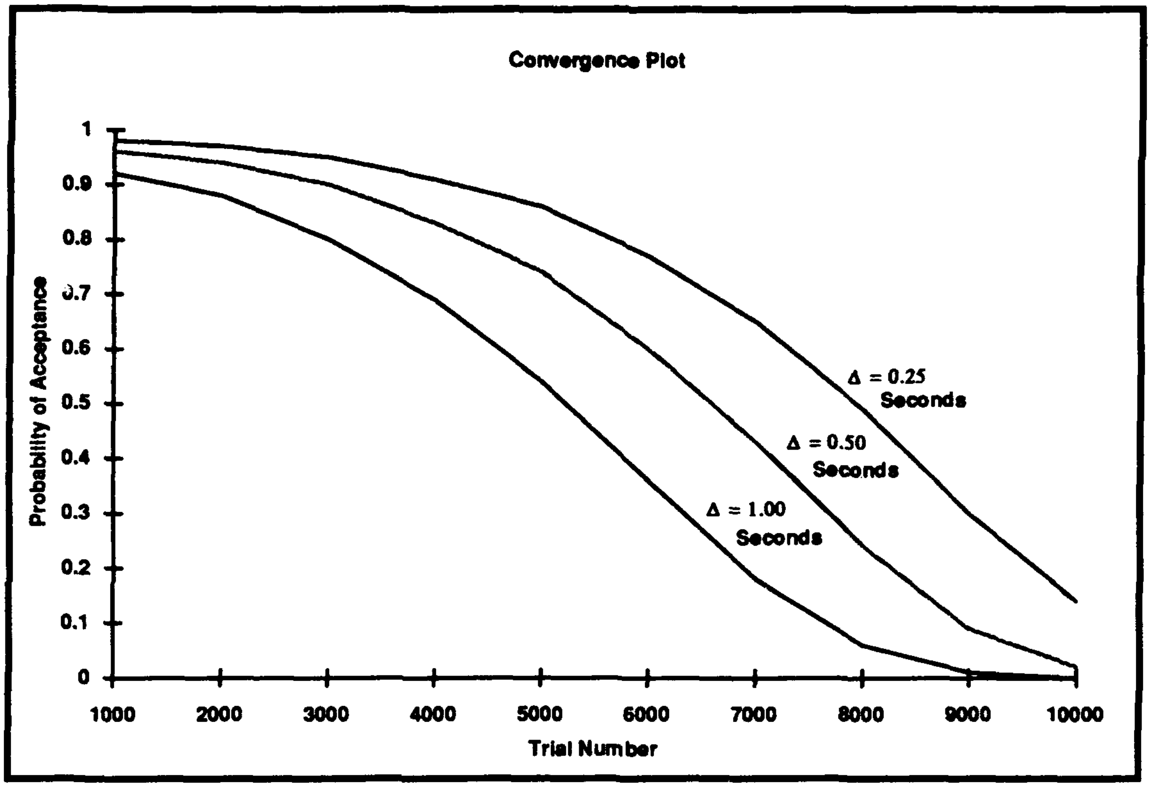


FIGURE 3.7 DETERMINISTIC ALGORITHM

The convergence plot must be tailored to mitigate the limitations of an annealing run consisting of, at most, hundreds of trials. The first consideration is to choose between homogeneous and nonhomogeneous implementations. The next consideration is to modify the chosen implementation in two ways: to accept larger changes initially and to converge on a solution more quickly.

Theory does not indicate a preference between homogeneous and nonhomogeneous algorithms. Nonhomogeneous algorithms, however, yield a practical advantages. Given short annealing runs, the homogeneous algorithm would require large drops in the acceptance probabilities between successive segments. Such large drops invalidate the assumption of gradual cooling. The nonhomogeneous algorithm generates a smoother plot since the temperature is adjusted at each trial. This smoothness avoids the large discontinuities between successive segments. Therefore, a nonhomogeneous implementation appears to be the better choice when the annealing run is time limited.

Modifications to a nonhomogeneous algorithm can allow larger changes in the response value to be accepted initially yet converge to a solution more quickly. These modifications constitute a compromise between pure Local Search and pure Simulated Annealing. The resulting search is able to escape some poor local optima without requiring too many trials. Both of these aims are related. In order to ensure the search can escape poor local optima, some way of relating the probability of acceptance to the model's specific behavior must be found. This modification is termed *tuning*. The acceptance probability could not be high initially and low towards the end of the annealing run unless the probability converges to zero in a finite number of trials. This modification is termed *forced convergence.*

3.3.2.1. Tuning.

Tuning relates the behavior of the model to the probability of acceptance so that the search can escape some poor local optima while not accepting too many poor moves. Tuning lies more in the domain of art than of pure objectivity. Traditionally, the selection of a method for the tuning results from trial and error, experience, and intuition (15:869). The tuning method adopted in this research relates the variance in the estimate to a certain probability for a given change in the response at a set time during the annealing run. The tuning method is explained by examining the general form of the acceptance test and showing how the relationship is determined.

Equation 3.1 shows the general form of the acceptance test, where α represents the tuning parameter and *c(t)* is a coefficient function that may be used to force convergence.

$$P(\Delta,t) = \begin{matrix} 1 & ;\Delta \leq 0 \\ c(t)e^{-\Delta/\alpha T(t)} & ;\Delta > 0 \end{matrix} \quad (3.1)$$

The value for α establishes the desired relationship. The given change in response (Δ) is proportional to the variance of the estimate (σ). The certain probability of acceptance occurs when the trial (t) reaches the half-way point during the annealing run (which consists of n trials). The desired relationship is given by:

$$P(\Delta,t) = 0.05; \quad \Delta = \sigma/3; \quad \text{and} \quad \mathrm{t = n/2} \quad (3.2)$$

Applying these values to Equation 3.1 gives the value for α as:

$$\alpha = -(\sigma/3)/[\ln(0.05/c(n/2)) \bullet T(n/2)] \quad (3.3)$$

Although this tuning method seems arbitrary, it achieves good solution quality in the first two models studied. It allows all of the acceptance functions to be tuned in the same way so that differences in performance are due more to differences in the functions themselves than to differences in the tuning method used. The tuning method adopted allows the probability of acceptance to be related to the accuracy of the response estimate. Each response is estimated using thirty batch means. The variance of the estimate (σ) is the variance among those batch means. In practice, σ is estimated from the variances of several response estimates. Each response is the mean of the thirty batch means. The standard deviation of the mear ($\sigma(\bar{y})$) is given by (17:5-12):

$$\sigma(\bar{y}) = \sigma / \sqrt{30} \quad (3.4)$$

This implies that the change in response used for tuning ($\sigma/3$) measures the difference between means that are apart by *1.826 standard deviations of the mean.* In this way, the tuning method relates the probability of acceptance to the resolution of the responses.

3.3.2.2. Forced Convergence.

Forced convergence means that the temperature converges to zero, not in an infinite number of trials, but in a finite number of trials. This yields another advantage. If each trial executes within one minute and the algorithr terminates after 100 trials, then at most 100 minutes of computing time will be used. Conversely, if only 70 minutes are available, then the algorithm can be forced to terminate after 70 trials.

The modification can be made in one of two ways: using the temperature function (*T(t)*) to force convergence; or using the coefficient function (*c(t)*) to force convergence. Examining the general form of the acceptance test shows how either modification can be made:

$$P(\Delta,t)=\begin{matrix}1 & ;\Delta\leq 0\\ c(t)e^{-\Delta/\alpha T(t)} & ;\Delta>0\end{matrix}\quad(3.5)$$

The first of these methods use the temperature function to force the probability of acceptance to zero in a fixed number of trials (n). This is accomplished by letting the temperature function range between 1 and 0, resulting in:

$$P(\Delta,1)=\begin{matrix}1 & ;\Delta\leq 0\\ e^{-\Delta/\alpha} & ;\Delta>0\end{matrix};\quad P(\Delta,n)=\begin{matrix}1; & \Delta\leq 0\\ 0; & \Delta>0\end{matrix}\quad(3.6)$$

Note that once the trial number reaches n, the algorithm reverts to Local Search. This modification achieves a compromise between pure Simulated Annealing and pure Local Search. The second method achieves the same compromise by letting the coefficient function range between 1 and 0 as the trials progress from 1 to n. The resulting algorithm behaves similarly, except that the temperature function is unaffected:

$$P(\Delta,1)=\begin{matrix}1 & ;\Delta\leq 0\\ e^{-\Delta/\alpha T(t)} & ;\Delta>0\end{matrix};\quad P(\Delta,n)=\begin{matrix}1; & \Delta\leq 0\\ 0; & \Delta>0\end{matrix}\quad(3.7)$$

This allows a temperature function traditionally used on deterministic problems to be used on simulation models.

3.3.3. Alternative Acceptance Tests Devised.

Several alternatives are developed based on these modifications. The tuning method of Equation 3.3 is fixed for all of the alternatives. Two of the alternatives use traditional methods without forced convergence. Three alternatives use variations on the temperature function to force convergence. Two alternatives use variations on the coefficient function to force convergence. Each of these alternatives is presented.

3.3.3.1. Geometric Temperature.

The first alternative uses a traditional temperature function without forced convergence. It is termed geometric because each successive temperature is a proportion of the previous temperature (9:211). The acceptance test can be given by:

$$P(\Delta,t)=\begin{matrix} 1 & ;\Delta\leq 0 \\ e^{-\Delta/\alpha T(t)} & ;\Delta>0 \end{matrix};\quad T(t)=r^{t-1};\quad 0<\mathrm{r}<1 \quad (3.8)$$

Where *r* is the proportion of the temperature retained from one trial to the next. Figure 3.9 shows the convergence plot for *n = 100*, and *r = 0.995*. It also shows effect of tuning using a *tuning plot*. The tuning plot shows the acceptance probability as a function of the change in response value (Δ) for several trial numbers.

This alternative causes the acceptance probability to converge on zero very slowly. The rate of convergence is not keyed to a predetermined run length (*n*). It depends solely on the tuning parameter α to ensure that the algorithm converges on a solution in a reasonable amount of time.

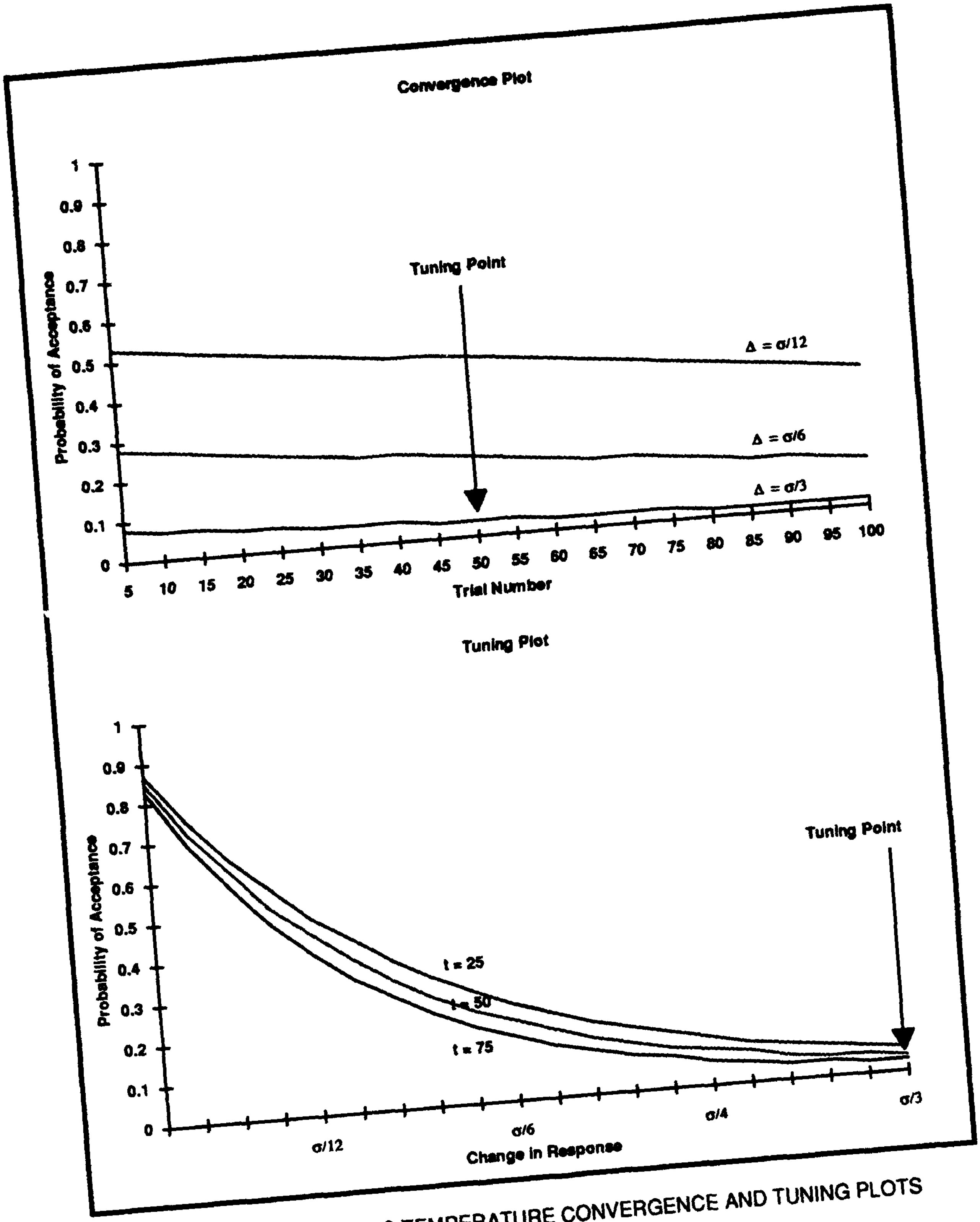


FIGURE 3.8 GEOMETRIC TEMPERATURE CONVERGENCE AND TUNING PLOTS

3.3.3.2. Linear Temperature.

This alternative forces convergence using the temperature function. This causes the acceptance function to converge more quickly and can be tailored to a predetermined run length. The temperature is reduced at a constant rate going from 1 to 0 in n trials:

$$P(\Delta,t)=\begin{matrix} 1 & ;\Delta\leq 0 \\ e^{-\Delta/\alpha T(t)} & ;\Delta>0 \end{matrix};\quad T(t)=\frac{n+1-t}{n} \quad (3.9)$$

Figure 3.9 shows the resulting convergence plot and Figure 3.10 the tuning plot.

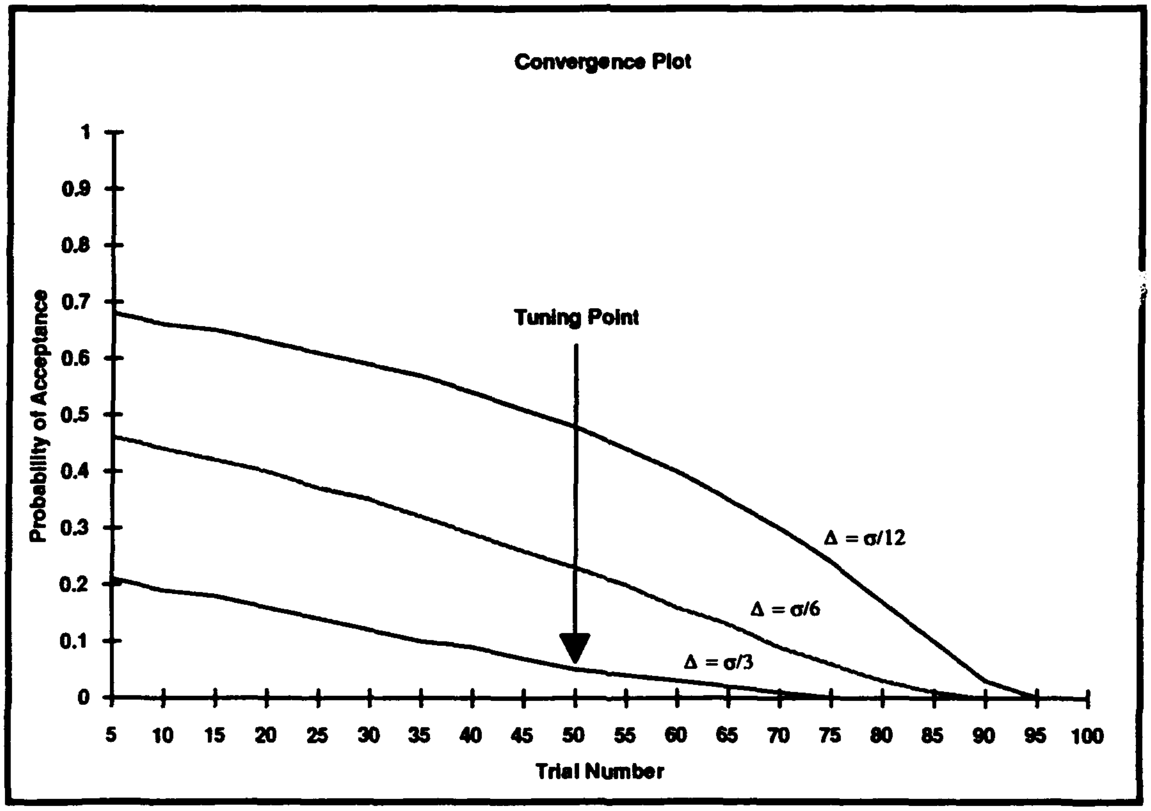


FIGURE 3.9 LINEAR TEMPERATURE CONVERGENCE PLOT

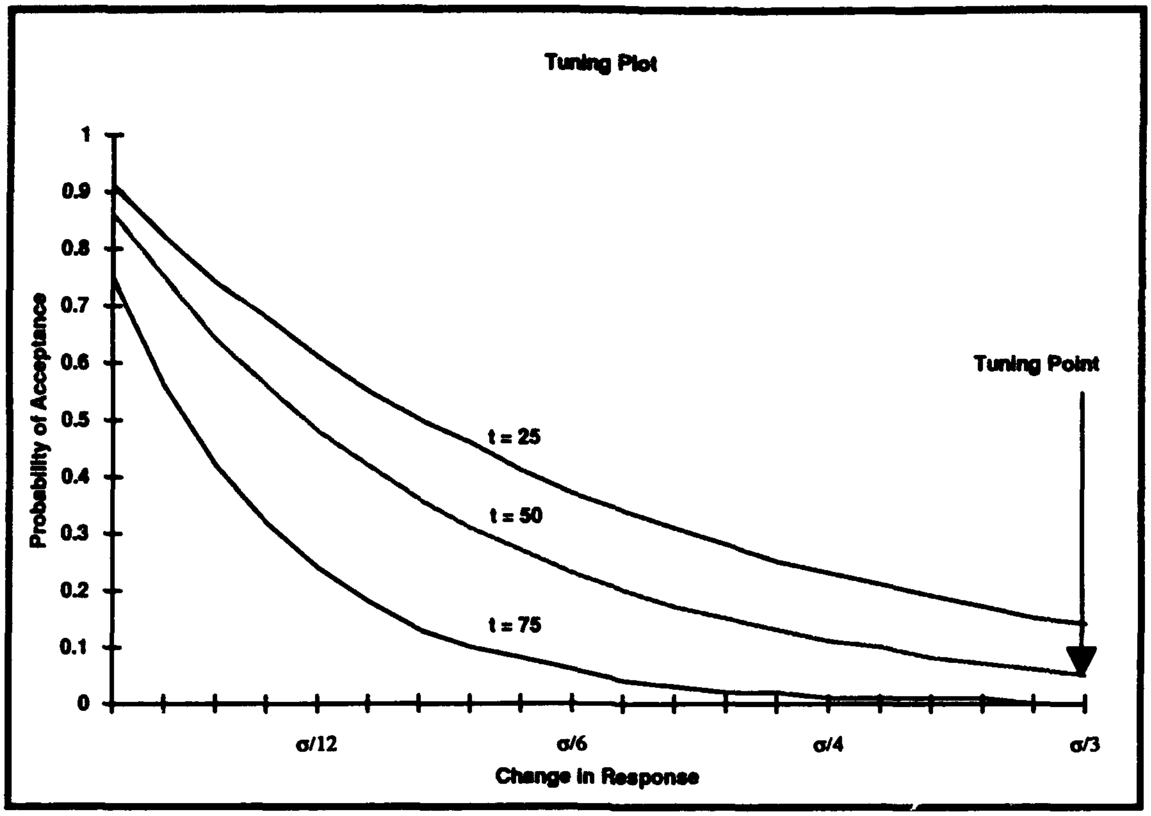


FIGURE 3.10 LINEAR TEMPERATURE TUNING PLOT

Note how this tuning plot reveals a wider separation among the probability curves at different trial numbers than the previous tuning plot. This difference in shape may correspond to a difference in performance when the annealing run is time-limited.

3.3.3.3. Adaptive Temperature.

This alternative differs from Linear Temperature by making use of the variance in the response. The cooling rate captures the system's variability by pooling the sample variance from the challenger and incumbent responses; the concept of a pooled variance is derived from the same method used by a two sample *t*-test (17:6-16):

$$P(\Delta,t)=\begin{matrix}1 & ;\Delta\le 0\\ e^{-\Delta/\alpha T(t)} & ;\Delta>0\end{matrix};\quad T(t)=\sigma'\bullet\frac{n+1-t}{n};\quad \sigma'=\sqrt{\frac{(m_t-1)\sigma'_t+(m_{t-1}-1)\sigma'_{t-1}}{m_t+m_{t-1}-2}}\quad (3.10)$$

Where σ' is the pooled standard deviation, σ'^2_t is the sample variance for the challenger, σ'^2_{t-1} is the sample variance for the incumbent, m_t is the number of batches used to estimate the challenger's response, and m_{t-1} is the number of batches used to estimate the incumbent's response. Figure 3.11 shows the result of three different challenger variances in the convergence plot. The plot was generated using a fixed change in the response $\Delta = \sigma\ /6$, and a fixed standard deviation in the incumbent's response $\sigma'_{t-1} = \sigma$.

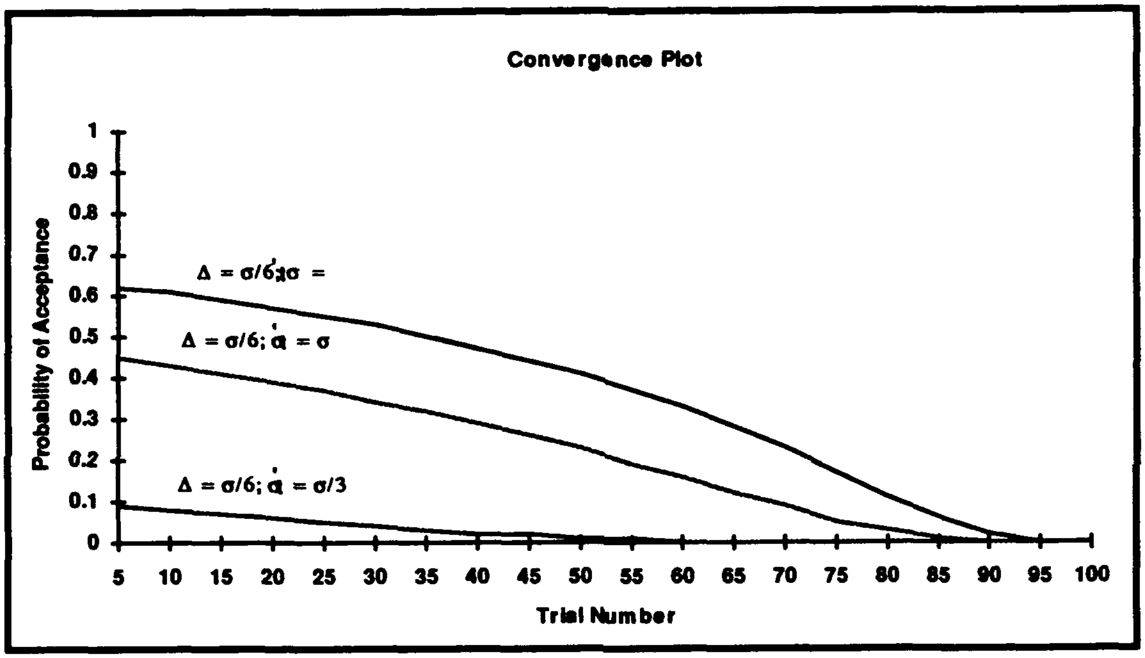


FIGURE 3.11 ADAPTIVE TEMPERATURE CONVERGENCE PLOT

The significance of this alternative lies in the effect that a change in the challenger's variance has on the probability of acceptance. It turns out that $\alpha_{[\text{Adaptive Temperature}]} = \alpha_{[\text{Linear Temperature}]}/\sigma$. If the response variance does not change between trials, then this alternative is equivalent to Linear Temperature ($\sigma' = \sigma$, so that $\sigma \bullet \alpha_{[\text{Adaptive Temperature}]} = \alpha_{[\text{Linear Temperature}]}$). If the response variance does change, then this alternative will react by accepting trials with larger variance more readily and rejecting trials with smaller variance more frequently. This reflects the degree of certainty implied by a smaller variance. Statistically, the smaller variance implies more certainty for the same difference in response value than a large variance. Recall that the tuning method uses an estimate for σ. This alternative modifies the acceptance probability to account for variations in σ among the trials.

3.3.3.4. Elliptic Temperature.

This alternative uses a temperature function that changes little initially and accelerates the rate of change towards the end of the run. The functional form is that of an ellipse. In this case, the independent variable (t) and the dependent variable ($T(t)$) are always non-negative:

$$P(\Delta,t)=\begin{matrix}1 & ;\Delta\leq 0\\ e^{-\Delta/\alpha T(t)} & ;\Delta>0\end{matrix};\quad T(t)=\sqrt{1-\frac{(t-1)^2}{n^2}}\quad (3.11)$$

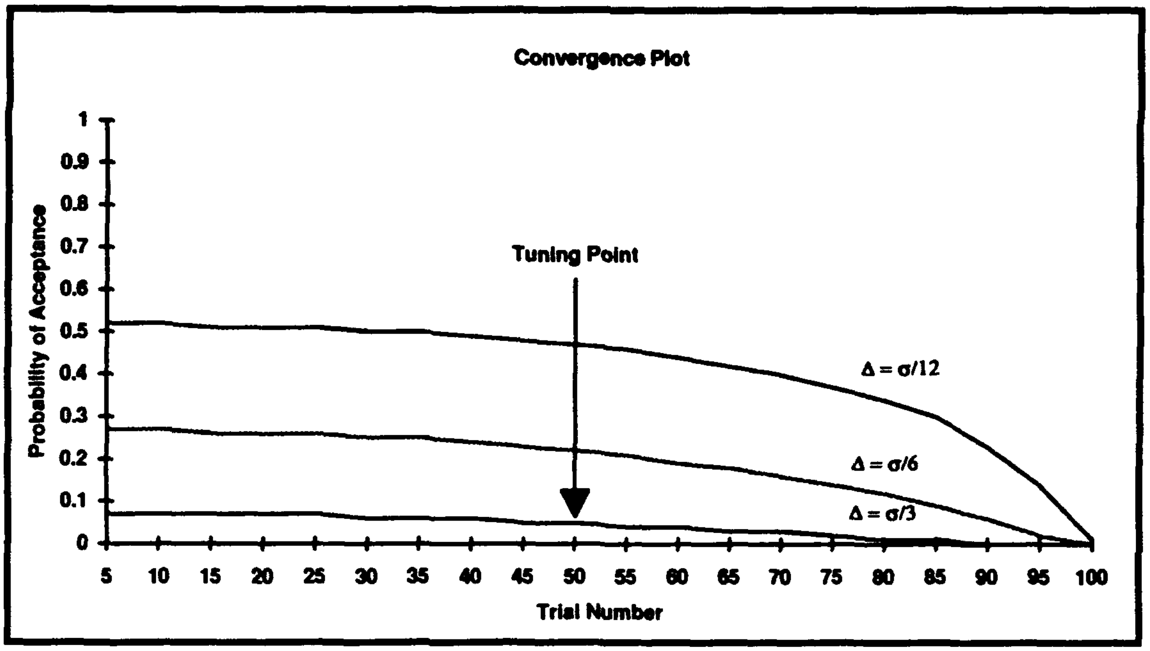


FIGURE 3.12 ELLIPTIC TEMPERATURE CONVERGENCE PLOT

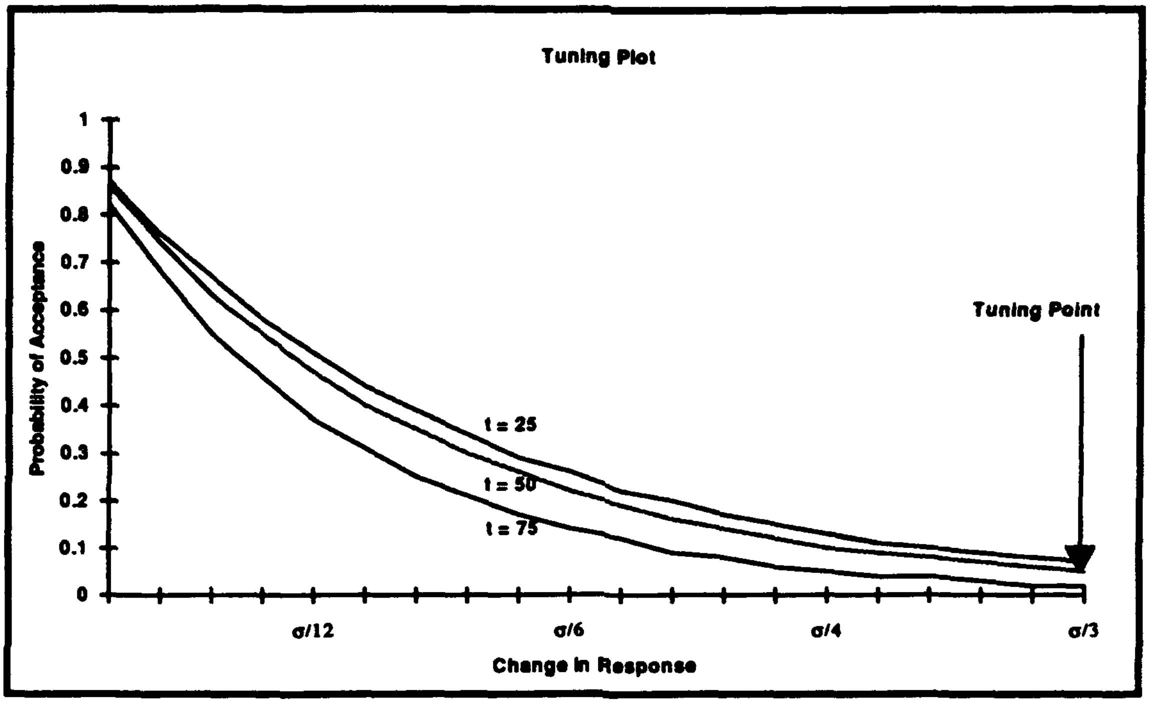


FIGURE 3.13 ELLIPTIC TEMPERATURE TUNING PLOT

Note that this alternative preserves the shape of the Geometric Temperature plots more than Linear Temperature and Adaptive Temperature. It constitutes a compromise between the Geometric and Linear Temperature techniques and will help to show which dominates.

3.3.3.5. Logarithmic Temperature.

This alternative uses another traditional temperature function without forced convergence. It provides the basis for the next two alternatives that use the logarithmic temperature function in conjunction with a coefficient function to force convergence. It also provides a basis for measuring the performance differences that result. The logarithmic temperature acceptance test is given by:

$$P(\Delta,t)=\begin{matrix}1 & ;\Delta\leq 0 \\ e^{-\Delta/\alpha T(t)} & ;\Delta>0\end{matrix}; \quad T(t)=1/\ln(t+1) \quad (3.12)$$

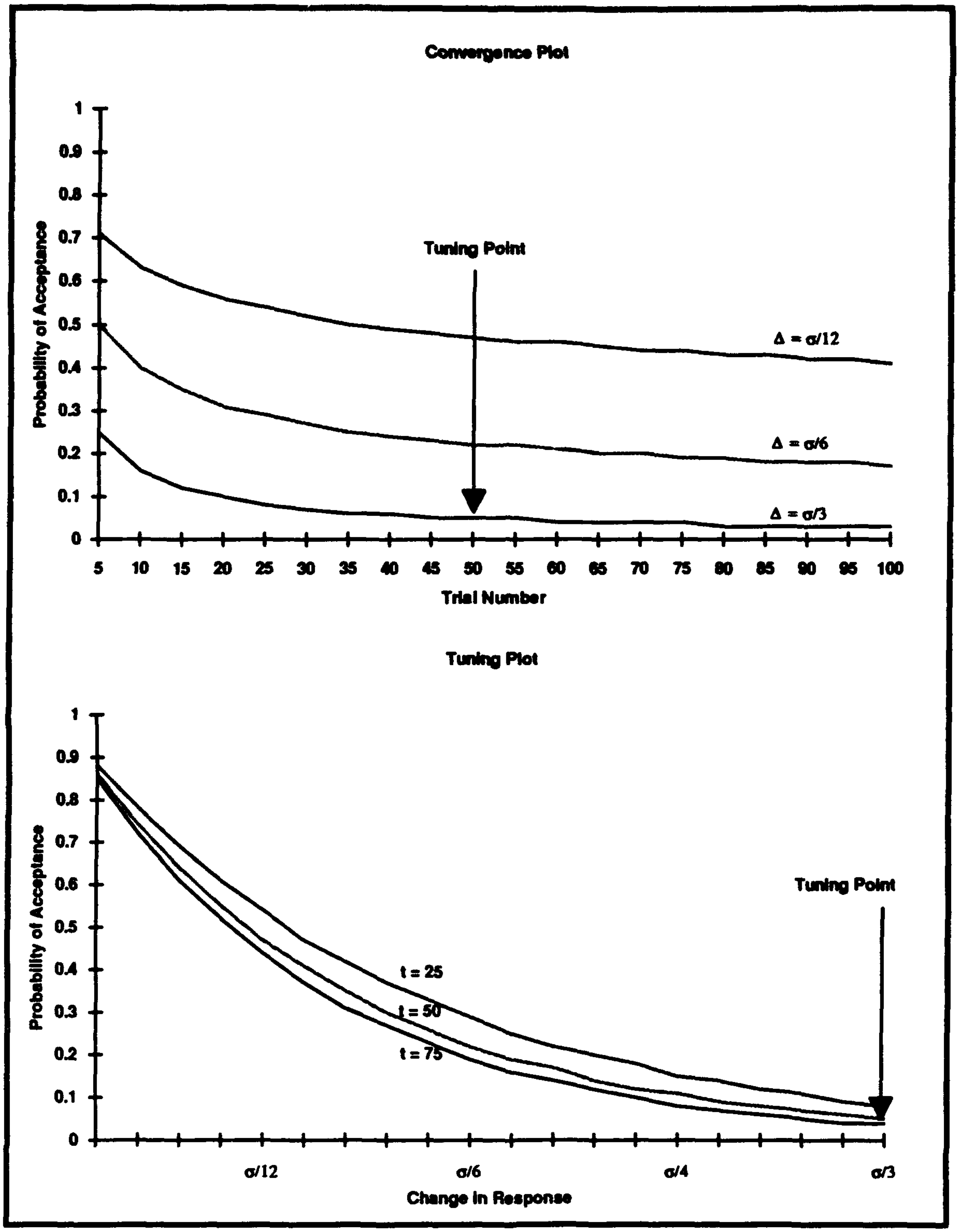


FIGURE 3.14 LOGARITHMIC TEMPERATURE CONVERGENCE AND TUNING PLOTS

Note that a significant difference between previous cooling rates and the logarithmic cooling rate lies in the probabilities of the first few trials. Probabilities based on the logarithmic function decrease rapidly in the first few trials but change gradually in later trials. As with the Geometric Temperature, the tuning plot shows little change in the acceptance probabilities over much of the annealing run.

3.3.3.6. Linear Coefficient.

This alternative uses a coefficient function that pre-multiplies the acceptance probability obtained from a logarithmic temperature function. Recall that the logarithmic temperature ensures quasi-equilibrium between trials. The intent of the coefficient is to keep this seeming equilibrium intact, but also to scale the resulting probability according to the trial number. This alternative forces convergence by using a linearly decreasing function for the coefficient:

$$P(\Delta,t)=\begin{matrix} 1 & ;\Delta\leq 0 \\ c(t)e^{-\Delta/\alpha T(t)} & ;\Delta>0 \end{matrix};\quad T(t)=1/\ln(t+1);\quad \mathrm{c(t)}=\frac{\mathrm{n-t}}{\mathrm{n}} \quad (3.13)$$

This acceptance test implies that, as the trials progress, the scaling factor reduces each probability at a constant rate. As with other alternatives, a different value for the tuning parameter results from the difference in the form of the acceptance function. The effect of this difference is to start with higher acceptance probabilities initially and reduce them more quickly near the end of the annealing run. Figure 3.15 shows the resulting plots. Note that the tuning plot shows a dramatic difference compared with previous alternatives. The acceptance probabilities for small adverse changes in the response are much less. Therefore, this alternative accepts fewer trials with small adverse changes in the response.

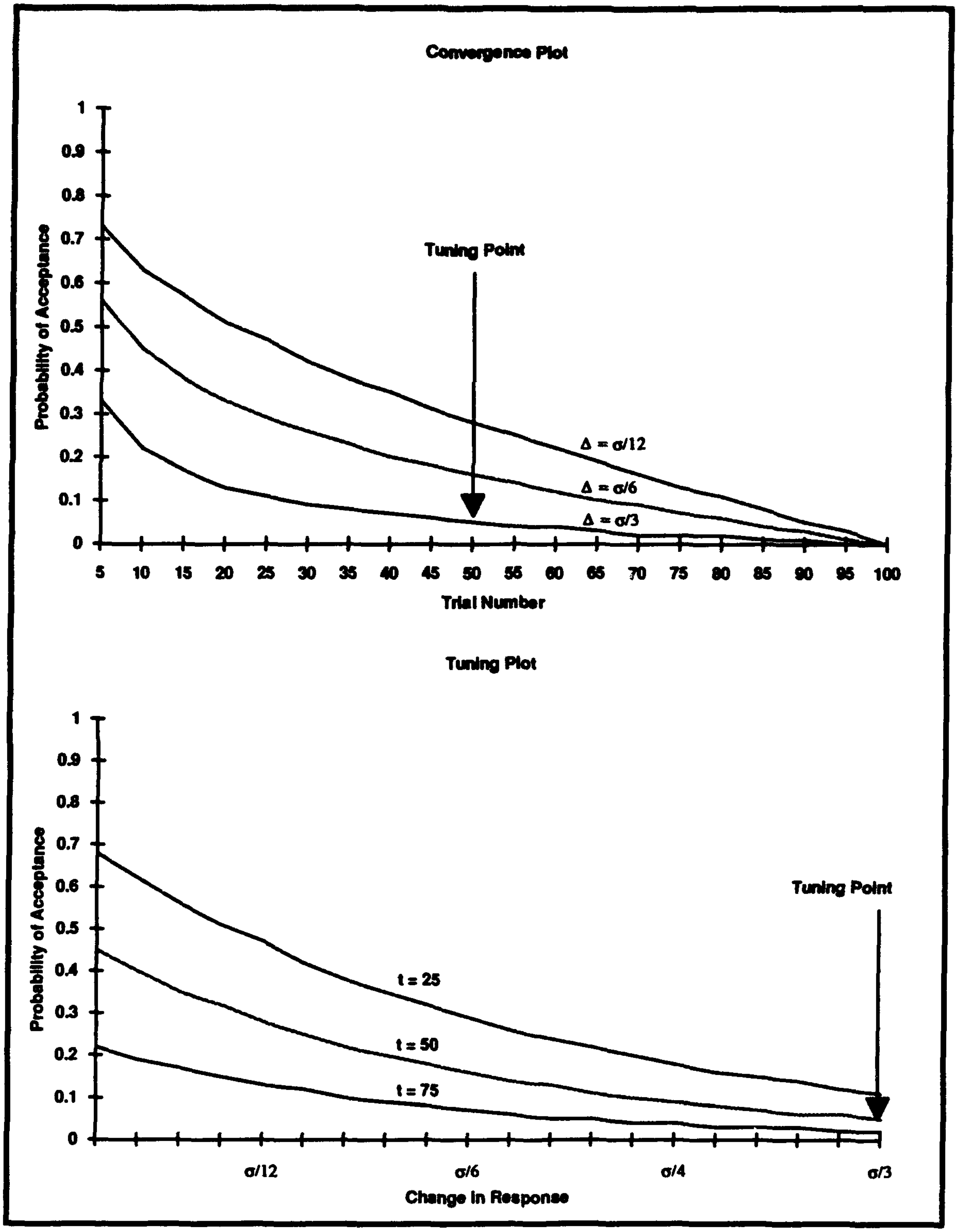


FIGURE 3.15 LINEAR COEFFICIENT CONVERGENCE AND TUNING PLOTS

3.3.3.7. Elliptic Coefficient.

This alternative uses a coefficient function to pre-multiply the acceptance probability. The coefficient function describes an ellipse, as before. The elliptically decreasing function preserves more of the Logarithmic Temperature's shape than the Linear Coefficient:

$$P(\Delta,t)=\begin{matrix} 1 & ;\Delta\leq 0 \\ c(t)e^{-\Delta/\alpha T(t)}; & \Delta>0 \end{matrix};\quad T(t)=1/\ln(t+1);\quad \mathrm{c(t)}=\sqrt{1-\frac{\mathrm{t}^2}{\mathrm{n}^2}}\quad (3.14)$$

This alternative also has the effect of reducing the acceptance probabilities of small adverse changes in the response. As seen in Figure 3.17, the effect is less pronounced than with Linear Coefficient.

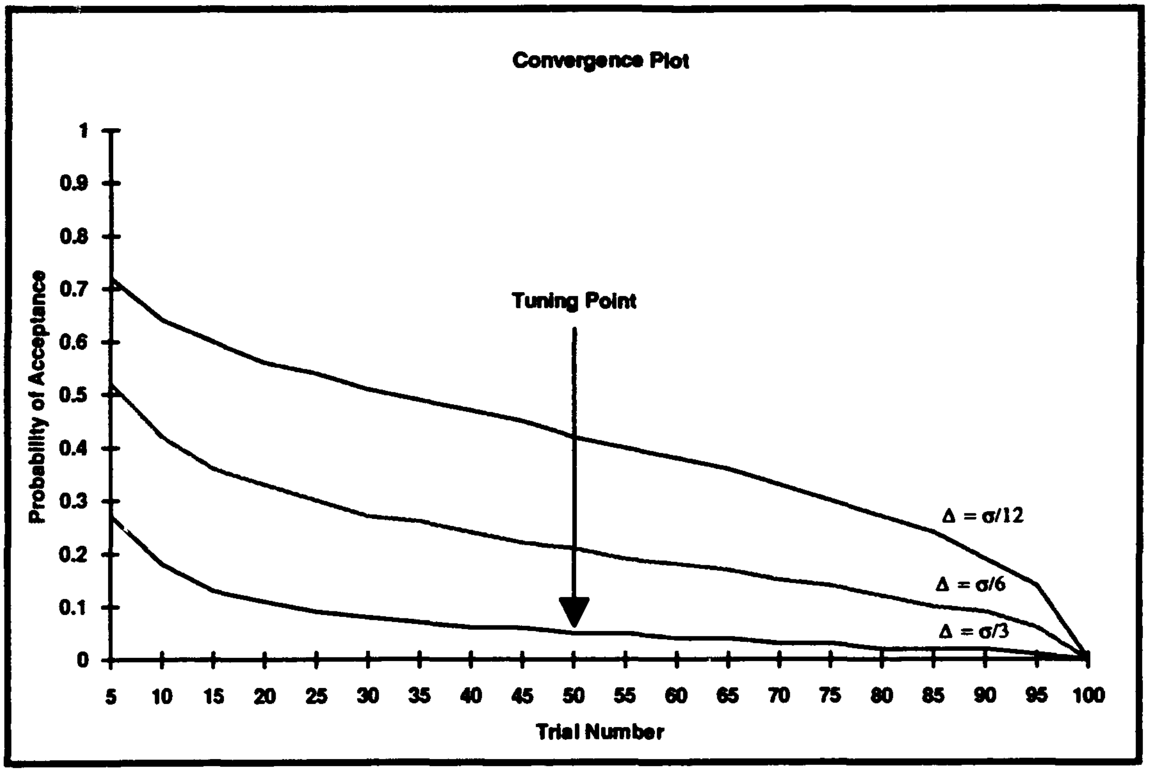


FIGURE 3.16 ELLIPTIC COEFFICIENT CONVERGENCE PLOT

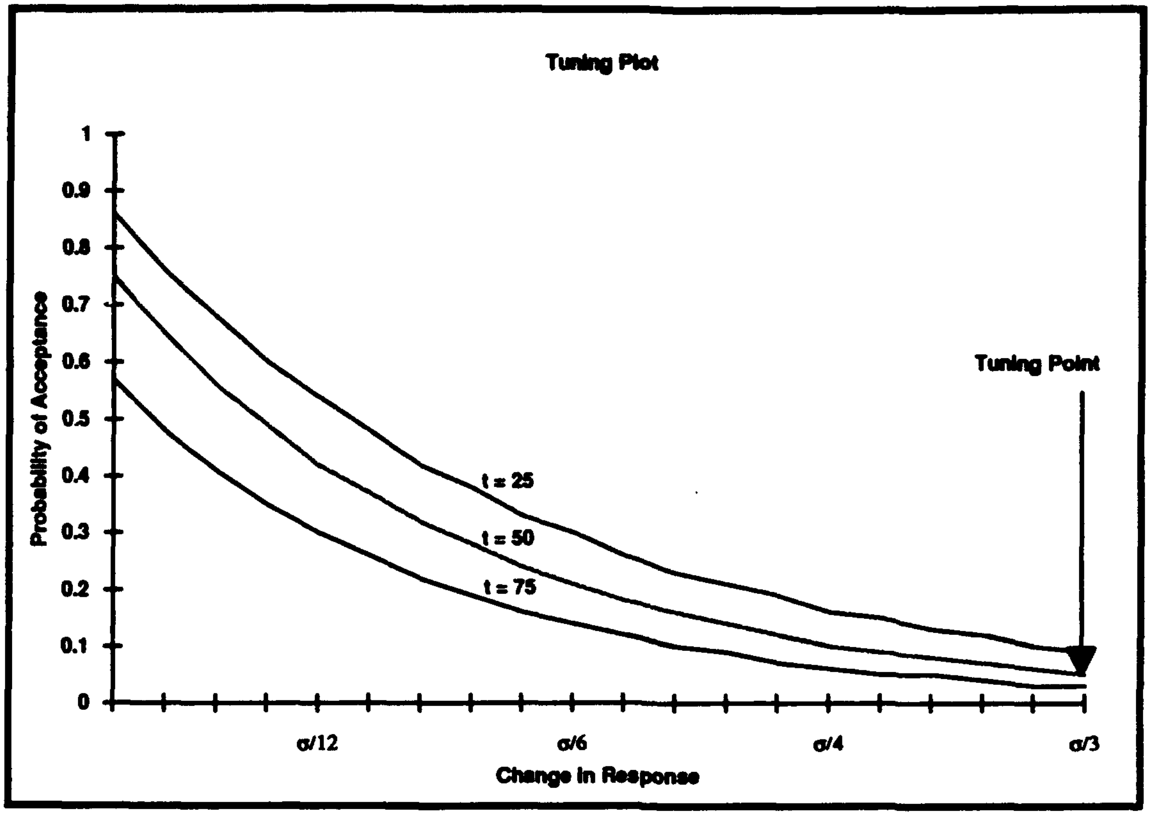


FIGURE 3.17 ELLIPTIC COEFFICIENT TUNING PLOT

The differing characteristics apparent in the convergence and tuning plots may result in differing performance in solution quality and efficiency. Those characteristics that result in better performance may support the development of superior acceptance functions.

3.4. The Comparisons.

As discussed in Section 2.1, the comparison of heuristic algorithms is made for a given problem based on their average performance. The performanc: of the alternative Simulated Annealing algorithms is measured using the average solution quality obtained over two fixed anne:ling run lengths. The two best candidates for each model are chosen and their performance is compared against Local Search. This two-stage comparison is explained in greater detail.

3.4.1. The Simulated Annealing Alternatives Compared.

The seven alternative acceptance functions are compared for the first and third models and a subset of the seven are compared for the second model. In every case, the solutions obtained from each alternative over a set of (*m*) starting locations is used to compute the average solution quality. The set of starting locations is the same for each alternative.

The average solution quality for a given alternative tuned for a set run length is calculated as follows. First, a base solution is determined (y_b). When the value for the global optimum is known, it is used for the base value. When the global optimum is not known, the best estimate is used. Next, the performance of each solution (y_i) is measured by taking its difference from the base value:

$$\partial_i = |y_b - y_i| \quad (3.15)$$

These values are used to calculate a measure of performance that relates the differences to the base value. This measure of performance is called the *mean percent difference*, given by:

$$\mathrm{MPD} = (100/\mathrm{m})\sum_{i=1}^{m} \partial_i / y_b \quad (3.16)$$

This relative measure provides an intuitive feel for an algorithm's general performance (21:47). The measure converts the raw difference in value to a more user-friendly percentage. It seems more reasonable to say that an algorithm has an average 5% difference than to simply state that the average difference is 183 (21:47). For this reason the MPD will be used to measure solution quality among the alternative acceptance functions.

3.4.2. The Best Alternatives Chosen.

Once the MPD for every alternative is measured, the two best alternatives can be chosen. This is not so straightforward as it seems. Each alternative is tested at two different annealing run lengths: m_1 annealing runs at $n_1 = 100$ trials, and m_2 runs at $n_2 = 200$ trials. The longer runs provide better solutions when compared head-to-head. However, the algorithm could be run twice at 100 trials for every run at 200 trials. Therefore, unless the solution quality improves dramatically at the longer run length, the shorter run is preferable. The two best alternatives at the selected annealing run length are chosen to compare against Local Search.

3.4.3. Local Search Compared.

Unlike the first comparison, the comparison of the chosen Simulated Annealing alternatives against Local Search is not based on the run length. Instead, the efficiency of each of the competing algorithms is determined. This is done by measuring the average CPU time required to converge on a solution. First, a set of starting locations is determined. Then the time to converge on a solution for each starting location is measured. Finally, the simple mean is calculated.

Local Search converges quickly. Simulated Annealing converges more slowly. Therefore, the efficiency for a single run of Local Search is much better than for a single run of Simulated Annealing. This does not reflect the solution quality. In order to determine which algorithm performs the best, the efficiencies must be equalized. For instance, Local Search may take 200 seconds for each run while both of the Simulated Annealing algorithms take 400 seconds for each run. The efficiency of the algorithms are equalized by letting the Local Search algorithm run twice for every run of Simulated Annealing. The solution quality for Local Search is obtained by taking the best out of every two solutions to compute each difference in solution quality (δ_i). Otherwise, the MPD is calculated in the same way as before. The algorithm with the best solution quality, adjusted for efficiency, is the more desirable algorithm. Sample calculations for solution quality and efficiency are given at the end of Appendix A.

IV. Results

Each of the alternative acceptance functions is applied against three models. The first model, the timing of traffic lights, facilitates the development of the SLAM implementation. The results from this model are of minor interest because the model contains only two decision variables. The results do, however, show the expected behavior of Simulated Annealing: the solution quality dominates Local Search and improves with longer annealing runs. This achieves the first goal of the research, to demonstrate the viability of using Simulated Annealing to optimize a simulation model.

The second model, the configuration of machines, provides greater insight to the advantages and limitations of Simulated Annealing. This model is implemented in two variations: as an open-queuing network, and as a closed-queuing network. The first variation provides little additional information about the relative merits of the alternative acceptance tests. It does show a more pronounced improvement with longer annealing runs. This lends credibility to the idea that one long annealing run gives a better solution than the best solution from multiple iterations of Local Search.

The second variation allows the optimal configuration to be predicted based on the theory of closed queuing networks. The results from this variation are completely unexpected. Two of the alternatives are clearly superior than any of the others. These results suggest that an even better alternative might be developed.

The results are given for each of the models. The key elements to consider are the determination of a base value used to calculate the solution quality, the selection of the two best Simulated Annealing alternatives, and the comparison to Local Search. Sample calculations for solution quality and efficiency are given at the end of Appendix A.

4.1. The Timing of Traffic Lights Model

Recall that the timing of traffic lights model uses two decision variables. Each decision variable represents the amount of time that a traffic light remains green. The response value measures the average waiting time for cars arriving at a road-repair location. The traffic lights control the flow of traffic through the section of road under repair.

Each alternative acceptance function is tried several times using this model. In order to compare the alternatives on the same basis, the annealing run length is fixed for either 100 trials or 200 trials. Then each alternative is tried 25 times and the average performance measured. Two settings for the number of trials show how the performance of Simulated Annealing improves with increasing run lengths. Comparing the average solution quality shows that all of the alternatives perform well.

4.1.1. Establishing the Base Response Value.

The average solution quality is the discriminator between the alternative acceptance tests. In order to establish the difference between the solutions obtained and the base value, the base value had to be found. For this problem, the base value is the global optimum. Since the configuration space is reasonably small, the global optimum is found by complete enumeration of the response values. These values are obtained by taking 100 batches for each configuration. Every batch contains 50 observations of the waiting time. These values are recorded in a grid with the time for light 1 on the horizontal axis and the time for light 2 on the vertical axis. The result is a map of the responses, showing many local optima and several global optima. The values are rounded down to the nearest integer value in order to enhance readability. Appendix A contains this response map.

Since the smallest value obtained is 76 seconds, this value is assumed to be the optimal response and the corresponding configurations are assumed to be multiple optimal configurations. The observed difference in solution quality is established by using this value and subtracting each predicted value from it. The observed values are averaged using Equation 3.16.

4.1.2. Comparing the Alternatives.

After establishing the optimum response value and generating the solutions, the performance is measured. Appendix A contains the solutions obtained and the differences from the base value used to calculate the average solution quality. The results of applying the alternatives against the timing of traffic lights model are summarized in Table 4.1. The Simulated Annealing alternatives all found solutions within about 4% difference given 100 trials and within about 3% difference given 200 trials.

Alternative	MPD at 100 Trials using 25 Starting Locations	MPD at 200 Trials using 25 Starting Locations
Geometric Temperature	4.17	3.32
Linear Temperature	4.13	2.99
Adaptive Temperature	4.19	3.04
Elliptic Temperature	4.38	3.04
Logarithmic Temperature	4.07	2.93
with Linear Coefficient	4.79	3.18
with Elliptic Coefficient	3.89	3.17

TABLE 4.1 SOLUTION QUALITY

This demonstrates the expected behavior of Simulated Annealing: that longer runs give better results. Note that, in this case, the additional trials only improve the solution by about 1%. For this problem, it appears that the shorter annealing runs are preferable since two iterations can be performed for each of the longer runs.

4.1.3. Choosing the Best Alternatives.

The two best alternatives at the shorter run length both use a logarithmic temperature function. The best alternative uses an elliptic coefficient function to force convergence with an average 3.89 percent difference. The next best alternative, Logarithmic Temperature, does not force convergence. It achieved a 4.07 percent difference. Note that Logarithmic Temperature is the dominant alternative at the longer run length with an average 2.93 percent difference. The convergence and tuning plots for the Elliptic Coefficient approximate the shape of Logarithmic Temperature's plots more closely than Linear Coefficient. These observations indicate that tuning alone is sufficient to achieve good results. The forced convergence methods do not appear to have a significant affect on performance in this problem.

4.1.4. Comparing with Local Search.

The comparison of the two best Simulated Annealing alternatives against Local Search is made in two steps. The first step is to measure the efficiency of each algorithm. The measure is the average CPU time required by an algorithm to converge on a solution. Since Local Search converges quickly, multiple runs can be made for every run of Simulated Annealing. Table 4.2 shows the results.

Local Search is applied seven times for every iteration of Simulated Annealing with a combined CPU time of 179.80 seconds on average. The two Simulated Annealing alternatives range between 168.10 seconds on average and 185.10 seconds on average. Note that the average solution quality obtained shows that both Simulated Annealing alternatives dominate Local Search. Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient appears to be the most desirable algorithm.

Algorithm	Starting Locations	Efficiency Mean CPU Time	Solution Quality Mean % Difference
Local Search (1 Iteration)	100	26.38	11.23
Local Search (7 Iterations)	70 (10 Solutions)	179.80	4.92
Simulated Annealing with Logarithmic Temperature (1 Iteration)	10	185.10	3.58
Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient (1 Iteration)	10	168.10	3.47

TABLE 4.2 COMPARISON WITH LOCAL SEARCH

4.2. The Configuration of Machines Model -- Open Queuing Network

This model is an open queuing network. Recall that the model contains six machine stations. Each station consists of several machines. The number of machines at a given station is a decision variable. Hence, there are six decision variables. The objective is to allocate 25 machines among the six stations so as to minimize the average processing time for parts.

4.2.1. Establishing the Base Response Value.

The configuration space generated by the six decision variables is too large to completely enumerate the response values. There is no theoretical basis at present for determining what the optimum configuration should be. The only alternative left is to sort through all of the solutions obtained and pick the minimal value. Appendix B contains the data collected from the experiment. The lowest value, found throughout the data, is 1.48 seconds and this is used in calculating the solution quality.

4.2.2 Comparing the Alternatives.

The open-queuing model is implemented using a random trial generator for the Simulated Annealing alternatives. Only the alternatives shown in Table 4.2 are tried because of time constraints in performing the research.

Alternative	MPD at 100 Trials using 25 Starting Locations	MPD at 200 Trials using 25 Starting Locations
Linear Temperature	103.70	14.27
Adaptive Temperature	323.95	29.14
Logarithmic Temperature with Linear Coefficient	131.32	20.70
with Elliptic Coefficient	136.89	18.57

TABLE 4.3 SOLUTION QUALITY

The Simulated Annealing alternatives have an average difference of about 130% given 100 trials, except for Adaptive Temperature. Adaptive Temperature performs significantly worse with an average difference over 300%. It does, however, perform better at 200 trials with an average difference of about 30%. Note that all of the alternatives improve the average difference 100% at the longer run. Therefore the longer run is preferable.

4.2.3. Choosing the Best Alternatives.

Linear Temperature dominates at the longer run with an average difference of 14.27. The next best alternative is the Logarithmic Temperature with Elliptic Coefficient. This alternative gives an average difference of 18.57. Since all of the alternatives use a convergence method, no conclusion about the usefulness of forced convergence can be made. It does appear, however, that the temperature function is a better way to force convergence than the coefficient function for this problem.

4.2.4. Comparing with Local Search.

Local Search is implemented using a structured trial generator. Five iterations of Local Search can be run for every iteration of Simulated Annealing, on average. The solution quality for one iteration of Local Search averages 82.48% in difference, but improves to 37.16% when the best solution out of every five iterations is used. The results are given below.

Algorithm	Starting Locations	Efficiency Mean CPU Time	Solution Quality Mean % Difference
Local Search (1 Iteration)	40	477.05	82.48
Local Search (5 Iterations)	40 (8 Solutions)	2385.25	37.16
Simulated Annealing with Linear Temperature (1 Iteration)	10	2320.90	16.49
Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient (1 Iteration)	10	2253.60	16.96

TABLE 4.4 COMPARISON WITH LOCAL SEARCH

Both of the Simulated Annealing alternatives dominate Local Search with five iterations. The solution quality improves about 20% using either Simulated Annealing algorithm. Both alternatives are also more efficient than Local Search with five iterations. The dominant algorithm appears to be Simulated Annealing with Linear Temperature. This data indicates that Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient is a close competitor.

4.3. The Configuration of Machines Model -- Closed Queuing Network

The closed queuing network is a variation of the configuration of machines model. Instead of letting the parts arrive exponentially and depart the system after processing, the closed network contains a fixed number of parts that never leave the system. There are 30 parts in the system. These parts cycle from station 1 through station 6, as before. Instead of leaving the system after station 6, the parts return to station 1 and the processing time is reset to zero.

4.3.1. Establishing the Base Response Value.

The optimum configuration is established using a PASCAL program written by Lt Col Dietz. Given the one-step probability transition matrix (which describes the flow of parts among the machine stations) and a configuration of machines, the program calculates the steady-state waiting time at each station. The optimal solution is found by permuting the configuration until the steady-state waiting time reaches a minimum. Applying the optimal configuration to the simulation model results in a total waiting time of 15.49 seconds. Both theoretical and simulation data agree on this optimal configuration. The PASCAL program, the results obtained from the program, and the data obtained from the simulation model are contained in Appendix C. The base value of 15.49 seconds is used to calculate solution quality.

4.3.2. Comparing the Alternatives.

The alternatives are implemented using a structured trial generator. The results from the closed queuing network differ from the results of the previous two models.

Simulated Annealing does not perform universally well. In fact, only the two alternatives based on Logarithmic Temperature with a coefficient function performed well.

Alternative	MPD at 100 Trials using 25 Starting Locations	MPD at 200 Trials using 10 Starting Locations
Geometric Temperature	167.28	185.33
Linear Temperature	90.91	126.62
Adaptive Temperature	186.12	131.43
Elliptic Temperature	169.72	132.54
Logarithmic Temperature	245.98	157.97
with Linear Coefficient	20.30	7.64
with Elliptic Coefficient	17.81	12.76

TABLE 4.5 SOLUTION QUALITY

Note that, except for the two alternatives mentioned, all of the alternatives performed poorly. This may be due to the tuning method used. The tuning method in Equation 3.3, used for the previous two models, is blindly applied to this model. In general, the alternatives using a forced convergence technique performed better than those using tuning alone. The results suggest, among forced convergence techniques, that a coefficient function dramatically outperforms a temperature function. Solution quality for these methods seems less sensitive to deviations in the tuning parameter, α. The solution quality improved about 10% at 200 trials in the average difference for the two best alternatives. Assuming that this is a significant improvement, the longer run is preferable to the shorter run.

4.3.3. Choosing the Best Alternatives.

The Logarithmic Temperature with Linear Coefficient performs best at 200 trials with an average difference of 7.64%. The Logarithmic Temperature with Elliptic Coefficient performs second best at 200 trials with an average difference of 12.7%. None of the other alternatives come close. The next best alternative is more than 100% worse on average.

4.3.4. Comparing with Local Search.

The comparison of efficiency shows that two to three iterations of Local Search can be run for every iteration of Simulated Annealing. Taking the pessimistic approach, the solution quality for Local Search with three iterations is used for comparison. The results are shown below:

Algorithm	Starting Locations	Efficiency Mean CPU Time	Solution Quality Mean % Difference
Local Search (1 Iteration)	40	454.20	34.23
Local Search (3 Iterations)	30 (10 Solutions)	1388.50	19.94
Simulated Annealing with Logarithmic Temperature and Linear Coefficient (1 Iteration)	10	1013.30	16.05
Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient (1 Iteration)	10	1137.80	12.12

TABLE 4.6 COMPARISON WITH LOCAL SEARCH

Local Search with three iterations produces a solution quality of 19.94%. Both of the Simulated Annealing alternatives dominate with solution qualities ranging between 12.12% and 16.05%. Both Simulated Annealing algorithms are also more efficient. The dominant algorithm, based on this data, is Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient.

4.3.5. Additional Alternatives.

The two best Simulated Annealing alternatives both use a coefficient function in conjunction with a Logarithmic Temperature. Among the temperature functions, the

Geometric Temperature and Linear Temperature performed best. Two more alternative acceptance functions are devised using these temperature functions in place of the Logarithmic Temperature. The results are given below.

Alternative	MPD at 200 Trials using 25 Starting Locations
Logarithmic Temperature with Linear Coefficient	7.64
Linear Temperature with Linear Coefficient	16.38
Geometric Temperature with Linear Coefficient	12.58

TABLE 4.7 ADDITIONAL ALTERNATIVES

Although Geometric Temperature and Linear Temperature outperform Logarithmic Temperature, the alternatives resulting from the Linear Coefficient function do not follow this same precedenı e. There are many other acceptance functions that may be tried. The selection of new alternatives can be guided by the results already obtained and some knowledge of the simulation model to be optimized.

4.4. Summary

The three models investigated demonstrate the utility of Simulated Annealing in optimizing simulation models. Simulated Annealing dominates Local Search and improves with increased run length. These are general qualitative remarks. The quantitative performance of Simulated Annealing depends on the model being used. For small problems, like the Timing of Traffic Lights with two decision variables, a short annealing run appears best. For large problems, like the Configuration of Machines with six decision variables, a long annealing run seems more appropriate. In some, cases proper tuning is sufficient to achieve good solutions. In other cases, the use of forced convergence helps find better solutions. Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient performs well on all of the models tested. Simulated Annealing works well for optimizing stochastic models, but the traditional acceptance functions are not always among the best alternatives.

V. Conclusions and Recommendations

The research presented shows the applicability of Simulated Annealing as an optimization technique for simulation models. The primary goal is to demonstrate how the technique could be used with an existing simulation language, namely SLAM. A secondary goal is to examine the affect that several alternative acceptance functions had on performance, measured by solution quality and efficiency. No attempt is made to justify the use of Simulated Annealing from a theoretical perspective. Rather, the empirical results from three test cases are used to infer the practical utility of the algorithm. The conclusions drawn as well as the recommendations for additional research follow from these test cases.

5.1 Conclusions

The goal of demonstrating a SLAM based implementation of Simulated Annealing is achieved as part of developing the three test cases. The SLAM implementation consists of three parts: the network model, the control statements, and the user-inserts (of which there were ten). A method for using generic programming structures and common variable names allows four of the same user-inserts to be used for all test cases. The remaining five user-inserts only need minor changes to adapt to a given model. The control statements are also developed generically. The result is a general purpose optimization scheme that allows both Local Search and Simulated Annealing to be implemented.

Experiments for each test case are conducted as follows:

1. Each of the optimization techniques was implemented.
2. The number of trial configurations was established.
3. The number of starting configurations was established (Every technique used the same set of starting configurations).
4. The best solution found in each instance was recorded.
5. The average performance of the technique was computed.

In all three cases, some variation of Simulated Annealing dominates Local Search in solution quality and efficiency. In the timing of traffic lights, all variations of Simulated Annealing dominate Local Search. In the configuration of machines with open queuing, all but one variation of Simulated Annealing dominate Local Search. In the configuration of machines with closed queuing, only two of the seven Simulated Annealing variations dominate Local Search. These results indicate that Simulated Annealing may have some practical use for optimizing constrained simulation models.

There are two major concerns with applying Simulated Annealing to simulation models: how to cope with random noise in the estimate, and how to obtain good results within a predetermined amount of time. Seven different acceptance functions are devised, using the concepts of tuning and forced convergence, to answer these concerns. All of the alternatives are tuned the same way using Equation 3.3. The tuning parameter in each acceptance function is set to a predetermined value. A specified difference in the response values is accepted with 5% probability at the mid-way point through the trials. That specified difference (Δ) is keyed to the variance in response estimates (σ^2) by the relationship: $\Delta = \sigma/3$. While the tuning method is fixed, the means for forcing the convergence of the acceptance probabilities is not: two of the alternatives use a standard acceptance function; three alternatives use different temperature functions to force convergence; and two alternatives use a coefficient function to cause faster convergence in an otherwise standard acceptance function. The three test cases demonstrate how these different acceptance functions affect performance.

The first test case is a simple model involving two decision variables. The number of trial configurations used to establish a solution is a relatively large portion of the entire configuration space. The precaution of sampling each response with multiple batches reduces the affect of random noise. As a result, the difference in most neighboring responses is larger than the error in estimates. Local Search obtains solutions which are 5% above the optimum on average while Simulated Annealing obtains solutions 3.5% above the optimum on average. In this case, adequate results are obtained in shorter time spans (100 trials).

The second test case involves six decision variables. The number of trial configurations used to establish a solution is relatively small with respect to the configuration space. Because this test case models an open queuing system, the average number of parts in the system depends on the configuration. As a result, most neighboring responses differ by more than the error in their estimates. Local Search with one iteration obtains solutions 82% above the optimum. Simulated Annealing with 100 trials obtains solutions more than 100% above the optimum. Local Search with five iterations obtains solutions 37% above the optimum on average. Simulated Annealing tuned for 200 trials obtains solutions 17% above the optimum on average. The results imply that Simulated Annealing does not compare favorably against Local Search in shorter search times when the model involves many decision variables, but clearly dominates with longer search times available.

The third test case involves the same six decision variables, but models the system as a closed queuing network. There are always thirty parts in the system regardless of the configuration used. As a result, the response difference between many neighboring configurations becomes so small that the error in their estimates is large in comparison. This explains the unusual results obtained. Local Search obtains solutions 20% above the optimum on average and the two best Simulated Annealing alternatives obtain between 12% and 16% above the optimum on average. The remaining five Simulated Annealing alternatives obtain solutions 160% above the optimum on average in 100 trials (130% in 200 trials). The only difference between the two best alternatives and the five worst alternatives is the method used to force convergence. All five of the worst alternatives use a temperature function while both of the best alternatives use a coefficient function. The coefficient function accepts fewer challengers with response values close to the incumbent's response. This implies that the poor alternatives are accepting too many small increases in the response value. To confirm this pattern, two more alternatives are tried that used coefficient functions. The results were similar. Therefore, Simulated Annealing alternatives that use a coefficient function are less sensitive to errors in tuning. The coefficient function appears to dominate the bias introduced by the temperature function.

Simulated Annealing can be applied to a simulation model with good results. The SLAM language can be used tu implement the algorithm. The selection of an acceptance function and the determination of an adequate search times remain problem dependent. In general, an acceptance function with both a logarithmic temperature function and an elliptic coefficient function appears to be a consistently good alternative.

5.2 Recommendations

To date, there have been few articles written about the application of Simulated Annealing to simulation models. Many of the results obtained from the application of Simulated Annealing to deterministic functions may or may not be applicable. In the case of the present study, many elements were fixed that might have been better implemented in another way. One of these areas provides the subject matter for future research:

1. What are the affects of different permutation methods on solution quality?

Many of the questions raised in the conduct of this research might also generate enough interest to merit further study:

2. How do different batching algorithms impact efficiency?
3. How does high estimate variance affect performance?
4. How does the number of trials needed to obtain good results increase as the number of decision variables increase?

Another recommendation concerns the application of Simulated Annealing to simulation models in an operational context. Imagine that an actual system performs under a given set of constraints, but those constraints vary as machines break down or get replaced. The optimal configuration of resources changes on a dynamic basis. Now imagine that an operator has access to a simulation model conformant with the new constraints and can use an optimization technique to find a good allocation of those resources. What capabilities would the analyst want to have? These questions may have the most significance:

5. Given a specified time constraint, which technique should be used?
6. Can the performance of Simulated Annealing be improved interactively, where the analyst varies the temperature at will?
7. Can Simulated Annealing be made artificially intelligent?

Most of all, it is hoped that the present research will help future researchers to find good solutions to real problems.

Appendix A: The Timing of Traffic Lights

A.1 Scenario

The scenario definition links the pieces together by name. It lists the filename for itself as SCE1, the control statements as CONT, the network model as NET1, and each of the user inserts. Filename extensions are omitted since they must conform to a the standard convention. The scenario definition used in the Job-Shop problem follows the same format.

Although the scenario definition does not specify the contents of each module, it does require that each named module exist. The SLAM executive coordinates the instructions within each module to maintain consistency. If a module is improperly named, is inconsistent with syntax requirements, or exceeds the program's limitations, then the executive will generate an error message. The scenario will not run correctly if the modules fail to meet these guidelines.

The control statements are identical for all three simulation models. The statements themselves specify initialization conditions for each simulation run as well as executive constraints used throughout the program's execution. These control statements support three main ideas:

1. To maintain program flow and syntax,
2. To transfer executive control to the user inserts, and
3. To achieve a common basis of comparison for each trial configuration.

The GEN statement constrains the program's execution to 100,000 simulation runs and suppresses the generation of unneeded output reports. These constraints allow the external subroutines to generate the required number of simulation runs and report the results in a desired format without conflicting with the SLAM executive.

The LIMITS statement identifies the number of files required to store *entities* (6), the number of attributes needed to describe each entity (2), and the maximum number of entities allowed within the model at any given time (500). An entity is any object which can enter a model, can change the model, can be changed by the model, or can exit the model (25:64). These settings minimize the memory requirements of each entity while maximizing the capacity for entities within the model.

The STAT statement allocates a file for the collection of statistics based on observations recorded in an external subroutine. This enables the external subroutines to do two things: 1) to collect the desired observations needed to estimate the response, and 2) to use the statistical tools embedded in SLAM.

The SEEDS statement causes the pseudo-random number streams to be reinitialized for each simulation run. This reduces the variability between the incumbent and challenger responses. As a result, each simulation run uses common random numbers. The acceptance test can more accurately compare responses because each configuration faces a common set of experimental conditions (25:745).

The ARRAY statements make memory available to all external subroutines and ensure that memory does not get reinitialized between simulation runs. The external subroutines use the memory space to record solutions and to maintain control parameters needed by the algorithm.

The NETWORK, INITIALIZE, and FIN statements are required by the SLAM executive to maintain program flow and syntax. The FIN statement terminates the list of control statements. The INITIALIZE statement constrains each simulation run to 1,000,000; more time than is actually needed. This allows the user inserts to determine the end of each run. The NETWORK statement transfers control to network model.

The network model consists of four segments: the resource and gate definitions, the control of traffic at light 1, the control of traffic at light 2, and the control of the cycle. The resources, START1 and START2, model the time-lag between stopped cars. The gates. LIGHT1 and LIGHT2, model the traffic lights.

The control of traffic models the arrival of cars with an exponential distribution. The use of the exponential distribution implies that the arrival of cars is "memoryless": the arrival a car neither depends on the arrival of the previous car nor affects the arrival of the next car (27:203). On average, though, the cars arrive nine seconds apart at light 1 and twelve seconds apart at light 2.

Examine the arrival of cars at light 1. The traffic waits for resource START1 and proceeds across the stretch of road when gate LIGHT1 is open (the light is green). If the car was stopped, it takes two more seconds to get started and up to the light. If the car was already moving, it proceeds directly across the stretch of road. Statistics are collected for waiting time at *LIGHT1 using a COLCT statement. Statistics are collected for the* overall waiting time using the EVENT statement which links to a user insert. The observations collected using EVENT are used to calculate the response value. The control of traffic at light 2 works in an identical manner.

The timing of the lights uses a single entity to cycle through each of the four stages. At first both gates are closed (both lights are red). Next gate LIGHT1 is opened (green) for time XX(4) and then closed for 55 seconds. Finally gate LIGHT2 is opened for XX(5) seconds and the closed for 55 seconds. The cycle repeats until the simulation run ends. An entity placed into the network by a user insert immediately terminates the run.

A.1.1 Scenario Definition.

```
Scenario:
SCE1
Control:
CONT
Network:
NET1
Script:
Facility:
User Insert:
EVENT1
FIRST1
INTLC
NEXT1
OTPUT
OUT1
PUT1
RAN
TEST1
TIME
Notes:
Data:
Curchange:
00000000
Definition:
```

A.1.2 Control Statements.

```
GEN,WARRENDER,TRAFFIC,7/20/93,1000,N,NO,Y/N,NO,N/1,72;
LIMITS,4,3,550;
STAT,3,WAIT TIME;
SEEDS,9375295(1)/Y;
ARRAY(1,21);
ARRAY(2,21);
ARRAY(3,21);
NETWORK;
INITIALIZE,,100000,Y;
FIN;
```

A.1.3 Network Model.

```
RESOURCE/START1,1/START2,3;
           GATES/LIGHT1,CLOSE,2/LIGHT2,CLOSE,4;
;TRAFFIC FROM DIRECTION 1
           CREATE,EXPON(9,1),,1;
           AWAIT(1),START1;
           AWAIT(2),LIGHT1;
           COLCT(2),INT(1),WAIT TIME 1;
           EVENT,1,1;
           ACT,2,TNOW.GT.ATRIB(1);
           ACT;
           FREE,START1;
           TERM;
;TRAFFIC FROM DIRECTION 2
           CREATE,EXPON(12,1),,1;
           AWAIT(3),START2;
      AWAIT(4),LIGHT2;
           COLCT(3),INT(1),WAIT TIME 2;
           EVENT,1,1;
           ACT,2,TNOW.GT.ATRIB(1);
           ACT;
           FREE,START2;
           TERM;
;TRAFFIC LIGHTS
           CREATE,,,,1;
           ACT,55;
LOOP       OPEN,LIGHT1;
           ACT,XX(4);
           CLOSE,LIGHT1;
           ACT,55;
           OPEN,LIGHT2;
           ACT,XX(5);
           CLOSE,LIGHT2;
           ACT,55,,LOOP;
           ENTER,1,1;
           TERM,1;
      END;
```

A.1.4 User Inserts.

A.1.4.1 Subroutine INTLC.

```
$INCLUDE:'PRCTL.FOR'
C******************************************************
C**   ANNEALING ROUTINE CONTROL                      **
C******************************************************
      SUBROUTINE INTLC
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      SELECT CASE (II)
C******************************************************
C**   NEIGHBORING SOLUTIONS                          **
C******************************************************
      CASE (2, 12, 22, 32, 42, 52, 62, 72, 82, 92)
           CALL NEXT
C******************************************************
C**   NEW INITIAL SOLUTIONS                          **
C******************************************************
           CASE (1, 11, 21, 31, 41, 51, 61, 71, 81, 91)
                II = II + 1
           CALL FIRST
C******************************************************
C**   LOCAL SEARCH INITIAL SOLUTION (N=100, M=25)    **
C******************************************************
           CASE (0)
           II =  2
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   0.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C*GEOMETRIC TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (10)
           II = 12
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   1.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C******************************************************
C**  LINEAR TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (20)
           II = 22
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   2.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ADAPTIVE TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (30)
           II = 32
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   3.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (40)
           II = 42
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   4.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
CLOGARITHMIC TEMPERATURE INITIAL SOLUTION (N=100, M=25)
C******************************************************
           CASE (50)
           II = 52
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   5.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C******************************************************
C** LINEAR COEFFICIENT INITIAL SOLUTION (N=100, M=25)**
C******************************************************
           CASE (60)
           II = 62
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   6.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC COEFFICIENT INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (70)
           II = 72
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   7.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**   LOCAL SEARCH INITIAL SOLUTION (N=200, M=25)    **
C******************************************************
           CASE (80)
           II = 82
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   8.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C*GEOMETRIC TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (90)
           II = 92
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   9.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C*****************************************************
C** LINEAR TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C*****************************************************
           CASE (100)
           II = 102
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5, 102.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ADAPTIVE TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (110)
           II = 112
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  11.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (120)
           II = 122
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  12.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
CLOGARITHMIC TEMPERATURE INITIAL SOLUTION (N=200, M=25)
C******************************************************
           CASE (130)
           II = 132
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  13.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
```

```
C******************************************************
C** LINEAR COEFFICIENT INITIAL SOLUTION (N=200, M=25)**
C******************************************************
           CASE (140)
           II = 142
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  14.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC COEFFICIENT INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (150)
           II = 152
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  15.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
C******************************************************
    END SELECT
    WRITE(*,10) INT(II/10+1.0), INT(GETARY(3,4)+1.0),
   + INT(GETARY(3, 2) + 1.0)
10  FORMAT(3I5)
    RETURN
    END
```

A.1.4.2 Subroutine FIRST.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE FIRST
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C******************************************************
C**     STARTING GREEN LIGHT TIMES,                  **
C**     XX(4) = LIGHT 1, XX(5) = LIGHT 2             **
C******************************************************
 CALL RAN(6,R1)
 CALL RAN(6,R2)
 XX(4) =  50.0 + REAL(INT(R1*40.0))
 XX(5) =  40.0 + REAL(INT(R2*40.0))
```

```
C******************************************************
C**    INITIALIZE XX(1) = SAMPLES AND XX(2) = BATCHES**
C******************************************************
 XX(1) =   0.0
 XX(2) =   0.0
 XX(3) =   0.0
C******************************************************
C**  INITIALIZE INCUMBENT AND SOLUTION VALUES        **
C******************************************************
 CALL PUTARY( 1, 1, 99999.9)
 CALL PUTARY( 1, 2,     0.0)
 CALL PUTARY( 1, 3,    30.0)
 CALL PUTARY( 2, 1, 99999.9)
 CALL PUTARY( 2, 2,     0.0)
 CALL PUTARY( 2, 3,    30.0)
C******************************************************
C**  INITIALIZE ARRAY(3,1) = TOTAL RUNS              **
C**  ARRAY(3,2) = CURRENT RUN                        **
C******************************************************
 RT = 100.0 + 100.0 * INT(GETARY(3, 5)/8)
 CALL PUTARY( 3, 1,   RT)
 CALL PUTARY( 3, 2,  0.0)
 CALL PUTARY( 3, 9,  0.0)
 CALL PUTARY( 3,10,  0.0)
 CALL PUTARY( 3,11,  0.0)
 CALL PUTARY( 3,12,  0.0)
 CALL SETTIM( 0, 0, 0, 0)
C******************************************************
 RETURN
 END
```

A.1.4.3 Subroutine NEXT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE NEXT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C****************************************************
C**  INITIALIZE INCUMBENT SOLUTION                 **
C**  AND SELECT A GUIDED DIRECTION                 **
C****************************************************
      XX(4) = GETARY(1,4)
      XX(5) = GETARY(1,5)
      R = GETARY(3,2) - GETARY(3,10) - GETARY(3,11)
      I = INT(MOD(R, 12.0))
```

```
SELECT CASE (I)
 CASE ( 0)
   XX(5) = XX(5) + 1.0
 CASE ( 1)
   XX(5) = XX(5) + 2.0
 CASE ( 2)
   XX(4) = XX(4) + 1.0
   XX(5) = XX(5) + 1.0
 CASE ( 3)
   XX(4) = XX(4) + 1.0
 CASE ( 4)
   XX(4) = XX(4) + 2.0
 CASE ( 5)
   XX(4) = XX(4) + 1.0
   XX(5) = XX(5) - 1.0
 CASE ( 6)
   XX(5) = XX(5) - 1.0
 CASE ( 7)
   XX(5) = XX(5) - 2.0
 CASE ( 8)
   XX(4) = XX(4) - 1.0
   XX(5) = XX(5) - 1.0
 CASE ( 9)
   XX(4) = XX(4) - 1.0
 CASE (10)
   XX(4) = XX(4) - 2.0
 CASE (11)
   XX(4) = XX(4) - 1.0
   XX(5) = XX(5) + 1.0
END SELECT
RETURN
END
```

A.1.4.4 Subroutine RAN.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE RAN(IS,R)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL Z, R, C, G
      C = 4.294967296E+9
      Z = GETARY(3, IS)
      G = 5*Z + 99991
      Z = MOD(G,C)
      R = Z/C
      CALL PUTARY(3, IS, Z)
      RETURN
      END
```

A.1.4.5 Subroutine EVENT.

```
$INCLUDE:'PRCTL.FOR'
C
      SUBROUTINE EVENT(I)
C
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C
      DIMENSION A(10)
      REAL BATCHES, PROBABILITY
C*****************************************************
C**  COLLECT XX(1) SAMPLES AND TOTAL WAIT XX(2)     **
C*****************************************************
      IF (TNOW.LT.500.0) RETURN
      XX(1)   = XX(1) + 1
      XX(2)   = XX(2) + TNOW - ATRIB(1)
C*****************************************************
C**   RECORD BATCH MEAN                             **
C*****************************************************
      IF (XX(1).LT.50) RETURN
      WAIT    = XX(2)/50
      XX(1)   = 0.0
      XX(2)   = 0.0
      CALL COLCT(WAIT,1)
      BATCHES = CCNUM(1) - 10.0
C*****************************************************
C**  STOP COLLECTING BATCHES WHEN CRITERIA MET:     **
C** LESS THAN 100% CHANCE OF ACCEPTANCE OR 30 BATCHES*
C*****************************************************
      IF (BATCHES.LT.0) RETURN
      CALL TEST(PROBABILITY)
      TERM = PROBABILITY*(1 - BATCHES/31.25)
      IF (TERM.LE.0.20) THEN
        CALL ENTER(1,A)
        RETURN
      END IF
      RETURN
      END
```

A.1.4.6 Subroutine TEST.

```
$INCLUDE:'PRCTL.FOR'
C
      SUBROUTINE TEST(PT)
C
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C
      REAL RT,RC,T,Y1,S1,N1,Y2,S2,N2,SP,DY,L,PT
C****************************************************
C**  DELTA FUNCTION                                **
C****************************************************
      Y1 = GETARY(1,1)
      Y2 = CCAVG(1)
      DY = (Y2-Y1)
C****************************************************
C**   CALCULATE PT; THE PROBABILITY OF ACCEPTANCE    **
C****************************************************
      IF (DY.LT.0.0) THEN
        PT = 1.0
        RETURN
      END IF
      IT = INT(GETARY(3, 5))
C****************************************************
C**   ACCEPTANCE FUNCTION                          **
C****************************************************
      SELECT CASE(IT)
      CASE (0, 8)
        PT = 0.0
        RETURN
C****************************************************
      CASE (1)
        RC = GETARY(3, 2)
        T  = 1.60*0.95**(INT(RC/10.0)-1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (9)
        RC = GETARY(3, 2)
        T  = 2.08*0.95**(INT(RC/10.0)-1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (2, 10)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.67 * (RT - RC) / RT
        PT = EXP(-DY/T)
        RETURN
      CASE (3, 11)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        S1 = GETARY(1,2) + 0.0001
        N1 = GETARY(1,3)
        S2 = CCSTD(1) + 0.0001
        N2 = CCNUM(1)
        SP = SQRT(((N1-1)*S1**2+(N2-1)*S2**2)/(N1+N2-2))
        T  = 0.22 * SP * (RT - RC) / RT
        PT = EXP(-DY/T)
        RETURN
      CASE (4, 12)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 1.54 * SQRT(1 - (RC/RT)**2)
        PT = EXP(-DY/T)
      RETURN
C*****************************************************
      CASE (5)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 5.24/LOG(RC + 1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (6)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 6.84/LOG(RC + 1.0)
        L  = (RT - RC)/RT
        PT = L*EXP(-DY/T)
        RETURN
      CASE(7)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 5.52/LOG(RC + 1.0)
        L  = SQRT(1 - RC**2/RT**2)
        PT = L*EXP(-DY/T)
        RETURN
      CASE (13)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 6.15/LOG(RC + 1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (14)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 8.04/LOG(RC + 1.0)
        L  = (RT - RC)/RT
        PT = L*EXP(-DY/T)
        RETURN
      CASE(15)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 6.48/LOG(RC + 1.0)
        L  = SQRT(1 - RC**2/RT**2)
        PT = L*EXP(-DY/T)
        RETURN
C****************************************************
      END SELECT
      END
```

A.1.4.7 Subroutine OTPUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE OTPUT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL YC, YI, YB, RT, RC, PT, PC, SC, ST
C*****************************************************
C**   INCREMENT THE NUMBER OF RUNS                 **
C**   RETRIEVE YC AND YI SOLUTIONS                 **
C*****************************************************
      RT = GETARY(3, 1)
      RC = GETARY(3, 2)
      CALL TEST(PT)
      RC = RC + 1.0
      CALL PUTARY(3, 2, RC)
      CALL RAN(8,PC)
      YC = CCAVG(1)
      YI = GETARY(1,  1)
      YB = GETARY(2,  1)
      A0 = GETARY(3,  9)
      A1 = GETARY(3, 10)
      A2 = GETARY(3, 11)
      A3 = GETARY(3, 12)
C*****************************************************
C**  YC IS LESS THAN PREVIOUS YB SOLUTION          **
C*****************************************************
      IF (YC.LT.YB) THEN
       A0 = A0 + 1.0
       A1 = A1 + 1.0
       CALL PUTARY(3,  9, A0)
       CALL PUTARY(3, 10, A1)
       CALL PUT(1)
       CALL PUT(2)
       GOTO 100
      END IF
C*****************************************************
C**  YC IS LESS THAN YI SOLUTION                    **
C*****************************************************
      IF (PT.EQ.1) THEN
       A0 = A0 + 1.0
       A2 = A2 + 1.0
       CALL PUTARY(3, 9, A0)
       CALL PUTARY(3, 11, A2)
       CALL PUT(1)
       GOTO 100
      END IF
C*****************************************************
C**DETERMINE ACCEPTANCE OF A MOVE AWAY FROM OPTIMUM **
C**COMPARE THE PC AGAINST PT OF ACCEPTANCE          **
C*****************************************************
      IF (PC.LT.PT) THEN
       A0 = A0 + 1.0
       A3 = A3 + 1.0
       CALL PUTARY(3,  9, A0)
       CALL PUTARY(3, 12, A3)
       CALL PUT(1)
      END IF
C****************************************************
C**  ANNEALING COMPLETE, START FROM NEW SOLUTION   **
C****************************************************
100   IF (RC.LT.RT) RETURN
      II = II - 1
      ST = GETARY(3, 3)
      SC = GETARY(3, 4) + 1.0
      CALL PUTARY(3, 4, SC)
      CALL OUT
C****************************************************
C**  TEST COMPLETE, START NEW TEST                 **
C****************************************************
      IF (SC.LT.ST) RETURN
      II = II + 9
C*****************************************************
C**   PROBLEM COMPLETE                              **
C*****************************************************
      IF   (II.LT.160)     RETURN
      WRITE(*,*) 'TYPE ENTER TO CONTINUE'
      READ (*,*)
      STOP
      END
```

A.1.4.8 Subroutine PUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE PUT(I1)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      CALL PUTARY(I1, 1, CCAVG(1))
      CALL PUTARY(I1, 2, CCSTD(1))
      CALL PUTARY(I1, 3, CCNUM(1))
      CALL PUTARY(I1, 4,    XX(4))
      CALL PUTARY(I1, 5,    XX(5))
      RETURN
      END
```

A.1.4.9 Subroutine OUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE OUT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C*****************************************************
      CALL TIME(RTIME)
      IT = INT(RTIME)
      IA = INT(II/10) + 1
      IS = INT(GETARY(3,  4))
      IR = INT(GETARY(3,  2))
      IM0 = INT(GETARY(3, 9))
      IM1 = INT(GETARY(3,10))
      IM2 = INT(GETARY(3,11))
      IM3 = INT(GETARY(3,12))
      OPEN(UNIT=8,ACCESS='APPEND',FILE='VAL1.DOC')
      OPEN(UNIT=9,ACCESS='APPEND',FILE='SOL1.DAT')
      OPEN(UNIT=10,ACCESS='APPEND',FILE ='REC1.DOC')
      WRITE(8, 10)   IA, ',' , IS , ',' ,
     + INT(GETARY(1,4)) , ',' , INT(GETARY(1,5))
10    FORMAT(I5, A1, I5, A1, I5, A1, I5, A1)
      WRITE(9, 20) IA , ',' , IS , ',' , IT , ',' ,
     + IR , ',' , IM0 , ',' , IM1 , ',' , IM2 , ',' ,
     + IM3 , ',' , GETARY(2,1) , ',' , GETARY(2,2) , ','
20    FORMAT(I5, A1, I5, A1, I5, A1,
     + I5, A1, I5, A1, I5, A1, I5, A1,
     + I5, A1, F8.2, A1, F8.2, A1)
      WRITE(10, *) IA, IS
      WRITE(10, *) GETARY(3, 6)
      WRITE(10, *) GETARY(3, 7)
      WRITE(10, *) GETARY(3, 8)
      CLOSE(UNIT = 8, STATUS = 'KEEP')
      CLOSE(UNIT = 9, STATUS = 'KEEP')
      CLOSE(UNIT =10, STATUS = 'KEEP')
      RETURN
      END
```

A.1.4.10 Subroutine TIME.

```
C*************************************************
C**   RETURNS SYSTEM TIME IN SECONDS            **
C*************************************************
      SUBROUTINE TIME(RTIME)
C
      REAL RTIME
      CALL GETTIM(IHR, IMIN, ISEC, I100TH)
      RTIME = 3600*REAL(IHR)+60*REAL(IMIN)+REAL(ISEC)+
     +     REAL(I100TH)/100.0
      RETURN
      END
```

A.2 Optimal Solution.

The following two pages contain the response map generated by completely enumerating the configurations and finding their responses. The responses were obtained for each configuration by taking the mean of 100 batch means. Each batch contained 50 samples. The global optimum was found by scanning the response map and taking the minimum value. Several global optima were found, as shown in bold font. The minimum value was found to be 76 seconds.

	80	999	893	972	336	415	112	119	103	103	96	96	91	91	88	89	87	87	86	86	84	83
	79	999	820	894	257	337	117	112	101	103	95	95	90	91	88	88	85	87	86	86	85	84
T	78	999	738	821	204	258	108	118	98	101	94	94	90	89	87	88	86	85	86	86	84	85
	77	999	660	738	178	204	110	107	102	97	92	93	88	90	85	87	86	86	84	86	84	84
R	76	999	579	660	149	179	104	109	96	102	92	92	88	88	87	85	84	85	84	84	85	84
	75	999	496	580	130	149	105	104	98	96	93	92	87	88	85	87	83	84	84	83	82	85
A	74	999	417	496	116	130	101	105	94	98	90	93	88	87	85	85	85	83	82	84	83	82
	73	991	332	417	113	115	100	101	96	93	91	90	87	87	84	85	83	85	82	83	84	83
F	72	916	251	332	111	113	99	100	92	96	87	91	86	87	85	84	83	83	84	82	81	83
	71	834	199	251	104	111	97	99	91	91	91	87	86	86	85	84	83	83	82	84	81	81
F	70	751	168	199	105	104	98	96	91	91	86	90	84	86	84	85	82	83	82	81	83	80
	69	671	149	169	99	105	92	98	88	91	86	85	87	83	83	84	84	82	82	82	81	83
I	68	586	123	149	102	99	94	92	90	87	85	86	83	87	82	83	82	84	81	82	82	81
	67	503	115	123	100	101	90	94	87	90	83	85	84	83	85	82	82	82	83	81	81	82
C	66	417	111	115	97	101	91	90	87	86	85	83	83	84	81	85	81	82	81	83	80	82
	65	330	107	111	98	96	90	91	85	87	83	85	81	83	83	81	84	81	81	81	82	80
	64	245	106	107	96	98	89	90	86	84	83	83	83	80	81	82	80	84	80	80	80	82
	.	195	101	105	94	95	89	89	84	86	82	83	81	83	79	81	82	80	83	80	80	81
L	62	170	101	101	93	94	88	89	84	84	82	81	81	80	82	79	80	81	79	83	80	79
	61	140	99	101	91	93	88	88	84	83	82	82	80	81	79	81	79	80	80	79	82	80
I	60	121	97	99	91	91	86	87	84	84	81	81	80	80	80	79	80	79	79	80	79	82
	59	114	98	96	88	90	85	86	84	84	82	81	80	80	79	80	78	80	78	79	80	78
G	58	107	96	98	87	88	85	85	83	84	82	82	79	80	79	79	79	78	80	78	78	80
	57	106	90	96	89	88	84	85	81	83	81	81	81	79	79	79	78	80	77	80	78	78
H	56	104	93	90	88	89	83	84	81	81	81	81	80	81	78	79	78	78	80	77	80	78
	55	100	92	93	84	88	85	83	81	82	79	81	80	79	80	78	78	78	78	79	76	80
T	54	102	88	92	86	84	83	85	80	81	80	80	80	80	79	80	77	78	78	78	80	77
	53	98	90	88	84	86	81	83	83	80	79	80	79	79	79	79	79	77	78	78	77	79
	52	95	88	90	83	84	82	81	80	83	79	79	79	79	79	79	78	79	77	78	78	77
	51	94	86	87	84	84	81	82	79	80	81	79	77	79	78	79	78	78	79	77	78	78
2	50	95	86	86	82	84	80	81	80	79	79	81	79	78	78	78	78	79	79	79	77	78
	49	92	87	87	80	82	81	80	79	79	78	79	80	79	77	78	78	78	78	79	79	77
	48	91	85	88	83	80	80	81	78	80	78	78	79	80	78	78	79	78	79	79	79	80
	47	92	83	85	84	83	77	80	79	78	78	78	77	79	79	79	77	78	78	78	79	79
T	46	90	86	84	82	83	81	77	79	79	78	79	79	78	79	80	79	78	79	79	79	81
	45	91	84	87	80	81	80	81	76	78	78	78	78	79	77	79	79	79	78	79	79	80
I	44	88	85	84	84	81	81	81	81	77	78	79	79	80	80	79	81	81	81	80	82	82
	43	85	82	85	81	83	79	81	80	81	76	78	79	79	80	80	79	81	81	81	80	82
M	42	87	82	84	83	83	83	81	80	81	82	78	80	81	82	83	83	82	85	85	85	84
	41	84	82	82	81	83	81	83	80	81	80	82	78	80	80	81	82	83	82	85	84	85
E	40	87	81	84	82	84	84	85	85	84	84	85	86	83	84	86	88	90	92	91	95	96
		50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70

TRAFFIC LIGHT 1 TIME

	80	83	85	83	85	83	83	86	82	85	82	84	84	85	82	84	84	83	82	87	81	86
	79	84	82	85	82	85	83	83	86	82	85	82	84	84	85	82	84	84	83	82	87	81
T	78	85	83	82	84	81	84	83	82	85	81	85	82	84	83	85	82	84	84	83	82	87
	77	84	84	83	82	84	81	84	83	82	85	81	85	82	84	83	85	81	83	83	83	82
R	76	84	84	84	82	81	84	81	84	82	82	85	81	85	81	84	83	84	81	83	83	83
	75	85	83	83	84	82	81	84	81	84	82	81	85	81	85	81	84	83	84	81	83	83
A	74	82	84	83	83	84	[illegible]	81	83	81	84	82	81	85	80	85	81	84	83	85	81	83
	73	83	81	84	83	83	83	81	81	83	81	84	82	81	85	81	85	81	84	83	84	81
F	72	83	82	81	83	82	82	83	81	81	83	81	84	83	81	85	81	85	81	84	83	84
	71	81	83	82	81	83	82	82	83	81	81	83	80	83	82	81	84	81	85	81	83	82
F	70	80	81	83	81	80	83	82	82	83	81	81	83	80	83	82	81	84	80	85	81	84
	69	83	80	80	82	81	80	82	81	81	83	81	80	83	80	83	82	81	84	80	85	81
I	68	81	82	80	80	82	81	80	82	82	81	83	81	80	83	80	82	83	81	84	80	85
	67	82	80	81	80	80	82	81	80	82	81	81	82	81	80	83	80	82	82	81	84	80
C	66	82	81	80	81	79	80	82	80	80	82	81	81	82	81	80	83	80	82	82	81	83
	65	80	81	81	79	81	79	80	81	80	80	82	81	81	82	80	79	83	80	82	82	81
	64	82	79	81	81	79	81	79	80	81	80	80	82	81	81	82	80	79	83	80	82	82
	63	81	82	79	80	81	79	81	79	79	81	81	80	82	81	81	81	80	79	83	80	82
L	62	79	80	82	79	80	81	79	81	79	79	81	81	80	82	81	81	81	81	79	83	80
	61	80	79	80	82	79	80	80	79	81	79	80	81	81	80	82	81	81	81	80	79	83
I	60	82	80	79	80	81	79	80	80	79	81	79	79	81	81	80	82	81	81	81	80	79
	59	78	81	80	79	80	81	79	80	80	79	81	79	80	81	81	80	82	81	81	81	81
G	58	80	78	81	79	79	80	81	79	80	80	79	80	79	80	81	81	80	82	81	81	80
	57	78	80	78	81	79	79	80	81	79	79	79	79	80	79	80	81	81	80	82	81	81
H	56	78	78	80	78	81	79	79	80	81	79	80	80	79	80	80	80	81	81	81	82	81
	55	80	78	77	79	78	81	79	79	80	81	79	79	79	79	80	80	80	81	80	80	82
T	54	77	80	78	77	79	78	81	79	79	80	82	79	79	79	80	81	80	81	81	81	81
	53	79	**76**	80	78	77	79	78	81	79	79	80	82	79	79	79	80	81	80	81	81	81
	52	77	79	**76**	80	78	78	79	78	81	79	79	80	82	80	80	80	81	81	81	82	82
	51	78	77	79	**76**	79	78	78	79	78	81	79	79	80	82	80	80	80	81	81	82	82
2	50	78	78	78	80	**76**	80	78	78	80	79	82	80	80	81	83	82	81	82	82	83	83
	49	77	78	78	78	79	**76**	80	78	78	80	78	82	80	79	81	83	82	81	81	82	83
	48	80	78	80	79	78	81	77	82	79	80	81	81	84	81	81	84	85	85	83	84	85
	47	79	80	78	80	79	78	81	77	82	79	80	81	81	84	81	81	84	85	85	83	84
T	46	81	81	81	79	81	81	80	83	80	84	82	83	83	85	87	85	85	87	89	90	87
	45	80	81	81	81	79	81	81	80	83	79	84	82	83	83	84	87	85	85	87	89	90
I	44	82	82	84	83	84	82	84	85	84	88	84	90	88	89	91	92	96	95	95	100	101
	43	82	82	82	84	83	84	82	84	85	84	88	84	90	88	89	91	91	96	95	95	100
M	42	84	86	88	87	90	90	91	91	92	97	95	101	96	105	112	114	119	122	134	146	153
	41	85	84	86	88	86	89	90	91	91	92	97	94	01	96	105	112	114	119	122	134	146
E	40	96	97	98	101	105	109	113	124	126	135	147	161	177	200	215	267	305	341	366	414	449
		70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

TRAFFIC LIGHT 1 TIME

A.3 Data.

A.3.1 Alternatives Compared.

Geometric Temperature Solution Quality at 100 Trials

RUN	XX(1)	XX(2)	δ_i
1	63	47	4.94
2	68	50	2.04
3	86	57	2.12
4	85	64	2.65
5	87	62	2.30
6	92	56	2.45
7	88	64	2.65
8	65	51	1.44
9	88	61	2.01
10	63	55	2.48
11	94	84	7.09
12	87	66	2.53
13	74	58	2.30
14	79	59	3.42
15	60	47	5.11
16	86	66	4.27
17	61	45	4.94
18	68	56	2.48
19	83	59	2.45
20	79	57	2.12
21	89	86	7.55
22	93	61	3.15
23	85	58	2.72
24	65	53	1.44
25	83	66	2.65

Linear Temperature Solution Quality at 100 Trials

RUN	XX(1)	XX(2)	δ_i
1	65	45	4.94
2	65	53	1.44
3	86	61	2.72
4	84	57	2.12
5	80	57	2.01
6	67	51	2.04
7	86	57	2.12
8	67	58	2.04
9	78	69	3.81
10	83	57	2.12
11	95	77	7.55
12	71	60	2.01
13	86	55	2.72
14	88	61	2.30
15	63	45	5.45
16	67	59	4.86
17	65	53	1.44
18	71	54	2.48
19	83	61	2.78
20	80	59	2.30
21	90	82	7.55
22	89	57	2.45
23	96	71	4.33
24	62	56	2.48
25	84	60	2.45

Adaptive Temperature Solution Quality at 100 Trials

RUN	XX(1)	XX(2)	δ_i
1	61	45	4.94
2	69	49	2.81
3	86	55	2.12
4	78	61	3.05
5	80	57	2.01
6	71	55	3.27
7	86	55	2.12
8	75	51	2.81
9	74	62	3.58
10	67	50	2.04
11	93	74	6.16
12	86	61	2.12
13	71	65	5.09
14	86	55	2.12
15	66	45	5.57
16	65	53	1.44
17	64	47	4.13
18	65	53	1.44
19	85	64	2.12
20	84	64	2.30
21	97	70	4.33
22	82	57	2.12
23	102	65	4.33
24	79	55	3.05
25	73	65	4.55

Elliptic Temperature Solution Quality at 100 Trials

RUN	XX(1)	XX(2)	δ_i
1	63	48	4.94
2	69	49	2.04
3	82	59	2.12
4	79	53	3.69
5	85	64	2.01
6	83	57	2.45
7	87	59	2.65
8	75	51	2.81
9	85	66	4.69
10	80	61	2.01
11	91	87	7.09
12	90	59	2.12
13	78	61	2.53
14	77	55	4.31
15	66	46	5.11
16	84	66	4.15
17	64	47	4.94
18	69	57	2.48
19	82	59	2.12
20	66	51	1.44
21	84	84	7.55
22	85	54	3.15
23	82	68	2.78
24	70	53	1.44
25	86	65	2.65

Logarithmic Temperature Solution Quality at 100 Trials			
RUN	XX(1)	XX(2)	δ_i
1	63	39	4.94
2	63	59	2.48
3	87	54	2.45
4	81	60	2.01
5	75	56	2.30
6	79	65	2.53
7	84	61	2.65
8	70	51	2.04
9	83	70	2.65
10	85	56	2.30
11	103	86	7.55
12	88	60	2.12
13	66	50	2.04
14	81	66	3.25
15	59	42	6.23
16	74	54	3.30
17	63	48	2.81
18	69	53	2.04
19	91	62	2.12
20	82	59	2.12
21	99	72	6.19
22	88	55	2.78
23	98	66	4.33
24	65	54	1.44
25	86	64	2.65

Logarithmic Temperature with Linear Coefficient Solution Quality at 100 Trials

RUN	XX(1)	XX(2)	δ_i
1	61	45	4.94
2	61	47	2.04
3	86	63	2.12
4	89	56	3.15
5	80	64	3.55
6	69	49	2.56
7	86	62	2.78
8	67	51	2.04
9	87	54	3.81
10	80	68	3.81
11	97	80	7.55
12	81	60	2.12
13	74	65	5.00
14	86	57	2.45
15	61	45	5.45
16	71	66	4.57
17	79	53	3.30
18	73	58	4.72
19	82	59	2.12
20	88	63	2.65
21	97	81	7.55
22	87	58	3.15
23	97	70	4.33
24	62	56	2.48
25	86	63	2.78

Logarithmic Temperature with Elliptic Coefficient Solution Quality at 100 Trials

RUN	XX(1)	XX(2)	δ_i
1	63	43	4.94
2	66	51	2.04
3	85	64	2.30
4	69	61	4.20
5	85	63	2.30
6	67	52	2.04
7	87	62	2.65
8	67	51	1.44
9	79	63	2.01
10	86	57	2.01
11	91	76	6.19
12	88	61	2.12
13	66	56	3.27
14	81	60	2.12
15	63	45	4.94
16	66	52	1.44
17	64	47	2.81
18	63	55	2.48
19	90	56	2.12
20	87	58	2.78
21	89	81	7.55
22	89	57	2.45
23	97	71	4.33
24	65	53	1.44
25	82	65	2.01

Geometric Temperature Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	64	45	4.94
2	68	50	1.44
3	68	49	1.44
4	81	60	2.01
5	86	66	2.65
6	66	52	1.44
7	82	59	2.01
8	67	51	1.44
9	86	61	2.78
10	84	53	2.45
11	98	73	4.33
12	87	65	2.01
13	69	53	1.44
14	88	60	2.01
15	63	45	4.94
16	75	62	2.81
17	65	54	1.44
18	65	52	1.44
19	88	56	2.12
20	88	55	2.01
21	101	66	4.36
22	80	56	3.15
23	104	61	4.33
24	65	54	1.44
25	87	60	2.65

Linear Temperature Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	68	55	2.04
2	72	54	1.44
3	67	51	1.44
4	88	61	2.12
5	78	61	2.01
6	79	65	2.12
7	88	61	2.30
8	65	45	1.44
9	85	64	2.01
10	73	50	2.01
11	76	74	4.62
12	84	56	2.12
13	69	52	1.44
14	69	49	2.04
15	71	54	2.04
16	78	61	3.99
17	65	47	1.44
18	64	47	1.44
19	69	54	2.01
20	71	47	2.04
21	100	67	4.36
22	85	64	2.12
23	98	73	4.33
24	65	53	1.44
25	86	55	2.45

Adaptive Temperature Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	69	54	2.04
2	70	56	1.44
3	67	51	1.44
4	65	53	1.44
5	85	64	2.45
6	89	57	2.53
7	81	59	2.01
8	85	49	2.01
9	88	63	2.65
10	64	54	1.44
11	91	59	2.65
12	86	56	2.01
13	65	52	1.44
14	89	59	2.72
15	71	51	4.13
16	88	61	2.45
17	67	51	1.44
18	61	45	2.56
19	85	64	2.01
20	85	64	2.12
21	100	67	4.36
22	85	64	2.45
23	100	72	4.33
24	71	54	1.44
25	85	58	2.12

Elliptic Temperature Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	72	54	2.81
2	67	51	1.44
3	67	51	2.04
4	75	64	2.53
5	67	51	2.01
6	63	55	2.01
7	64	43	2.04
8	65	53	1.44
9	79	63	2.01
10	65	54	2.12
11	109	69	4.33
12	87	62	2.45
13	65	53	1.44
14	68	53	1.44
15	67	51	2.04
16	67	51	2.04
17	67	52	2.04
18	65	46	1.44
19	88	64	2.12
20	78	54	3.30
21	97	70	4.33
22	85	64	2.01
23	102	64	4.33
24	67	51	1.44
25	86	60	2.65

Logarithmic Temperature Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	63	41	5.11
2	70	56	1.44
3	78	64	2.30
4	84	60	2.01
5	89	62	2.30
6	64	54	1.44
7	86	57	2.01
8	67	50	1.44
9	65	46	2.04
10	92	59	2.30
11	71	62	2.12
12	102	62	4.33
13	80	61	2.01
14	86	63	2.01
15	71	48	1.44
16	75	61	2.01
17	67	51	1.44
18	85	56	2.01
19	101	68	4.33
20	86	60	2.01
21	67	55	1.44
22	84	63	2.65
23	84	55	2.12
24	73	53	1.44
25	83	57	2.01

Logarithmic Temperature with Linear Coefficient Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	70	53	2.04
2	64	54	1.44
3	86	61	2.30
4	87	55	2.01
5	92	59	2.30
6	92	59	2.72
7	85	64	2.01
8	63	55	1.44
9	85	64	2.30
10	85	64	2.65
11	97	70	4.33
12	87	62	2.12
13	85	64	2.12
14	80	63	2.12
15	71	47	2.04
16	78	50	4.12
17	63	48	2.04
18	63	55	1.44
19	80	54	2.65
20	67	51	1.44
21	100	70	4.33
22	89	66	2.45
23	91	76	4.33
24	66	52	1.44
25	82	56	2.30

Logarithmic Temperature with Elliptic Coefficient Solution Quality at 200 Trials

RUN	XX(1)	XX(2)	δ_i
1	66	56	1.44
2	70	56	1.44
3	88	64	2.30
4	90	59	2.01
5	69	49	2.04
6	66	53	1.44
7	82	58	2.01
8	71	50	1.44
9	76	60	4.18
10	75	61	2.01
11	83	95	6.19
12	83	57	2.01
13	68	55	3.27
14	92	59	2.01
15	59	50	1.44
16	66	52	1.44
17	72	54	2.56
18	67	51	2.04
19	84	58	2.01
20	65	52	1.44
21	103	64	4.33
22	86	63	2.78
23	100	68	4.36
24	65	53	1.44
25	85	56	2.65

A.3.2 Local Search Compared.

Local Search Efficiency (1 Iteration)

Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	19	2	2	0	0	21.00
2	19	3	3	0	0	25.00
3	19	6	6	0	0	35.00
4	19	1	1	0	0	15.00
5	19	2	2	0	0	25.00
6	19	2	2	0	0	25.00
7	19	2	2	0	0	19.00
8	19	2	2	0	0	29.00
9	19	1	1	0	0	13.00
10	19	1	1	0	0	13.00
11	19	4	4	0	0	33.00
12	19	3	3	0	0	27.00
13	19	2	2	0	0	20.00
14	19	1	1	0	0	15.00
15	19	4	4	0	0	29.00
16	19	2	2	0	0	21.00
17	19	4	4	0	0	27.00
18	19	4	4	0	0	25.00
19	19	2	2	0	0	19.00
20	19	3	3	0	0	34.00
21	19	3	3	0	0	30.00
22	19	6	6	0	0	51.00
23	19	1	1	0	0	13.00
24	19	3	3	0	0	23.00
25	19	9	9	0	0	52.00

Local Search Efficiency (1 Iteration)						
Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
26	19	6	6	0	0	51.00
27	19	4	4	0	0	26.00
28	19	1	1	0	0	14.00
29	19	1	1	0	0	15.00
30	19	3	3	0	0	24.00
31	19	3	3	0	0	24.00
32	19	5	5	0	0	48.00
33	19	2	2	0	0	18.00
34	19	3	3	0	0	22.00
35	19	1	1	0	0	14.00
36	19	1	1	0	0	14.00
37	19	2	2	0	0	20.00
38	19	2	2	0	0	45.00
39	19	1	1	0	0	15.00
40	19	1	1	0	0	13.00
41	19	1	1	0	0	14.00
42	19	1	1	0	0	14.00
43	19	2	2	0	0	22.00
44	19	4	4	0	0	25.00
45	19	1	1	0	0	23.00
46	19	2	2	0	0	20.00
47	19	2	2	0	0	23.00
48	19	7	7	0	0	38.00
49	19	3	3	0	0	60.00
50	19	2	2	0	0	25.00

Local Search Efficiency (1 Iteration)

Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
51	19	3	3	0	0	22.00
52	19	7	7	0	0	44.00
53	19	4	4	0	0	22.00
54	19	3	3	0	0	23.00
55	19	3	3	0	0	24.00
56	19	4	4	0	0	25.00
57	19	1	1	0	0	16.00
58	19	8	8	0	0	44.00
59	19	2	2	0	0	29.00
60	19	1	1	0	0	13.00
61	19	3	3	0	0	30.00
62	19	4	4	0	0	29.00
63	19	1	1	0	0	16.00
64	19	8	8	0	0	44.00
65	19	4	4	0	0	29.00
66	19	3	3	0	0	47.00
67	19	1	1	0	0	13.00
68	19	3	3	0	0	22.00
69	19	1	1	0	0	14.00
70	19	3	3	0	0	26.00
71	19	3	3	0	0	27.00
72	19	3	3	0	0	30.00
73	19	1	1	0	0	21.00
74	19	3	3	0	0	24.00

Local Search Efficiency (1 Iteration)						
Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
75	19	1	1	0	0	14.00
76	19	3	3	0	0	39.00
77	19	2	2	0	0	21.00
78	19	8	8	0	0	44.00
79	19	4	4	0	0	22.00
80	19	1	1	0	0	14.00
81	19	3	3	0	0	34.00
82	19	1	1	0	0	13.00
83	19	3	3	0	0	34.00
84	19	1	1	0	0	14.00
85	19	1	1	0	0	13.00
86	19	2	2	0	0	28.00
87	19	3	3	0	0	22.00
88	19	2	2	0	0	17.00
89	19	8	8	0	0	41.00
90	19	3	3	0	0	23.00
91	19	3	3	0	0	24.00
92	19	3	3	0	0	30.00
93	19	3	3	0	0	34.00
94	19	4	4	0	0	35.00
95	19	8	8	0	0	63.00
96	19	2	2	0	0	37.00
97	19	3	3	0	0	22.00
98	19	2	2	0	0	23.00
99	19	6	6	0	0	51.00
100	19	4	4	0	0	26.00

Local Search Solution Quality (1 Iteration)

RUN	XX(1)	XX(2)	δ_i
1	51	41	14.38
2	57	63	17.56
3	82	51	5.84
4	71	69	6.79
5	79	59	4.90
6	68	55	5.36
7	80	68	4.54
8	69	53	5.03
9	73	76	6.41
10	70	63	5.62
11	91	76	6.19
12	79	59	4.90
13	65	66	11.45
14	62	64	14.51
15	55	43	7.35
16	65	67	9.73
17	66	45	6.13
18	63	56	4.93
19	80	68	4.54
20	69	53	5.03
21	91	76	6.19
22	81	53	5.59
23	88	72	10.09
24	57	61	13.39
25	79	65	5.68

Local Search Solution Quality (1 Iteration)

RUN	XX(1)	XX(2)	δ_i
26	81	53	5.59
27	86	63	2.78
28	69	58	6.63
29	64	73	18.51
30	61	78	23.99
31	84	56	3.78
32	81	53	5.59
33	51	47	12.35
34	80	68	4.54
35	64	72	17.30
36	53	59	23.99
37	71	59	5.73
38	67	43	7.59
39	78	63	3.05
40	88	74	9.79
41	64	70	15.04
42	81	76	9.13
43	73	66	5.39
44	63	56	4.93
45	79	65	5.68
46	59	47	6.30
47	81	76	9.13
48	82	51	5.84
49	57	41	6.23
50	68	55	5.36

Local Search Solution Quality (1 Iteration)

RUN	XX(1)	XX(2)	δ_i
51	75	51	4.63
52	57	41	6.23
53	67	51	2.04
54	57	61	13.39
55	75	73	4.76
56	61	73	21.24
57	51	48	14.91
58	59	47	6.30
59	69	53	5.03
60	73	75	5.60
61	69	49	2.81
62	66	45	6.13
63	51	48	14.91
64	59	47	6.30
65	68	78	13.02
66	85	51	6.12
67	69	57	3.99
68	61	75	23.40
69	69	56	4.56
70	57	65	20.98
71	53	53	15.32
72	81	72	6.15
73	85	56	3.15
74	57	62	16.54
75	77	57	4.91

Local Search Solution Quality (1 Iteration)

RUN	XX(1)	XX(2)	δ_i
76	57	41	6.23
77	51	41	14.38
78	59	47	6.30
79	67	51	2.04
80	53	59	23.99
81	69	53	5.03
82	89	75	7.28
83	69	53	5.03
84	88	71	10.19
85	89	75	7.28
86	69	49	2.81
87	67	70	10.87
88	86	63	2.78
89	82	51	5.84
90	57	61	13.39
91	75	73	4.76
92	57	65	20.98
93	69	53	5.03
94	87	62	3.25
95	71	47	4.13
96	57	41	6.23
97	61	75	23.40
98	68	55	5.36
99	81	53	5.59
100	86	63	2.78

Local Search Efficiency (7 Iterations)						
Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	70	2	2	0	0	165.00
2	70	3	3	0	0	150.00
3	70	6	6	0	0	185.00
4	70	1	1	0	0	230.00
5	70	2	2	0	0	165.00
6	70	2	2	0	0	135.00
7	70	2	2	0	0	211.00
8	70	2	2	0	0	185.00
9	70	1	1	0	0	177.00
10	70	1	1	0	0	195.00

Local Search Solution Quality (7 Iterations)			
RUN	XX(1)	XX(2)	δ_i
1	51	41	4.54
2	57	63	4.90
3	82	51	4.54
4	71	69	2.78
5	79	59	3.78
6	68	55	3.05
7	80	68	4.93
8	69	53	2.04
9	73	76	2.81
10	70	63	3.99

Simulated Annealing with Logarithmic Temperature Efficiency (1 Iteration)

Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	62	58	10	19	29	266.00
2	62	31	13	6	12	132.00
3	62	68	10	24	34	263.00
4	62	21	5	4	12	89.00
5	62	71	5	32	34	248.00
6	62	52	6	23	23	192.00
7	62	60	6	23	31	223.00
8	62	22	8	4	10	74.00
9	62	69	10	23	36	256.00
10	62	27	5	8	14	108.00

Simulated Annealing with Logarithmic Temperature Solution Quality (1 Iteration)

RUN	XX(1)	XX(2)	δ_i
1	63	43	4.94
2	63	55	1.44
3	86	63	2.12
4	78	69	3.81
5	82	68	2.78
6	70	55	1.44
7	74	60	2.01
8	70	56	2.56
9	86	61	2.78
10	69	57	3.30

Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient Efficiency (1 Iteration)

Run	Total Moves	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	57	51	10	16	25	241.00
2	57	50	15	14	21	193.00
3	57	49	11	15	23	217.00
4	57	21	5	6	10	79.00
5	57	45	6	15	24	205.00
6	57	52	7	19	26	220.00
7	57	56	5	21	30	230.00
8	57	15	4	6	5	62.00
9	57	29	7	12	10	111.00
10	57	35	5	12	18	123.00

SAMPLE CALCULATION: Mean CPUTime $= \sum_{i=1}^{10} \frac{(\text{CPU Time})_i}{10} = \frac{1681}{10} = 168.10$

Simulated Annealing with Logarithmic Temperature Solution Quality (1 Iteration)

RUN	XX(1)	XX(2)	δ_i
1	63	43	4.94
2	72	54	2.04
3	81	59	2.01
4	78	69	3.81
5	71	55	2.01
6	69	49	1.44
7	86	61	2.01
8	67	58	3.79
9	80	61	2.30
10	82	60	2.01

SAMPLE CALCULATION: Mean Percent Difference $= \frac{100}{10} \sum_{i=1}^{10} \frac{\delta_i}{76} = 10 \bullet \frac{26.36}{76} = 3.47$

Appendix B: The Configuration of Machines -- Open Queuing Network

Page

B.1 Scenario

The network model shows the segmentation of the machine stations, the processing time distributions for each station, and the transition probabilities between stations. The model consists of nine segments. The first segment defines the decision variables **RI** (where **I** represents the numbers 1 through 6), the number of resources available at each station. Although the model initializes the number of machines per station to zero, a user insert alters the number according to each trial configuration. The second segment models the arrival of parts to the job-shop using an exponential distribution with an average inter-arrival rate of one part every five seconds.

Segments three through eight model the machine stations using identical logic. As each part arrives at a station, it is "tagged" with the time TNOW using ATRIB(2). When a machine becomes available to process that part, the waiting time spent at the station is added to ATRIB(1). Processing time for each machine is exponentially distributed with an average of five seconds. After this processing, the part is routed probabilistically to the next station. The shipment from one station to the next occurs with an exponential distribution at an average of two seconds. Stations one through five all have two transition probabilities; each probability equal to 50%. Station six has a 100% transition probability.

Segment nine collects the statistics on the overall waiting time through the EVENT statement, linking to a user insert. Once enough samples have been collected for a given simulation run, the user insert enters an entity into the model which then terminates the simulation.

The network models uses generic structures: the XX(I) array represents decision variables and the ARRAY(I,J) array represents control parameters. The generic structure of the network model facilitates the transfer of control to the user inserts.

B.1.1 Scenario Definition.

```
Scenario:
SCE1
Control:
CONT
Network:
NET2
Script:
Facility:
User Insert:
EVENT2
FIRST2
INTLC
NEXT2
OTPUT
OUT2
PUT2
RAN
TEST2
TIME
Notes:
Data:
Curchange:
00000000
Definition:
```

B.1.2 Control Statements.

```
GEN,WARRENDER,MACHINES,7/20/93,100000,N,N,Y/N,N,N/1,72;
LIMITS,6,2,500;
STAT,1,WAIT TIME;
SEEDS,8653713(10)/Y,6470899(9)/Y,4286515(8)/Y,
4399739(7)/Y,6475819(6)/Y,9113213(5)/Y,8126355(4)/Y,
2734681(3)/Y,6315779(2)/Y,9375295(1)/Y;
ARRAY(1,21);
ARRAY(2,21);
ARRAY(3,21);
NETWORK;
INITIALIZE,,1000000,Y;
FIN;
```

B.1.3 Network Model.

```
        RESOURCE/1,R1(0),1;
        RESOURCE/2,R2(0),2;
        RESOURCE/3,R3(0),3;
        RESOURCE/4,R4(0),4;
        RESOURCE/5,R5(0),5;
        RESOURCE/6,R6(0),6;
;
INPUT   CREATE,EXPON(5,2),,,,1;
IN      ASSIGN,ATRIB(1)=0.0,1;
        ACTIVITY,,,A1;
;
A1      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(1),R1,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/1,EXPON(5,3),,F1;
F1      FREE,R1/1,1;
        ACTIVITY/12,EXPON(2,9),0.5,A2;
        ACTIVITY/13,EXPON(2,10),0.5,A3;
;
A2      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(2),R2,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/2,EXPON(5,4),,F2;
F2      FREE,R2/1,1;
        ACTIVITY/21,EXPON(2,9),0.5,A1;
        ACTIVITY/24,EXPON(2,10),0.5,A4;
;
A3      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(3),R3,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/3,EXPON(5,5),,F3;
F3      FREE,R3/1,1;
        ACTIVITY/31,EXPON(2,9),0.5,A1;
        ACTIVITY/35,EXPON(2,10),0.5,A5;
;
A4      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(4),R4,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/4,EXPON(5,6),,F4;
F4      FREE,R4/1,1;
        ACTIVITY/42,EXPON(2,9),0.5,A2;
        ACTIVITY/46,EXPON(2,10),0.5,A6;
;
A5      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(5),R5,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/5,EXPON(5,7),,F5;
F5      FREE,R5/1,1;
        ACTIVITY/53,EXPON(2,9),0.5,A3;
        ACTIVITY/56,EXPON(2,10),0.5,A6;
;
A6      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(6),R6,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/6,EXPON(5,8),,F6;
F6      FREE,R6/1,1;
        ACTIVITY,,,E1;
E1      EVENT,1,1;
        ACTIVITY,,,OUT;
OUT     TERMINATE,;
        ENTER,1,1;
        ACTIVITY,,,END;
END     TERMINATE,1;
        END;
```

B.1.4 User Inserts.

B.1.4.1 Subroutine INTLC.

```
$INCLUDE:'PRCTL.FOR'
C******************************************************
C**   ANNEALING ROUTINE CONTROL                      **
C******************************************************
      SUBROUTINE INTLC
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      SELECT CASE (II)
C******************************************************
C**   NEIGHBORING SOLUTIONS                          **
C******************************************************
      CASE (2, 12, 22, 32, 42, 52, 62, 72, 82, 92)
           CALL NEXT
C******************************************************
C**   NEW INITIAL SOLUTIONS                          **
C******************************************************
           CASE (1, 11, 21, 31, 41, 51, 61, 71, 81, 91)
                II = II + 1
           CALL FIRST'
C******************************************************
C**   LOCAL SEARCH INITIAL SOLUTION (N=100, M=25)    **
C******************************************************
           CASE (0)
           II =  2
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   0.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**  LINEAR TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (10)
           II = 12
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   1.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C****************************************************
C**ADAPTIVE TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C****************************************************
           CASE (20)
           II = 22
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   2.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*****************************************************
C** LINEAR COEFFICIENT INITIAL SOLUTION (N=100, M=25)**
C*****************************************************
           CASE (30)
           II = 32
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   3.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*****************************************************
C**ELLIPTIC COEFFICIENT INITIAL SOLUTION (N=100, M=25)*
C*****************************************************
           CASE (40)
           II = 42
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   4.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*****************************************************
C**   LOCAL SEARCH INITIAL SOLUTION (N=200, M=25)    **
C*****************************************************
           CASE (50)
           II = 52
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   5.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*****************************************************
```

```
C******************************************************
C** LINEAR TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (60)
           II = 62
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   6.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ADAPTIVE TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (70)
           II = 72
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   7.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C** LINEAR COEFFICIENT INITIAL SOLUTION (N=200, M=25)**
C******************************************************
           CASE (80)
           II = 82
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   8.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC COEFFICIENT INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (90)
           II = 92
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   9.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
    END SELECT
    WRITE(*,10) INT(II/10+1.0),INT(GETARY(3,4)+1.0),
   + INT(GETARY(3, 2) + 1.0)
10  FORMAT(3I5)
    RETURN
    END
```

B.1.4.2 Subroutine FIRST.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE FIRST
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C****************************************************
C**  INITIALIZE RESOURCES R1 - R6                  **
C****************************************************
 CALL RAN(6,R1)
 CALL RAN(6,R2)
 CALL RAN(6,R3)
 CALL RAN(6,R4)
 CALL RAN(6,R5)
 CALL RAN(6,R6)
 I1 = 3 + INT(R1*2.0)
 I2 = 3 + INT(R2*2.0)
 I3 = 3 + INT(R3*2.0)
 I4 = 3 + INT(R4*2.0)
 I5 = 3 + INT(R5*2.0)
 I6 = 3 + INT(R6*2.0)
 CALL ALTER(1, I1)
 CALL ALTER(2, I2)
 CALL ALTER(3, I3)
 CALL ALTER(4, I4)
 CALL ALTER(5, I5)
 CALL ALTER(6, I6)
 XX(4) =  REAL(I1)
 XX(5) =  REAL(I2)
 XX(6) =  REAL(I3)
 XX(7) =  REAL(I4)
 XX(8) =  REAL(I5)
 XX(9) =  REAL(I6)
C****************************************************
C**  INITIALIZE XX(1) = SAMPLES AND XX(2) = BATCHES **
C****************************************************
 XX(1) =   0.0
 XX(2) =   0.0
 XX(3) =   0.0
C****************************************************
C**  INITIALIZE INCUMBENT AND SOLUTION VALUES      **
C****************************************************
```

```
 CALL PUTARY( 1, 1, 99999.9)
 CALL PUTARY( 1, 2,     0.0)
 CALL PUTARY( 1, 3,    30.0)
 CALL PUTARY( 2, 1, 99999.9)
 CALL PUTARY( 2, 2,     0.0)
 CALL PUTARY( 2, 3,    30.0)
C*****************************************************
C**  INITIALIZE ARRAY(3,1) = TOTAL RUNS             **
C**  ARRAY(3,2) = CURRENT RUN                       **
C*****************************************************
C     RT = 100.0 + 100.0 * INT(GETARY(3, 5)/8)
 RT = 200.0
 CALL PUTARY( 3, 1,   RT)
 CALL PUTARY( 3, 2,  0.0)
 CALL SETTIM( 0, 0, 0, 0)
C*****************************************************
 RETURN
 END
```

B.1.4.3 Subroutine NEXT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE NEXT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C*****************************************************
C**  INITIALIZE INCUMBENT SOLUTION                  **
C**  AND SELECT A RANDOM  DIRECTION                 **
C*****************************************************
      CHARACTER*3 LABEL
10    I1 = INT(GETARY(1,1))
      I2 = INT(GETARY(1,2))
      I3 = INT(GETARY(1,3))
      I4 = INT(GETARY(1,4))
      I5 = INT(GETARY(1,5))
      I6 = INT(GETARY(1,6))
      IF ((I1 + I2 + I3 + I4 + I5 + I6).LE.24) THEN
      IM = INT(DRAND(2)*2)
      IM = 2*IM - 1
      ELSE
      IM = -1
      END IF
      SELECT CASE (INT(DRAND(2)*6) + 1)
      CASE (1)
       LABEL = ' R1'
       I1    = I1 + IM
       IM    = I1
       IMIN  = 3
      CASE (2)
       LABEL = ' R2'
       I2    = I2 + IM
       IM    = I2
       IMIN  = 2
      CASE (3)
       LABEL = ' R3'
       I3    = I3 + IM
       IM    = I3
       IMIN  = 2
      CASE (4)
       LABEL = ' R4'
       I4    = I4 + IM
       IM    = I4
       IMIN  = 2
      CASE (5)
       LABEL = ' R5'
       I5    = I5 + IM
       IM    = I5
       IMIN  = 2
      CASE (6)
       LABEL = ' R6'
       I6    = I6 + IM
       IM    = I6
       IMIN  = 2
      END SELECT
      IF (IM.LT.IMIN) GOTO 10
      WRITE(*,20) LABEL
      WRITE(*,30) IM
20    FORMAT(A5)
30    FORMAT(I5)
C******************************************************
C**  INITIALIZE NEIGHBORING SOLUTION                 **
C******************************************************
      CALL ALTER(1, I1)
      CALL ALTER(2, I2)
      CALL ALTER(3, I3)
      CALL ALTER(4, I4)
      CALL ALTER(5, I5)
      CALL ALTER(6, I6)
      XX(1) =  REAL(I1)
      XX(2) =  REAL(I2)
      XX(3) =  REAL(I3)
      XX(4) =  REAL(I4)
      XX(5) =  REAL(I5)
      XX(6) =  REAL(I6)
      RETURN
      END
```

B.1.4.4 Subroutine RAN.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE RAN(IS,R)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL Z, R, C, G
      C = 4.294967296E+9
      Z = GETARY(3, IS)
      G = 5*Z + 99991
      Z = MOD(G,C)
      R = Z/C
      CALL PUTARY(3, IS, Z)
      RETURN
      END
```

B.1.4.5 Subroutine EVENT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE EVENT(I)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C
      DIMENSION A(10)
      REAL BATCHES, PROBABILITY
C*****************************************************
C**   COLLECT XX(1) SAMPLES AND TOTAL WAIT XX(2)    **
C*****************************************************
      IF (TNOW.LT.500.0) RETURN
      XX(1)   = XX(1) + 1
      XX(2)   = XX(2) + ATRIB(1)
C*****************************************************
C**  RECORD BATCH MEAN                              **
C*****************************************************
      IF (XX(1).LT.50) RETURN
      WAIT    = XX(2)/50
      XX(1)   = 0.0
      XX(2)   = 0.0
      CALL COLCT(WAIT,1)
      BATCHES = CCNUM(1) - 10.0
C*****************************************************
C**   STOP COLLECTING BATCHES WHEN CRITERIA MET:    **
C** LESS THAN 100% CHANCE OF ACCEPTANCE OR 30 BATCHES*
C*****************************************************
      IF (BATCHES.LT.0) RETURN
      CALL TEST(PROBABILITY)
      TERM = PROBABILITY*(1 - BATCHES/31.25)
      IF (TERM.LE.0.20) THEN
```

```
        CALL ENTER(1,A)
        RETURN
      END IF
      RETURN
      END
```

B.1.4.6 Subroutine TEST.

```
$INCLUDE:'PRCTL.FOR'
     SUBROUTINE TEST(PT)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL RT,RC,T,Y1,S1,N1,Y2,S2,N2,SP,DY,L,PT
C*****************************************************
C**  DELTA FUNCTION                                 **
C*****************************************************
      Y1 = GETARY(1,1)
      Y2 = CCAVG(1)
      DY = (Y2-Y1)
C*****************************************************
C**   CALCULATE PT; THE PROBABILITY OF ACCEPTANCE  **
C*****************************************************
      IF (DY.LT.0.0) THEN
        PT = 1.0
        RETURN
      END IF
      IT = INT(GETARY(3, 5))
C*****************************************************
C**   ACCEPTANCE FUNCTION                           **
C*****************************************************
      SELECT CASE(IT)
      CASE (0, 5)
        PT = 0.0
        RETURN
C*****************************************************
      CASE (1, 6)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 0.67 * (RT - RC) / RT
        PT = EXP(-DY/T)
        RETURN
      CASE (2, 7)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        S1 = GETARY(1,2) + 0.0001
        N1 = GETARY(1,3)
        S2 = CCSTD(1) + 0.0001
        N2 = CCNUM(1)
        SP = SQRT(((N1-1)*S1**2+(N2-1)*S2**2)/(N1+N2-2))
        T  = 0.22 * SP * (RT - RC) / RT
        PT = EXP(-DY/T)
        RETURN
C****************************************************
      CASE (3)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 1.71/LOG(RC + 1.0)
        L  = (RT - RC)/RT
        PT = L*EXP(-DY/T)
        RETURN
      CASE(4)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 1.38/LOG(RC + 1.0)
        L  = SQRT(1 - RC**2/RT**2)
        PT = L*EXP(-DY/T)
        RETURN
      CASE (8)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.01/LOG(RC + 1.0)
        L  = (RT - RC)/RT
        PT = L*EXP(-DY/T)
        RETURN
      CASE(9)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 1.62/LOG(RC + 1.0)
        L  = SQRT(1 - RC**2/RT**2)
        PT = L*EXP(-DY/T)
        RETURN
C****************************************************
      END SELECT
      END
```

B.1.4.7 Subroutine OTPUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE OTPUT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL YC,YI,YB,RT,RC,PT,PC,SC,ST
C*****************************************************
C**  INCREMENT THE NUMBER OF RUNS                   **
C**  RETRIEVE YC AND YI SOLUTIONS                   **
C*****************************************************
```

```
      RT = GETARY(3, 1)
      RC = GETARY(3, 2)
      CALL TEST(PT)
      RC = RC + 1.0
      CALL PUTARY(3, 2, RC)
      CALL RAN(8,PC)
      YC = CCAVG(1)
      YI = GETARY(1,  1)
      YB = GETARY(2,  1)
C*****************************************************
C**    YC IS LESS THAN PREVIOUS YB SOLUTION         **
C*****************************************************
      IF (YC.LT.YB) THEN
       CALL PUTARY(3,  9, A0)
       CALL PUTARY(3, 10, A1)
       CALL PUT(1)
       CALL PUT(2)
       GOTO 100
      END IF
C*****************************************************
C**  YC IS LESS THAN YI SOLUTION                    **
C*****************************************************
      IF (PT.EQ.1) THEN
       CALL PUTARY(3, 9, A0)
       CALL PUTARY(3, 11, A2)
       CALL PUT(1)
       GOTO 100
      END IF
C*****************************************************
C** DETERMINE ACCEPTANCE OF A MOVE AWAY FROM OPTIMUM *
C**  COMPARE THE PC AGAINST PT OF ACCEPTANCE        **
C*****************************************************
      IF (PC.LT.PT) THEN
       CALL PUTARY(3,  9, A0)
       CALL PUTARY(3, 12, A3)
       CALL PUT(1)
      END IF
C****************************************************
C**  ANNEALING COMPLETE, START FROM NEW SOLUTION   **
C****************************************************
100   IF (RC.LT.RT) RETURN
      II = II - 1
      ST = GETARY(3, 3)
      SC = GETARY(3, 4) + 1.0
      CALL PUTARY(3, 4, SC)
      CALL OUT
C****************************************************
C**   TEST COMPLETE, START NEW TEST                **
C****************************************************
      IF (SC.LT.ST) RETURN
      II = II + 9
C*****************************************************
C**  PROBLEM COMPLETE                               **
C*****************************************************
      IF   (II.LT.100)     RETURN
      WRITE(*,*) 'TYPE ENTER TO CONTINUE'
      READ (*,*)
      STOP
      END
```

B.1.4.8 Subroutine PUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE PUT(I1)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      CALL PUTARY(I1, 1, CCAVG(1))
      CALL PUTARY(I1, 2, CCSTD(1))
      CALL PUTARY(I1, 3, CCNUM(1))
      CALL PUTARY(I1, 4,    XX(4))
      CALL PUTARY(I1, 5,    XX(5))
      CALL PUTARY(I1, 6,    XX(6))
      CALL PUTARY(I1, 7,    XX(7))
      CALL PUTARY(I1, 8,    XX(8))
      CALL PUTARY(I1, 9,    XX(9))
      RETURN
      END
```

B.1.4.9 Subroutine OUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE OUT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C*****************************************************
      CALL TIME(RTIME)
      IT = INT(RTIME)
      IA = INT(II/10) + 1
      IS = INT(GETARY(3,   4))
      IR = INT(GETARY(3,   2))
      OPEN(UNIT= 8,ACCESS='APPEND',FILE='VAL2.DOC')
      OPEN(UNIT= 9,ACCESS='APPEND',FILE='SOL2.DAT')
      OPEN(UNIT=10,ACCESS='APPEND',FILE='REC2.DOC')
      WRITE(8, 10)   IA, ',' , IS , ',' ,
     + INT(GETARY(2,4)) , ',' , INT(GETARY(2,5)) , ',' ,
     + INT(GETARY(2,6)) , ',' , INT(GETARY(2,7)) , ',' ,
     + INT(GETARY(2,8)) , ',' , INT(GETARY(2,9))
10    FORMAT(I5, A1, I5, A1
     + I5, A1, I5, A1,
     + I5, A1, I5, A1,
     + I5, A1, I5)
      WRITE(9, 20) IA , ',' , IS , ',' , IT , ',' ,
     + IR , ',' , GETARY(2,1) , ',' , GETARY(2,2) , ','
20    FORMAT(I5, A1, I5, A1, I5, A1,
     + I5, A1, F5.2, A1, F5.2, A1)
      WRITE(10, *) IA, IS
      WRITE(10, *) GETARY(3, 6)
      WRITE(10, *) GETARY(3, 7)
      WRITE(10, *) GETARY(3, 8)
      CLOSE(UNIT = 8, STATUS = 'KEEP')
      CLOSE(UNIT = 9, STATUS = 'KEEP')
      CLOSE(UNIT =10, STATUS = 'KEEP')
C******************************************************
      RETURN
      END
```

B.1.4.10 Subroutine TIME.

```
C*************************************************
C**  RETURNS SYSTEM TIME IN SECONDS             **
C*************************************************
      SUBROUTINE TIME(RTIME)
      REAL RTIME
      CALL GETTIM(IHR, IMIN, ISEC, I100TH)
      RTIME = 3600*REAL(IHR)+60*REAL(IMIN)+REAL(ISEC)+
     +     REAL(I100TH)/100.0
      RETURN
      END
```

B.2 Optimal Solution.

The configuration space generated by the six decision variables is too large to completely enumerate the response values. There is no theoretical basis at present for determining what the optimum configuration should be. The only alternative left is to sort through all of the solutions obtained and pick the most optimal value. Since this is a minimization problem the, the lowest value found is used as the optimum response value. The lowest value is found to be 1.48 and this is used in calculating the solution quality.

B.3 Data.

B.3.1 Comparing the Alternatives.

Linear Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	5	5	4	3	3	5	1.11
2	4	7	4	3	4	3	5.50
3	6	4	4	5	3	3	.43
4	6	5	4	4	2	4	.79
5	6	6	4	3	3	3	.08
6	6	4	5	4	3	3	.23
7	6	4	4	5	3	3	.43
8	7	4	5	3	3	3	.00
9	4	4	4	3	4	3	6.24
10	4	4	5	3	3	6	5.95
11	6	5	5	3	3	3	.06
12	6	5	5	3	3	3	.06
13	6	4	3	3	4	5	3.33
14	6	6	4	3	3	3	.08
15	6	4	4	4	4	3	.76
16	7	5	3	4	3	3	2.39
17	6	4	5	4	4	2	1.30
18	6	4	4	5	2	4	1.24
19	6	5	5	3	3	3	.06
20	5	4	3	4	3	6	4.63
21	6	5	4	3	4	3	.29
22	6	6	4	3	3	3	.08
23	7	3	5	4	3	3	2.84
24	7	5	4	3	3	3	.26
25	6	4	5	4	3	3	.23

Adaptive Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	5	5	4	3	3	5	1.11
2	7	4	5	3	3	3	.00
3	7	3	4	3	3	5	3.55
4	5	6	4	3	3	4	.86
5	6	5	4	3	4	3	.29
6	6	4	5	3	3	3	.58
7	6	4	5	4	3	3	.23
8	6	4	4	5	3	3	.43
9	5	4	4	5	5	2	2.72
10	4	6	4	4	4	3	5.30
11	7	5	4	3	3	3	.26
12	6	4	5	4	3	3	.23
13	6	4	4	3	4	4	1.22
14	5	4	4	3	3	5	1.79
15	6	5	5	4	2	3	.78
16	5	5	6	3	3	3	.68
17	5	4	5	4	3	3	.85
18	3	3	3	3	2	5	87.46
19	6	4	5	4	3	3	.23
20	6	4	5	4	3	3	.23
21	6	4	5	5	3	2	1.90
22	5	4	4	3	6	2	3.77
23	5	8	3	4	2	3	4.11
24	5	5	5	3	3	4	1.05
25	6	4	5	4	3	3	.23

Linear Coefficient Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	5	4	5	4	2	5	2.17
2	5	4	5	4	3	4	.65
3	7	4	5	3	3	3	.00
4	6	4	5	4	3	3	.23
5	6	4	4	3	4	4	1.22
6	7	5	4	3	3	3	.26
7	6	4	4	5	3	3	.43
8	4	5	5	6	2	3	5.23
9	6	4	4	4	4	3	.76
10	4	6	4	5	3	3	5.33
11	5	5	4	3	3	5	1.11
12	5	5	6	3	3	3	.68
13	7	5	4	3	3	3	.26
14	6	5	5	3	3	3	.06
15	5	3	4	3	4	6	4.34
16	6	5	4	3	4	3	.29
17	5	3	6	2	6	3	5.26
18	5	4	4	6	2	4	3.01
19	9	4	4	2	3	3	1.65
20	5	3	4	3	3	7	4.99
21	7	5	4	3	2	4	1.26
22	5	5	6	3	3	3	.68
23	6	6	4	3	3	3	.08
24	4	6	3	4	4	4	7.53
25	5	5	4	3	3	5	1.11

Elliptic Coefficient Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	5	4	4	3	3	6	1.79
2	6	5	4	3	4	3	.29
3	6	4	5	4	3	3	.23
4	6	6	4	3	3	3	.08
5	6	6	4	3	3	3	.08
6	6	3	4	4	5	2	4.47
7	6	4	4	5	3	3	.43
8	5	4	4	5	2	5	2.54
9	7	4	4	3	3	4	.87
10	4	4	4	5	3	5	6.29
11	6	4	5	4	3	3	.23
12	6	4	4	5	3	3	.43
13	6	4	3	5	3	3	3.89
14	5	5	4	4	3	4	1.08
15	7	4	5	3	3	3	.00
16	4	4	4	3	4	5	6.02
17	6	5	5	3	3	3	.06
18	6	4	3	3	4	3	3.46
19	5	4	3	4	2	7	6.12
20	6	4	5	3	3	4	.41
21	7	5	4	2	4	3	1.25
22	5	3	4	5	5	2	5.71
23	5	4	4	5	4	3	1.83
24	7	5	3	4	3	3	2.39
25	6	4	4	4	3	4	.70

Linear Temperature Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	5	5	3	3	3	.06
2	7	4	5	3	3	3	.00
3	6	4	5	3	4	3	.45
4	6	6	4	3	3	3	.08
5	5	4	5	4	3	4	.65
6	5	5	5	3	4	3	.68
7	7	5	4	3	3	3	.26
8	7	4	5	3	3	3	.00
9	7	4	5	3	3	3	.00
10	7	5	4	3	3	3	.26
11	7	4	5	3	3	3	.00
12	6	4	5	4	3	3	.23
13	6	6	4	3	3	3	.08
14	7	4	5	3	3	3	.00
15	6	4	5	4	3	3	.23
16	6	4	5	4	3	3	.23
17	6	4	4	5	3	3	.43
18	7	5	4	3	3	3	.26
19	7	4	4	3	4	3	.45
20	6	6	4	3	3	3	.08
21	6	4	5	4	3	3	.00
22	6	4	5	4	3	3	.00
23	6	4	5	4	3	3	.00
24	8	4	4	3	3	3	.00
25	6	5	4	4	3	3	.85

Adaptive Temperature Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	4	5	3	3	3	.00
2	6	5	5	3	3	3	.06
3	6	5	5	3	3	3	.06
4	6	4	5	4	3	3	.23
5	7	4	5	3	3	3	.00
6	5	5	4	4	3	4	1.08
7	6	4	4	2	6	3	1.92
8	6	5	5	3	3	3	.06
9	6	5	4	3	4	3	.29
10	6	5	4	3	4	3	.29
11	7	4	5	3	3	3	.00
12	7	4	5	3	3	3	.00
13	7	4	5	3	3	3	.00
14	6	4	5	4	3	3	.23
15	6	5	5	3	3	3	.06
16	5	4	4	6	3	3	1.97
17	6	5	5	3	3	3	.06
18	6	6	4	3	3	3	.08
19	6	4	5	4	3	3	.23
20	6	4	3	3	6	3	3.01
21	6	5	5	3	3	3	.06
22	6	4	5	4	3	3	.23
23	6	4	4	5	3	3	.43
24	6	4	4	5	3	3	.43
25	7	4	5	3	3	3	.00

Linear Coefficient Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	5	5	3	3	3	.06
2	6	4	5	3	3	4	.41
3	6	6	4	3	3	3	.08
4	6	4	4	5	3	3	.43
5	6	4	5	4	3	3	.23
6	6	5	4	3	4	3	.29
7	6	5	5	3	3	3	.06
8	7	4	5	3	3	3	.00
9	6	4	5	4	3	3	.23
10	7	6	4	3	2	3	.82
11	6	4	5	4	3	3	.23
12	8	4	4	3	3	3	.58
13	7	4	5	3	3	3	.00
14	5	5	5	3	4	3	.68
15	6	5	5	3	3	3	.06
16	5	4	5	4	3	4	.65
17	6	6	4	3	3	3	.08
18	5	5	4	3	4	4	1.14
19	7	5	4	3	3	3	.26
20	6	6	4	3	3	3	.08
21	6	5	5	3	3	3	.06
22	6	4	5	3	4	3	.45
23	6	6	4	3	3	3	.08
24	7	4	5	3	3	3	.00
25	6	4	4	4	3	4	.70

Elliptic Coefficient Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	5	6	4	3	3	4	.86
2	7	5	4	3	3	3	.26
3	7	4	5	3	3	3	.00
4	6	5	5	3	3	3	.06
5	6	4	5	4	3	3	.23
6	7	4	5	3	3	3	.00
7	7	4	5	3	3	3	.00
8	6	4	5	4	3	3	.23
9	6	6	4	3	3	3	.08
10	6	5	4	4	2	4	.79
11	7	4	5	3	3	3	.00
12	7	5	4	3	3	3	.26
13	7	4	5	2	3	4	1.44
14	6	5	5	3	3	3	.06
15	6	5	5	3	3	3	.06
16	7	4	5	3	3	3	.00
17	6	5	5	3	3	3	.06
18	7	5	4	3	3	3	.26
19	7	4	5	3	3	3	.00
20	7	4	5	3	3	3	.00
21	6	5	4	4	2	4	.79
22	5	5	5	3	4	3	.68
23	6	4	5	4	3	3	.23
24	6	5	4	3	4	3	.29
25	6	4	5	4	3	3	.23

B.3.2 Comparing Local Search.

Local Search Efficiency (1 Iteration)

Run	Moves Accepted	Best Moves	Other Moves	CPU Time
1	12	12	0	702.00
2	7	7	0	416.00
3	4	4	0	331.00
4	5	5	0	315.00
5	10	10	0	619.00
6	9	9	0	585.00
7	4	4	0	331.00
8	6	6	0	351.00
9	4	4	0	287.00
10	8	8	0	513.00
11	6	6	0	333.00
12	7	7	0	504.00
13	10	10	0	670.00
14	5	5	0	341.00
15	4	4	0	330.00
16	9	9	0	534.00
17	11	11	0	661.00
18	9	9	0	640.00
19	7	7	0	504.00
20	10	10	0	671.00

Local Search Efficiency (1 Iteration)

Run	Moves Accepted	Best Moves	Other Moves	CPU Time
21	10	10	0	631.00
22	12	12	0	703.00
23	11	11	0	686.00
24	5	5	0	366.00
25	7	7	0	423.00
26	10	10	0	630.00
27	10	10	0	630.00
28	4	4	0	331.00
29	5	5	0	329.00
30	6	6	0	412.00
31	7	7	0	504.00
32	7	7	0	415.00
33	6	6	0	392.00
34	6	6	0	396.00
35	10	10	0	619.00
36	5	5	0	315.00
37	7	7	0	416.00
38	10	10	0	562.00
39	3	3	0	261.00
40	7	7	0	423.00

Local Search Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	4	5	4	3	3	.42
2	9	4	4	2	3	3	2.02
3	6	4	4	3	4	4	1.25
4	7	4	4	3	4	3	.91
5	6	4	7	2	3	3	1.91
6	6	4	7	2	3	3	1.91
7	6	4	4	3	4	4	1.25
8	7	4	4	3	3	4	.90
9	7	4	4	3	4	3	.91
10	5	5	4	4	3	4	1.40
11	7	5	4	3	3	3	.33
12	6	4	5	4	3	3	.42
13	6	4	6	2	3	4	1.90
14	5	5	4	4	3	4	1.40
15	6	4	4	3	4	4	1.25
16	6	5	5	4	2	3	.92
17	6	4	7	2	3	3	1.91
18	6	4	6	2	3	4	1.90
19	6	4	5	4	3	3	.42
20	6	4	6	2	3	4	1.90

Local Search Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
21	6	4	7	2	3	3	1.91
22	6	4	5	4	3	3	.42
23	6	4	5	4	3	3	.42
24	6	4	4	3	4	4	1.25
25	6	4	5	3	4	3	.36
26	6	4	7	2	3	3	1.91
27	6	4	7	2	3	3	1.91
28	6	4	4	3	4	4	1.25
29	7	5	3	3	3	4	2.00
30	8	4	4	2	3	4	2.09
31	6	4	5	4	3	3	.42
32	9	4	4	2	3	3	2.02
33	6	4	5	3	4	3	.36
34	6	4	5	3	4	3	.36
35	6	4	7	2	3	3	1.91
36	7	4	4	3	4	3	.91
37	9	4	4	2	3	3	2.02
38	6	5	5	4	2	3	.92
39	6	4	4	4	3	4	.80
40	6	4	5	3	4	3	.36

Local Search Efficiency (5 Iterations)

Run	Moves Accepted	Best Moves	Other Moves	CPU Time
1	12	12	0	2383.00
2	7	7	0	2067.00
3	4	4	0	2178.00
4	5	5	0	3010.00
5	10	10	0	2809.00
6	9	9	0	2332.00
7	4	4	0	2326.00
8	6	6	0	1977.00

Local Search Solution Quality (5 Iterations)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	4	5	4	3	3	.42
2	9	4	4	2	3	3	.90
3	6	4	4	3	4	4	.33
4	7	4	4	3	4	3	.42
5	6	4	7	2	3	3	.36
6	6	4	7	2	3	3	1.25
7	6	4	4	3	4	4	.36
8	7	4	4	3	3	4	.36

Simulated Annealing with Linear Temperature Efficiency (1 Iteration)

Run	Moves Accepted	Best Moves	Other Moves	CPU Time
1	66	16	50	2473.00
2	65	8	57	2358.00
3	42	7	35	2311.00
4	45	8	37	2100.00
5	34	12	22	2193.00
6	39	12	27	2211.00
7	72	12	60	2358.00
8	64	10	54	2475.00
9	48	5	43	2213.00
10	74	13	61	2517.00

Simulated Annealing with Linear Temperature Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	4	5	4	3	3	.06
2	7	5	4	3	3	3	.00
3	8	5	3	2	3	3	.45
4	6	6	4	3	3	3	.08
5	6	4	7	2	3	3	.65
6	6	4	7	2	3	3	.68
7	7	5	4	2	3	4	.26
8	8	4	4	3	3	3	.00
9	8	4	4	3	3	3	.00
10	5	5	4	4	3	4	.26

Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient Efficiency (1 Iteration)

Run	Moves Accepted	Best Moves	Other Moves	CPU Time
1	72	15	57	2332.00
2	49	7	42	2156.00
3	76	10	66	2362.00
4	65	10	55	2180.00
5	66	11	55	2331.00
6	61	9	52	2349.00
7	54	13	41	2181.00
8	54	7	47	2349.00
9	44	6	38	2105.00
10	58	10	48	2191.00

Simulated Annealing with Logarithmic Temperature and Elliptic Coefficient Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	6	4	3	3	3	.86
2	8	4	4	3	3	3	.26
3	7	5	4	3	3	3	.00
4	6	5	5	3	3	3	.06
5	8	5	4	2	3	3	.23
6	9	4	4	2	3	3	.00
7	8	6	4	2	2	3	.00
8	6	4	5	4	3	3	.23
9	7	5	4	2	4	3	.08
10	6	5	5	3	3	3	.79

Appendix C: The Configuration of Machines -- Closed Queuing Network

Page

C.1 Scenario

The network model shows the segmentation of the machine stations, the processing time distributions for each station, and the transition probabilities between stations. The model consists of nine segments. The first segment defines the decision variables **RI** (where I represents the numbers 1 through 6), the number of resources available at each station. Although the model initializes the number of machines per station to zero, a user insert alters the number according to each trial configuration. The second segment models the arrival of parts to the job-shop using an exponential distribution with an average inter-arrival rate of one part every five seconds. Segments three through eight model the machine stations using identical logic. As each part arrives at a station, it is "tagged" with the time TNOW using ATRIB(2). When a machine becomes available to process that part, the waiting time spent at the station is added to ATRIB(1). Processing time for each machine is exponentially distributed with an average of five seconds. After this processing, the part is routed probabilistically to the next station as indicated in Figure 3.5. The shipment from one station to the next occurs with an exponential distribution at an average of two seconds. Stations one through five all have two transition probabilities; each probability equal to 50%. Station six has a 100% transition probability. Segment nine collects the statistics on the overall waiting time through the EVENT statement, linking to a user insert. Once enough samples have been collected for a given simulation run, the user insert enters an entity into the model which then terminates the simulation. The network models uses generic structures: the XX(I) array represents decision variables and the ARRAY(I,J) array represents control parameters. The generic structure of the network model facilitates the transfer of control to the user inserts.

C.1.1 Scenario Definition.

```
Scenario:
SCE1
Control:
CONT
Network:
NET2
Script:
Facility:
User Insert:
EVENT2
FIRST2
INTLC
NEXT2
OTPUT
OUT2
PUT2
RAN
TEST2
TIME
Notes:
Data:
Curchange:
00000000
Definition:
```

C.1.2 Control Statements.

```
GEN,WARRENDER,MACHINES,7/20/93,100000,N,N,Y/N,N,N/1,72;
LIMITS,6,2,500;
STAT,1,WAIT TIME;
SEEDS,8653713(10)/Y,6470899(9)/Y,4286515(8)/Y,
4399739(7)/Y,6475819(6)/Y,9113213(5)/Y,8126355(4)/Y,
2734681(3)/Y,6315779(2)/Y,9375295(1)/Y;
ARRAY(1,21);
ARRAY(2,21);
ARRAY(3,21);
NETWORK;
INITIALIZE,,1000000,Y;
FIN;
```

C.1.3 Network Model.

```
        RESOURCE/1,R1(0),1;
        RESOURCE/2,R2(0),2;
        RESOURCE/3,R3(0),3;
        RESOURCE/4,R4(0),4;
        RESOURCE/5,R5(0),5;
        RESOURCE/6,R6(0),6;
;
INPUT   CREATE,EXPON(5,2),,,30,1;
IN      ASSIGN,ATRIB(1)=0.0,1;
        ACTIVITY,,,A1;
;
A1      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(1),R1,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/1,EXPON(5,3),,F1;
F1      FREE,R1/1,1;
        ACTIVITY/12,EXPON(2,9),0.5,A2;
        ACTIVITY/13,EXPON(2,10),0.5,A3;
;
A2      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(2),R2,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/2,EXPON(5,4),,F2;
F2      FREE,R2/1,1;
        ACTIVITY/21,EXPON(2,9),0.5,A1;
        ACTIVITY/24,EXPON(2,10),0.5,A4;
;
A3      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(3),R3,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/3,EXPON(5,5),,F3;
F3      FREE,R3/1,1;
        ACTIVITY/31,EXPON(2,9),0.5,A1;
        ACTIVITY/35,EXPON(2,10),0.5,A5;
;
A4      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(4),R4,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/4,EXPON(5,6),,F4;
F4      FREE,R4/1,1;
        ACTIVITY/42,EXPON(2,9),0.5,A2;
        ACTIVITY/46,EXPON(2,10),0.5,A6;
;
A5      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(5),R5,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/5,EXPON(5,7),,F5;
F5      FREE,R5/1,1;
        ACTIVITY/53,EXPON(2,9),0.5,A3;
        ACTIVITY/56,EXPON(2,10),0.5,A6;
;
A6      ASSIGN,ATRIB(2)=TNOW,1;
        AWAIT(6),R6,,1;
        ASSIGN,ATRIB(1)=ATRIB(1)+TNOW-ATRIB(2),1;
        ACTIVITY/6,EXPON(5,8),,F6;
F6      FREE,R6/1,1;
        ACTIVITY,,,E1;
E1      EVENT,1,1;
        ACTIVITY,,,IN;
;
        ENTER,1,1;
        ACTIVITY,,,END;
END     TERMINATE,1;
        END;
```

C.1.4 User Inserts.

C.1.4.1 Subroutine INTLC.

```
$INCLUDE:'PRCTL.FOR'
C******************************************************
C**   ANNEALING ROUTINE CONTROL                      **
C******************************************************
      SUBROUTINE INTLC
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      SELECT CASE (II)
C******************************************************
C**   NEIGHBORING SOLUTIONS                          **
C******************************************************
      CASE (2, 12, 22, 32, 42, 52, 62, 72, 82, 92)
           CALL NEXT
C******************************************************
C**   NEW INITIAL SOLUTIONS                          **
C******************************************************
           CASE (1, 11, 21, 31, 41, 51, 61, 71, 81, 91)
                II = II + 1
           CALL FIRST
C******************************************************
C**   LOCAL SEARCH INITIAL SOLUTION (N=100, M=25)    **
C******************************************************
           CASE (0)
           II =  2
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   0.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C*GEOMETRIC TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (10)
           II = 12
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   1.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C*****************************************************
C**  LINEAR TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C*****************************************************
           CASE (20)
           II = 22
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   2.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ADAPTIVE TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (30)
           II = 32
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   3.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC TEMPERATURE INITIAL SOLUTION (N=100, M=25)*
C******************************************************
           CASE (40)
           II = 42
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   4.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
CLOGARITHMIC TEMPERATURE INITIAL SOLUTION (N=100, M=25)
C******************************************************
           CASE (50)
           II = 52
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   5.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C*******************************************************
C** LINEAR COEFFICIENT INITIAL SOLUTION (N=100, M=25)**
C*******************************************************
           CASE (60)
           II = 62
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   6.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*******************************************************
C**ELLIPTIC COEFFICIENT INITIAL SOLUTION (N=100, M=25)*
C*******************************************************
           CASE (70)
           II = 72
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   7.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*******************************************************
C**   LOCAL SEARCH INITIAL SOLUTION (N=200, M=25)     **
C*******************************************************
           CASE (80)
           II = 82
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   8.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C*******************************************************
C*GEOMETRIC TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C*******************************************************
           CASE (90)
           II = 92
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,   9.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C*****************************************************
C** LINEAR TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C*****************************************************
           CASE (100)
           II = 102
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5, 102.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ADAPTIVE TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (110)
           II = 112
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  11.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC TEMPERATURE INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (120)
           II = 122
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  12.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
CLOGARITHMIC TEMPERATURE INITIAL SOLUTION (N=200, M=25)
C******************************************************
           CASE (130)
           II = 132
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  13.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
```

```
C******************************************************
C** LINEAR COEFFICIENT INITIAL SOLUTION (N=200, M=25)**
C******************************************************
           CASE (140)
           II = 142
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  14.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
           CALL FIRST
C******************************************************
C**ELLIPTIC COEFFICIENT INITIAL SOLUTION (N=200, M=25)*
C******************************************************
           CASE (150)
           II = 152
           CALL PUTARY( 3,  3,  25.0)
           CALL PUTARY( 3,  4,   0.0)
           CALL PUTARY( 3,  5,  15.0)
           CALL PUTARY( 3,  6, 3853417.0)
           CALL PUTARY( 3,  7, 5113297.0)
           CALL PUTARY( 3,  8, 1522731.0)
C******************************************************
    END SELECT
    WRITE(*,10) INT(II/10+1.0),INT(GETARY(3,4)+1.0),
   + INT(GETARY(3, 2) + 1.0)
10  FORMAT(3I5)
    RETURN
    END
```

C.1.4.2 Subroutine FIRST.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE FIRST
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C*****************************************************
C**  INITIALIZE RESOURCES R1 - R6                   **
C*****************************************************
 CALL RAN(6,R1)
 CALL RAN(6,R2)
 CALL RAN(6,R3)
 CALL RAN(6,R4)
 CALL RAN(6,R5)
 CALL RAN(6,R6)
 I1 = 3 + INT(R1*2.0)
 I2 = 3 + INT(R2*2.0)
 I3 = 3 + INT(R3*2.0)
 I4 = 3 + INT(R4*2.0)
 I5 = 3 + INT(R5*2.0)
 I6 = 3 + INT(R6*2.0)
 CALL ALTER(1, I1)
 CALL ALTER(2, I2)
 CALL ALTER(3, I3)
 CALL ALTER(4, I4)
 CALL ALTER(5, I5)
 CALL ALTER(6, I6)
 XX(4) =  REAL(I1)
 XX(5) =  REAL(I2)
 XX(6) =  REAL(I3)
 XX(7) =  REAL(I4)
 XX(8) =  REAL(I5)
 XX(9) =  REAL(I6)
C*****************************************************
C**  INITIALIZE XX(1) = SAMPLES AND XX(2) = BATCHES**
C*****************************************************
 XX(1) =   0.0
 XX(2) =   0.0
 XX(3) =   0.0
C*****************************************************
C**  INITIALIZE INCUMBENT AND SOLUTION VALUES       **
C*****************************************************
 CALL PUTARY( 1, 1, 99999.9)
 CALL PUTARY( 1, 2,     0.0)
 CALL PUTARY( 1, 3,    30.0)
 CALL PUTARY( 2, 1, 99999.9)
 CALL PUTARY( 2, 2,     0 0)
 CALL PUTARY( 2, 3,    30.0)
C*****************************************************
C**  INITIALIZE ARRAY(3,1) = TOTAL RUNS             **
C**  ARRAY(3,2) = CURRENT RUN                       **
C*****************************************************
C    RT = 100.0 + 100.0 * INT(GETARY(3, 5)/8)
 RT = 200.0
 CALL PUTARY( 3, 1,   RT)
 CALL PUTARY( 3, 2,  0.0)
 CALL PUTARY( 3, 9,  0.0)
 CALL PUTARY( 3,10,  0.0)
 CALL PUTARY( 3,11,  0.0)
 CALL PUTARY( 3,12,  0.0)
 CALL SETTIM( 0, 0, 0, 0)
C*****************************************************
 RETURN
 END
```

C.1.4.3 Subroutine NEXT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE NEXT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C*****************************************************
C**  INITIALIZE INCUMBENT SOLUTION                  **
C**  AND SELECT A RANDOM  DIRECTION                 **
C*****************************************************
      I1 = INT(GETARY(1,4))
      I2 = INT(GETARY(1,5))
      I3 = INT(GETARY(1,6))
      I4 = INT(GETARY(1,7))
      I5 = INT(GETARY(1,8))
      I6 = INT(GETARY(1,9))
      R = GETARY(3,2) - GETARY(3,10) - GETARY(3,11)
      IC = INT(MOD(R, 12.0))
      IM = 2*INT(IC/6.0) - 1.0
      IF ((I1 + I2 + I3 + I4 + I5 + I6).GT.24) IM = -1
10    SELECT CASE (IC)
      CASE (0, 6)
       I1    = I1 + IM
       IM    = I1
       IMIN  = 3
      CASE (1, 7)
       I2    = I2 + IM
       IM    = I2
       IMIN  = 2
      CASE (2, 8)
       I3    = I3 + IM
       IM    = I3
       IMIN  = 2
      CASE (3, 9)
       I4    = I4 + IM
       IM    = I4
       IMIN  = 2
      CASE (4, 10)
       I5    = I5 + IM
       IM    = I5
       IMIN  = 2
      CASE (5, 11)
       I6    = I6 + IM
       IM    = I6
       IMIN  = 2
      END SELECT
      IF (IM.LT.IMIN) THEN
        IM = 1.0
        GOTO 10
      END IF
C*****************************************************
C**  INITIALIZE NEIGHBORING SOLUTION                **
C*****************************************************
      CALL ALTER(1, I1)
      CALL ALTER(2, I2)
      CALL ALTER(3, I3)
      CALL ALTER(4, I4)
      CALL ALTER(5, I5)
      CALL ALTER(6, I6)
      XX(4) =  REAL(I1)
      XX(5) =  REAL(I2)
      XX(6) =  REAL(I3)
      XX(7) =  REAL(I4)
      XX(8) =  REAL(I5)
      XX(9) =  REAL(I6)
      RETURN
      END
```

C.1.4.4 Subroutine RAN.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE RAN(IS,R)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL Z, R, C, G
      C = 4.294967296E+9
      Z = GETARY(3, IS)
      G = 5*Z + 99991
      Z = MOD(G,C)
      R = Z/C
      CALL PUTARY(3, IS, Z)
      RETURN
      END
```

C.1.4.5 Subroutine EVENT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE EVENT(I)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C
      DIMENSION A(10)
      REAL BATCHES, PROBABILITY
C******************************************************
C**  COLLECT XX(1) SAMPLES AND TOTAL WAIT XX(2)     **
C******************************************************
      IF (TNOW.LT.500.0) RETURN
      XX(1)   = XX(1) + 1
      XX(2)   = XX(2) + ATRIB(1)
C******************************************************
C**  RECORD BATCH MEAN                              **
C******************************************************
      IF (XX(1).LT.50) RETURN
      WAIT    = XX(2)/50
      XX(1)   = 0.0
      XX(2)   = 0.0
      CALL COLCT(WAIT,1)
      BATCHES = CCNUM(1) - 10.0
C******************************************************
C**  STOP COLLECTING BATCHES WHEN CRITERIA MET:     **
C**  LESS THAN 100% CHANCE OF ACCEPTANCE OR 30 BATCHES
C******************************************************
      IF (BATCHES.LT.0) RETURN
      CALL TEST(PROBABILITY)
      TERM = PROBABILITY*(1 - BATCHES/31.25)
      IF (TERM.LE.0.20) THEN
        CALL ENTER(1,A)
        RETURN
      END IF
      RETURN
      END
```

C.1.4.6 Subroutine TEST.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE TEST(PT)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL RT,RC,T,Y1,S1,N1,Y2,S2,N2,SP,DY,L,PT
C****************************************************
C**  DELTA FUNCTION                                **
C****************************************************
      Y1 = GETARY(1,1)
      Y2 = CCAVG(1)
      DY = (Y2-Y1)
C****************************************************
C**  CALCULATE PT; THE PROBABILITY OF ACCEPTANCE   **
C****************************************************
      IF (DY.LT.0.0) THEN
        PT = 1.0
        RETURN
      END IF
      IT = INT(GETARY(3, 5))
C****************************************************
C**  ACCEPTANCE FUNCTION                           **
C****************************************************
      SELECT CASE(IT)
      CASE (0, 8)
        PT = 0.0
        RETURN
C****************************************************
      CASE (1)
        RC = GETARY(3, 2)
        T  = 0.58*0.95**(INT(RC/10.0)-1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (9)
        RC = GETARY(3, 2)
        T  = 0.75*0.95**(INT(RC/10.0)-1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (2, 10)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 0.975 * (RT - RC) / RT
        PT = EXP(-DY/T)
        RETURN
      CASE (3, 11)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        S1 = GETARY(1,2) + 0.0001
        N1 = GETARY(1,3)
        S2 = CCSTD(1) + 0.0001
        N2 = CCNUM(1)
        SP = SQRT(((N1-1)*S1**2+(N2-1)*S2**2)/(N1+N2-2))
        T  = 0.22 * SP * (RT - RC) / RT
        PT = EXP(-DY/T)
        RETURN
      CASE (4, 12)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 0.563 * SQRT(1 - (RC/RT)**2)
        PT = EXP(-DY/T)
        RETURN
C*****************************************************
      CASE (5)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 1.92/LOG(RC + 1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (6)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.49/LOG(RC + 1.0)
        L  = (RT - RC)/RT
        PT = L*EXP(-DY/T)
        RETURN
      CASE(7)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.01/LOG(RC + 1.0)
        L  = SQRT(1 - RC**2/RT**2)
        PT = L*EXP(-DY/T)
        RETURN
      CASE (13)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.25/LOG(RC + 1.0)
        PT = EXP(-DY/T)
        RETURN
      CASE (14)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.93/LOG(RC + 1.0)
        L  = (RT - RC)/RT
        PT = L*EXP(-DY/T)
        RETURN
      CASE(15)
        RT = GETARY(3, 1)
        RC = GETARY(3, 2)
        T  = 2.36/LOG(RC + 1.0)
        L  = SQRT(1 - RC**2/RT**2)
        PT = L*EXP(-DY/T)
        RETURN
C***************************************************
      END SELECT
      END
```

C.1.4.7 Subroutine OTPUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE OTPUT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      REAL YC,YI,YB,RT,RC,PT,PC,SC,ST
C****************************************************
C**INCREMENT THE NUMBER OF RUNS                    **
C**RETRIEVE YC AND YI SOLUTIONS                    **
C****************************************************
      RT = GETARY(3, 1)
      RC = GETARY(3, 2)
      CALL TEST(PT)
      RC = RC + 1.0
      CALL PUTARY(3, 2, RC)
      CALL RAN(8,PC)
      YC = CCAVG(1)
      YI = GETARY(1,  1)
      YB = GETARY(2,  1)
      A0 = GETARY(3,  9)
      A1 = GETARY(3, 10)
      A2 = GETARY(3, 11)
      A3 = GETARY(3, 12)
C****************************************************
C**  YC IS LESS THAN PREVIOUS YB SOLUTION          **
C****************************************************
      IF (YC.LT.YB) THEN
       A0 = A0 + 1.0
       A1 = A1 + 1.0
       CALL PUTARY(3,  9, A0)
       CALL PUTARY(3, 10, A1)
       CALL PUT(1)
       CALL PUT(2)
       GOTO 100
      END IF
C****************************************************
C**  YC IS LESS THAN YI SOLUTION                   **
C****************************************************
      IF (PT.EQ.1) THEN
```

```
        A0 = A0 + 1.0
        A2 = A2 + 1.0
        CALL PUTARY(3, 9, A0)
        CALL PUTARY(3, 11, A2)
        CALL PUT(1)
        GOTO 100
       END IF
C*****************************************************
C**  DETERMINE ACCEPTANCE OF A MOVE AWAY FROM OPTIMUM*
C**  COMPARE THE PC AGAINST PT OF ACCEPTANCE        **
C*****************************************************
       IF (PC.LT.PT) THEN
        A0 = A0 + 1.0
        A3 = A3 + 1.0
        CALL PUTARY(3,  9, A0)
        CALL PUTARY(3, 12, A3)
        CALL PUT(1)
       END IF
C****************************************************
C**  ANNEALING COMPLETE, START FROM NEW SOLUTION   **
C****************************************************
100    IF (RC.LT.RT) RETURN
       II = II - 1
       ST = GETARY(3, 3)
       SC = GETARY(3, 4) + 1.0
       CALL PUTARY(3, 4, SC)
       CALL OUT
C****************************************************
C**  TEST COMPLETE, START NEW TEST                 **
C****************************************************
       IF (SC.LT.ST) RETURN
       II = II + 9
C*****************************************************
C**   PROBLEM COMPLETE                              **
C*****************************************************
       IF   (II.LT.160)     RETURN
       WRITE(*,*) 'TYPE ENTER TO CONTINUE'
       READ (*,*)
       STOP
       END
```

C.1.4.8 Subroutine PUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE PUT(I1)
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
      CALL PUTARY(I1, 1, CCAVG(1))
      CALL PUTARY(I1, 2, CCSTD(1))
      CALL PUTARY(I1, 3, CCNUM(1))
      CALL PUTARY(I1, 4,    XX(4))
      CALL PUTARY(I1, 5,    XX(5))
      CALL PUTARY(I1, 6,    XX(6))
      CALL PUTARY(I1, 7,    XX(7))
      CALL PUTARY(I1, 8,    XX(8))
      CALL PUTARY(I1, 9,    XX(9))
      RETURN
      END
```

C.1.4.9 Subroutine OUT.

```
$INCLUDE:'PRCTL.FOR'
      SUBROUTINE OUT
$INCLUDE:'PARAM.INC'
$INCLUDE:'SCOM1.COM'
C*****************************************************
      CALL TIME(RTIME)
      IT = INT(RTIME)
      IA = INT(II/10) + 1
      IS = INT(GETARY(3,  4))
      IR = INT(GETARY(3,  2))
      IM0 = INT(GETARY(3, 9))
      IM1 = INT(GETARY(3,10))
      IM2 = INT(GETARY(3,11))
      IM3 = INT(GETARY(3,12))
      OPEN(UNIT= 8,ACCESS='APPEND',FILE='VAL2.DOC')
      OPEN(UNIT= 9,ACCESS='APPEND',FILE='SOL2.DAT')
      OPEN(UNIT=10,ACCESS='APPEND',FILE='REC2.DOC')
      WRITE(8, 10)   IA, ',' , IS , ',' ,
     + INT(GETARY(2,4)) , ',' , INT(GETARY(2,5)) , ',' ,
     + INT(GETARY(2,6)) , ',' , INT(GETARY(2,7)) , ',' ,
     + INT(GETARY(2,8)) , ',' , INT(GETARY(2,9))
10    FORMAT(I5, A1, I5, A1
     + I5, A1, I5, A1,
     + I5, A1, I5, A1,
     + I5, A1, I5)
      WRITE(9, 20) IA , ',' , IS , ',' , IT , ',' ,
     + IR , ',' , IM0 , ',' , IM1 , ',' , IM2 , ',' ,
     + IM3 , ',' , GETARY(2,1) , ',' , GETARY(2,2) , ','
```

```
20     FORMAT(I5, A1, I5, A1, I5, A1,
     + I5, A1, I5, A1, I5, A1, I5, A1,
     + I5, A1, F5.2, A1, F5.2, A1)
       WRITE(10, *) IA, IS
       WRITE(10, *) GETARY(3, 6)
       WRITE(10, *) GETARY(3, 7)
       WRITE(10, *) GETARY(3, 8)
       CLOSE(UNIT = 8, STATUS = 'KEEP')
       CLOSE(UNIT = 9, STATUS = 'KEEP')
       CLOSE(UNIT =10, STATUS = 'KEEP')
C*****************************************************
       RETURN
       END
```

C.1.4.10 Subroutine TIME.

```
C************************************************
C**  RETURNS SYSTEM TIME IN SECONDS            **
C************************************************
       SUBROUTINE TIME(RTIME)
       REAL RTIME
       CALL GETTIM(IHR, IMIN, ISEC, I100TH)
       RTIME = 3600*REAL(IHR)+60*REAL(IMIN)+REAL(ISEC)+
     +     REAL(I100TH)/100.0
       RETURN
       END
```

C.2 Optimal Solution.

The optimum configuration is established using a PASCAL program written by Lt Col Dietz. Given the one-step probability transition matrix, which describes the flow of parts among the machine stations, and the number of machines at each station, the program calculates the steady-state waiting time at each station. By changing the number of machines at each station, the optimal configuration is inferred. Locating the optimal configuration in the data and using the corresponding response value as the base response value allows the solution quality to be measured. Since this value is also the minimum response value, the validity of the choice is strengthened. The program and the results inferred from the program are presented below. The base value selected is 15.49.

The program was used to generate the expected server utilizations for each of the queues in a network with unlimited machines. These server utilizations were used to determine which queues get used most and which get used least. The ideal allocation of machines was identified using these relationships. Unfortunately, the optimum allocation included fractional numberts of machines. Therefore, a minimum allocation of 22 machines was made. The average response times were determined by using the code with the desired configuration. The Queues with the three highest response times were allocated the remaining 3 machines. The total response time for the new configuration was determined. A possible improvement was identified and the total response time measured. This allowed a global optimum to be identified because the total response time increased, rather than decreased.

Simulated Annealing and Local Search were applied to the simulation model. The optimum configuration found using the theoretical inference was found to generete the mimimum response value. The value was found to be 15.49 seconds of total waiting time, not including service.

C.2.1 Mean Value Analysis.

```
program MVA;

{*****************************************************}
{*                                                   *}
{*  Applies Mean Value Analysis to determine average *}
{*  queue lengths response times, and utilizations   *}
{*  for stations in a closed queueing network.       *}
{*  Language: (standard Pascal except file mgt)      *}
{*   Turbo-Pascal 5.5                                *}
{*  Source:                                          *}
{*   D C Dietz, (513) 255-3362, ddietz@afit.af.mil   *}
{*   AFIT/ENS, Wright-Patterson AFB OH 45433-7765    *}
{*                                                   *}
{*****************************************************}

const Nmax=50; {maximum number of customers in network}
      Mmax=30; {maximum number of stations in network}

type Age=(old,new);
     MIntArray=array[1..Mmax] of integer;
     MRealArray=array[1..Mmax] of real;
     PMatrix=array[1..Mmax,1..Mmax] of real;
     OutMatrix=array[1..Mmax,0..Nmax] of real;

var I,J,K,M,N: integer;
    CT,Lambda: real;
    NServe: MIntArray;
    S,V: MRealArray;
    P: PMatrix;
    C: array[1..Mmax,1..Nmax] of integer;
    Q,R,U: OutMatrix;
    Pr: array[old..new] of OutMatrix;
    Dat,Out: text;
{****************************************************}

procedure Error(ErrCode,I,K: integer);

{Reports errors}

begin
    writeln;
    case ErrCode of
        1: writeln('ERROR: N>Nmax (',Nmax:3,')');
        2: writeln('ERROR: M>Mmax-1 (',Mmax-1:3,')');
        3: writeln('ERROR: routing probs from station ',
                I:3,' do not sum to 1.000');
        4: writeln('WARNING: Pr{',I:2,',0,',K:2,'}<0; ',
            'may have numerical problems');
      end; {case}
    if (ErrCode=4) then begin
        writeln('Press <enter> to resume');
        readln;
      end else begin
        writeln('Program terminated;
                 press <enter> to exit');
        readln;
        halt;
      end; {else}
  end; {Error}
{*****************************************************}
procedure ReadData(var N,M: integer;
                   var NServe: MIntArray;
                   var S: MRealArray;
                   var P: PMatrix;
                   var Dat: text);
{Reads input data from file assigned to text var 'Dat'}
var I,J: integer;
    Psum: real;
begin
    readln(Dat,N,M);
    if not (N in [1..Nmax]) then Error(1,0,0);
    if not (M in [1..Mmax-1]) then Error(2,0,0);
    for I:=1 to M do read(Dat,NServe[I]);
    for I:=1 to M do read(Dat,S[I]);
    for I:=1 to M do begin
        Psum:=0;
        for J:=1 to M do begin
            read(Dat,P[J,I]);
            Psum:=Psum+P[J,I];
          end; {for}
        if (Psum<0.999) or (Psum>1.001) then
           Error(3,I,0);
      end; {for}
    P[1,1]:=1;
    for J:=2 to M do P[1,J]:=0;
    P[1,M+1]:=1;
    for I:=2 to M do begin
        P[I,I]:=P[I,I]-1;
        P[I,M+1]:=0;
      end; {for}
  end; {ReadData}

{****************************************************}

procedure VSolve(P: PMatrix;
                 M: integer;
                 var V: MRealArray);

{Solves simultaneous equations to obtain visit ratios}

var I,J,K,IC,KK,MM,IT,IS: integer;
    B,W,C: real;
    ID: MIntArray;
    Y: MRealArray;
begin
    MM:=M+1;
    for I:=1 to M do ID[I]:=I;
    K:=1;
    repeat
        KK:=K+1;
        IS:=K;
        IT:=K;
        B:=abs(P[K,K]);
        for I:=K to M do for J:=K to M do
          if (abs(P[I,J])>B) then begin
              IS:=I;
              IT:=J;
              B:=abs(P[I,J]);
            end; {if}
        if (IS>K) then
          for J:=K to MM do begin
              C:=P[IS,J];
              P[IS,J]:=P[K,J];
              P[K,J]:=C;
            end; {for}
        if (IT>K) then begin
            IC:=ID[K];
            ID[K]:=ID[IT];
            ID[IT]:=IC;
            for I:=1 to M do begin
                C:=P[I,IT];
                P[I,IT]:=P[I,K];
                P[I,K]:=C;
              end; {for}
          end; {if}
        for J:=KK to MM do begin
            P[K,J]:=P[K,J]/P[K,K];
            for I:=KK to M do begin
                W:=P[I,K]*P[K,J];
                P[I,J]:=P[I,J]-W;
                if (abs(P[I,J])<0.00001*abs(W)) then
                   P[I,J]:=0;
              end; {for}
          end; {for}
        K:=KK;
      until (K=M);
    Y[M]:=P[M,MM]/P[M,M];
    for I:=1 to M-1 do begin
        K:=M-I;
        KK:=K+1;
        Y[K]:=P[K,MM];
        for J:=KK to M do Y[K]:=Y[K]-P[K,J]*Y[J];
      end; {for}
    for I:=1 to M do for J:=1 to M do
      if (ID[J]=I) then V[I]:=Y[J];
  end; {VSolve}
{****************************************************}
procedure WriteOut(N,M: integer;
                   X: OutMatrix;
                   var Out: text);
{Writes output to file assigned to text var 'Out'}
var I,K: integer;
begin
    writeln(Out);
    write(Out,'  N');
    for I:=1 to M do write(Out,'   i=',I:2);
    writeln(Out);
    writeln(Out);
    for K:=1 to N do begin
        write(Out,K:3);
        for I:=1 to M do write(Out,X[I,K]:7:3);
        writeln(Out);
      end; {for}
    writeln(Out);
    writeln(Out);
  end; {WriteOut}
{****************************************************}
begin
    assign(Dat,'MVA.DAT');
    assign(Out,'MVA.OUT');
    reset(Dat);
    rewrite(Out);
    writeln;
    writeln('*** Running program MVA ***');
    writeln;
    writeln('> Reading data ...');
    ReadData(N,M,NServe,S,P,Dat);
    writeln('> Calculating visit ratios ...');
    VSolve(P,M,V);
    writeln('> Calculating performance measures ...');
    for I:=1 to M do begin;
        for K:=1 to N do
          if (K<NServe[I]) then C[I,K]:=K else
             C[I,K]:=NServe[I];
        Pr[old,I,0]:=1;
        Q[I,0]:=0;
    end; {for}
  for K:=1 to N do begin
      for I:=1 to M do begin
       if (NServe[I]>=N) then R[I,K]:=S[I] else
        if (NServe[I]=1) then
         R[I,K]:=S[I]*(1+Q[I,K-1]) else begin
         R[I,K]:=0;
           for J:=1 to K do
            R[I,K]:=R[I,K]+J*Pr[old,I,J-1]/C[I,J];
            R[I,K]:=R[I,K]*S[I];
         end; {else}
        end; {for}
      CT:=0;
      for I:=1 to M do CT:=CT+V[I]*R[I,K];
      Lambda:=K/CT;
      for I:=1 to M do begin
          Q[I,K]:=R[I,K]*Lambda*V[I];
          U[I,K]:=S[I]*Lambda*V[I];
        end; {for}
      for I:=1 to M do if (NServe[I]<N) then begin
       for J:=1 to K do Pr[new,I,J]:=U[I,K]*
            Pr[old,I,J-1]/C[I,J];
       Pr[new,I,0]:=1;
       for J:=1 to K do Pr[new,I,0]:=Pr[new,I,0]-
            Pr[new,I,J];
       if (Pr[new,I,0]<0) then Error(4,I,K);
       end; {for}
      Pr[old]:=Pr[new];
    end; {for}
  writeln('> Writing output ...');
  writeln(Out,'AVERAGE QUEUE LENGTHS
         (including service)');
  WriteOut(N,M,Q,Out);
  writeln(Out,'AVERAGE RESPONSE TIMES
         (including service)');
  WriteOut(N,M,R,Out);
  writeln(Out,'AVERAGE UTILIZATIONS');
  WriteOut(N,M,U,Out);
  close(Dat);
  close(Out);
  writeln;
  writeln('Program complete; output to file MVA.OUT');
  writeln;
end. {MVA}•
```

C.2.2 The Inferred Optimum.

Service Time

Queue	1	Ins1	2	Ins2	3	Ins3	4	Ins4	5	Ins5	6
Sec	5.0	2.0	5.0	2.0	5.0	2.0	5.0	2.0	5.0	2.0	5.0

One-Step Transition Probability Matrix

Queue	1	Ins1	2	Ins2	3	Ins3	4	Ins4	5	Ins5	6
1	0	1	0	0	0	0	0	0	0	0	0
Ins1	0	0	0.5	0	0.5	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0
Ins2	0.5	0	0	0	0	0	0.5	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	0
Ins3	0.5	0	0	0	0	0	0	0	0.5	0	0
4	0	0	0	0	0	0	0	1	0	0	0
Ins4	0	0	0.5	0	0	0	0	0	0	0	0.5
5	0	0	0	0	0	0	0	0	0	1	0
Ins5	0	0	0	0	0.5	0	0	0	0	0	0.5
6	1	0	0	0	0	0	0	0	0	0	0

Theoretical Performance (Including Service)

QUEUE	1	Ins1	2	Ins2	3	Ins3	4	Ins4	5	Ins5	6
SERVERS	99	99	99	99	99	99	99	99	99	99	99
AVERAGE QUEUE LENGTHS	6.618	2.647	4.412	1.765	4.412	1.765	2.206	0.882	2.206	0.882	2.206
AVERAGE RESPONSE TIMES	5.000	2.000	5.000	2.000	5.000	2.000	5.000	2.000	5.000	2.000	5.000
AVERAGE SERVER UTILIZATION	6.618	2.647	4.412	1.765	4.412	1.765	2.206	0.882	2.206	0.882	2.206

Forming Ratios for Initial Allocation (22 Machines)

Queue	Avgerage Response Time	Common Ratio	Allocating 25 Machines	Minimum Allocation
1	6.618	3	7.5	6
2	4.412	2	5	5
3	4.412	2	5	5
4	2.206	1	2.5	2
5	2.206	1	2.5	2
6	2.206	1	2.5	2

Identifying Most Probable Locations for Remaining Allocation (3 Machines)

SERVERS	6	99	5	99	5	99	2	99	2	99	2
AVERAGE QUEUE LENGTHS	6.092	1.914	3.594	1.276	3.594	1.276	3.659	0.638	3.659	0.638	3.659
AVERAGE RESPONSE TIMES	6.365	2.000	5.632	2.000	5.632	2.000	11.47	2.000	11.47	2.000	11.47
AVERAGE SERVER UTILIZATION	4.786	1.914	3.191	1.276	3.191	1.276	1.595	0.638	1.595	0.638	1.595

Identifiying Possible Improvement (Swap 1 Machine from Queue 2 to Queue 1)

SERVERS	6	99	5	99	5	99	3	99	3	99	3
AVERAGE QUEUE LENGTHS	8.096	2.156	4.349	1.437	4.349	1.437	2.246	0.719	2.246	0.719	2.246
AVERAGE RESPONSE TIMES	7.511	2.000	6.051	2.000	6.051	2.000	6.252	2.000	6.252	2.000	6.252
AVERAGE SERVER UTILIZATION	5.390	2.156	3.593	1.437	3.593	1.437	1.797	0.719	1.797	0.719	1.797

Total Time: 38.369 Seconds

Identifying Move Away from the Global Optimum

SERVERS	7	99	4	99	5	99	3	99	3	99	3
AVERAGE QUEUE LENGTHS	6.174	2.135	6.511	1.424	4.276	1.424	2.211	0.712	2.211	0.712	2.211
AVERAGE RESPONSE TIMES	5.783	2.000	9.148	2.000	6.007	2.000	6.213	2.000	6.213	2.000	6.213
AVERAGE SERVER UTILIZATION	5.338	2.135	3.559	1.424	3.559	1.424	1.779	0.712	1.779	0.712	1.779

Total Time: 39.577 Seconds

C.3 Data.

C.3.1 Comparing the Alternatives.

Geometric Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	3	31.15
3	6	5	5	3	3	3	.00
4	7	5	4	3	3	3	.53
5	7	4	6	3	2	3	4.07
6	6	6	6	2	2	2	8.80
7	8	4	4	3	3	3	3.30
8	7	5	4	3	3	3	.53
9	6	5	4	3	3	4	2.32
10	3	3	3	4	3	4	70.06
11	6	6	4	3	3	3	2.08
12	9	4	5	2	2	3	6.58
13	6	6	4	2	4	3	6.83
14	7	4	5	3	3	3	1.54
15	8	4	4	3	3	3	3.30
16	3	3	4	4	4	3	67.01
17	7	4	6	2	4	2	7.83
18	3	3	3	3	3	4	68.36
19	7	5	5	2	3	3	1.46
20	5	6	4	2	3	5	12.76
21	3	3	3	3	3	3	70.43
22	3	3	3	4	4	3	69.57
23	3	3	3	4	3	3	69.48
24	3	4	3	3	4	4	69.71
25	7	5	4	3	3	3	.53

Linear Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	2	29.88
3	8	4	4	3	3	3	3.30
4	7	4	4	3	4	3	4.29
5	6	6	5	3	3	2	3.07
6	6	5	5	5	2	2	5.29
7	7	4	5	2	4	3	5.73
8	6	6	4	3	3	3	2.08
9	7	4	4	3	3	3	4.14
10	3	3	3	4	3	4	70.06
11	7	5	4	3	3	3	.53
12	7	5	5	3	3	2	1.04
13	6	5	5	2	3	3	5.17
14	8	4	4	3	3	3	3.30
15	8	4	4	3	3	3	3.30
16	3	3	4	4	4	3	67.01
17	7	5	5	3	3	2	1.04
18	5	6	5	2	3	4	12.43
19	8	5	4	3	3	2	3.51
20	7	5	4	3	2	4	3.17
21	4	4	5	4	3	3	30.01
22	7	5	5	3	2	3	3.11
23	5	5	6	3	2	2	12.35
24	8	5	5	2	2	3	5.37
25	8	4	4	3	3	3	3.30

Adaptive Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	2	29.88
3	8	4	4	3	3	3	3.30
4	8	5	3	2	4	3	17.73
5	7	5	5	3	2	3	3.11
6	5	5	4	4	3	3	10.14
7	9	4	4	2	3	3	7.53
8	5	5	5	3	3	3	9.71
9	9	5	4	2	3	2	6.52
10	3	3	3	4	3	4	70.06
11	7	6	4	3	3	2	3.04
12	6	5	4	3	4	2	5.34
13	6	6	4	3	4	2	5.32
14	7	5	4	4	3	2	5.50
15	9	4	4	2	3	3	7.53
16	3	3	4	4	4	3	67.01
17	6	5	5	3	4	2	3.50
18	3	3	3	3	3	4	68.36
19	9	5	3	2	3	3	18.32
20	7	4	6	2	4	2	7.83
21	3	3	3	3	3	3	70.43
22	3	3	3	4	4	3	69.57
23	3	3	3	4	3	3	69.48
24	3	4	3	3	4	4	69.71
25	5	5	3	2	3	4	22.27

Elliptic Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	2	29.88
3	8	4	4	3	3	3	3.30
4	7	5	4	3	3	3	.53
5	7	4	6	3	2	3	4.07
6	9	5	4	2	2	3	7.25
7	8	4	4	3	3	3	3.30
8	7	5	4	3	3	3	.53
9	6	5	4	3	3	4	2.32
10	3	3	3	4	3	4	70.06
11	6	6	4	3	3	3	2.08
12	9	4	5	2	2	3	6.58
13	6	6	4	2	4	3	6.83
14	7	5	4	3	2	4	3.17
15	8	4	4	3	3	3	3.30
16	3	3	4	4	4	3	67.01
17	6	6	4	2	4	3	6.83
18	3	3	3	3	3	4	68.36
19	7	5	5	2	3	3	1.46
20	8	4	3	2	4	4	18.11
21	3	3	3	3	3	3	70.43
22	3	3	3	4	4	3	69.57
23	3	3	3	4	3	3	69.48
24	3	4	3	3	4	4	69.71
25	8	5	4	3	3	2	3.51

Logarithmic Temperature Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	3	31.15
3	8	4	4	3	3	3	3.30
4	6	5	4	3	3	4	2.32
5	7	5	5	2	3	3	1.46
6	3	4	4	4	3	4	67.93
7	3	4	4	4	4	3	71.89
8	6	5	4	3	4	3	4.16
9	9	4	3	2	4	3	19.62
10	4	4	3	3	2	4	32.67
11	7	5	4	3	3	3	.53
12	3	3	4	2	4	4	68.26
13	4	3	3	2	3	4	35.04
14	8	5	5	2	2	3	5.37
15	7	5	5	3	3	2	1.04
16	8	4	3	2	4	4	18.11
17	3	3	3	4	4	4	69.58
18	3	3	4	3	3	3	68.79
19	4	4	4	3	2	3	31.15
20	3	4	4	4	3	4	67.93
21	3	3	3	3	3	3	70.43
22	3	3	3	4	4	3	69.57
23	3	3	3	4	3	3	69.48
24	3	4	3	3	4	4	69.71
25	8	5	4	3	3	2	3.51

Linear Coefficient Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	4	5	3	3	3	1.54
2	7	5	5	4	2	2	3.78
3	8	4	4	3	3	3	3.30
4	7	5	4	3	3	3	.53
5	8	5	5	3	2	2	3.84
6	7	5	5	2	3	3	1.46
7	7	4	5	3	3	3	1.54
8	5	7	4	2	3	4	12.50
9	7	4	4	3	3	3	4.14
10	7	5	5	4	2	2	3.78
11	7	6	4	3	3	2	3.04
12	7	5	5	3	3	2	1.04
13	6	5	5	3	3	3	.00
14	8	5	4	3	2	3	4.34
15	7	5	5	2	3	3	1.46
16	7	5	5	3	3	2	1.04
17	7	5	6	2	2	3	5.26
18	8	5	4	3	3	2	3.51
19	7	5	5	3	2	3	3.11
20	8	5	5	2	2	3	5.37
21	6	5	5	3	3	3	.00
22	6	5	5	5	2	2	5.29
23	7	5	4	3	2	4	3.17
24	7	6	4	2	3	3	5.05
25	7	5	4	3	3	3	.53

Elliptic Coefficient Solution Quality at 100 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	4	5	3	3	3	1.54
2	8	5	5	3	2	2	3.84
3	8	4	4	3	3	3	3.30
4	6	5	5	3	3	3	.00
5	7	5	5	3	3	2	1.04
6	8	5	4	3	2	3	4.34
7	8	4	4	3	3	3	3.30
8	7	5	6	2	2	3	5.26
9	7	5	5	3	2	3	3.11
10	7	5	5	2	3	3	1.46
11	7	6	4	3	3	2	3.04
12	7	5	5	4	2	2	3.78
13	7	5	4	2	3	4	4.89
14	7	5	5	3	3	2	1.04
15	7	5	5	2	3	3	1.46
16	8	4	4	3	3	3	3.30
17	8	4	4	3	3	3	3.30
18	7	5	4	3	3	3	.53
19	7	5	5	3	2	3	3.11
20	8	5	4	3	3	2	3.51
21	7	5	5	3	2	3	3.11
22	7	5	4	3	2	3	3.95
23	7	5	5	2	3	3	1.46
24	7	5	5	2	3	3	1.46
25	8	5	5	3	2	2	3.84

Geometric Temperature Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	3	31.15
3	7	5	5	3	3	2	1.04
4	7	4	5	2	4	3	5.73
5	7	5	4	3	3	3	.53
6	3	4	4	4	3	2	67.93
7	3	4	4	4	4	3	71.89
8	6	5	5	3	3	3	.00
9	9	4	3	2	4	3	19.62
10	9	4	3	2	4	3	19.62

Linear Temperature Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	2	29.88
3	7	6	4	2	3	3	5.05
4	6	6	5	3	3	2	3.07
5	6	6	5	3	3	2	3.07
6	8	5	5	2	2	3	5.37
7	8	4	5	3	3	2	4.03
8	6	5	5	3	3	3	.00
9	8	5	4	2	3	2	6.03
10	3	3	3	4	3	4	70.06

Adaptive Temperature Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	2	29.88
3	7	5	4	3	3	3	.53
4	8	4	4	2	4	3	6.36
5	7	4	4	3	5	2	6.30
6	5	4	6	5	2	3	11.69
7	6	5	5	3	3	3	.00
8	7	5	4	3	2	4	3.17
9	8	5	4	2	3	2	6.03
10	3	3	3	4	3	4	70.06

Elliptic Temperature Solution Quality at 200 Trials

Run	[illegible]	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	2	29.88
3	7	5	5	2	3	3	1.46
4	8	4	5	3	3	2	4.03
5	7	5	4	3	3	3	.53
6	8	4	4	4	3	2	6.88
7	3	3	4	4	4	3	67.01
8	6	5	5	3	3	3	.00
9	7	6	3	2	4	3	19.67
10	6	4	5	3	2	4	6.28

Logarithmic Temperature Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	3	3	3	4	4	3	69.57
2	4	4	4	3	2	3	31.15
3	7	5	5	3	3	2	1.04
4	8	4	4	2	4	3	6.36
5	6	6	5	3	3	2	3.07
6	3	4	3	3	4	3	70.42
7	7	5	4	3	3	3	.53
8	7	5	5	3	3	2	1.04
9	4	4	4	4	3	3	31.63
10	4	4	4	3	2	2	29.88

Linear Coefficient Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	5	5	3	3	3	.00
2	7	5	4	3	3	3	.53
3	7	5	5	2	3	3	1.46
4	7	5	5	2	3	3	1.46
5	7	5	5	3	3	2	1.04
6	8	5	5	3	2	2	3.84
7	6	5	5	3	3	3	.00
8	6	5	5	3	3	3	.00
9	6	5	5	3	3	3	.00
10	8	5	4	3	3	2	3.51

Elliptic Coefficient Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	5	5	2	3	3	1.46
2	7	5	4	3	3	3	.53
3	7	5	5	3	3	2	1.04
4	7	5	5	2	3	3	1.46
5	8	5	4	3	3	2	3.51
6	8	5	5	3	2	2	3.84
7	8	5	5	3	2	2	3.84
8	7	5	5	3	3	2	1.04
9	7	6	4	3	3	2	3.04
10	6	5	5	3	3	3	.00

Linear Temperature with Linaer Coefficient Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	5	5	2	3	3	1.46
2	8	4	4	3	3	3	3.30
3	8	4	4	3	3	3	3.30
4	6	5	5	3	3	3	.00
5	7	6	5	3	2	2	3.74
6	8	5	5	3	2	2	3.84
7	7	5	5	3	2	3	3.11
8	6	5	5	3	3	3	.00
9	8	5	4	3	3	2	3.51
10	7	5	5	3	2	3	3.11

Geometric Temperature with Linaer Coefficient Solution Quality at 200 Trials

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	5	5	3	3	2	1.04
2	7	6	5	3	2	2	3.74
3	7	5	5	3	3	2	1.04
4	7	6	5	3	2	2	3.74
5	7	5	4	3	3	3	.53
6	6	5	5	3	3	3	.00
7	7	5	5	3	3	2	1.04
8	7	5	5	3	3	2	1.04
9	7	5	5	4	2	2	3.78
10	7	5	4	4	2	3	3.53

C.3.2 Comparing Local Search.

Local Search Efficiency (1 Iteration)					
Run	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	8	8	0	0	466.00
2	5	5	0	0	313.00
3	4	4	0	0	308.00
4	3	3	0	0	268.00
5	8	8	0	0	439.00
6	9	9	0	0	554.00
7	4	4	0	0	307.00
8	6	6	0	0	346.00
9	6	6	0	0	399.00
10	8	8	0	0	415.00
11	6	6	0	0	347.00
12	7	7	0	0	402.00
13	8	8	0	0	426.00
14	5	5	0	0	434.00
15	4	[illegible]	0	0	308.00
16	7	[illegible]	0	0	410.00
17	13	13	0	0	878.00
18	7	7	0	0	388.00
19	7	7	0	0	402.00
20	8	8	0	0	427.00

Local Search Efficiency (1 Iteration)					
Run	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
21	14	14	0	0	801.00
22	8	8	0	0	465.00
23	9	9	0	0	462.00
24	9	9	0	0	561.00
25	7	7	0	0	406.00
26	14	14	0	0	799.00
27	14	14	0	0	800.00
28	4	4	0	0	308.00
29	7	7	0	0	431.00
30	8	8	0	0	615.00
31	7	7	0	0	403.00
32	5	5	0	0	313.00
33	14	14	0	0	922.00
34	6	6	0	0	379.00
35	8	8	0	0	439.00
36	3	3	0	0	268.00
37	5	5	0	0	314.00
38	8	8	0	0	432.00
39	7	7	0	0	408.00
40	7	7	0	0	405.00

Local Search Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	4	5	4	4	2	5.73
2	8	4	4	3	3	3	3.30
3	5	4	5	3	4	4	11.07
4	6	4	4	4	4	3	4.94
5	7	4	4	3	3	4	3.56
6	6	5	4	4	3	3	3.49
7	5	4	5	3	4	4	11.07
8	7	4	4	3	3	4	3.56
9	7	4	4	3	3	4	3.56
10	6	5	4	3	3	4	2.32
11	6	6	4	3	3	3	2.08
12	7	4	4	3	3	4	3.56
13	8	4	3	2	4	4	18.11
14	6	4	4	4	3	4	4.97
15	5	4	5	3	4	4	11.07
16	7	4	4	4	4	2	6.36
17	7	5	5	2	3	3	1.46
18	6	5	4	3	3	4	2.32
19	7	4	4	3	3	4	3.56
20	8	4	3	2	4	4	18.11

Local Search Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
21	7	5	6	2	2	3	5.26
22	6	4	5	4	4	2	5.73
23	7	4	4	3	3	4	3.56
24	6	5	5	3	3	3	.00
25	6	4	5	3	4	3	2.86
26	7	5	6	2	2	3	5.26
27	7	5	6	2	2	3	5.26
28	5	4	5	3	4	4	11.07
29	6	4	6	3	2	4	5.61
30	9	4	4	2	3	3	7.53
31	7	4	4	3	3	4	3.56
32	8	4	4	3	3	3	3.30
33	7	5	5	2	3	3	1.46
34	6	4	5	3	4	3	2.86
35	7	4	4	3	3	4	3.56
36	6	4	4	4	4	3	4.94
37	8	4	4	3	3	3	3.30
38	7	4	4	4	4	2	6.36
39	7	4	4	3	3	4	3.56
40	6	4	5	3	4	3	2.86

Local Search Efficiency (3 Iterations)					
Run	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	8	8	0	0	1087.00
2	5	5	0	0	1261.00
3	4	4	0	0	1052.00
4	3	3	0	0	1164.00
5	8	8	0	0	1168.00
6	9	9	0	0	1676.00
7	4	4	0	0	1630.00
8	6	6	0	0	1488.00
9	6	6	0	0	2005.00
10	8	8	0	0	1354.00

Local Search Solution Quality (3 Iterations)							
Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	4	5	4	4	2	3.30
2	8	4	4	3	3	3	3.49
3	5	4	5	3	4	4	3.56
4	6	4	4	4	4	3	2.08
5	7	4	4	3	3	4	4.97
6	6	5	4	4	3	3	1.46
7	5	4	5	3	4	4	3.56
8	7	4	4	3	3	4	.00
9	7	4	4	3	3	4	2.86
10	6	5	4	3	3	4	5.61

Simulated Annealing with Logarithmic Cooling and
Linear Coefficient Efficiency (1 Iteration)

Run	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	25	12	6	7	934.00
2	21	11	4	6	934.00
3	47	12	10	25	1739.00
4	17	7	1	9	693.00
5	29	8	7	14	1119.00
6	24	12	0	12	1042.00
7	18	7	2	9	998.00
8	37	13	4	20	1739.00
9	10	7	0	3	425.00
10	13	9	0	4	510.00

Simulated Annealing with Logarithmic Cooling and
Linear Coefficient Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	6	5	5	3	3	3	.00
2	8	5	5	3	2	2	3.84
3	6	6	5	3	3	2	3.07
4	7	5	5	2	3	3	1.46
5	7	5	5	3	3	2	1.04
6	6	5	5	5	2	2	5.29
7	7	5	5	3	3	2	1.04
8	6	5	5	3	3	3	.00
9	7	6	4	3	3	2	3.04
10	6	6	4	2	3	4	6.08

Simulated Annealing with Logarithmic Cooling and
Elliptic Coefficient Efficiency (1 Iteration)

Run	Accepted Moves	Best Moves	Worst Moves	Better Moves	CPU Time
1	36	11	5	20	1844.00
2	34	14	4	16	1348.00
3	10	7	1	2	581.00
4	20	9	1	10	916.00
5	19	9	2	8	821.00
6	41	15	5	21	1656.00
7	19	9	3	7	923.00
8	10	8	0	2	583.00
9	29	9	4	16	1479.00
10	27	10	3	14	1227.00

Simulated Annealing with Logarithmic Cooling and
Elliptic Coefficient Solution Quality (1 Iteration)

Run	R1	R2	R3	R4	R5	R6	δ_i
1	7	5	5	2	3	3	1.46
2	7	4	5	3	3	3	1.54
3	7	4	5	3	3	3	1.54
4	7	5	4	3	2	4	3.17
5	7	4	5	4	3	2	4.03
6	7	4	5	3	3	3	1.54
7	6	5	5	3	3	3	.00
8	7	5	4	3	3	3	.53
9	8	5	4	3	3	2	3.51
10	7	5	5	2	3	3	1.46

Bibliography

1. Aldous, David. Probability Approximations via the Poisson Clumping Heuristic. New York: Springer-Verlag, 1989.

2. Andradottir, Sigrun. "Discrete Optimization in Simulation: A Method and Application," Proceedings of the 1992 Winter Simulation Conference: 483-486 (1992).

3. "Annealing," Encyclopedia Brittanica, 1: 429. Chicago: Encyclopedia Brittanica, Inc., 1990.

4. Azadivar, Farhad. "A Tutorial on Simulation Optimization," Proceedings of the 1992 Winter Simulation Conference: 198-204 (1992).

5. Bohachevsky, Ihor O., Mark E. Johnson, and Myron L. Stein. "Generalized Simulated Annealing for Function Optimization," Technometrics,28: 209-217 (August 1986).

6. "Boltzmann Constant," McGraw-Hill Encyclopedia of Physics: 83. New York: McGraw-Hill, 1983.

7. Cerny, V. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," Journal of Optimization Theory and Application, 45: 41-51 (January 1985).

8. Cheh, Kah Mun, Jeffrey B. Goldberg, and Ronald G. Askin. "A Note on the Effect of Neighborhood Structure in Simulated Annealing," Computers and Operations Research,18: 537-547 (January 1991).

9. Collins, N. E. "Simulated Annealing--An Annotated Bibliography," American Journal of Mathematical and Management Sciences, 8: 209-245 (1988).

10. Ferrara, Antonella and Riccardo Minciardi. "Resource Constrained Sheduling via Simulated Annealing: A Discrete Event Approach," Proceedings of the 1990 European Simulation Symposium: 177-181 (1990).

11. Gelfand, Saul B. and others. "Theory and Application of Annealing Algorithms for Continuous Optimization," Proceedings of the 1992 Winter Simulation Conference: 494-499 (1992).

12. Glynn, Peter W. "Optimization of Stochastic Systems," Proceedings of the 1986 Winter Simulation Conference: 52-59 (1986).

13. Haddock, Jorge and John Mittenthal. "Simulation Optimization using Simulated Annealing," Computers in Engineering, 22: 387-395 (1992).

14. Hartsfield, Nora and Gerhard Ringel. Pearls in Graph Theory: A Comprehensive Introduction. San Diego: Academic Press, Inc., 1990.

15. Johnson, David S., and others. "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," Operations Research, 37: 865-892 (November-Decenber 1989).

16. Johnson, David S., and others. "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning," Operations Research, 39: 378-406 (May-June 1991).

17. Kiemele, Mark J. and Stephen R. Schmidt. Basic Statistics: Tools for Continuous Improvement. Colorado Springs: Air Academy Press, 1990.

18. Kirkpatrick, S., C. D. Gellat, Jr., and M. P. Vecchi. "Optimization by Simulated Annealing," Science, 220: 671-680 (13 May 1983).

19. Larson, Richard J. and Morris L. Marx. An Introduction to Statistics and its Applications, 2nd Edition, New Jersey: Prentice-Hall, 1986.

20. Lin, S. "Heuristic Programming as an Aid to Network Design," Networks, 5: 33-43 (1975).

21. Makridakis, Spyros and others. Forecasting Methods and Applications, 2nd Ed. New York: John Wiley and Sons, 1983.

22. Metropolis, Nicholas, and others. "Equation of State Calculations by Fast Computing Machines," The Journal of Chemical Physics, 21: 1087-1092 (June 1953).

23. Mihram, G. Arthur. Simulation: Statistical Foundations and Methodology. New York: Academic Press, 1972.

24. Pegden, Dennis C. and Michael P. Gately. "A Decision-Optimization Module for SLAM," Simulation, 34: 18-25 (1980).

25. Pritsker, A. Alan B. Simulation and SLAM II, 3rd Edition. New York: Systems Publishing Corporation, 1986.

26. Romeo, Fabio, and Alberto Sangiovanni-Vincentelli. "A Theoretical Framework for Simulated Annealing," Algorithmica, 6: 302-345 (1991).

27. Ross, Sheldon M. Introduction to Probability Models. San Diego: Academic Press, Inc., 1989.

28. Tovey, Craig A. "Simulated Simulated Annealing," American Journal of Mathematical and Management Sciences, 8: 389-407 (1988).

29. van Laarhoven, P. J. M., and E. H. L. Aarts. Simulated Annealing: Theory and Applications. Dordrecht: D. Reidel Publishing Company, 1987.

30. van Laarhoven, P. J. M., and others. "Job Shop Scheduling by Simulated Annealing," Operations Research, 40: 113-125 (Jan-Feb 1992).

31. Wong, D. F., H. W. Leong, and C. L. Liu. Simulated Annealing for VLSI Design. Boston: Kluwer Academic Publishers, 1988.

32. Yan, Di and H. Mukai. "Stochastic Discrete Optimization," SIAM Journal of Control and Optimization, 30: 594-612 (May 1992).

33. Yao, Xin. "Simulated Annealing with Extended Neighborhood," International Journal of Computer Mathematics, 40: 169-189 (1991).

Vita

Captain Charles B. Warrender was born on 15 September 1961 in Wichita, Kansas. He graduated from high school in Evergreen, Colorado in 1980 and attended the United States Air Force Academy. He received the degree of Bachelor of Science in Mathematical Sciences in May 1984. Upon graduation from the Academy, he entered Space Operations training and received the Distinguished Graduate award. He participated as a cadre member among the first Air Force officers to conduct crew operations for the Global Positioning System (GPS). He developed many of the simulator training scenarios for qualification training and certification as lead Planning and Analysis Evaluations Officer. He developed and conducted launch and early orbit training scenarios as a member of the launch rehearsal committee. After serving in this capacity for three years, he transferred to the Combat Crew Training Squadron where he took charge of the GPS Initial Qualification Training branch. He transitioned the training program from a contractor-taught course to an Air Force course, saving the contractor's expense, halving the training time, and standardizing the instructional material. Two years later he assumed command of the Simulation Development Flight where he supervised the development, maintenance, and employment of training scenarios for seven satellite programs. In May of 1992 he entered the Graduate School of Engineering, Air Force Institute of Technology, in pursuit of a Master's degree in Space Operations. He is a member of Tau Beta Pi.

Permanent address: 31678 Gallery Lane
Evergreen, Colorado 80439

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
December 1993

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE

THE APPLICATION OF SIMULATED ANNEALING
TO STOCHASTIC SYSTEMS

5. FUNDING NUMBERS

6. AUTHOR(S)

Charles B. Warrender, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION
REPORT NUMBER

AFIT/GSO/ENS/93D-16

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Simulated Annealing was used to optimize three constrained simulation models. For each of these models, seven different acceptance functions were evaluated and compared against the performance of Local Search. These comparisons demonstrated the effect that different acceptance functions have on the performance of the algorithm. The performance was measured by the average solution quality and average efficiency.

The first model facilitated the implementation of Simulated Annealing using the SLAM simulation language. The configuration space was described by only two decision variables. It demonstrated the viability of using Simulated Annealing to optimize the variable settings in a simulation model. The second model (with six decision variables) provided greater insight to the advantages and limitations of Simulated Annealing. This model was implemented as an open queuing network. The third model, similar to the second, was implemented as a closed queuing network. The results from this variation were completely unexpected. They showed a wide performance separation among the different acceptance functions that was not present in the first two models.

No attempt was made to justify the use of Simulated Annealing from a theoretical perspective. Rather, empirical results from the three models were used to infer the practical utility of the algorithm.

14. SUBJECT TERMS

Simulated Annealing, Simulation, Stochastic Models, Optimization

15. NUMBER OF PAGES
219

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL