THE APPLICATION OF SIMULATED ANNEALING TO THE MIXED MODEL, DETERMINISTIC ASSEMBLY LINE BALANCING PROBLEM

by

Sherry L. Edwards

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Industrial and Systems Engineering

APPROVED:

Dr. O. K. Eyada, Chairman

Dr. S. C. Sarin

Dr. R. T. Sumichrast

July 1993

Blacksburg, Virginia

The Application of Simulated Annealing to the Mixed Model, Deterministic Assembly Line Balancing Problem

by

Sherry L. Edwards

Dr. Osama K. Eyada, Chairman

Industrial and Systems Engineering

(ABSTRACT)

With the trend towards greater product customization and shorter delivery time, the use of mixed model assembly lines is increasing. A line balancing approach is needed that can address the complex nature of the mixed model line and produce near-optimal solutions to problems of realistic size. Due to the combinatorial nature of the line balancing problem, exact solution techniques are limited to small problems. Heuristic methods, on the other hand, are often too simplistic to find good solutions. Furthermore, many of the existing techniques cannot be expanded to handle the mixed model problem.

Simulated Annealing (SA) is a search methodology which has exhibited good results when applied to combinatorial optimization problems. In fact, researchers have found that SA is able to find near-optimal solutions while its processing time increases only as a polynomial function of problem size. However, none of the applications found in the literature fully explore the technique's ability to handle a highly-constrained problem such as line balancing.

The objectives of this research, therefore, were to assess the feasibility and effectiveness of applying the simulated annealing technique to the mixed-model line balancing problem, and to propose a set of guidelines for applying the technique to problems with different characteristics. Achieving these objectives involved investigating the effects of the algorithm's parameters on solution quality and speed of execution and testing the algorithm's flexibility and robustness using problems of differing size and complexity.

The results of the research indicate that simulated annealing is quite effective for solving the mixed model line balancing problem. The technique is capable both of modeling the more complicated mixed model objective function and of handling the highly-constrained nature of the line balancing problem. SA was found to be particularly well suited for large line balancing problems, as it provides a solution to the problem in a fraction of the time required for an exhaustive search.

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Drs. Eyada, Sarin, and Sumichrast, for their time and helpful advice. In particular, I would like to thank my advisor, Dr. Eyada, for believing in me, for treating me with respect, for being a friend as well as an advisor, and for always being available when I needed him.

A special thanks is given to Dave Nicks for providing me with portions of the C code for the exhaustive search algorithm, and also for sharing his laser printer. In addition, my other officemates–especially Janis Terpenny, Jin Ong, and Jeff Fithian–deserve a hearty thank you for listening to me and being my friends.

I further acknowledge my parents, who have always been a strong motivational force behind my success. Each has supported me in his/her own unique way, yet a special thank you should go to my mother for maintaining unwavering confidence in me and for insisting that I strive to be the best I can be.

Finally, I would like to thank Jim Heron for his love and loyal support. His willingness to listen patiently to me throughout my time in Blacksburg was enormously helpful. I greatly appreciate all of his valuable ideas and suggestions. Furthermore, Jim helped to instill in me the confidence to believe in myself and my ideas.

TABLE OF CONTENTS

CHAPTER 3: METHODOLOGY

CHAPTER 4: EXPERIMENTAL RESULTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 ASSEMBLY LINES

Many manufactured products require the assembly of individually-processed components. When the demand for the product is small, or when the assembly requires only a few steps, the assembly is usually performed by one individual at a fixed location. When the product is being mass produced, however, and the assembly is more complex, assembly lines are desirable.

On an assembly line, work is divided into groups of tasks which are assigned to stations along the line. The worker (or machine) at each station is responsible for only a portion of the overall assembly. As a result, the amount of training needed is minimized, each worker becomes very skilled at his/her job, there is no duplication of tools along the line, and materials need to be delivered to only one location on the line.

Kalpakjian (1989) states that assembly costs on manual lines typically comprise 25-50 percent of total manufacturing cost. Thus, improvement in this area would greatly benefit a manufacturing organization.

1.2 DESIGN CONSIDERATIONS FOR ASSEMBLY LINES

There are many objectives which should be considered in assembly line design. One objective is minimizing cost, which includes costs associated with labor, physical space, inventory, and quality. Other objectives include maximizing flexibility, reliability,

quality, and throughput, minimizing manufacturing lead time, and decreasing the system's sensitivity to variation.

Many design alternatives exist which allow a manufacturing organization to develop a line to meet the objectives which are most important to its operations. One such design decision is whether the work will be performed by humans or by machines or by a combination of both. Another decision is whether the material transfer system will be manual or automatic. If the transfer system is mechanized, it can be continuous, synchronous, or asynchronous. In a continuous line, the product constantly moves and the worker may need to move along with the product. A synchronous, or indexed, line moves all of the partial assemblies to their next stations simultaneously. The assemblies then remain stationary until the next move takes place. An asynchronous transfer system differs in that the worker can release the assembly whenever he/she is finished with the work and is therefore not constrained by the handling system. Asynchronous lines typically have buffer storage between the stations to reduce the effect of variability in worker task times.

A third design decision involves the number of models which will be produced on the line. If the line is dedicated to one model only, it is known as a single model line. If the line produces several models, it is known as either a multi-model or mixed model line. If the models are produced in batches (for example, a batch of one model is produced, the line is stopped and the setup changed, and a batch of another model is run), then the line is a multi-model type. If, however, the line can produce several models without setup changes between models, it is a mixed-model line. Models can be

sequenced onto a mixed model line in any order, and therefore this type of line can produce according to demand fluctuations. This aspect of mixed model lines results in quick response to customer need and reduced inventory of both work-in-process and finished goods. Nonetheless, mixed model lines inherently have inefficiencies due to the differences in work times for each model. Depending on how the models are sequenced, some operators will be idle at times and some may have more work than they can do in the available station time.

A fourth decision, which is only made in the case of a continuous, manual line, is whether the stations will be open or closed. If the stations are closed, this means that the workers must complete their tasks within a fixed distance along the line. Open stations, however, allow the workers flexibility to move outside of the stations' limits by a certain distance. Closed stations are sometimes dictated by equipment or plant layout constraints.

Several other decisions exist, such as whether stations should be duplicated (parallel stations), whether multiple workers should be allowed to man a station (expanded stations), and whether incomplete assemblies should be completed on- or off-line. Once the basic structure of the line has been chosen, there are more design decisions to be made in order to optimize the selected system to minimize cost and maximize quality.

From the above discussion, it can be concluded that the design of assembly lines is quite complex. Therefore, the scope of the problem to be addressed by this thesis is limited to the case of continuous, mixed-model lines with manual work. This type of line is frequently used in the assembly of automobiles, large domestic appliances, and high-

volume electronics products [Buzacott, 1993]. The focus of this thesis, then, is on one of the design decisions necessary to optimize a continuous, mixed-model, manual-work assembly line. This decision is commonly referred to as the line balancing problem.

1.3 LINE BALANCING

The assembly of any product can be broken down into a finite number of tasks, known as work elements. These work elements must be allocated to the stations along the assembly line. This allocation is the focus of line balancing.

1.3.1 Objectives for Line Balancing

The objective of line balancing is to assign assembly tasks, or work elements, to stations in a feasible, efficient way. For the balance to be feasible, the work elements must be assigned such that they do not violate the precedence constraints, which are a set of rules about the order in which elements are performed. An example of a precedence constraint is the restriction that lug nuts cannot be tightened until after the tire has been placed on the wheel of a car.

The other constraint on the line balance is either a minimum production rate or a maximum number of stations. One of these two design aspects is fixed, and the other is to be optimized. In other words, if a particular production rate is required, the objective of line balancing is to find an allocation of elements to stations which requires the least number of stations. This allocation will reduce labor cost and will also reduce idle time. On the other hand, if the number of stations is fixed, the objective will be to

find an allocation which maximizes the production rate. This solution will reduce idle time. These two alternate formulations of the line balancing problem are commonly recognized, and the first formulation (minimizing the number of stations) is the more frequently used. The first formulation will be used in this thesis.

For a given problem, there may be several solutions which require the minimal number of stations. From these solutions, one would like to choose the one which results in the most efficient production. This will be discussed in greater detail later in this thesis. However, the perfectly efficient line is one in which the work elements divide into the stations such that each station requires the same amount of time. In this line, there will be no time when operators are idle (assuming deterministic work times).

1.3.2 Special Considerations for Mixed Model Lines

The problem of line balancing becomes significantly more difficult when the assembly line produces several models. Mixed model lines are designed such that all models can run on the line and setups are not needed between models. However, since the models differ somewhat in the tasks required and the time required to perform the tasks, there is likely to be inefficiency in the design of the line.

Early solutions for balancing mixed model lines involved balancing each model separately. Thus, the operator at a given station might perform different tasks depending on which model arrived at the station. This would mean duplication of training, raw materials, and tools.

A solution to this problem was proposed by Thomopoulos in 1967. Thomopoulos' concept was to consider the work to be done on an entire shift, and to allocate work elements to stations on a shift basis as opposed to a cycle time basis. This approach involves developing a fictitious aggregate model, using the relative frequencies of production of each model to determine a weighted average of the time required for each work element. This model is then used to balance the line.

1.4 SIMULATED ANNEALING

Simulated Annealing (SA) is a heuristic search approach to solving combinatorial optimization problems. The idea was developed by Kirkpatrick, Gelatt, Jr., and Vecchi in 1983 and resulted from a concept used in statistical mechanics (a branch of condensed matter physics). In statistical mechanics, annealing is used to study the behavior of atoms at low temperatures. As a liquid solidifies, the atoms will form crystals if they can reach the ground-state, or the lowest energy state. Thus, by growing a crystal (through the annealing process) one can determine the low-energy state. The annealing process involves melting the substance, and then cooling the substance slowly, with the time between successive temperature drops increasing as the temperature approaches the freezing point.

Kirkpatrick, et al. drew an analogy between finding the minimum amount of energy in a group of atoms and finding the minimum of the objective function of a combinatorial optimization problem. Three significant benefits of the simulated annealing search procedure are: (1) the approach avoids local optima, (2) computational effort

increases slowly with increasing problem size (effort scales as N or a small power of N, where N is the number of atoms or other elements to be arranged) [Kirkpatrick, 1983], and (3) the solution of an SA problem does not strongly depend on the initial solution [Aarts and Korst, 1989]. The literature indicates that simulated annealing has been successfully applied to a large number of different problems, including facility location, plant layout, AGV routing, the traveling salesman problem, and integrated circuit design. However, in all of these applications, either constraints were not addressed, or only one type of constraint was considered. In contrast to these problems, the mixed model assembly line balancing problem is highly constrained, requiring two different types of constraints.

1.5 PROBLEM DEFINITION

The assembly line balancing problem has been formulated by integer programming, mixed integer programming, and dynamic programming models. These methods are not very effective for problems of realistically-large size because as the size of the problem grows, the number of possible solutions grows exponentially.

> "There is a class of combinatorial optimization problems which has defied solution by efficient algorithms. The travelling salesman problem, job shop scheduling and line balancing all fall into this class. These problems all have a finite but extremely large number of feasible solutions, so that total enumeration is impractical. Furthermore, they all may be formulated as integer linear programs, but unfortunately this approach does not lead to efficient algorithms." [Gutjahr and Nemhauser, 1964]

Other exact methods exist, such as branch and bound and shortest route algorithms, but these too become quite cumbersome as the problem becomes large.

Furthermore, the techniques used to solve the simple, one-model line balancing problem are often not sufficient to address the mixed model case with its more complex objective function. Smith (1990) and Pantovanous (1992) developed computerized methodologies to exhaustively search the solution space, and these techniques use Thomopoulos' (1970) objective function which was specifically designed for the mixed model case. However, these exhaustive searches also require a great deal of processing time. Ghosh and Gagnon (1989) state:

> "Despite the extensive research on the basic [single model, deterministic work element time] problem, optimal solution procedures capable of solving realistically complex, large-scale problems within existing computer computational restrictions are rare. ...All exact algorithms become intractable with increasing problem size."

Some heuristic methods, including those by Arcus (1966) and Smith (1990), have been developed to address the mixed model problem; however, by definition, these are not guaranteed to produce an optimal solution. These techniques can be unpredictable and are frequently very dependent on the particular problem instance and on the choice of initial solution.

Although simulated annealing has been hailed for its generality, simplicity, robustness, and success in producing high-quality solutions for large-scale combinatorial problems [Aarts and Korst, 1989, and Ku and Karimi, 1991], no one has attempted to use the approach for line balancing. This is perhaps due to the large number of constraints inherent in this problem.

1.6 RESEARCH OBJECTIVES

The research objective was to determine the effectiveness and appropriateness of using simulated annealing to solve the mixed model, deterministic assembly line balancing problem. This involved assessing the feasibility of applying the technique to the line balancing problem. It also involved investigating the effects of the algorithm's parameters on the quality of the solution and the speed of execution. The algorithm was tested for flexibility and robustness using problems of differing size and complexity. Based on the results, general guidelines were developed for applying simulated annealing to mixed model, deterministic line balancing problems.

1.7 THESIS OUTLINE

Chapter 2 provides a review of both line balancing and simulated annealing literature. In Section 2.1, line balancing is discussed in great detail, including both single model and mixed model line balancing techniques assuming deterministic work element times. The review includes the very earliest as well as recent works in this area. In Section 2.2, the simulated annealing concept is described in detail, and then various applications of the technique to problems similar in nature to line balancing are reviewed. Both sections conclude with a summary of the research efforts to date.

Chapter 3 describes the proposed methodology for achieving the research objectives. First, the method for developing the initial formulation of the simulated annealing problem is discussed. The selection of certain techniques is justified, the coding of the algorithm is explained, and the results from initial experimentation are

presented. Next, a set of experimental problems is presented and justified. Finally, the experimentation plan is described for optimization of the algorithm and development of a framework for applying simulated annealing to mixed model line balancing problems.

Chapter 4 provides results from the exhaustive searches and the simulated annealing experiments. Since exhaustive search results were not obtainable for the larger problems, the heuristic technique suggested by Smith (1990) was performed for these problems to provide a basis for comparison. Chapter 5 presents conclusions from the research findings and provides suggestions for future work.

CHAPTER 2: LITERATURE REVIEW

2.1 ASSEMBLY LINE BALANCING

2.1.1 Single Model Assembly Line Balancing

2.1.1.1 Exact methods

Salveson (1955) was the first to publish in the area of assembly line balancing. He formulated the problem and introduced much of the commonly used terminology. He introduced the concept of the precedence diagram, which is a means of displaying the work elements and their precedence relationships. He also proved that if elements are grouped into stations such that between stations no precedence constraints are violated, then the elements can be organized within the station such that no precedence constraints are violated. Salveson developed a linear programming (LP) model for the single model assembly line balancing problem, which could be solved by the simplex method. The problem with linear programming formulations is that they can produce solutions where elements are split between stations, which are impractical if not impossible.

Bowman (1960) developed two integer programming models which were improvements over Salveson's LP model because they prevent elements from being split between stations. The decision variables of the first model represent the number of time units devoted to a task at a station. The other model's decision variables represent the clock time when tasks start. In the second formulation, Bowman used zero-one variables to insure that no tasks use the same clock time. Kilbridge and Wester (1962) offer the following critique:

> "The constraints and objective function are set up easily, but the computations required for a balancing problem of even modest size are considerable, so that Bowman's analytical approach to the problem is of more academic than practical value."

White (1961) modified Bowman's model to use a different decision variable, namely a zero-one variable which indicates if task i is assigned to station j, for all i and j. Thangavelu and Shetty (1971) as well as Patterson and Albracht (1975) made improvements to the Bowman-White model in both the objective function and the constraints. Talbot and Patterson (1984) proposed a formulation which did not require zero-one variables, by letting the decision variables represent the number of the station to which each task i is assigned. Despite these improvements, it has been noted that while integer and dynamic programming formulations have been extensively used to express the assembly line balancing problem, neither can be efficiently solved [Ghosh and Gagnon, 1989]. This is due to the combinatorial complexity of the problem.

Jackson (1956) was the first to formulate a dynamic programming (DP) model to solve the single-model assembly line balancing problem (SALBP). In Jackson's procedure, all possible assignments for the first station are generated. For each of these assignments, the feasible assignments for the second station are generated. For each of these first-second station assignments, the feasible third-station assignments are generated, and so forth until an optimal solution is found. Clearly, the computational and storage requirements of this method are substantial [Baybars, 1986]. Although the algorithm will provide an optimal solution, it is impractical for realistic problems [Kilbridge & Wester, 1962].

Held et al. (1963) developed a dynamic programming model for small line balancing problems. They also proposed an iterative procedure for approximating the solution to larger problems. They programmed the exact algorithm for an IBM 7090, and noted that it balanced a 36 task line in 20 seconds. However, storage limitations were exceeded for larger problems, and so a technique for approximation was necessary. The approximation technique grouped the N tasks into groups of tasks, which were used in place of the original tasks in the algorithm.

According to Ghosh & Gagnon, although Schrage and Baker (1978) and Kao and Queyranne (1982) made improvements in DP methods in terms of time and storage requirements, DP approaches are computationally intractable. "This problem is true for all DP methods: the computational demands of a DP method grow at an exponential rate with increasing problem size." [Baybars, 1986]

Betts & Mahmoud (1989) developed a branch and bound algorithm which was an improvement over the one presented by Johnson in 1981. The first step in this approach is to generate all possible assignments for the first station. Solutions which are subsets of other solutions are eliminated. The solution which results in the least amount of idle time is then selected and the other solutions are kept as "waiting nodes." The best solution from station one is then used to begin filling station two. The stations are filled one at a time in the same manner, each time branching off from only the best solution and keeping the rest as waiting nodes. Once all the work elements have been assigned, the number of stations required by this solution is compared with the pre-calculated minimum number of stations. If the two are equal, the solution is considered optimal.

Otherwise, one backtracks to investigate the waiting nodes to see if a solution exists with fewer stations.

To reduce the number of branches which must be investigated, 'bounds' are used. As stations are filled, the lower bound on the number of stations required on each branch (or path) is calculated to determine which branches should be pruned. Once pruned, the branches will not be investigated further.

A very important thing to note with this methodology is the rather loose definition of optimal. Since any balance which requires only the minimum number of stations is considered optimal, there are likely to be a great many 'optimal' solutions. This may explain the small processing times associated with this technique. Nonetheless, the processing time required will be very much a function of the problem, and thus the method could perform very quickly or very slowly. In the case of very large problems, there may be a problem with memory storage. A final aspect to note is that this technique can only work if a goal is known. In other words, it cannot find the 'best' balance; it only works if it knows when it has reached a solution that is good enough to warrant stopping. Without a goal, this method becomes an exhaustive search, which will be quite time-consuming.

Mansoor (1964) proposed an exact solution method which builds on a heuristic technique developed by Helgeson and Birnie in 1961. Helgeson and Birnie's technique, called the Ranked Positional Weight (RPW) method, fills the stations one at a time, selecting first those elements whose successors in the precedence diagram require the most time. Mansoor uses the RPW method but backtracks when necessary. Specifically,

while assigning stations via RPW, Mansoor keeps track of the total amount of idle time in the assignment. When a certain threshold value of idle time is exceeded (indicating a poor solution), he backtracks by removing elements one at a time, starting with the last element. He then resumes the process of adding elements. However, if this still produces a poor solution, more elements must be removed, and so forth. It should be obvious that a very large amount of computational time would be needed for all but the smallest of problems.

2.1.1.2 Heuristic Methods

Although Salveson was the first to formally publish in the area of line balancing, Benjamin Bryton, in his master's thesis, was the first to suggest an analytical procedure for assembly line balancing. Bryton's method involves creating an initial solution and then interchanging work elements between stations until no further improvement is found. The station with the largest assigned time as well as the station with the least assigned time are chosen for possible interchange of elements. An element from each of these two stations is selected for interchange on the basis that their time difference is the closest to one-half of the difference between the two stations' times. Elements are interchanged according to this rule until no improvement is made.

This method, even though it was the first procedure developed for line balancing, is similar to the simulated annealing procedure used in this thesis. This is not particularly surprising, as one of the heralded benefits of simulated annealing is its simplicity. However, Bryton's method has serious drawbacks that simulated annealing does not have.

For example, Bryton's procedure is very likely to find only a local minimum. This is because the rules for interchanging elements are very restrictive, preventing the consideration of a large number of feasible, and possibly better, balances. Also, only interchanges which result in improved solutions will be accepted, thus many feasible solutions will be overlooked, and the procedure will be very heavily dependent on the selection of the initial solution. A further disadvantage is that the complexity of the rules for transferring elements increases the computational requirements.

It should also be noted that Bryton's algorithm solves a different objective than the one in this thesis. Bryton chooses a set number of stations and tries to minimize the balance delay. This thesis instead fixes the desired production rate and minimizes the number of stations, and thus the production cost. This is a more realistic objective, as rarely in industry is the number of stations fixed.

Kilbridge and Wester (1961) developed a method in which elements are assigned to stations serially, meaning that once the first station is filled, the next station is filled, and so on. The procedure involves placing elements in columns according to their position in the precedence diagram, then assigning elements into stations column by column in an effort to achieve the ideal station time. The lack of specific rules makes this method dependent on the user's skill and confines it to small problems. Also, the method is not guaranteed to work well with all problems. For example, problems with small cycle times will require more computational effort. Kilbridge and Wester's method, as with all serial methods, is likely to find only a local minimum. In serial methods, once an element is placed, no other placements will be considered. Thus, an enormous

number of feasible, and potentially better, solutions will be overlooked. Therefore, serial methods in general produce relatively poor solutions. The tradeoff, however, is speed, since serial methods are quick to perform because of their simplicity.

Helgeson & Birnie (1961) initiated the Ranked Positional Weight (RPW) technique, which is one of the most common heuristics used in industry [Ghosh and Gagnon, 1989]. The theory behind this approach is that elements whose successors require the largest amount of time should be placed first. Elements are given a weight which is the sum of the element times for all elements which are successors. Then elements are assigned to stations, station by station, according to decreasing weight. This procedure is not guaranteed to produce an optimal solution, but it is easy to computerize. This method is a serial method, and serial methods have already been characterized as relatively poor.

A modification of the Ranked Positional Weight technique is the Largest Candidate Rule, developed by Moodie and Young (1965). Rather than assigning elements based on the time requirements of their successors, this heuristic assigns elements on the basis of their own time requirements. Given the set of feasible elements that will fit into a station, the heuristic rule is to select the element which requires the most time to perform. Once an initial balance is obtained, Moodie and Young suggest improving it by shifting tasks between stations to reduce idle time.

Moodie and Young's trade and transfer methodology is rather complex, and is as follows. The first step is to calculate half the difference between the largest and smallest stations' total station times and call this 'goal.' Next, consider all elements in the largest

station that have times less than twice the value of 'goal' and would not violate precedence relations if transferred to the smallest station. Determine all the possible trades between the largest and smallest stations such that the resulting increase in the small station and decrease in the large station is less than twice the value of 'goal.' The final step is to execute the trade or transfer for the element which is closest in absolute difference to 'goal.'

If there is no such trade or transfer possible between the smallest and largest stations, Moodie and Young suggest a search of all station combinations, as follows: beginning with station one, try each of its elements with the elements in station n, then with station n-1, then station n-2, and so forth. Next, try station two with every other station, and so forth. If there is still no possible trade or transfer after trying every possible combination, the authors suggest dropping the 'goal' rule and accepting any feasible trade or transfer. They further suggest that this procedure can be repeated although they do not say how many iterations should be performed.

There are flaws in this methodology. First, there is no stopping criterion. One of the examples given continues to iterate until the optimum objective function value is reached. This may require quite a significant amount of computation time, since for each iteration many operations and computations must be performed. In fact, the method seems to be essentially an exhaustive search. Thus, the amount of computation time required is likely to grow exponentially with the problem size. Also, if an exhaustive search is not performed, then the structured rules for trades and transfers may make this algorithm dependent on its initial solution.

The method allows the acceptance of worse solutions, and this technique has merit. However, with no stopping methodology, the algorithm could be stopped after the acceptance of a balance which is worse than previous balances. One additional comment is that the objective function used for the trade and transfer procedure is related to, but not the same as, the objective function for the best balance.

Tonge (1961) developed a heuristic methodology which is somewhat similar to Moodie and Young's procedure, except Tonge's method involves three phases. Phase One includes generating a tree of all the feasible sets (a set is a group of tasks with same predecessors and followers which can be performed in any order) and all chains (a chain is a group of tasks with same predecessors and followers but must be performed in specific order). Phase Two involves determining the lower bound on the number of stations and trying to fit the tasks into this number of stations using five heuristic rules. If the tasks do not fit, the number of stations is increased until they will fit. Phase Three involves rearranging tasks to reduce the cycle time of the solution obtained from Phase Two (while keeping the same number of stations). This method is good in that it attempts to both reduce the number of stations and even the workload over the stations. However, the final solution is likely to be quite dependent on the initial solution.

In 1973, Dar-El presented a heuristic version of the exact backtracking algorithm he developed in 1964. The heuristic method is called MALB, and seems to be rather successful. It follows the same basic logic as the exact method from 1964, but there are many heuristic rules which are employed to significantly reduce the number of backtracking operations. The objective function used in MALB is to minimize the cycle

time for a given number of stations. The algorithm determines the theoretical minimum cycle time and attempts to find a feasible assignment of elements that fits in this cycle time. If no feasible solution exists, the cycle time is increased until a solution is found.

Dar-El states that MALB dominates COMSOAL and 10-SP (10-SP is a heuristic which selects the best solution from ten single pass attempts, each using different ranking methods for assigning stations, e.g., RPW, Largest Candidate Rule, etc.). He claims that the algorithm is "extremely fast," but he does not provide data to prove this. It would also be desirable to know how fast the algorithm is when tackling large problems. This method is not guaranteed to produce a near-optimal solution. It is likely to find only a local optima since it does not accept 'poor' partial solutions. Also, the heuristic rules used to limit the backtracking make this method complex.

COMSOAL (Arcus, 1966) is a computerized heuristic approach which is best known for loosening the strict assumptions that most of the other line balancing methods make. Thus, COMSOAL can more readily address realistic industry situations. However, the basic procedure for generating balances is somewhat flawed. The program generates randomly a fixed number of sequences (usually 1000) and then chooses the one requiring the fewest stations. Stations are filled as follows: if there are ten tasks which could be assigned to the station, each has probability of one-tenth of being selected. The remaining nine have probability of one-ninth of being selected next, and so forth. Arcus also experimented with weighting elements according to their times and/or number of successor elements.

The obvious problem with this method is that it considers only a small fraction of the possible solutions, so achieving the optimal solution is unlikely. Also, there is no intelligent mechanism for moving from one solution to the other--solutions are completely independent. Thus, the benefits of a good arrangement will not be acknowledged and improved upon. A potential advantage of this method is speed. This advantage is offset, however, by the fact that without any intelligent search procedure, a very large number of random solutions will need to be generated in order to find a near-optimal solution.

Another problem with this method is that it does not prohibit a given sequence from being generated more than once. In addition, several sequences may be generated which are, in effect, the same balance. This is because the elements assigned to a particular station could be assigned in many different orders without affecting the quality of the balance. Since Arcus' algorithm does not avoid this duplication, it wastes a certain amount of computational time.

Arcus recognized the fact that his algorithm generates the same solution repeatedly, and attempted to deal with this by finding several feasible, but different, first-station assignments at a time and then proceeding to fill the rest of the stations. To do this, though, he filled the first stations to maximum potential. Arcus admits that this procedure may overlook many good solutions.

Arcus states that optimality may be hard to define. He says that a solution is optimal if the total idle time (defined as "the total available time of all workers during one work cycle, minus the total standard time of all tasks") is less than the cycle time.

Others would argue that a solution is not optimal unless the idle time is zero. Thus, one must be careful in comparing the results of different researchers.

Dar-El and Rubinovitch (1979) developed MUST, another computerized procedure. MUST stands for 'MUltiple Solutions Technique,' and thus one of its primary advantages is that it produces several good solutions as opposed to many other packages which only provide the best solution. The flexibility of selecting between several good solutions may be needed in cases where certain operations can only be performed at certain stations, or when the user prefers to group certain elements together. This is certainly a convenient feature; however, MUST is not unique in providing it.

The authors note that there are practical problems associated with this procedure. Specifically, they state, "The information that must be maintained on the list of subsets can get very large since the entire solution space is generated." When this happens, heuristic rules are invoked which essentially curtail the amount of subsets that are stored. Thus, for large problems, the algorithm is forced to change from an optimal-seeking one to a heuristic one. MUST uses clever programming to make wise use of computer storage space and to speed processing time. Nonetheless, the processing time requirements for the technique, though not excessive, are relatively high. It should also be noted that, similar to MALB, the objective of MUST is to minimize cycle time for a given number of stations.

2.1.2 Mixed Model, Deterministic Assembly Line Balancing

As discussed previously, mixed model lines complicate the line balancing process. This is because some models require different work elements than others, and the work element times are likely to vary between the different models. Nonetheless, mixed model production has significant advantages, and may become even more common as society continues to demand made-to-order products. Following is a discussion of the research done on mixed-model assembly line balancing assuming deterministic work element times.

Wester and Kilbridge in 1964 suggested that a mixed model line could be balanced by separately balancing the line for each model. This approach might result in an operator at a given station performing different work elements depending on the model which arrives to the station. This is not a good situation since it would require duplication of tools, training, and material handling/storage.

A preferable alternative is to balance the line as if only one model, sort of a weighted average of all the models, were to be produced. This approach involves creating a combined precedence diagram which includes the elements used in all of the models. One could derive the precedence diagram for any given model from the combined precedence diagram by setting to zero the element times for those elements that the model does not require. A representative model is then determined by using the relative frequencies of the models (i.e., the demand mix) to calculate a weighted average of the element times. The representative model, then, can be used to balance the line.

The other, very important aspect of this alternate method is that the line is balanced using a shift approach rather than the cycle-time approach used in single-model balancing. This idea was proposed by Nick Thomopoulos in 1967. In the shift approach, the demand mix for a shift is considered in balancing the line. The cycle time is calculated as the total work time required during the shift divided by the total number of units to be produced. The shift approach has the effect of placing more emphasis on the models which will be produced most often during the shift. For example, if model C is the only model which requires element six, and model C is produced infrequently, element six should not be given much emphasis when balancing the line. In the shift approach, average times are used to balance the line, thereby making use of the fact that certain models will take longer than the average, but other models will take less time than the average, thus 'making up' for the longer models. Variation is accepted as a given in mixed model lines, and the way to handle it is to properly sequence the models onto the line. Proper sequencing, for example, might mean that if model C requires more assembly time than the average, then a model requiring less time should be launched after model C.

These ideas, as well as others, were presented by Thomopoulos in two papers, published in 1967 and 1970. In the 1967 paper, once the problem is formulated as discussed above, Thomopoulos uses Kilbridge and Wester's 1961 approach for balancing. The line is balanced serially, which is likely to result in a poor balance.

In the 1970 paper, Thomopoulos provides a new objective function for smoothing station assignments on a model by model basis. Prior to this work, lines were balanced

without considering the smoothness of work assignments along the line for the individual models. As a result, there could be an uneven flow of work for a given model. If several of one model were run at a time, or if batch production were needed to satisfy a demand spike, the line might be very poorly balanced. Thomopoulos' objective function addresses this problem. The objective function is:

$$\min. \quad \sum_{i=1}^{n} \sum_{j=1}^{J} \left| P_j - P_{ij} \right|$$

where:

$$P_j = \frac{N_j}{n} \left(\sum_{k=1}^{K} t_{jk} \right)$$

$$P_{ij} = N_j \, p_{ij}$$

The variables are defined as follows. There are n stations, J models, and K work elements. N_j is the number of units of model j to be made on a shift, t_{jk} is the work element time for element k and model j, and p_{ij} is the amount of time assigned (through a line balancing procedure) to station i per unit of model j. P_j is the total time needed to produce a shift's worth of model j divided by the number of stations on the line. This is the station time for every station if the line is perfectly balanced for model j. The objective function then sums, over all stations and all models, the difference between this ideal and P_{ij}, the actual total time (for a shift) assigned to station i on model j.

Thomopoulos also introduced the idea of allowing a range for cycle time by setting upper and lower limits. This allows flexibility for exceeding cycle time, which is appropriate for a mixed model line since variability is inherent. The lower limit serves as a threshold for acceptable solutions; this concept is used in the balancing approach in Thomopoulos' 1970 article. The balancing approach assigns elements to stations serially. A finite number of alternatives is searched until an acceptable one is found. Again, it should be clear that this method of balancing is likely to result in a sub-optimal solution.

COMSOAL, as discussed previously, allows for many realistic features of an assembly line, and one of these is mixed model production. Arcus uses the relative frequency of models to generate a representative model and then balances the line with a random generation method as described earlier. This method is not elegant as there is very little logic involved and there is a high probability that balances will be repeated in the generation. The objective function used in COMSOAL is different from the one proposed by Thomopoulos.

In 1969, a project team at the IIT Research Institute developed another computerized method which was designed for industrial use. This software package, known as CALB, addresses mixed model lines using combined precedence diagrams and the shift approach. Very little has been published about this technique. [Kovach, 1969]

Roberts and Villa (1970) developed an integer programming formulation to the mixed model problem. The objective function is to minimize the excess work content of each station. The authors solve the problem via Gutjahr and Nemhauser (1964)'s shortest route algorithm. This algorithm is not well suited to solving problems of realistic size.

Macaskill (1972) developed a computerized technique which uses Thomopolous' shift approach. A 'fit list' of tasks without predecessors is made, and either the RPW or largest candidate rule is used to select elements from the fit list. This is a serial method, and is therefore unlikely to produce an optimal solution. Macaskill placed greater emphasis on computational speed than accuracy, with the reasoning that even the best mixed model balance can result in inefficient production if the sequencing is not well planned. Macaskill speeds computation and decreases storage requirements by storing precedence information in bit positions in computer words. The program does not use the objective function proposed by Thomopoulos (1970), so as Macaskill admits, "an immediate difficulty of the method is that tasks for a given model will often by shared unevenly between the stations." This problem reduces the productivity of the line and makes the line more sensitive to model sequencing. It also leaves the line poorly balanced during periods of batch processing.

Vrat and Virani (1976) address the mixed model balancing problem when work element times are stochastic. They reformulate the mixed model problem into a single model problem, and then balance the line using an approach similar to Kottas and Lau (1973). The approach is serial, and fills each station by making lists of elements which will feasibly fit into the station and selecting among them according to incompletion cost. Elements with low probability of incompletion should be considered first, giving additional priority to the ones of these which have the highest incompletion cost. The problem with this method is that it is a serial approach, and therefore will likely select only a local optimum.

Schofield (1979) developed a computerized line balancing methodology called NULISP. This method falls into the category of general purpose line balancing techniques, such as COMSOAL and CALB. The balancing method is similar to that used in COMSOAL, where a large enough number of random sequences is generated such that the likelihood of achieving a good balance is high. NULISP uses a weighted approach, similar to that tried by Arcus, in which tasks are not really randomly selected, but are weighted according to either task time, number of followers, time required by followers, et cetera. Arcus noted in his paper that the weighted approach was not as successful as the simple random search. Schofield, however, does not provide comparisons between NULISP and any other packages. Since NULISP is a general purpose technique, it handles mixed model balancing. However, only a brief description of the mixed model methodology is provided for copyright reasons.

Chakravarty and Shtub (1985) formulated the mixed model problem to consider inventory and setup costs in addition to idle-time costs. The two solution methods suggested are a shortest-path method and a single-pass heuristic procedure. The single-pass heuristic is simple, but it is a serial approach. The shortest path solution provides better solutions but requires more processing time.

Smith (1990) developed a computerized methodology for exhaustively searching the feasible balances and choosing the best solution. Smith programmed three methodologies: the algorithm used by Thomopoulos in his 1970 paper (a serial approach in which only a fixed number of alternatives is evaluated per station), a serial approach where all feasible assignments are examined for each station, and a fully exhaustive

search. He calculated balance delay for each of these algorithms using two different objective functions: one to minimize the difference between station times (the 'perfect balance') and the other to minimize the objective function proposed by Thomopoulos. In all three cases, using Thomopoulos' function as the objective minimizes the balance delay.

Smith performed a similar comparison using operator inefficiency, rather than balance delay, as a measure. To do this, he determined the optimal sequence for each solution using Thomopoulos' penalty method (1967). Operator inefficiency was defined as the sum of idle time and utility work. Again, using Thomopoulos' function as the objective resulted in smaller values of operator inefficiency.

Smith's exhaustive search requires a large amount of time to generate optimal solutions. For large problems, the algorithm can balance a few stations at a time, which speeds up the search at the cost of solution quality. This technique resembles a serial approach, which is not appropriate for the mixed model objective function since it assesses solution quality by evaluating the entire balance.

Pantouvanos (1992) developed a methodology for determining the best combination of balance and sequence for mixed model lines with stochastic work element times. Model sequencing is a critical factor in mixed model lines, since variation exists between models. Pantouvanos' premise is that merely finding the best balance and then determining the best sequence for that balance does not guarantee the optimal combination of balance and sequence. Thus, he provides an exhaustive search for the best balances, similar to Smith (1990). This method also can balance subsets of stations

(known as 'lookup horizons'), again similar to Smith. The objective function used is Thomopoulos' function. Pantouvanos also generates the best sequences, independent of balance, and then compares each combination of balance and sequence using a calculation of expected incompletion cost. Additional experimentation should be performed with this method to conclusively determine its merits. This method requires a great deal of processing time, even with the lookup horizon approach.

2.1.3 Summary

Many approaches have been developed for solving the assembly line balancing problem. However, there are not many that address complex problems, including the case of mixed model assembly. Also, very few approaches can handle large problems.

In the case of large problems, exact methods fail. In his 1990 book, *Assembly Line Design*, We-Min Chow states, "No efficient computational methods for the exact solution are known." Ghosh and Gagnon, in their 1989 review article, note that:

> "Despite recent advances in problem formulation and solution procedure efficiency, mathematical programming/network-based optimization techniques are still computationally prohibitive beyond limited problem dimensions. It is also unlikely that computational efficiency will progress sufficiently in the near term to allow the use of even the most efficient exact techniques to realistically sized GALB problems. Therefore, heuristic and inexact techniques still remain the only computationally efficient and sufficiently flexible methodologies capable of addressing large-scale, real-world ALB situations, particularly for the multi/mixed model and GALB categories."

While the exact techniques require too much processing time or memory storage to be of practical use, many of the heuristic techniques are too simplistic to find good solutions. The best candidates for the problem seem to be MUST, MALB and

COMSOAL. MUST and MALB both make use of heuristic rules that curtail the search space, making the quality of solution questionable. The processing times for MUST are relatively, though not excessively, large, and there is not enough experimental data presented in the MALB paper to document either solution quality or processing speed. COMSOAL is not a very elegant technique because there is little intelligence involved with it. It relies on chance to find an acceptable, but not necessarily good, solution. As with MUST and MALB, there is not enough data given for COMSOAL with regard to either solution quality or processing speed.

Although some of the approaches for balancing single model lines may be suitable for use in balancing mixed model lines, few authors have experimented with this. One cannot compare the solution quality and processing time achieved by a single model approach with the same measures of a mixed model approach, because the problem is more complex and the definition of optimal can be quite different. The only works which provide a significant amount of experimental data for mixed model balancing are those by Smith (1990) and Pantouvanos (1992), and these are both exhaustive search methodologies which require substantial processing time.

On the basis of the findings of this literature survey, there is still room for improvement in the area of balancing mixed model lines of realistic size. An approach is needed that can find near-optimal solutions while requiring a minimum of processing time. Simulated annealing can be such an approach.

2.2 SIMULATED ANNEALING

2.2.1 Introduction

Simulated annealing is a general approach for solving combinatorial optimization problems. These types of problems are often computationally intensive, and optimal-seeking solution procedures require computational time which grows exponentially with problem size. Simulated annealing, though, has been shown to produce high quality solutions to this class of problems in polynomial time. [Aarts and Korst, 1989]

In 1953, Metropolis, et al. developed a technique for simulating the annealing of solids into a state of thermal equilibrium. Thirty years later, Kirkpatrick, et al. (1983) and Cerny (1983) independently recognized a link between this technique and a process for solving optimization problems. They noticed a similarity between arranging atoms to minimize energy and arranging components of an optimization problem to minimize an objective function.

Annealing is the process of melting a solid and then slowly cooling it in an effort to produce a purer, crystalline-lattice structure. Annealing must be performed very carefully: the solid must be heated until the particles freely move, and then gradually cooled. The temperature of the substance is lowered slowly, in a step-wise manner. Each temperature must be held constant until thermal equilibrium is reached at that temperature. If this is not done, the resulting solid will contain imperfections.

At each temperature, the atoms of the material move around, searching for the arrangement which will minimize the material's energy state. As the temperature continues to drop, the atoms move more slowly until eventually they do not move and are

considered 'frozen.' To simulate this behavior, Metropolis, et al. used a Monte Carlo method (or randomized method) to generate potential states of the material. Given an initial arrangement of atoms, the researchers create a small perturbation by moving the position of a randomly-selected atom. If this new arrangement results in a lower energy state, the atom remains in this new position and another perturbation is performed. If the arrangement increases the energy, the arrangement is accepted with probability

$$P(\Delta E) = e^{\left(\frac{-\Delta E}{k_b T}\right)}$$

(where k_b is Boltzmann's constant, T is the current temperature, and ΔE is the change in the system's energy caused by the perturbation), and rejected otherwise. If rejected, the atom is moved back into its original position and a new perturbation is sampled. The acceptance probability rule is known as the Metropolis criterion.

Simulated annealing (the optimization technique) is very similar to the simulation of physical annealing. First, the function representing the energy of the physical system is replaced by the objective function that is to be optimized. Thus, rather than minimizing energy, we might minimize cost. To perform simulated annealing, we must formulate the optimization problem such that we can generate feasible solutions as well as small perturbations to these solutions. Finally, we must have an annealing schedule, similar to the simulation of physical annealing.

The annealing schedule dictates the original temperature, the conditions under which the temperature will be decremented, the amount by which the temperature will be decremented, and the conditions for stopping the annealing. In the case of simulated

annealing, temperature has none of the connotations commonly associated with the name; rather, it is a parameter which affects the probability of accepting worse solutions.

As temperature decreases, the probability of accepting worse solutions decreases. This means that at the beginning of the annealing process, the algorithm is fairly liberal, and 'atoms' can move around with substantial freedom. However, as time passes, and the quality of the solution improves, the more selective the algorithm becomes. If this were not the case, the likelihood that a poor solution could be selected at the end of the annealing could be great; that is, there would be no focusing on a good solution. The gross features of the eventual state of the system appear at high temperatures, while small refinements are made at the lower temperatures [Kirkpatrick, et al., 1983].

Temperature is not the only factor affecting the probability of accepting worse solutions. The potential change in the objective function value is also a factor. The larger the change, the smaller the probability of accepting the worse solution. In other words, the Metropolis criterion is conservative. It wants to avoid choosing solutions that are much worse than the current solution. There are other acceptance criteria which could be used instead, and several are discussed later. However, the Metropolis criterion seems to be the most effective and the most popular.

The rules for lowering the temperature affect how many arrangements are sampled or accepted at any given temperature. The more arrangements sampled at each temperature, the higher the solution quality is likely to be. There is a tradeoff, however, between solution quality and computation time. Obviously, the more arrangements sampled, the more computation time is required. A stopping criterion is included in the

annealing schedule as a means of identifying when a sufficiently good solution has been found. This can be used to prevent the algorithm from expending excessive computational time to make very small improvements in the solution.

The pseudoalgorithm for simulated annealing is provided below, and a flowchart is provided in Figure 2.1.

1. Select an initial feasible solution, S_0. Compute the associated objective function value, $E(S_0)$.

2. Select an initial temperature, $T > 0$.

3. Perturb S_0 slightly to generate a solution in the neighborhood of S_0, S_1. Compute the associated objective function value, $E(S_1)$.

4. Calculate ΔE, $E(S_1) - E(S_0)$.

5. If $\Delta E \leq 0$, set $S_0 = S_1$; otherwise set $S_0 = S_1$ with probability $\exp(-\Delta E / T)$.

6. If equilibrium has been reached at this temperature, decrease temperature.

7. If the stopping condition has been reached, stop annealing. Otherwise, go to step 3.

2.2.2 Advantages of Simulated Annealing

There are several advantages to simulated annealing. First, it is capable of finding high-quality solutions to combinatorial optimization problems in polynomial time [Aarts and Korst, 1989]. There exists a class of combinatorial optimization problems for which the amount of computational effort needed to find optimal solutions grows exponentially with problem size. These problems are called NP-hard, or NP-complete. It is extremely time-consuming to optimally solve these problems. However, if simulated annealing is

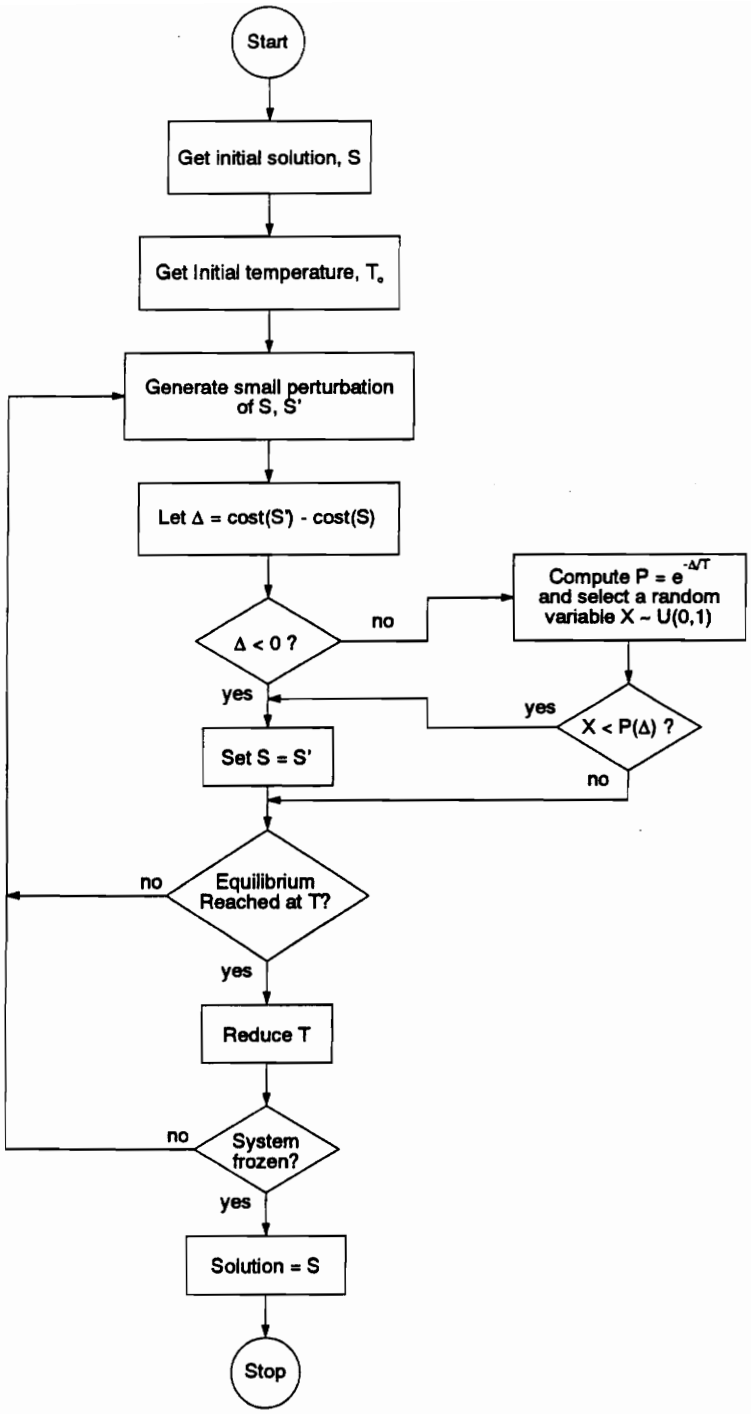

Figure 2.1: Flowchart of the Simulated Annealing Algorithm.

used with appropriately-chosen parameters, computational effort will increase with problem size, but at a much smaller and more manageable rate than optimal-seeking approaches.

Although simulated annealing is a heuristic approach, it is known to produce high-quality solutions. This is a result of its fundamental rule of accepting, with a certain probability, solutions which worsen the objective function value. It is this feature which allows the simulated annealing algorithm to 'jump out' of local minima. This feature also makes simulated annealing independent of the choice of initial solution.

An additional advantage of simulated annealing is that it is a simple approach. The algorithm requires few steps and is somewhat intuitive. Also, application of the technique generally does not require a great deal of prior knowledge about the problem. A final advantage of the approach is its applicability to many different types of problems. If the problem can be set up in such a way to generate feasible solutions, and if there is a defined objective function, then the technique can be applied.

2.2.3 Disadvantages of Simulated Annealing

The most significant disadvantage of simulated annealing is that it is sensitive to the parameters chosen for the annealing schedule. Each problem may require an unique set of parameters for efficient and successful annealing. Furthermore, although there are ways to intelligently estimate the parameters, there is no method which guarantees the optimal choice of parameters. In many cases, parameters are chosen through

experimentation. The better the solution space is understood, the easier it is to generate the annealing parameters.

Another possible disadvantage is that some amount of inefficiency results from the fact that arrangements of 'atoms' may be selected for evaluation more than once. This is due to the Monte Carlo method of generating arrangements. Although conceptually this may be inefficient, it is a necessary evil so that when a good solution is lost due to the acceptance of a worse solution, the good solution has the potential to be regenerated.

Simulated annealing provides the greatest benefits when used to solve large problems, where optimal-seeking procedures become extremely time-consuming, and heuristic approaches overlook very large numbers of possible solutions. For solving small problems, however, optimal or heuristic solutions may be superior in solution quality and/or time.

2.2.4 Applications of Simulated Annealing

Simulated annealing has been applied to optimization problems in many diverse subject areas. These areas include computer network design, VLSI design, code generation, pattern recognition, and a number of operations research problems. Applications in the area of manufacturing include plant layout, cell formation, AGV routing, and flowshop scheduling. Examples of specific applications are provided below to show the range and nature of problems which have been addressed by simulated annealing.

Kirkpatrick, et al. (1983), in their paper introducing the technique, applied simulated annealing to the traveling salesman problem. In this problem, a salesman has a number of cities to visit and the optimal path connecting these cities is to be determined. The objective function used was simply the minimization of total distance.

Also in the 1983 paper, Kirkpatrick, et al. applied the technique to problems in the area of circuit board layout and wiring. In particular, they addressed the partitioning problem, which involves assigning circuits to chips in order to minimize the number of connections needed between the chips while also maintaining a balance between the number of circuits assigned to each chip. To handle this multiobjective function, the authors used a coefficient to represent the relative importance of the two conflicting objectives.

Kirkpatrick, et al. also tackled the placement problem, which involves locating chips on a board such that wiring distance is minimized but chips will not be so congested that wiring will become impossible. This problem also has two objectives, thus the objective function used in the simulated annealing formulation was a sum of the length of wire required and a measure of wire congestion.

In the area of chemical engineering, simulated annealing has been applied by Das, et al. (1990) and Ku and Karimi (1991) to the scheduling of batch processing. Specifically, this problem involves determining the order in which products should be processed in order to minimize total production time. The problem assumes a serial flowshop, which means that the products are so similar that they require the same path through the processors, though certain products may skip some processors. As a result

of this assumption, there are no constraints on the schedule, as there are in the case of line balancing. Ku and Karimi assumed unlimited storage capacity between each pair of processors, while Das, et al. formulated the problem for four types of storage situations.

Quite similarly, Ogbu and Smith (1990) have applied simulated annealing to flowshop scheduling in a manufacturing environment. These authors also chose to minimize makespan. They note:

> "For problems involving very small number of jobs (n), optimal sequences may be determined by complete enumeration, branch and bound techniques, or integer programming. However, the enormous computation time and computer memory requirements associated with using these methods preclude their practical application to larger problems."

Wilhelm and Ward (1987) used simulated annealing to solve quadratic assignment problems. In these problems, elements are to be assigned to sites, and there is interaction between the elements which is affected by their locations. As a result of the interaction, there is a 'cost' associated with assigning pairs of elements, and the amount of this cost depends on the locations of assignment (as an example, there is a cost associated with assigning element 1 to site C and element 2 to site A). The objective function then is to minimize the total cost of the assignments. Facility location, facility layout, placement of electronic components on circuit boards, or placement of controls on a control panel are all examples of quadratic assignment problems.

Although there are likely to be constraints involved with these problems, this issue was not addressed in Wilhelm and Ward's paper. Some constraints could be handled by placing a particularly high cost on infeasible assignment pairs, but this technique is not likely to handle all of the constraints. It should be noted that line balancing problems

are not quadratic assignment problems. It is not the assignment of *pairs* of elements which determine the cost/effectiveness of the balance, but the assignment of the entire set of elements.

Abramson (1991) used simulated annealing to solve the school timetabling problem. In this problem, there exist combinations of classes, teachers, and classrooms that need to be scheduled into time slots. The assignments must be made such that there are no overlaps, or clashes. This problem involves constraints; however, the problem is such that it can be formulated by using these constraints as the objective function. To do this, Abramson associates a cost with each type of clash--teacher, class, and room. The objective function is then to minimize the total cost. The cost function allows for weights to be placed on the three types of clashes so that certain types can be made more significant than others. Abramson also introduced the idea of determining the incremental change on the objective function, rather than recalculating the entire objective function value with each rearrangement. If this can be done in any application of simulated annealing, it could save significant amounts of processing time.

Harhalakis, et al. (1990) addressed the problem of manufacturing cell design. In this problem, there are a number of machines, M, and a number of part types, N. Each part type has a specific routing through the M machines. The problem is to assign machines into cells such that the inter-cell traffic cost is minimized. The inter-cell traffic cost results from transportation costs which are incurred if a product must leave its cell and travel to a different cell for processing. Weights were used to incorporate the

production volume of the different products. There was one constraint used in this problem, which was an upper limit on the number of machines in each cell.

Derin (1986) addressed two problems, resource allocation in an interconnected computing network and task allocation in a distributed computing environment. The resource allocation problem is a network of nodes, where each node is connected to a set of other nodes. Each node may be connected to a different number of nodes. The problem is as follows: given the demand at each node, determine the optimal allocation to each node.

The objective function is to minimize a cost function, C (A,D), which is the cost of allocating an amount A when the demand is D. This function is the sum of two components. One represents the cost of allocating a certain A when the demand is D for a certain node. The second represents the cost of making allocation A on a set of nodes. It is through the second component that the author says that constraints on the allocation can be implemented. The author does not elaborate, but it is assumed that this means assigning a high cost penalty to infeasible solutions. This would work because the probability of accepting a particular arrangement is a function of the resulting change in the objective function (assuming that the Metropolis criterion is used). The higher the change in objective function, the lower the probability of keeping the arrangement. Thus, very high penalties will greatly reduce the probability of accepting a constrained arrangement. Although this method may work for this application, it might not work for applications where several types of constraints are needed.

Derin also addressed the allocation problem, which involves assigning N tasks to M processors with the constraint that some tasks require communication to exist between processors. Again, a cost function is minimized. The function includes an assignment cost which is independent of other assignments, and a cost which is dependent on the other assignments.

Bulgak and Sanders (1991) used a hybrid approach, involving analytical techniques, discrete event simulation and simulated annealing, to determine optimal buffer sizes for an asynchronous flexible assembly system with statistical process control and repair loops. The first step of this approach is to analytically estimate the optimal number of pallets to be used in the system. The next step is to perform simulated annealing to determine the optimal buffer sizes. The annealing process would be applied as usual, except that discrete event simulation would be used to determine expected value of the objective function for each perturbation. The objective function was maximization of the production rate. Simulation takes into account the stochastic nature of the problem, for example, the possibility of jamming and the probabilistic nature of downtime. This is a good demonstration of applying SA to a stochastic combinatorial optimization problem.

Manz, et al. (1989) used simulated annealing in conjunction with a simulation model to optimize an automated manufacturing system. Three parameters of the system were considered: size of arrival batches, distribution of the different products within the arrival batches, and buffer sizes. The objective function used for this problem was maximizing profit.

In conclusion, the applications of simulated annealing to date have not fully tested the ability of the algorithm to handle highly constrained problems. The applications either involve no constraints, or very simple ones. Kouvelis, et. al (1992) directly addressed the issue of constraints in formulating a simulated annealing algorithm for the plant layout problem in the presence of zoning constraints. The authors proposed two methods, one in which successive perturbations must be feasible in order to be considered. The second method allows infeasible arrangements to be considered; however, the objective function is structured to place such a high penalty on these arrangements that they will not be accepted. Although this work included one type of constraint in the SA formulation, the experimental cases used only involved one or two of the simplest cases (e.g., two machines required to be neighbors). Therefore, the ability of SA to cope with realistically constrained problems has yet to be fully explored.

2.2.5 Formulation of Problems for Simulated Annealing

To formulate a problem to be solved by simulated annealing involves first determining an appropriate objective function and a method for generating alternative solutions. The next step is to decide upon certain rules and parameters. Specifically, these rules and parameters are: an acceptance criterion, a value for initial temperature, a rule for determining how much to decrement the temperature, a rule for determining when to decrement the temperature, and a stopping criterion. These six decisions, and techniques for addressing them, are presented below.

2.2.5.1 Generation of solutions

This section discusses methods for generating small perturbations of an existing, feasible solution. Examples from research on various problems will be presented, and although the techniques have been used on specific problems, the concepts are usually applicable to many different types of problems. This section should provide an appreciation of the flexibility of simulated annealing and the potential freedom of choice in modeling a problem for solution by simulated annealing.

Probably the most common means of generating alternate solutions is pairwise interchange, or simply swapping the position of two elements. This is the procedure used by Wilhelm and Ward (1987) in solving QAP problems. The potential difficulty with interchange is that it will result in the same number of elements in each group. Thus, sometimes it is preferred to use insertion, where an element is simply moved from one group to another. For example, Kirkpatrick, et al. (1983) used this technique in the partitioning problem by moving one circuit from the current chip to the other chip. These authors used a variation of this procedure in the placement problem, where interchanges were made between the contents of two locations on the board. In some cases, this would entail the swapping of the locations of two chips. In other cases, this might mean the interchange of a chip and a vacancy. Kirkpatrick, et al. also suggested limiting the distance between interchanges at low temperatures for more efficient annealing.

Another common method of generating alternate solutions is rearranging a portion of an existing arrangement. In solving the traveling salesman problem, Kirkpatrick, et

al. used a technique where each move involved reversing the direction of a subsequence of the tour.

Das, et al. (1990) experimented with five rearrangement schemes: the reversal of a subsequence, product insertion, product interchange, movement of the position of a subsequence, and the interchange of two adjacent products. They found that product insertion performed the best. The authors also experimented with combining two types of rearrangement strategies, using one for high temperatures and one for lower temperatures.

Likewise, Ku and Karimi (1991), in solving the batch process scheduling problem, experimented with several forms of sequence rearrangement, including combinations of product insertions, product interchanges, and pairwise interchanges of adjacent products. They found that no rearrangement strategy worked well unless pairwise interchanges of adjacent products were included. As a result, this is the strategy that was chosen for their final implementation.

Ogbu and Smith (1990) tried two different schemes, pairwise exchange and insertion. They discovered in their experiments that insertion produced better results. The authors believe this is due to the fact that insertion produces a neighborhood of greater cardinality.

Ogbu and Smith deviate from the common simulated annealing algorithm in that they search the neighborhood of a seed sequence, noting all the accepted rearrangements, and replace the original sequence only with the last accepted sequence at a given temperature. They believe that this practice, combined with their acceptance criterion

which does not consider the change in objective function value, will result in a faster attainment of equilibrium at a given temperature. According to the authors, this will happen because the modified algorithm will 'reach out' over a broader area of the problem's solution space.

Harhalakis, et al. (1990) generate solutions for the cell formation problem in three ways: (1) removing a machine from a cell and placing it in another cell, (2) exchanging two machines (from two different cells), and (3) taking a machine out of an existing cell and creating a new cell with it. All three of these methods are used in one simulated annealing run, by using probabilities which govern the selection of each of these three methods. Specifically, option three is chosen with probability $1/(N_c + 1)$, while each of the other methods is chosen with probability $[1-1/(N_c+1)]/2$, where N_c is the number of cells in the current solution.

In the buffer size problem of Bulgak and Sanders (1991), rearrangements are generated as follows. The initial solution will be a vector of buffer sizes for each station in the system. The buffer size for a particular station will then be modified by 0, +1, -1, +2, or -2, with probabilities 0.3, 0.25, 0.25, 0.1, and 0.1. These sizes are constrained, however, to the interval [1, N/3], where N is the number of pallets in the system. This set of constraints is a heuristic rule which is used to limit the amount of processing required.

2.2.5.2 Acceptance Criterion

The acceptance criterion, as discussed previously, is the criterion for determining whether or not to accept a given arrangement of elements. Ku and Karimi (1991) state that the Metropolis Criterion is the simplest and most often used acceptance criterion. This certainly seems to be the case, as Ku and Karimi, Das. et al. (1990), Wilhelm and Ward (1987), Abramson (1991), Aarts and Korst (1989), Kirkpatrick, et al. (1983), Harhalakis, et al. (1990), Bulgak and Sanders (1991), and Manz, et al. (1989) used this criterion in their work. Derin (1986) tried both the Metropolis criterion and the Gibbs Sampler Algorithm (which selects a new value for one element of the solution from a conditional probability mass function) and concluded that the algorithm using the Metropolis criterion was more stable.

It should be noted that for simulated annealing (as opposed to the simulation of physical annealing), the Boltzmann constant is not needed. Thus, for simulated annealing, the Metropolis Criterion is as follows:

$$p = e^{\left(\frac{-\Delta E}{T}\right)}$$

Das, et al. (1990) experimented with the Glauber algorithm in addition to the Metropolis algorithm. The Glauber algorithm (Glauber, 1963) does not accept all moves that improve the objective function value. Instead, the probability of accepting any move, whether better or worse than the current solution, is given by:

$$p = \frac{e^{-\Delta E/T}}{1 + e^{-\Delta E/T}}$$

At high temperatures, this algorithm will accept all moves with probability 0.5. As the temperature decreases, the probability of accepting an improvement increases gradually toward one, while the probability of accepting an 'uphill' move decreases to zero. The Glauber algorithm was designed for systems with discrete objective function values rather than continuous [Glauber, 1963; cited in Das, et al., 1990]. The algorithm will converge to the optimal solution more slowly than the Metropolis algorithm; however, it will avoid local minima better than the Metropolis algorithm [Das, et al., 1990].

Ogbu and Smith (1990) use an acceptance probability which is not dependent on the change in the objective function. The acceptance probability is given either by 1.0 if the perturbation improves the objective function value or by

$$AP(k) = AP(1)(pfac)^{k-1}$$

where *pfac* is a reducing factor, $AP(1)$ is the initial acceptance probability, and k is the stage, otherwise. The objective of this algorithm is to increase the speed of annealing.

2.2.5.3 Initial Temperature

The objective in choosing an initial temperature, T_o, is to choose one high enough that the 'atoms' of the system will flow freely. In other words, we would like an initial

temperature high enough that the probability of accepting an uphill move will be great. This will allow the algorithm to freely search the solution frontier, and thus eliminate dependence on the original solution. Kirkpatrick, et al. (1983) note that the temperature at which elements will flow freely will be of the order $N^{1/2}$, where N is the number of elements.

A number of researchers suggest taking samples to determine the initial temperature. This is likely to produce a good annealing schedule, but it will involve additional processing time. Ku and Karimi (1991) took samples from 3000 rearrangements and calculated the difference in objective function for each change. They then took the average of these differences and set the initial temperature equal to 1.5 times the average.

Das, et al. (1990) cite a technique developed by Aarts and van Laarhoven (1985), in which the initial temperature is determined as follows. Knowing that the initial temperature should allow the acceptance of the maximum delta objective function with a high probability, then if we decide what this probability should be, we can rearrange the Metropolis relation to find the associated temperature, as follows:

$$p = e^{-\Delta E/T}$$

$$T_0 = \frac{-\Delta E_{max}}{\ln p_0}$$

If p_o is set to 0.9, then T_o is approximately 10 times the value of ΔE_{max}. So, Aarts and van Laarhoven recommend taking a number of samples, recording the maximum change in objective function, multiplying by 10 and using this as T_o.

Aarts and Korst (1989) propose a somewhat more complicated means of finding T_o, which again involves taking a number of sample rearrangements. For the first rearrangement, T_o should be set to zero. After each sample, T is recalculated using the following equation:

$$T = \frac{\overline{\Delta E^{(+)}}}{\ln\left(\frac{m_2}{m_2\chi - m_1(1-\chi)}\right)}$$

where m_1 is the number of energy-decreasing trials, m_2 is the number of energy-increasing trials, χ is the initial acceptance ratio, and the numerator is the average difference in energy over the m_2 energy-increasing trials. The final value of T is used as T_o. Aarts and Korst also suggest a more simple means of generating a T_o, which is to choose one, run samples to determine the ratio of accepted to attempted samples, and increase T_o until the ratio is sufficiently high.

2.2.5.4 Temperature decrements

As discussed, it is important to lower the temperature slowly to insure that a good final solution is reached. However, if the temperature is lowered too slowly, excessive processing time may be required.

A fairly common approach for determining the amount to decrement temperature is an exponential cooling scheme, where the temperature is lowered by multiplying by a constant. Thus, $T_n = T_{n-1}R$, where R is a cooling rate. (Note that this is the same as $T_n = T_oR^n$.) Abramson (1991) experimented with cooling rates ranging from 0.1 to 0.99.

Aarts and Korst (1989) suggested a cooling rate between 0.8 and 0.99. Kirkpatrick, et al. (1983) suggested a cooling rate of 0.9. Das, et al. (1990) tried cooling rates varying from 0.6 to 0.99. Harhalakis, et al. (1990) used R = 0.95. Wilhelm and Ward (1987) suggested that R = 0.9 for all quadratic assignment problems.

In addition to the exponential cooling scheme, Aarts and Korst also suggest the following rule for decreasing temperature:

$$T_{k+1} = \frac{T_k}{1 + \frac{T_k \ln(1+\delta)}{3\sigma_{T_k}}}$$

δ is a measure of the desired closeness to equilibrium, and σ_{Tk} is the standard deviation of the objective function values at the current temperature. The larger the value of the non-constant multiplicative factor (the denominator), the smaller the value of the temperature and the faster the annealing. The factor will be larger if the standard deviation of the objective function is small, or equivalently, if the system is close to equilibrium. Thus, it seems reasonable to conclude that the closer the system is to equilibrium, the faster the annealing. This is significantly different from the exponential approach, which reduces the temperature increasingly slowly, and thus anneals more slowly as temperature is reduced. Das, et al. conclude from their experiments that this annealing schedule is superior to the exponential schedule. However, this schedule is more complex to calculate and will therefore involve more computational time.

A very simplistic method is noted in van Laarhoven and Aarts (1987). This method involves determining an initial temperature and a number of decrement steps for

the temperature, K. In this way, the temperature decreases by a constant amount, as per the following equation:

$$T_k = \frac{(K-k)}{K} T_0$$

Geman and Geman (1984) developed a function for decreasing T which they show will cause the simulated annealing algorithm to converge on the optimal solution for deterministic problems, given that the value of the parameter, d, is correctly chosen. This formula is as follows (cited in Derin, 1986):

$$T_k \geq \frac{d}{\log(k+1)}$$

The parameter d must be larger than or equal to the depth of the deepest local minimum that is not a global minimum configuration [Bulgak and Sanders, 1991].

Derin notes that it is likely to be very difficult to determine the value of d, especially if little is understood about the objective function frontier. Further, he states that d is likely to be such a large value that annealing would be too slow to be practical.

In Derin's experiments, he used Geman and Geman's function, but with arbitrarily-chosen values of d. This presented problems, as small values of d sometimes resulted in local minima, but large values of d would not even converge. Derin then tried setting T_k equal to $1/k^{\alpha}$, although he found that this did not work well, either. His best experience was with an accelerated logarithmic function,

$$T_k = \frac{d}{q_k}$$

where

$$q_k = n_k[\log(k_i + k_p + 1) - \log(k_i + 1)] + \log(k - n_k k_p + 1)$$

$$n_k = \left\lfloor \frac{k - k_i}{k_p} \right\rfloor$$

and $\lfloor\ \rfloor$ denotes the integer part. Although this function allows the annealing to converge, it is quite a complex function to calculate, and also requires values to be chosen for k_i and k_p.

2.2.5.5 Number of attempts at each temperature

In addition to determining how much the temperature should decrease with each decrement, one must also determine *when* the temperature will be lowered. If the relationship with physical annealing is maintained, the temperature should be lowered only when the system has reached a state of equilibrium at the current temperature. The following are examples of how researchers have addressed this issue.

Harhalakis, et al. (1990) and Bulgak and Sanders (1991) suggest the possibility of reducing the temperature after each rearrangement. No explanation is given for this

approach. Obviously this is an easy method which requires little planning. However, it seems unlikely to produce good results.

Ku and Karimi (1991) suggest fixing a number of sequences, dividing this number into twenty equally-sized groups, and lowering the temperature after each group of sequences. This rather arbitrary method is simple to implement, but again the tradeoff may be solution quality.

Das, et al. (1990) set the number of attempts at a given temperature equal to the number of different configurations that can be achieved from the current solution by making only one move. The authors do not present the logic behind this suggestion.

Kirkpatrick's, et al. (1983) method is to continue to attempt different configurations until either a certain number of configurations is accepted per group, or the total number of attempts exceeds a constant times the number of groups. The latter condition prevents the algorithm from getting stuck. If the number of accepted configurations is not achieved for three consecutive temperatures, the system is considered frozen and annealing is stopped. This is a logical method, because as the temperature drops and the solution becomes closer to the optimal, it will become increasingly difficult to find/accept new configurations. The algorithm will notice the behavior of the system and stop the annealing. This method seems more logical than the method of simply choosing a number of configurations a priori (as used, for example, in Manz, et al. (1989)), since this method will adapt to the particular nature of the problem. The difficulty with this procedure is in choosing the values of the constants.

Abramson (1991) uses a similar method. He recommends choosing two constants, one for the maximum number of attempts and one for the maximum number of accepted attempts. At each temperature, configurations are attempted until one of these two limits is reached. The limits should be chosen to be proportional to the number of elements in the system such that larger problems are given more annealing time.

Wilhelm and Ward (1987) make use of an idea proposed by Golden and Skiscim (1983), which is to sample an 'epoch' of configurations before testing for equilibrium at a given temperature. The epoch, e, is a pre-determined number of samples. After an epoch's worth of configurations have been sampled and accepted at a given temperature, the mean of the objective function from all of the accepted samples during the epoch is compared with the grand mean, which is the mean of all accepted samples from all previous epochs at that temperature. If the mean of the current epoch is within a certain error, epsilon, of the grand mean, the system is assumed to be at equilibrium at that temperature, and the temperature is lowered.

Wilhelm and Ward combine this approach with the approach of Kirkpatrick, et al. (1983). The algorithm will sample enough configurations at each temperature that either at least N interchanges per group are accepted, or the number of attempted samples exceeds an a priori constant N' times the number of groups, n. If the desired number of accepted solutions is not achieved at three successive temperatures, the system is considered frozen.

2.2.5.6 Stopping criterion

Simulated annealing is not an exhaustive search methodology. Thus, it will not search the entire solution frontier, and so it must be able to determine when to stop searching. Obviously there is a certain tradeoff involved, as extensive searching may produce a better solution at the cost of processing time.

Ku and Karimi (1991) stop their algorithm after a fixed number of configurations have been attempted. Similarly, Ogbu and Smith (1990) stop annealing after a fixed number of temperature changes. Abramson (1991) stops after a fixed number of temperature changes have occurred without a change in the objective function value. Kirkpatrick, et al. (1983) and Wilhelm and Ward (1987) stop annealing if the desired number of accepted configurations has not been achieved after three temperature changes.

Das, et al. (1990) present a more complex method, which is to terminate when the derivative of the smoothed average objective function value with respect to the temperature is less than an epsilon which is between zero and one. This is a logical method, as one would want to stop annealing when the rate of change of the objective function was zero, or approaching zero. However, the complexity of this method may create problems.

2.2.6 Summary

As this literature review indicates, simulated annealing has been used to solve a variety of problems. Many are similar in characteristics to the line balancing problem,

but are not the same. One significant distinction is the constrained nature of the line balancing problem.

There is not *one* simulated annealing algorithm. There are many variations of the technique, depending on decisions made regarding both the method and the parameters. Previous researchers have proposed guidelines for making these decisions, but for each new class of problems to be solved by simulated annealing, many of the decisions are likely to be made via experimentation with the particular type of problem.

Simulated annealing seems well suited for solving large line balancing problems. The issues left to be addressed are the formulation of the problem for efficient solution, the identification of a rule for choosing parameters such that all line balancing problems can be solved, and the assessment of the technique.

CHAPTER 3: RESEARCH METHODOLOGY

This chapter describes the methodology used to meet the research objectives. The first step involved tailoring the simulated annealing algorithm to model the mixed model assembly line balancing problem, as described in Section 3.1. Once the model was obtained, a methodology, documented in Section 3.2, was developed to identify the best parameters for problems of different characteristics. The methodology involved establishing a representative set of problems and performing experiments with these problems using different levels of the simulated annealing parameters. Results of the experiments were compared with optimal solutions when available through exhaustive search. For the case problems where an optimal solution could not be obtained (problems with a large number of work elements), the best known heuristic in the literature, from Smith (1990), was used to provide a baseline for comparison.

3.1 MODELING THE MIXED MODEL LINE BALANCING PROBLEM

In order to apply simulated annealing to a particular class of problems, one must be able to formulate the problem in terms of an objective function to be optimized and a procedure to generate samples from the solution space. Once these fundamentals have been determined, the next step in formulating the SA algorithm is to determine the annealing schedule, which includes an initial temperature, a procedure for choosing each successive temperature, and a means of determining when equilibrium has been reached for each temperature. Finally, a stopping rule must be established to determine when the

algorithm has frozen. The entire algorithm must then be written in computer code for efficient solution. Each of these steps will now be described for the mixed model line balancing problem.

3.1.1 The Objective Function

The objective function provides a means of assessing the quality of solutions. In the case of mixed model line balancing, Thomopoulos' (1970) measurement accomplishes this task well. However, it should be noted that this function (hereafter referred to as 'delta') does not explicitly penalize solutions with a large number of stations. Consider the single model situation where there are 50 minutes of assembly work required. This work could be divided evenly between five stations, where each performs ten minutes of work. Another alternative would be to divide the work between ten stations, where each performs five minutes of work. The first alternative minimizes the number of stations, while the second minimizes cycle time. Thomopoulos' measurement will not distinguish between the two alternatives, because they will both result in an even division of work among the stations. To address this, Thomopoulos suggests placing upper and lower limits on the cycle time. Placing a lower limit on cycle time may cause problems for the simulated annealing algorithm, as it will make rearrangement difficult and will likely prohibit creation of new stations. A solution to this problem is addressed in the following two sections.

3.1.2 Generating Perturbations

In order to use simulated annealing, one must develop a means of efficiently generating alternate solutions. The procedure should generate solutions which are minor modifications of previous solutions. Ultimately, this step involves developing an appropriate means of moving from one configuration to the next. The most common way is to simply swap the position of two randomly-selected elements. This technique will not suffice for the line balancing problem since it will always result in the same number of elements in each station. If this method was used in the case of line balancing, the number of stations would be fixed as the number of stations generated in the initial solution, which would often be suboptimal.

The method chosen instead was one used by Harhalakis, et al. (1990) In this method, there are three types of rearrangements: (1) elements can be swapped between stations, (2) an element can be transferred from one station to another, and (3) an element can be taken from one station and used to form a new station. Harhalakis, et al. set the probability of selecting the latter alternative equal to $1/(1 + n_c)$, where n_c is the number of cells in the current solution (similar to the number of stations in a line balancing problem). Thus, as the number of cells/stations increases, this formula will reduce the probability of creating new cells/stations. This result is desired in line balancing when the objective is to reduce the number of stations.

However, Harhalakis, et al.'s formula does not treat problems of different sizes in proportion to their size. For large line balancing problems, or problems where the cycle time is very small, a large number of stations will be required. In these cases, the

probability of creating a new station will be quite small, even though the optimization routine may need to create a new station in order to sufficiently re-order the elements.

In the case of line balancing, the simulated annealing algorithm should be able to create new stations, particularly so that there is enough slack for the system to rearrange the elements. This slack is necessary for line balancing problems since the cycle time constraint will tend to restrict the freedom of rearrangement.

There is another reason for allowing the addition of stations. Without this feature, the simulated annealing algorithm will tend to minimize the number of stations via element transfer. Depending on the nature of the problem, a smoother arrangement of stations may be possible using more than the minimum number of stations. Without the ability to add new stations, then, the algorithm will probably not achieve the optimal delta value.

However, a limit should be placed on the number of stations which can be added, so that the resulting solution will be within the desired ranges for cycle time and number of stations. Without this limit, the number of stations could increase drastically. The more stations that are created, the more time consuming it will be for the simulated annealing algorithm to return to a solution with a small delta value and few stations. This is because the probability of choosing any one station will be reduced, and likewise the probability of choosing the station with only one element (the station which can be eliminated) will be reduced. As the algorithm searches to find the station which will lead to improvement, it is likely (due to the probabilities) to accept a worse solution. In this way, the solution could get progressively worse.

Thus, there should be an upper limit on the total number of stations, and the probability of attempting to generate a new station should be small. For the purposes of this research, the upper limit of stations was set as the number generated in the initial solution. The initial solution is an inefficient solution, and thus it is unlikely that a better solution would be found with more than this number of stations. For future applications, a different limit could be used. The probability of attempting to generate a new station was set equal to 0.04, and new stations could be added anywhere along the line. For any given perturbation, the locations of the stations and the elements within them were selected at random.

The probabilities of attempting the two other types of perturbations (swapping and transferring elements) should be approximately equal, since they are both important functions. The pairwise exchange method is a good way to efficiently rearrange the solutions. The transfer method is needed to change the number of elements in each station. It may be easier for the system to find a feasible transfer than a feasible exchange, since the precedence requirements of only one element must be met. On the other hand, due to the constrained nature of the line balancing problem, it may be difficult to add elements to stations without violating cycle time. Therefore, the pairwise exchange method may be more effective, because by removing two elements, time is freed up, and so the elements have greater likelihood of being successfully replaced in the stations. For this research, the probabilities of attempting a swap and attempting a transfer were each set at 0.48. Thus, $P_{swap} + P_{transfer} + P_{new} = 0.48 + 0.48 + 0.04 = 1.0$.

The generation of perturbations was actually more difficult to program for computer generation than it may seem. For example, the program must be able to recognize when, in the process of a transfer, it has removed the last element from a station. It must then change its knowledge of the other stations accordingly. The concept of swapping and moving elements is conceptually very easy, but this is because we take for granted the reasoning capabilities of humans.

3.1.3 Constraints of Line Balancing Problems

The line balancing problem differs from other problems solved by simulated annealing in that it is more constrained. The fact that it is constrained means that it may be difficult for the simulated annealing algorithm to sufficiently move elements such that every feasible solution *could* be achieved. One might think that the constraints should be relaxed for the purposes of generating alternatives, but this is impossible in the context of simulated annealing. If one allowed the algorithm to accept an infeasible solution, it would be difficult to find (via *one* swap or transfer) a feasible solution which is better. It is possible that the system could get completely off track and never return to the set of feasible solutions.

As discussed previously, Thomopoulos (1970) suggested setting a range on cycle time over which to optimize delta. However, for simulated annealing it is desirable to loosen the cycle time constraints to the maximum possible extent. Using a large upper limit for the cycle time will provide the algorithm with more flexibility to rearrange elements. Setting a lower limit on cycle time would have the effect of stifling

rearrangement and would make improbable any attempt to create new stations. Thus, in this research, no lower limit on cycle time was set. However, the upper limit on the number of stations has the effect of a limit on the cycle time by preventing a station "explosion" which would result in very low cycle times.

The precedence constraints will also restrict the movement of elements. The fact that a solution can differ from the previous solution by only one or two elements makes the precedence constraints particularly restrictive. If many elements could be moved at a time, the constraints would not present a problem. Nonetheless, only arrangements which satisfy the precedence constraints can be accepted. Precedence constraints *within* a station *can* be violated. As proven by Salveson (1955), elements within a station can be arranged so that they do not violate constraints. We are not particularly concerned with the arrangement within a station, but rather the assignments to stations.

3.1.4 Cooling Schedule and Stopping Criterion

Two cooling schedule methodologies were considered in this work. The first was used (in slightly different forms) by Kirkpatrick, et al. (1983), Wilhelm and Ward (1987), and Abramson (1991). This methodology sets a limit on the maximum number of attempted arrangements and the maximum number of accepted arrangements at any temperature level. When either of these limits is reached, temperature is lowered. If MaxAccepted has not been achieved for three successive temperatures, the system is considered frozen.

In this scenario, if MaxAccepted is too high relative to MaxAttempted, the system will not be able to find enough good solutions and it will think it has frozen. However, if MaxAccepted is too low, the temperature will be lowered prematurely, before enough of the solution space has been sampled. Temperature will be lowered rapidly, making the system very selective before a good solution has been found. There really are no guidelines for choosing the parameters, except that they should increase as problem size and number of stations increases so that the solution space will be sufficiently searched.

In addition to deciding *when* to lower temperature, one must also determine *how much* to lower it. The most commonly used means for decrementing temperature is an exponential method in which the temperature is decreased by a smaller amount each time it is reduced. This approach is in accordance with Kirkpatrick, et al.'s (1983) concept of annealing more slowly as the system approaches equilibrium. Thus, this method of reducing temperature was used in formulating the line balancing problem.

The second cooling schedule methodology is a slight variation of a method advocated by Das, et al. (1990). This method suggests sampling at each temperature the number of configurations that can be achieved from any given solution by only one random move. In the line balancing case, this would generally be the number of combinations of size two which could be formed from the total number of stations, times the number of elements expected in one station:

$$Trials\ @\ each\ temperature = \binom{NumStations}{2} \frac{NumElements}{NumStations}$$

The temperature will be lowered incrementally until the derivative with respect to temperature of the smoothed average of the objective function at the current temperature is less than a predetermined ε value. The advantage of this method is that the stopping criterion is actually what we want it to be, which is when the algorithm is having difficulty finding better solutions. However, it seems likely that the system could get in a temporary 'rut' in which it is not be able to change the objective function much. Hopefully, the smoothing will allow for enough configurations to be sampled to avoid this problem. The key is to strike a balance--to make the algorithm loose enough to allow generation of a large portion of the solution space yet strict enough not to allow chance to worsen a good solution due to too many samples.

The only difference between the method advocated by Das, et al. and the one used for this research is that the exponential method for reducing temperature was used instead of the more complex method used by Das, et al. Das, et al. used the Aarts and van Laarhoven (1985) method, which reduces the temperature more drastically the closer the solution is to equilibrium at the given temperature. This method may help the system to converge if the ultimate solution is considered to be the final solution at the end of the annealing instead of the best solution found during the annealing. However, a large percentage of the processing time will be spent at high temperatures where the probability of accepting all solutions, whether good or poor, is high. Furthermore, the probability of achieving the true optimal solution may be lowered, since the "fine-tuning" phase of the annealing will be substantially reduced.

Both of the methods chosen for modeling the mixed model line balancing problem are very logical means of performing the annealing because they conform to the problem at hand. If a number of temperatures is pre-selected for the problem, it will be somewhat haphazardly chosen, without corresponding to the problem. These methods instead impart a form of intelligence to the algorithm, allowing it to observe the performance of the system and to stop when it has determined that a good solution has been found.

It should be noted that in the case of line balancing, an attempt is not just one rearrangement, since that rearrangement may be infeasible. Several rearrangements must often be performed in order to get one attempt.

3.1.5 Initial Temperature

As discussed in Chapter 2, there are several philosophies regarding the selection of initial temperature. The objective in choosing an initial temperature is to select one that is sufficiently high that the elements can freely flow among the stations, or in other words, that the algorithm will accept poor solutions. This removes the dependency on the initial solution.

Two philosophies for determining initial temperature were considered in this research. They are similar in the fact that both approaches take sample rearrangements to find an initial temperature which is appropriate for each particular problem. The first method, used by Ku and Karimi (1990), involves multiplying the average change in objective function observed in the sample, ΔE_{ave}, by 1.5. From the Metropolis relation, we know that this corresponds to a probability of 0.5 of accepting the average move. For

the line balancing problem, Ku and Karimi's method was modified by instead multiplying ΔE_{max} by 1.5, meaning that the probability of accepting the *worst* move will be 0.5 at the beginning of the annealing.

The second method, used by Das, et al. (1990), is quite similar, but results in a higher initial temperature. This method involves multiplying the maximum change in objective function by ten, which corresponds to a probability of 0.9 of accepting the worst move. In generating the sample moves to make this calculation, Das, et al. accept all moves, whether they are improvements or not. However, with this approach, one might continue making a bad solution even worse, making the resulting maximum difference in objective function unrealistically high. Thus, in the line balancing formulation, all sample moves were generated from the initial solution.

These two procedures for determining initial temperature are appealing because they are intuitive. It makes sense to take samples from the problem, since this is an effective way to characterize the problem and ensure that the parameter chosen fits the problem. The procedures are also easy to program; they can be coded such that they will be transparent to the user. They will, however, require processing time, though it should be minimal.

3.1.6 Coding the Algorithm

The algorithm was coded in the C programming language. Each element's precedence constraints were contained in a binary word. This concept is depicted in Figure 3.1, where each row of the figure represents one binary word. The binary word

		Element							
		1	2	3	4	5	6		
Element	1	0	0	0	0	0	0	0	0
Element	2	1	0	0	0	0	0	0	0
Element	3	1	0	0	0	0	0	0	0
Element	4	0	0	1	0	0	0	0	0
Element	5	0	0	1	1	0	0	0	0
Element	6	1	0	0	0	1	0	0	0
		0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0

Figure 3.1: Binary Representation of Precedence Relationships

can be viewed as a one-dimensional array, where each bit position is an element of the array. Using several binary words, we can, in effect, generate a two-dimensional array, which is needed in order to represent precedence relationships between work elements. Each constraint relationship can be represented by a '1' in the bit position which corresponds to the prerequisite element. A '0' indicates no relationship between the elements. In the example, Element 1 has no precedence constraints, which means that it may be performed before all other elements. Element 5, however, cannot be performed until Elements 3 and 4 have been completed. An eight-bit word is used in the example, but larger problems would require longer words. Zeros are stored in the excess bits. The binary word structure allows operations involving the precedence relationships to be performed using bit manipulation features of C, which greatly speed processing.

There are two arrays which are used to perform the basic annealing process. One array holds the current solution. The contents of this array are copied (via a memory move command in C) to the other array, on which the rearrangement is performed. This array is used for calculating the objective function. If this solution is accepted by the algorithm, it is copied onto the current solution array. Otherwise, the current solution is copied onto the 'experimental' array for the next attempt. A supplemental array, which contains information on the number of elements in each station (or row of the solution arrays) is also duplicated for the same purpose as the solution array.

To begin the simulated annealing process, an initial feasible solution is required from which exchanges and transfers can be performed. It is important in the case of line balancing that the solution be feasible, since otherwise the algorithm might never find

other feasible solutions. The initial solution routine for this research is a very coarse routine which places elements in stations without concern for optimization. There is no need to begin annealing with a better solution as the annealing procedure begins with shuffling elements to lose dependency on the initial solution. A listing of the computer code for both cooling schedules is provided in Appendix A.

3.1.7 Initial Experimentation

Initial experimentation was performed on a case problem to demonstrate the feasibility of using simulated annealing to solve the mixed model line balancing problem. The experimentation indicated that the algorithm could perform sufficient rearrangements to locate the optimal solution, even with the cycle time and precedence constraints. Often the algorithm did not recognize the optimal solution and cease annealing, as it did not know that no better solutions existed. The stopping rules are such that annealing continued and worse solutions were occasionally accepted. As a result of this, the algorithm was modified to compare each accepted solution with the best solution found up to that point, and store the best solution. In this way, the algorithm was less dependent on the stopping rules.

3.2 METHODOLOGY

Once feasibility had been determined, the objective was then to assess the flexibility of the technique to solve the general set of line balancing problems. The predominant concern was whether a set of parameters, or a set of rules for selecting

parameters, could be determined such that simulated annealing can be simply and quickly applied to any given mixed model line balancing problem.

The first step in achieving this objective was to establish a set of problems that would be representative of the class of mixed model deterministic line balancing problems. Three characteristics of mixed model line balancing problems were considered particularly important; thus, these three characteristics were considered at two levels each, for a total of eight problems.

The first of the three factors is problem size, measured by the number of work elements. As discussed previously, the solution space increases as an exponential function of problem size. Two problems, one with a relatively small number of elements and one with a large number of elements, were chosen from the literature. The first has nineteen work elements and three models and is taken from Thomopoulos (1970). The data for this problem is given in Figure 3.2 and Tables 3.1 and 3.2. The second problem has 50 work elements and eight models and is taken from Dar-El and Nadivi (1981). The problem data is given in Figure 3.3 and Tables 3.3 and 3.4 .

The second factor is the F-ratio of the problem. This measure, suggested by Dar-El (1973), expresses the degree to which the problem is constrained by its precedence relationships. Thus, it provides an indication of the number of feasible solutions which may exist, which is a significant determinant of processing time. Specifically, the F-ratio is the ratio of '0' entries in the precedence matrix to the total number of entries in the matrix, where '0's indicate no precedence relationships, '1's indicate that an element is an immediate predecessor to another element, and '2's indicate that an element is a non-

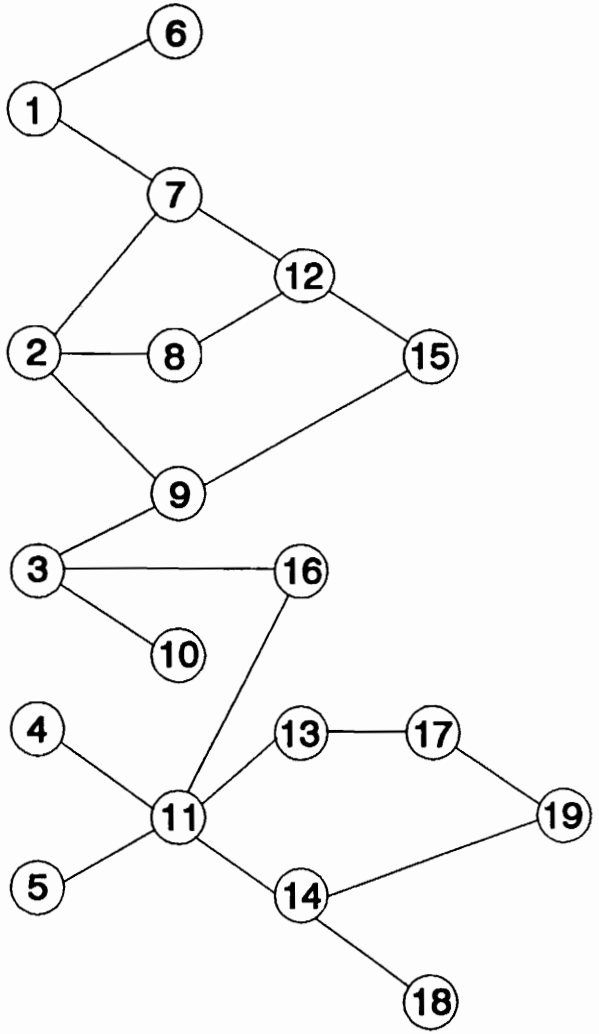

Figure 3.2: Precedence Diagram for the 19-Element Problem Taken from Thomopoulos (1970).

Table 3.1: Work Element Data for Each of the Three Models in the 19-Element Problem.

k	1	2	3	TTe
1	0.50	0.00	1.00	100
2	0.40	0.80	1.20	144
3	0.00	0.20	0.40	28
4	0.40	0.00	0.00	48
5	0.20	0.20	0.20	44
6	0.20	0.00	0.00	24
7	0.40	0.50	0.60	102
8	0.00	0.50	0.50	50
9	0.40	0.30	0.20	74
10	0.00	0.00	0.20	8
11	0.30	0.30	0.30	66
12	0.10	0.30	0.50	50
13	0.10	0.00	0.10	16
14	0.20	0.20	0.20	44
15	0.70	1.00	1.50	204
16	0.00	0.10	0.00	6
17	0.50	0.50	0.00	90
18	0.30	0.50	0.30	78
19	0.40	0.30	0.00	66
Total	5.10	5.70	7.20	1242

Table 3.2: Production Requirements Per Shift for the Three Models in the 19-Element Problem.

Model j	Nj
1	120
2	60
3	40
Total	220

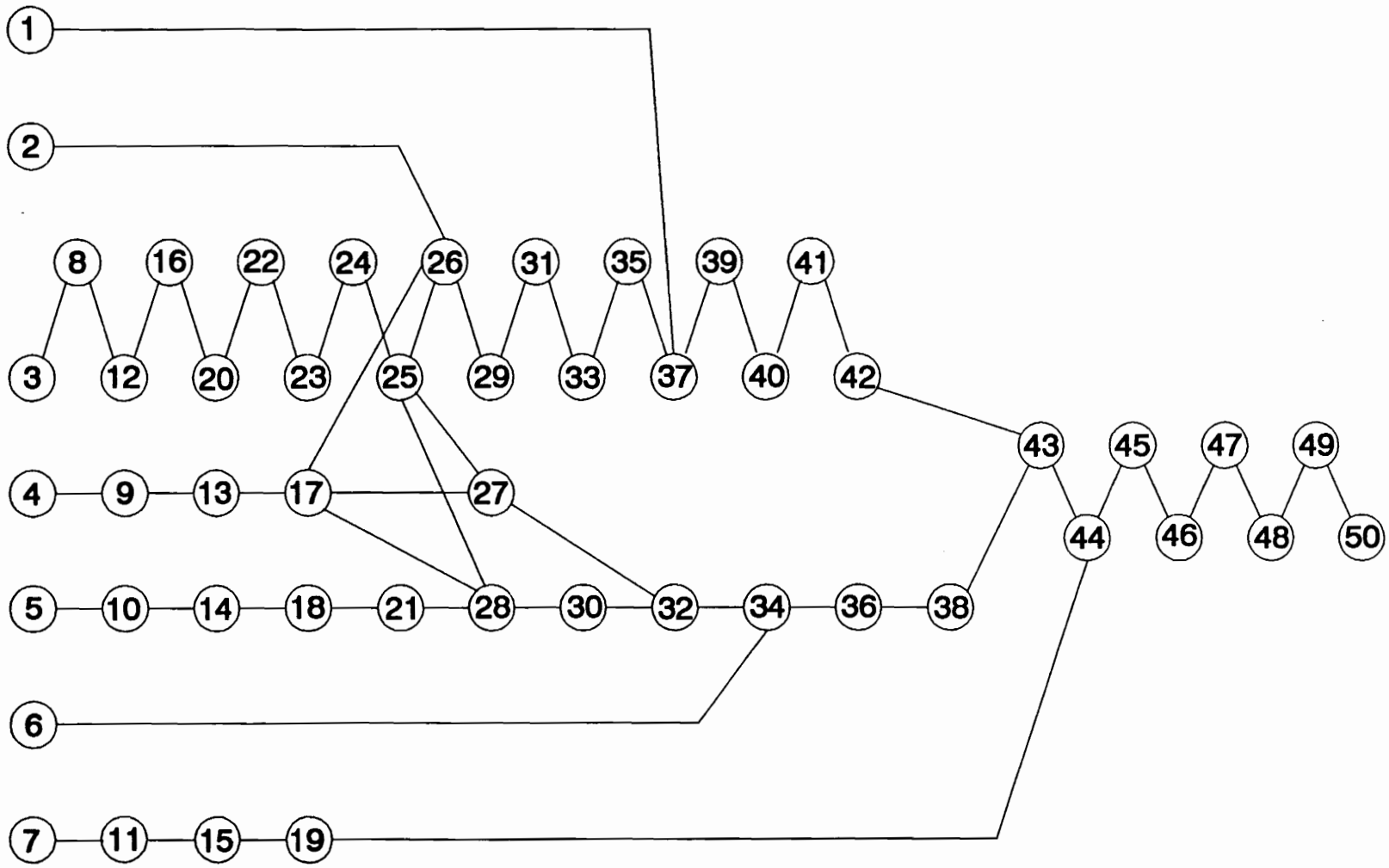

Figure 3.3: Precedence Diagram for the 50-Element Problem Taken From Dar-El and Nadivi (1981).

Table 3.3: Work Element Times for Each of the Eight Models of the 50-Element Problem.

	1	2	3	4	5	6	7	8	TTe
1	12.61	12.37	10.43	14.55	15.16	12.73	8.97	7.28	207
2	35.00	35.00	10.00	8.00	10.00	44.00	42.00	38.00	447
3	27.13	26.74	13.37	10.22	13.37	23.20	20.45	21.63	324
4	13.73	13.56	8.64	6.34	8.17	16.24	14.61	12.51	193
5	38.00	33.00	52.00	33.00	52.00	0.00	0.00	0.00	458
6	6.00	6.00	6.00	6.00	6.00	0.00	0.00	0.00	72
7	14.00	14.00	14.00	14.00	14.00	0.00	0.00	0.00	168
8	12.59	12.45	6.87	7.01	7.16	13.31	9.16	10.73	167
9	12.89	12.75	5.95	4.25	5.95	14.59	12.61	11.90	165
10	20.00	18.00	28.00	18.00	28.00	0.00	0.00	0.00	248
11	14.00	14.00	28.00	14.00	28.00	0.00	0.00	0.00	210
12	2.09	2.05	1.25	1.02	1.33	4.51	1.77	3.23	34
13	24.15	23.03	26.83	22.81	26.83	0.00	0.00	0.00	288
14	6.00	5.00	8.00	5.00	8.00	0.00	0.00	0.00	70
15	14.00	14.00	14.00	14.00	14.00	0.00	0.00	0.00	168
16	38.00	37.00	13.00	9.00	15.00	35.00	23.00	45.00	424
17	31.00	30.00	37.00	30.00	37.00	0.00	0.00	0.00	382
18	4.04	3.54	5.55	3.54	5.55	0.00	0.00	0.00	49
19	14.00	14.00	14.00	14.00	14.00	0.00	0.00	0.00	168
20	7.77	7.40	5.69	3.03	4.93	14.98	4.93	11.38	117
21	9.00	8.00	12.00	8.00	12.00	0.00	0.00	0.00	109
22	17.69	17.50	9.04	6.98	8.94	16.81	13.66	14.45	218
23	11.00	11.00	13.00	11.00	13.00	0.00	0.00	0.00	138
24	28.58	27.76	22.86	18.78	22.86	13.88	11.02	11.84	345
25	34.00	34.00	16.00	14.00	18.00	32.00	28.00	29.00	425
26	20.19	20.19	5.92	4.53	5.92	30.63	22.63	23.67	267
27	6.00	6.00	6.00	6.00	6.00	0.00	0.00	0.00	72
28	5.77	5.04	7.98	5.04	7.98	0.00	0.00	0.00	70
29	16.07	16.07	6.37	4.31	5.70	24.20	21.35	18.86	226
30	32.21	28.01	44.25	28.01	44.25	0.00	0.00	0.00	389
31	9.27	9.27	4.20	2.97	3.71	14.83	14.21	12.60	141
32	37.00	34.00	46.00	34.00	46.00	0.00	0.00	0.00	447
33	21.00	21.00	7.00	6.00	7.00	34.00	25.00	29.00	297
34	23.12	20.94	29.45	20.94	29.45	0.00	0.00	0.00	279
35	10.00	10.00	4.00	8.00	4.00	24.00	20.00	18.00	200
36	24.00	21.00	34.00	21.00	34.00	0.00	0.00	0.00	294
37	16.62	16.62	7.70	7.30	7.70	55.14	33.25	28.79	341
38	12.00	12.00	12.00	12.00	12.00	0.00	0.00	0.00	144
39	13.00	13.00	9.00	7.00	8.00	42.00	34.00	36.00	307
40	21.32	21.32	10.93	9.90	11.46	41.57	27.95	12.45	331
41	10.00	10.00	4.00	4.00	5.00	16.00	17.00	0.00	145
42	24.00	24.00	13.00	10.00	11.00	49.00	44.00	16.00	399
43	24.00	24.00	7.00	7.00	7.00	47.00	33.00	39.00	368
44	9.32	8.76	11.21	8.76	11.21	0.00	0.00	0.00	113
45	25.00	25.00	35.00	25.00	35.00	0.00	0.00	0.00	330
46	34.00	30.00	44.00	30.00	44.00	0.00	0.00	0.00	406
47	17.09	16.83	16.29	12.38	17.00	6.63	16.92	10.37	241
48	0.95	0.95	1.13	1.13	0.95	1.19	1.78	1.58	20
49	16.61	16.04	12.79	10.88	12.79	17.57	12.03	14.13	236
50	4.59	4.59	4.59	4.59	4.59	4.59	4.59	4.59	78
Total	880.40	846.78	765.29	587.27	770.96	649.60	517.89	481.99	11735

Table 3.4: Production Requirements Per Shift for the Eight Models of the 50-Element Problem.

Model j	Nj
1	1
2	4
3	2
4	4
5	1
6	2
7	2
8	1
Total	17

immediate predecessor. The two problems taken from the literature had F-ratios of 0.74 and 0.41 for the small and large problems, respectively. Two variants of these two problems were generated, such that the F-ratios were, respectively, 0.31 and 0.67. The precedence diagrams for these problems are given in Figures 3.4 and 3.5.

The final factor considered was the WEST ratio, or the Work Elements to STations ratio, proposed by Dar-El (1973). As Baybars (1986) notes, this measure by itself is not a strong indicator of processing time. However, for a given problem, if the cycle time is such that the WEST ratio is high, then there will be fewer unique feasible solutions than if the cycle time is such that the WEST ratio is low. Another important indicator of processing time is the allowable range given for the cycle time, as will be discussed later. The two original problems taken from the literature had WEST ratios of 6.3 and 2.1 for the small and large problems, respectively. In order to generate different levels of the WEST ratio, the ideal cycle times were adjusted from 414 and 500 minutes, respectively, to 205 and 1308 minutes. These modifications produced WEST ratios of 3.1 and 5.55 for the small and large problems, respectively. The set of eight experimental problems is summarized in Table 3.5.

Once these problems were established, a methodology was developed for determining appropriate SA parameters. Research has shown that the quality of the solution and the efficiency of the technique are significantly affected by the following four factors: (1) initial temperature, (2) the number of solutions generated at each temperature, (3) cooling rate, and (4) the efficiency of generating solutions from a

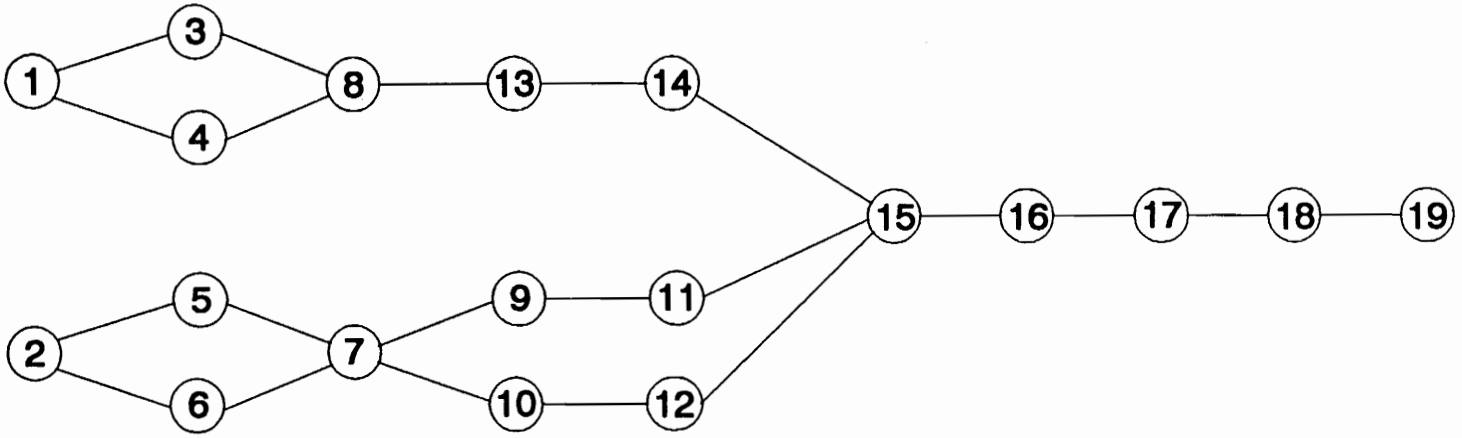

Figure 3.4: Precedence Diagram for the F-ratio Variant of the 19-Element Problem.

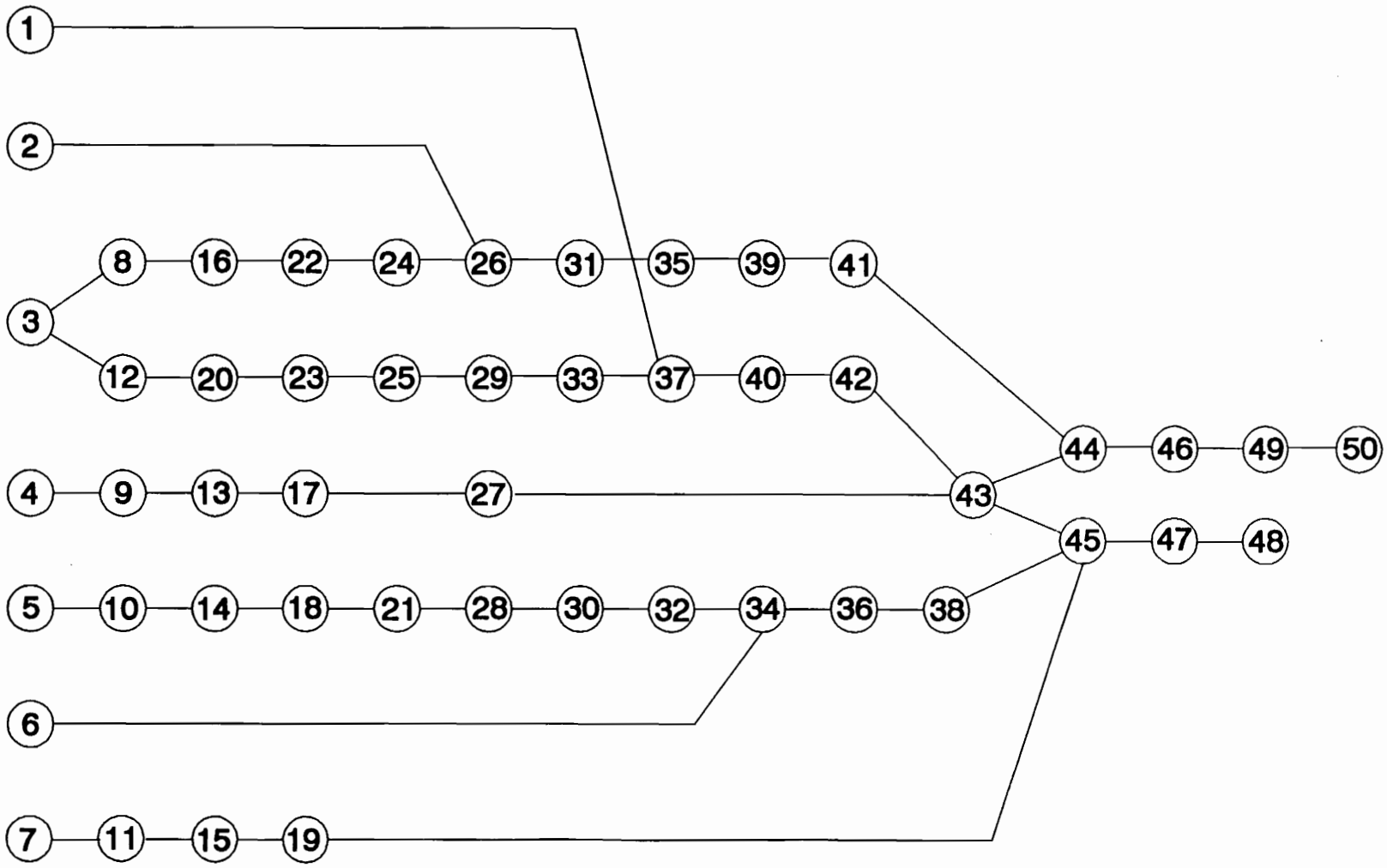

Figure 3.5: Precedence Diagram for the F-ratio Variant of the 50-Element Problem.

Table 3.5: The Set of Eight Experimental Problems.

		Small Problems (19 Elements)		Large Problems (50 Elements)	
		WEST ratio		WEST ratio	
		Low	High	Low	High
F-ratio	Low	Problem 1	Problem 3	Problem 5	Problem 7
	High	Problem 2	Problem 4	Problem 6	Problem 8

current solution [Ku and Karimi, 1991]. There are not many alternatives for solution generation in the line balancing case. Many of the generation techniques used in other problems are not applicable or are not practical for the line balancing problem. Thus, it was assumed that the generation method described in the previous section was acceptable. However, the remaining three factors were evaluated via experimentation.

Two cooling schedule methods were used, hereafter referred to as SA Method I and SA Method II. SA Method I requires values for MaxAttempted and MaxAccepted. The best values of these parameters were determined experimentally, by varying MaxAccepted and MaxAttempted and performing five replications at each level. SA Method II requires values for epsilon to be chosen. For this research, two values of epsilon were used, $\varepsilon = 0.0005$ and $\varepsilon = 0.00005$.

For SA Method I, two values of the cooling rate were tested, $\alpha = 0.9$ and $\alpha = 0.8$. For SA Method II, four values were used, $\alpha = 0.90, 0.95, 0.98$, and 0.99. A larger number of cooling rates were used for this technique because, unlike the first technique, the number of arrangements attempted at each temperature is fixed, and therefore the cooling rate may be the most important parameter in optimizing this method.

For both cooling schedule methods, two methods for generating initial temperature were used, $T_o = 1.5\ \Delta E_{max}$ and $T_o = 10\ \Delta E_{max}$. In all experiments, values for delta, processing time, and number of feasible solutions generated were recorded simultaneously.

The set of small problems was studied first. These problems were more suitable for extensive experimentation since they required less processing time. Also, since optimal solutions were available for these problems through exhaustive searches, the performance of SA could be accurately assessed. Efforts were focused on one problem at a time in order to study the behavior of the algorithm and to examine the effects of the parameters on solution quality and processing time.

Once the small problems were 'optimized', it was possible to seek trends and develop general rules. At this point, the information gained from the small problems was used in forming the experiments for the larger problems. Although a great deal of effort was expended to find optimal solutions to the four larger problems, these solutions could not be obtained, as explained in Chapter 4. To provide a basis for comparison for the larger problems, Smith's (1990) heuristic method was used.

After assessing the experimental results, a set of guidelines was developed to aid the assembly line designer. These guidelines suggest which line balancing methodologies may be most suitable for problems of different characteristics. They also suggest rules for selecting appropriate simulated annealing parameters should this be the balancing method chosen.

CHAPTER 4: EXPERIMENTAL RESULTS

This chapter presents the results of the experiments. Results from the exhaustive search and Smith's (1990) modified exhaustive search are provided first, since they serve as a baseline against which to compare the SA technique. SA results for the small problems are then presented for both Method I and Method II, followed by the large problem results for each method.

4.1 EXHAUSTIVE SEARCH RESULTS

4.1.1 Small Problems

The results of the exhaustive search runs are summarized in Table 4.1, while the actual station assignments are provided in Appendix B. The four small problems were run on IBM-compatible 80486 personal computers.

All but one of the four small problems were run for a completely exhaustive search, meaning that there were no limits placed on the cycle time of each station. For Problem 2, however, the lower limit on cycle time at each station was limited to 50% of the ideal cycle time. This limit was placed because of the very lengthy processing time which resulted without it. Placing a limit on cycle time has a profound effect on reducing processing time, because it drastically reduces the number of solutions the search must consider. The problem is that one does not know a priori what values to give the limits. If the limits are too tight, it may be impossible to find any feasible solutions. Also, solutions in which each station has a small deviation from the ideal cycle time will

Table 4.1: Exhaustive Search Results.

		WEST ratio: Low	WEST ratio: High
F-ratio	Low	**Problem 1** Delta: 254.0 Time: 268 min, 10 sec Feasible solns: 780,788	**Problem 3** Delta: 164.0 Time: 0 min, 45 sec Feasible solns: 2080
	High	**Problem 2** Delta: 161.7 * Time: 1976 min, 15 sec * Feasible solns: 112,542,808	**Problem 4** Delta: 32.0 Time: 136 min, 4 sec Feasible solns: 3,437,924

* This problem was run with a lower limit on cycle time of 50% of the ideal cycle time.

generally have smaller delta values, although this is not always the case. Delta is a measure of how much difference there is, for each model, between the cycle time of the assigned balance and an ideal balance. Minimizing delta is not the same as minimizing the difference between each station's cycle time and the ideal cycle time.

The upper limit on the number of stations also has a substantial effect on processing time. Again, however, one does not know a priori what value to assign. A very coarse assignment could be performed to get an upper bound, although reducing this number as much as possible will significantly reduce the amount of processing time. However, reducing the number of stations and reducing delta are not always compatible efforts. Sometimes a lower delta value can be achieved by increasing the number of stations. So again, it is difficult to choose proper limits for efficiently searching the solution space. And thus, the exhaustive search effort can require an inordinately large amount of processing time.

Looking at the small problems, the exhaustive search results are as one would expect. The two lowest delta values were achieved in the two most flexible arrangements, Problems 2 and 4. With a high F-ratio, the elements could be arranged in many different ways, increasing the probability that a good balance could be achieved.

As expected, the highest number of feasible solutions was found in Problem 2. Because of the high F-ratio, there are a large number of feasible solutions to the problem. Also, with a lower WEST ratio, the number of stations required to contain the elements is larger. The equation for permutations shows that with more stations, the number of possible arrangements is greater. Since Problem 1 also has a low WEST ratio, there are

many possible arrangements, but since the F-ratio is smaller, the number of *feasible* arrangements was less than for Problem 2. For Problems 3 and 4, there are fewer possible arrangements (due to the higher WEST ratio), and so the processing time was less. Again, the problem with the higher F-ratio required more time and generated more feasible solutions. The very substantial difference between processing time for Problems 2 and 3 (3½ days for Problem 2 run at only 50%, but less than a minute for Problem 3) very clearly shows the impact of the characteristics of a line balancing problem.

4.1.2 Large Problems

Despite quite a great deal of effort, optimal solutions for the set of four large problems were not obtained. The exhaustive search code was modified to support the larger problems. Solutions were generated using simulated annealing in order to determine feasible limits for the number of stations and cycle time. Even using the tightest feasible constraints known, the two small problems ran for four weeks on dedicated 80486 personal computers, and the larger problems ran for three weeks on RS-6000 series computers, without generating solutions. In all four cases, millions of feasible solutions had been generated; however, the best delta value achieved at that point was far higher than the values generated through simulated annealing.

It is very difficult to determine the number of solutions an exhaustive search would have to generate. The precedence and cycle time constraints will reduce the number significantly. However, a rough approximation of the number of solutions can be obtained via the following formula (adapted from Feller (1968)):

$$M! \binom{N-1}{M-1}$$

where N is the number of elements and M is the number of stations. Thus, for the 19-element problem, the number of solutions (feasible and infeasible permutations, where each station must have at least one element) would be 9.9×10^{19}, if we use M = 4. The 50-element problem would have 6×10^{73} permutations. The quickest 19-element problem was solved in 45 seconds. Even if the computer really did generate 2.2 x 1018 permutations per second (which is quite difficult to believe--the code must be structured to avoid a huge group of infeasible solutions), then to solve the quickest 50-element problem, (in which there would be 10 stations) would require 8.6×10^{47} years. Even if gross calculation errors have been made in this estimate, the orders of magnitude still suggest that exhaustive searches are quite impractical.

Since optimal solutions were not available via complete exhaustive search, Smith's (1990) modified exhaustive search technique was implemented for the four large problems to provide a baseline for comparison with simulated annealing. Smith's technique reduces the processing time by balancing a certain number of stations at a time. The method generates all possible balances for those stations, then the best balance is selected, and the remaining elements are considered for balancing the next group of stations. This procedure is continued until all work elements are assigned. The user of this technique decides the number of stations to be balanced at each step, and also decides what

tolerance to allow on the cycle time at each station. As a result, this method is likely to provide different results depending on what parameters it is given.

The advantage of this technique is that it can be quite time-effective. The code written by Smith is such that permutations will not be considered. This significantly reduces the solution space, and speeds up the search. Nonetheless, if the parameters on this problem are set such that the problem resembles an exhaustive search, the processing time will still make this technique infeasible for large problems.

There are inherent weaknesses in this technique. First, delta is a measure of smoothness of the entire balance, and thus it is somewhat inappropriate to assess delta for a subset of stations. If the cycle time limit is not set fairly tight on the first set of stations, a great deal of processing time will be required. Also, if the cycle time is not set fairly tightly, the algorithm will not try to pack elements into the stations. It will endeavor to minimize delta, which may mean placing fewer elements into the stations than need to be placed in order to obtain a desired number of stations. This is a significant problem. One solution is to set tight constraints on the first set, and gradually loosen the constraints. However, this will reduce the solution space searched, and will also probably produce a poor delta value, since delta measures smoothness along the line.

The results of this technique run for the four large problems are given in Table 4.2, while station assignments are provided in Appendix B. Problems 5 and 6 were run with a stepsize of five stations, while Problems 7 and 8 were run with a stepsize of two stations. The lower limit tolerances on cycle time per station are given in Table 4.2; the upper limit tolerance in all cases was zero. Time values are given as hours: minutes:

Table 4.2: Results of Smith's (1990) Modified Exhaustive Search for the Four Large Problems.

Problem 5		**Problem 7**	
Stations 1-5:	0.10	Stations 1-2:	0.10
Stations 6-10:	0.15	Stations 3-4:	0.15
Stations 11-15:	0.15	Stations 5-6:	0.15
Stations 16-20:	0.40	Stations 7-8:	0.15
Stations 21-27:	0.50	Stations 9-10:	0.30
Delta:	4449.47	Delta:	1288.04
Time:	0:31:14	Time:	0:42:27
Problem 6		**Problem 8**	
Stations 1-5:	0.03	Stations 1-2:	0.10
Stations 6-10:	0.05	Stations 3-4:	0.15
Stations 11-15:	0.20	Stations 5-6:	0.15
Stations 16-20:	0.35	Stations 7-8:	0.15
Stations 21-27:	0.35	Stations 9-10:	0.20
Delta:	4664.59	Delta:	1217.57
Time:	0:04:57	Time:	3:55:08

seconds. The delta values are the best results obtained after significant experimentation; however, this does not indicate that these are the best results obtainable by this method.

4.2 SIMULATED ANNEALING RESULTS

4.2.1 Small Problems

4.2.1.1 SA Method I

The first SA method that was explored was one used by many researchers. This method involves choosing two limits, one for the maximum number of attempted feasible solutions at a temperature, and the other for the maximum number of accepted solutions at a temperature. The temperature is lowered when one of these two limits is reached. If the maximum number of attempted solutions is reached before the maximum number of accepted solutions for three consecutive temperatures, the system is considered frozen.

Initial experiments were run with two values of cooling rate, α = 0.9 and 0.8, and two methods of calculating initial temperature as discussed in Chapter 3. Values for MaxAttempted were 100, 200, 300, 400, and 500, while values for MaxAccepted were 50, 100, 150, 200, 250, 300, 350, 400, and 450. All combinations were used, and five replications of each were generated. Data was collected on the best solution and its delta value, number of feasible solutions attempted, and processing time.

The data for Problem 3 is provided as an example in Tables 4.3 - 4.5. (Only the data for α = 0.9 is given to avoid overloading the reader. However, the same amount of data was collected for α = 0.8.) The shaded areas in the table of delta values indicate

Table 4.3: Delta Values: Problem 3, SA Method I

To: 1.5

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450
100	164								
	164								
	164								
	164								
	164								
200	164	164	164						
	164	164	164						
	164	164	164						
	164	164	164						
	164	164	164						
300	164	164	164	164	164				
	164	164	164	168	174				
	164	164	164	164	164				
	164	164	164	164	168				
	164	164	164	164	168				
400	164	164	164	164	164	164	164		
	164	164	164	164	164	164	174		
	164	164	164	164	164	164	164		
	164	164	164	164	164	164	180		
	164	164	164	164	164	168	174		
500	164	164	164	164	164	164	164	164	164
	164	164	164	164	164	164	164	164	164
	164	164	164	164	164	164	164	164	168
	164	164	164	164	164	164	164	164	186
	164	164	164	164	164	164	164	164	164

To: 10

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450
100	168								
	164								
	164								
	164								
	164								
200	164	164	164						
	164	164	164						
	164	164	164						
	164	164	168						
	164	164	164						
300	164	164	164	164	164				
	164	164	164	164	164				
	164	164	164	164	164				
	164	164	164	164	164				
	164	164	164	164	164				
400	164	164	164	164	164	164	164		
	164	164	164	164	164	164	164		
	164	164	164	164	164	164	164		
	164	164	164	164	164	164	164		
	164	164	164	164	164	164	168		
500	164	164	164	164	164	164	164	164	164
	164	164	164	164	164	164	164	164	164
	164	164	164	164	164	164	164	164	164
	164	164	164	164	164	164	164	164	164
	164	164	164	164	164	164	164	164	164

Table 4.4: Time Data: Problem 3, SA Method I

To: 1.5

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450
100	0:8								
	0:7								
	0:8								
	0:6								
	0:7								
200	0:12	0:13	0:11						
	0:14	0:13	0:9						
	0:11	0:13	0:12						
	0:11	0:14	0:12						
	0:12	0:12	0:11						
300	0:14	0:17	0:21	0:17	0:9				
	0:13	0:17	0:19	0:19	0:10				
	0:13	0:19	0:18	0:20	0:13				
	0:15	0:19	0:20	0:17	0:10				
	0:18	0:17	0:18	0:18	0:7				
400	0:17	0:20	0:23	0:24	0:25	0:22	0:10		
	0:17	0:23	0:24	0:28	0:25	0:20	0:12		
	0:18	0:21	0:24	0:26	0:25	0:17	0:9		
	0:19	0:22	0:23	0:27	0:25	0:21	0:8		
	0:17	0:20	0:24	0:24	0:24	0:22	0:8		
500	0:32	0:25	0:28	0:30	0:34	0:29	0:24	0:23	0:8
	0:19	0:23	0:28	0:29	0:34	0:29	0:27	0:23	0:8
	0:21	0:23	0:30	0:29	0:34	0:31	0:29	0:20	0:10
	0:27	0:25	0:29	0:30	0:32	0:31	0:25	0:20	0:8
	0:22	0:23	0:31	0:27	0:32	0:35	0:27	0:22	0:12

To: 10

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450
100	0:12								
	0:12								
	0:13								
	0:12								
	0:12								
200	0:15	0:24	0:25						
	0:15	0:23	0:25						
	0:15	0:22	0:24						
	0:15	0:24	0:24						
	0:17	0:23	0:24						
300	0:19	0:26	0:34	0:36	0:35				
	0:20	0:26	0:31	0:35	0:32				
	0:22	0:26	0:33	0:34	0:33				
	0:22	0:25	0:34	0:37	0:36				
	0:18	0:26	0:31	0:37	0:34				
400	0:24	0:32	0:37	0:42	0:50	0:48	0:42		
	0:20	0:30	0:39	0:44	0:48	0:50	0:42		
	0:23	0:29	0:37	0:43	0:46	0:46	0:40		
	0:26	0:31	0:37	0:43	0:46	0:49	0:42		
	0:26	0:29	0:38	0:45	0:48	0:44	0:37		
500	0:24	0:35	0:42	0:54	0:52	0:58	1:1	1:1	0:49
	0:25	0:33	0:39	0:48	0:54	0:58	1:0	0:57	0:51
	0:26	0:44	0:41	0:48	0:55	0:57	1:2	1:0	0:46
	0:23	0:34	0:45	0:49	0:53	0:58	0:59	0:55	0:51
	0:25	0:34	0:45	0:51	0:55	0:57	0:58	0:55	0:48

Table 4.5: Number of Feasible Solutions: Problem 3, SA Method I

To: 1.5

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450
100	1698								
	1568								
	1674								
	1383								
	1489								
200	2097	3150	2434						
	3442	2954	2033						
	2768	3143	2625						
	2589	3320	2607						
	3084	2862	2226						
300	3682	4413	4996	3908	1755				
	3469	4336	4746	4494	2056				
	3342	4636	4373	4565	2626				
	3830	4961	4668	3924	2046				
	4804	4141	4401	4176	1476				
400	4627	5027	5663	5803	5778	4876	1992		
	4493	5904	5965	6849	5734	4468	2390		
	5053	5352	6045	6291	6110	3702	1985		
	5223	5555	5695	6650	5790	4829	1595		
	4801	5185	5929	5891	5782	4866	1588		
500	8481	6422	7265	7398	8406	6940	6754	5296	1500
	5345	6177	7293	7037	8485	6902	6260	5311	1500
	5488	6164	7605	7358	8284	7414	6697	4340	1999
	7358	6434	7429	7499	7839	7361	5766	4276	1500
	6381	6171	8013	6916	7850	8491	6284	4809	2498

To: 10

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450
100	2580								
	2662								
	2948								
	2617								
	2480								
200	3657	5451	5333						
	3668	5036	5449						
	3614	5213	5263						
	3676	5284	4872						
	4357	5210	5288						
300	4902	6392	7855	7749	7378				
	4817	6232	7249	7740	6841				
	5337	5739	7523	7428	7079				
	5543	6112	7995	8293	7347				
	4309	6164	6902	8271	7347				
400	6417	8067	8960	9672	11252	10549	8642		
	5140	7209	9152	10042	10901	10961	8651		
	5995	7060	8591	10018	10480	10089	8254		
	6434	7231	8529	9741	10450	10570	8638		
	6960	7029	8964	10156	10904	9371	7835		
500	6093	8879	10444	12787	12186	13098	13441	13403	10037
	6866	8300	9241	11226	12065	13116	13391	12368	10555
	6613	11010	9740	11277	12801	13121	13923	12937	9573
	6148	8273	10566	11626	12126	13087	12828	11870	10530
	6463	8369	10666	12013	12643	12685	12920	11934	10074

sub-optimal solutions, while those regions which are not shaded indicate the optimal solutions found through the exhaustive search.

For each problem, the percentage of optimal solutions generated was calculated for each temperature/cooling rate combination. This data is shown in Table 4.6. In all cases, the higher cooling rate produced a higher percentage of optimal solutions. Thus, future experiments were run only with $\alpha = 0.9$. However, it was not clear which of the two methods for determining initial temperature was better, so experiments continued to include both methods.

As can be seen in Table 4.3, the initial set of parameters produced very good results for Problem 3. When $\alpha = 0.9$, the optimal solution (a delta value of 164) was generated over 92% of the time using either method for initial temperature. The smaller value of initial temperature showed some difficulty in cases where MaxAccepted was very close to MaxAttempted. This is understandable, given the way the cooling schedule works. Consider the case where MaxAttempted = 400 and MaxAccepted = 350. If, in 400 feasible solutions, 350 accepted solutions were not found, the temperature was dropped. If this occurred for three consecutive temperatures, the algorithm would consider itself frozen. This was not a problem for the higher initial temperature, because higher temperatures allow more solutions to be accepted. Thus more temperature drops were allowed, and more of the solution space was explored. Given the high success rate for this problem, additional experiments were not performed.

However, there were no significant patterns in the data for the other problems, so additional runs were made with higher values of MaxAttempted and MaxAccepted until

Table 4.6: Percentages of Optimal Solutions Generated During Initial Experiments with Two Cooling Rates and Two Initial Temperature Formulations.

Problem 1

	To: 1.5	To: 10
$\alpha = 0.9$	0.35	0.33
$\alpha = 0.8$	0.21	0.25

Problem 2

	To: 1.5	To: 10
$\alpha = 0.9$	0.30	0.27
$\alpha = 0.8$	0.20	0.21

Problem 3

	To: 1.5	To: 10
$\alpha = 0.9$	0.92	0.98
$\alpha = 0.8$	0.88	0.89

Problem 4

	To: 1.5	To: 10
$\alpha = 0.9$	0.62	0.72
$\alpha = 0.8$	0.34	0.46

patterns developed. Problem 4 exhibited very good results from parameter values ranging from 700 to 1900 for MaxAttempted and MaxAccepted values greater than 300, especially for the case where To = 10. In this region, the probability of achieving the optimal delta value was 98%. Delta values for these runs are shown in Table 4.7, while the processing times and the number of feasible solutions are provided in Appendix C. The bold line through the delta table indicates the area in which the value of MaxAccepted is one-half of the value of MaxAttempted. From the table, we see that parameters along this border provide excellent results.

The remaining two problems, Problems 1 and 2, required higher parameter values to achieve the same degree of success. In the case of Problem 2, this is because the number of feasible solutions is substantially higher than in any other problem. In the case of Problem 1, it is probably caused by the fact that the optimal solution requires eight stations, although the work elements will fit into as few as seven stations. The SA algorithm implemented in this research associates a small probability to the event of attempting to generate a new station, for reasons discussed in Chapter 3. Thus, the algorithm tends to minimize the number of stations, which for this problem is not necessarily optimal. Of the simulated annealing runs, Problem 1 required the most processing time because it has a large solution space but a small percentage of feasible solutions.

Delta values for Problems 1 and 2 are given in Tables 4.8 and 4.9, while the information on processing time and number of feasible solutions found is provided in Appendix C. Since values of MaxAccepted below 300 did not perform as well in early

Table 4.7: Delta Values: Problem 4, SA Method I

MaxAttempted \ MaxAccepted	50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
100	96																			
	32																			
	32																			
	32																			
	103																			
300	60	71	32	32	32															
	36	32	32	32	36															
	76	32	32	32	32															
	96	48	32	32	44															
	32	52	32	32	32															
500	32	32	32	32	32	32	32	32	32											
	32	32	56	64	32	32	32	32	32											
	32	60	36	32	32	48	32	32	32											
	32	32	56	32	32	32	32	32	48											
	32	52	32	32	48	32	32	32	52											
700	32	32	60	32	32	32	32	32	32	32	36									
	44	32	32	32	32	32	32	32	32	32	32									
	32	32	32	32	32	32	32	32	32	32	32									
	32	32	32	32	32	32	32	32	32	32	32									
	52	32	32	32	32	32	32	32	32	32	32									
900	52	32	32	32	32	32	32	32	32	32	32	32	32	32	32					
	56	32	32	32	32	32	32	32	32	32	32	32	32	32	32					
	60	32	32	32	40	32	32	32	32	32	32	32	32	32	32					
	32	32	48	32	32	32	32	32	32	32	32	32	32	32	32					
	48	32	32	32	32	32	32	32	32	32	32	32	32	32	32					

Table 4.7: Delta Values: Problem 4, SA Method I, continued

MaxAttempted	MaxAccepted 50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
1100	32	36	32	32	32	32	32	32	32	32	32	32	32	32	32					
	60	32	48	32	32	32	32	32	32	32	32	32	32	32	32					
	60	52	60	32	32	32	32	32	32	32	32	32	32	32	32					
	52	32	32	32	32	32	32	32	32	32	32	32	32	32	32					
	96	44	52	32	32	32	32	32	32	32	32	32	32	32	32					
1300	56	32	32	36	32	32	32	32	32	32	32	32	32	32	32	32	32			
	60	32	36	32	32	32	32	32	32	32	32	32	32	32	32	32	32			
	52	36	32	32	32	32	32	36	32	32	32	32	32	32	32	32	32			
	56	56	56	32	32	32	40	32	32	32	32	32	32	32	32	32	32			
	32	60	32	52	32	48	32	32	32	32	32	32	32	32	32	32	32			
1500	32	32	60	32	60	40	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	60	32	32	32	32	48	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	36	32	32	44	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	48	32	32	36	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	40	32	32	32	32	52	32	60	32	32	32	32	32	32	32	32	32	32	32	32
1700	48	32	32	32	48	32	32	32	36	32	32	32	32	32	32	32	32	32	32	32
	56	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	60	32	44	32	32	32	32	32	32	36	32	32	32	32	32	32	32	32	32	32
	32	32	44	32	32	32	32	36	32	32	32	32	32	32	32	32	32	32	32	32
	60	56	32	52	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
1900	60	32	32	48	32	32	52	32	32	32	32	32	32	44	32	32	32	32	32	32
	32	32	32	32	32	32	32	32	36	32	32	32	32	32	32	32	32	32	32	32
	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	60	32	32	56	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32
	60	32	32	52	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32

Table 4.8: Delta Values: Problem 1, SA Method I

MaxAccepted

MaxAttempted	MaxAccepted 300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
1100	254	254	254	254	254	254	254	254	254	254																			
	254	254	254	254	275.5	254	262	254	267.5	266																			
	254	254	254	262	254	254	254	254	262	254																			
	262	266	254	254	254	254	254	254	274	254																			
	254	254	254	254	254	254	262	254	254	262																			
1300	254	262	254	254	254	266	254	262	254	254	262	254																	
	254	254	254	262	254	254	254	262	254	254	254	254																	
	254	254	254	266	254	254	254	262	267.5	254	254	254																	
	254	254	254	254	254	254	267.5	254	254	254	262	254																	
	254	254	254	254	262	254	262	266	254	254	262	262																	
1500	254	254	278.3	254	254	254	254	254	254	254	254	254	254	254	254														
	262	254	254	254	262	254	254	254	254	262	254	254	254	254	254														
	278.3	254	262	254	254	254	254	254	254	262	254	254	254	278.3	254														
	254	267.5	275.5	254	254	254	254	254	254	254	254	266	266	254	262														
	267.5	254	254	254	254	262	254	254	254	262	254	254	254	254	254														
1700	278.3	278.3	275.5	266	254	254	254	254	254	254	266	254	266	254	254	254	275.5												
	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254												
	254	254	254	267.5	254	254	262	254	262	254	262	254	254	254	254	254	262												
	275.5	254	254	267.5	254	254	254	267.5	254	254	275.5	254	254	254	254	266	254												
	254	254	262	254	254	254	254	254	254	254	254	254	254	266	254	254	254												
1900	254	254	266	266	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254										
	262	254	278.3	254	254	254	262	254	254	254	266	254	254	254	254	254	254	262	254										
	254	254	254	254	254	266	254	254	254	254	254	254	254	254	254	254	254	262	254										
	278.3	254	254	254	254	254	254	254	262	254	254	254	262	254	254	254	262	254	254										
	254	266	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254										

Table 4.8: Delta Values: Problem 1, SA Method I, continued

MaxAccepted

MaxAttempted	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
2100	254	262	254	254	254	254	267.5	254	254	254	254	254	262	254	254	254	254	254	254	254	254								
	254	254	254	254	254	254	254	254	262	254	254	254	254	254	262	254	254	254	254	254	254								
	254	278.3	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	262	254	254								
	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254	266	254	254	254								
	254	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254								
2300	274	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254						
	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254						
	254	254	254	254	267.5	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254						
	266	254	254	262	262	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254						
	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254						
2500	262	254	254	266	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254				
	254	254	262	254	254	267.5	266	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	262				
	267.5	254	254	262	254	254	254	254	254	254	254	254	254	254	266	254	254	254	254	254	254	254	254	254	254				
	254	254	266	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254				
	254	254	254	254	254	254	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254				
2700	254	254	254	254	254	262	254	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254		
	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	266	254	254	254	254	254	254	254	254	254	254		
	254	267.5	254	254	254	254	254	254	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254		
	254	254	254	275.5	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254		
	254	254	254	254	254	262	254	254	254	266	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254		
2900	266	266	254	254	254	254	254	254	254	254	254	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254
	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254
	254	267.5	254	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254
	254	274	254	254	254	267.5	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254
	266	267.5	254	254	254	254	254	262	262	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	254	262	254

Table 4.9: Delta Values: Problem 2, SA Method I

MaxAccepted

MaxTemperature	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
1100	161.7	161.7	161.7	161.7	161.7	172.0	161.7	172.0	167.4	184.0																			
	161.7	161.7	161.7	161.7	161.7	161.7	167.4	173.7	161.7	179.4																			
	161.7	176.6	161.7	176.0	161.7	167.4	175.4	178.3	172.0	178.3																			
	172.0	161.7	167.4	161.7	161.7	161.7	167.4	161.7	173.7	172.0																			
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	178.3	172.0	167.4																			
1300	161.7	161.7	161.7	172.0	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7																	
	161.7	161.7	161.7	161.7	172.0	161.7	172.0	161.7	167.4	172.0	173.1	178.3																	
M	161.7	161.7	161.7	167.4	161.7	161.7	161.7	172.0	161.7	161.7	161.7	172.0																	
a	185.1	167.4	161.7	161.7	161.7	161.7	161.7	161.7	172.0	172.0	161.7	167.4																	
x	161.7	161.7	161.7	178.3	161.7	167.4	178.3	161.7	172.0	178.3	161.7	167.4																	
A 1500	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	167.4	161.7	161.7	161.7	167.4	189.7	172.0														
t	161.7	172.0	161.7	161.7	161.7	178.3	161.7	161.7	161.7	161.7	167.4	161.7	172.0	177.7	161.7														
t	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	175.4	161.7	178.3	161.7	161.7	167.4	177.7														
e	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	172.0	167.4														
m	172.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	172.0	161.7	172.0														
p 1700	161.7	161.7	176.0	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	176.0	167.4	161.7	184.0												
t	161.7	172.0	161.7	172.0	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	167.4	176.6	177.7	161.7												
e	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	186.3												
d	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	172.0	178.9												
	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	167.4	161.7	167.4												
1900	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	161.7	167.4	167.4										
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	178.3										
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	167.4	161.7	172.0										
	184.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	167.4	172.0										
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	172.0	161.7	176.0	167.4										

Table 4.9: Delta Values: Problem 2, SA Method I, continued

MaxAccepted

MaxAttempted	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
2100	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	177.7	161.7								
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	172.0	177.7								
	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	161.7	161.7	161.7	167.4	172.0	167.4	161.7	172.0	173.7								
	161.7	161.7	178.9	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	172.0	161.7	161.7	172.0	161.7	161.7	161.7	167.4	161.7	172.0								
	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	167.4	161.7	167.4	161.7	167.4	161.7	161.7	161.7	172.0	161.7	161.7	161.7	161.7								
2300	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	167.4	161.7	172.0						
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	175.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	177.7						
M	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	172.0	167.4						
a	172.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	172.0						
x	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	172.0	172.0	161.7	172.0						
A 2500	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7				
t	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	177.7				
t	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7					
e	161.7	172.0	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	172.0	172.0	161.7				
m	161.7	180.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	173.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	167.4	161.7				
p 2700	167.4	173.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7		
t	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7		
e	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7		
d	161.7	178.3	161.7	174.9	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	178.3		
	184.0	172.0	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7		
2900	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	173.7	172.0	161.7
	161.7	161.7	161.7	167.4	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7
	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	172.0	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7
	161.7	161.7	161.7	176.6	161.7	173.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	161.7	167.4

runs, these values were not used in the experiments. In the case of Problem 1, good results were obtained with MaxAttempted values of 2300 or larger, and MaxAccepted values greater than 600. In this region of the table, the optimal solution was found 97% of the time. Values along the border (where MaxAccepted is approximately one-half of MaxAttempted) seemed to produce the best results.

In the case of Problem 2, good results were obtained with values of MaxAttempted of 2300 or larger, and values of MaxAccepted less than or equal to one-half of MaxAttempted. In this region, the optimal solution was found 94% of the time. The data for this problem indicates that ratios of MaxAccepted to MaxAttempted of greater than one-half will produce sub-optimal answers.

In general, it is apparent that increasing the parameters (in proportion) improves the solution quality. The tradeoff is increased processing time, although processing times for these problems ranged from several seconds to less than five minutes. A general guideline seems to be to select values of MaxAccepted and MaxAttempted such that their ratio will be one-half or slightly less. This is a sensible result, since ratios which are too high will force the system to prematurely freeze and ratios which are too low may anneal too quickly and not search enough of the solution space.

4.2.1.2 SA Method II

The second simulated annealing method explored was a variation of a method used in Das, et al. (1990). Das, et al. used the Metropolis acceptance criterion and a cooling schedule suggested by Aarts and van Laarhoven (1985). This cooling schedule involves

a fixed number of attempts at each temperature, which is equal to the number of possible configurations which can be achieved by only one random move. The system is considered frozen when the derivative of the smoothed average value of the objective function at a temperature level with respect to the temperature is less than a specified value of a parameter, epsilon. Aarts and van Laarhoven also use a different formula for reducing temperature; however, in this research, the exponential method was used as in Method I.

The small problems were run with four different values of cooling rate, α = 0.90, 0.95, 0.98, and 0.99, and two different methods for calculating initial temperature, as discussed in Chapter 3. Also, two different values of epsilon were tried, ε = 0.0005 and 0.00005. Ten replications of each were run, and the results are provided in Tables 4.10-4.13. The results appear to indicate that the higher values of cooling rate and the higher starting temperature perform best. For the higher starting temperature and a cooling rate of α = 0.99, the percentage of optimal solutions achieved was 80%, 80%, 95% and 75% for Problems 1, 2, 3, and 4, respectively. While these values are high, the first SA method produced higher percentages. This method has an advantage, however, in that the user is given an equation for determining the number of attempts to generate at each temperature. This is particularly helpful if no guidelines are available or if no experimentation has been performed.

Table 4.10: Delta Values: Problem 1, SA Method II

To: 1.5

	α			
	0.90	0.95	0.98	0.99
0.00050	204.6	167.4	161.7	161.7
	161.7	172.0	161.7	161.7
	161.7	161.7	161.7	175.4
	161.7	161.7	161.7	161.7
	180.0	176.0	167.4	161.7
	161.7	199.4	161.7	161.7
	172.0	181.1	161.7	161.7
	172.0	198.3	161.7	161.7
	208.0	208.0	161.7	161.7
	259.4	176.0	172.0	161.7
0.00005	178.3	161.7	161.7	199.4
	177.7	202.9	172.0	161.7
	161.7	197.1	161.7	161.7
	192.0	161.7	161.7	161.7
	172.0	186.3	161.7	161.7
	195.4	161.7	161.7	161.7
	208.0	161.7	161.7	178.3
	161.7	161.7	161.7	161.7
	161.7	161.7	161.7	161.7
	176.0	172.0	161.7	161.7

To: 10

	α			
	0.90	0.95	0.98	0.99
0.00050	161.7	195.4	161.7	161.7
	210.9	161.7	199.4	161.7
	200.0	161.7	161.7	161.7
	161.7	161.7	161.7	236.6
	208.0	161.7	161.7	161.7
	161.7	184.0	161.7	161.7
	161.7	185.1	190.3	161.7
	208.0	167.4	167.4	167.4
	176.0	178.3	194.3	167.4
	176.6	161.7	178.3	161.7
0.00005	178.3	161.7	175.4	161.7
	161.7	176.0	176.6	161.7
	161.7	161.7	161.7	161.7
	213.1	161.7	172.0	161.7
	161.7	202.9	161.7	161.7
	167.4	185.7	172.0	161.7
	167.4	172.0	185.1	161.7
	161.7	161.7	161.7	161.7
	184.0	161.7	161.7	177.7
	178.3	161.7	161.7	161.7

Table 4.11: Delta Values: Problem 2, SA Method II

To: 1.5

	α			
	0.90	0.95	0.98	0.99
0.00050	254.0	278.3	254.0	275.5
	254.0	254.0	254.0	254.0
	266.0	278.3	254.0	254.0
	278.3	278.3	254.0	254.0
	278.3	254.0	254.0	262.0
	254.0	267.5	254.0	254.0
	275.5	278.3	254.0	254.0
	254.0	278.3	254.0	254.0
	281.1	262.0	262.0	267.5
	278.3	254.0	254.0	266.0
0.00005	254.0	254.0	254.0	267.5
	267.5	262.0	275.5	254.0
	278.3	254.0	254.0	254.0
	254.0	275.5	254.0	254.0
	278.3	262.0	254.0	254.0
	279.0	254.0	275.5	254.0
	278.3	278.3	262.0	274.0
	254.0	275.5	275.5	275.5
	278.3	278.3	278.3	254.0
	267.5	262.0	254.0	254.0

To: 10

	α			
	0.90	0.95	0.98	0.99
0.00050	278.3	278.3	254.0	254.0
	254.0	262.0	275.5	254.0
	275.5	254.0	254.0	254.0
	275.5	254.0	254.0	254.0
	278.3	274.0	274.0	254.0
	278.3	279.0	254.0	266.0
	266.0	275.5	254.0	254.0
	278.3	275.5	266.0	254.0
	278.3	274.0	254.0	254.0
	266.0	254.0	274.0	262.0
0.00005	254.0	254.0	262.0	254.0
	262.0	278.3	254.0	254.0
	280.5	254.0	254.0	254.0
	281.1	254.0	254.0	254.0
	254.0	266.0	267.5	274.0
	280.5	278.3	254.0	254.0
	278.3	278.3	254.0	254.0
	278.3	254.0	254.0	254.0
	278.3	275.5	254.0	262.0
	278.3	262.0	254.0	254.0

Table 4.12: Delta Values: Problem 3, SA Method II

To: 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	164	164	164	164
	164	164	164	164
	164	164	164	164
	174	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164
	174	164	164	164
0.00005	164	164	164	164
	164	164	164	164
	164	164	164	192
	164	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164
	164	164	164	164

To: 10

	α			
	0.9	0.95	0.98	0.99
0.00050	164	164	164	164
	164	164	164	164
	164	164	164	168
	164	168	164	164
	164	164	164	164
	164	164	164	164
	168	164	164	164
	164	164	164	164
	168	164	164	164
	164	186	164	164
0.00005	164	164	164	164
	168	164	164	164
	164	164	164	164
	164	164	164	164
	168	164	164	164
	164	164	164	164
	164	164	164	164
	168	164	164	164
	164	164	164	164
	174	164	164	164

Table 4.13: Delta Values: Problem 4, SA Method II.

To: 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	44	32	32	32
	112	96	60	32
	96	88	60	32
	108	60	32	32
	88	60	32	68
	96	96	36	44
	60	32	32	60
	36	32	44	48
	32	68	60	52
	32	96	32	32
0.00005	80	68	68	60
	88	80	32	32
	71	32	40	52
	60	60	68	32
	72	60	32	32
	127	60	72	32
	36	60	60	32
	96	60	32	32
	100	120	48	32
	96	60	32	32

To: 10

	α			
	0.9	0.95	0.98	0.99
0.00050	106	44	32	32
	56	60	32	32
	60	52	32	32
	52	56	32	32
	120	96	52	36
	104	72	32	32
	96	60	32	36
	56	32	36	32
	48	32	60	32
	108	64	32	32
0.00005	68	32	32	52
	96	56	48	56
	36	80	32	40
	32	32	32	32
	60	64	32	32
	68	52	32	32
	76	32	36	32
	60	32	60	32
	100	32	32	32
	32	36	32	32

4.2.2 Large Problems

4.2.2.1 SA Method I

The information gained from applying this technique to the set of small problems was used in selecting parameters for the larger problems. Since the solution space of the large problems is much greater than that for the small problems, the parameters should be larger. Also, it was determined from the small problems that problems with small WEST ratios require larger parameter values because the increased number of stations greatly increases the solution space. Of the two problems with higher WEST ratios, the less-constrained problem will require the smallest parameters. In addition, it was known that the ratio of MaxAccepted to MaxAttempted should be approximately 1:2. The question which remained was to what degree the parameters should be increased.

The parameters should be predominantly a function of the problem size, as noted by Abramson (1991), since the solution space will be much larger. For the small problem, with 19 elements, the range of MaxAttempted which produced very good results was approximately 50n to 150n, where n is the number of elements. Using this same formula to determine parameters for the larger problem would not likely be successful, since the solution space increases exponentially for the larger problem. Thus, experimentation was used to suggest good parameters.

Preliminary experiments were run for all four problems to establish a baseline set of delta values which could be used to indicate improvement. The parameters chosen were approximately double those used for the small problems. The lower initial temperature was used to reduce processing time. The data is provided in Table 4.14.

Table 4.14: Preliminary Data for Large Problems Using Method I.

Problem 5

MaxAttempted \ MaxAccepted	1500	2000	2500
3000	3531.46 3807.53 3827.64		
4000		3545.38 3626.67 3549.88	
5000			3564.63 3522.13 3539.40

Average Time: 59 mins

Problem 7

MaxAttempted \ MaxAccepted	1500	2000	2500
3000	1168.36 1179.96 1168.49		
4000		1182.11 1185.61 1225.45	
5000			1173.13 1147.52 1154.91

Average Time: 50 mins

Problem 6

MaxAttempted \ MaxAccepted	1500	2000	2500
3000	3309.09 3431.00 3325.36		
4000		3314.71 3436.64 3291.24	
5000			3304.66 3279.14 3278.29

Average Time: 50 mins

Problem 8

MaxAttempted \ MaxAccepted	1500	2000	2500
3000	873.51 930.78 943.65		
4000		837.86 910.27 908.26	
5000			826.57 838.56 890.28

Average Time: 32 mins

Every run of the simulated annealing algorithm produced a better solution than that found through Smith's heuristic (Table 4.2), even using small SA parameters.

However, since the preliminary SA data showed no convergence, Problem 7 (the problem with the high WEST ratio and low F-ratio--the characteristics shown in the small set to require the smallest parameters and give the best response) was used in several more experiments, each with increased parameters. Again, the lower initial temperature method, as well as MaxAttempted/MaxAccepted ratios of less than 50% were used in order to explore the range of parameters without requiring excessive processing time. The results of these runs are shown in Table 4.15.

Some degree of convergence was noted at the highest MaxAttempted values; in fact, the best solution found via SA Method II was achieved. These MaxAttempted values correspond to approximately 500n. This multiplier is a factor of ten greater than the multiplier for the 19 element problem. Thus, it was hypothesized that a range of 500n to 1500n for MaxAttempted would be successful for the set of large problems.

In keeping with the results from the small problems, Problem 7 was run with parameters at the low end of the range, Problem 8 with somewhat higher parameters, and Problems 5 and 6 with the highest parameters in the range. The runs were made with MaxAccepted/MaxAttempted ratios of both 40% and 50%. In all cases, $T_o = 10\Delta E_{ave}$ (the higher initial temperature formulation), and $\alpha = 0.9$. The data from these runs is shown in Tables 4.16 and 4.17.

Table 4.15: Experimental Data for Problem 7 Using Method I.

Problem 7

MaxAttempted \ MaxAccepted	2000	3000	4000	5000	6000	7000	8000
7000	1140.49 1152.30 1157.96						
9000		1154.66 1141.06 1140.49					
11000			1140.49 1156.71 1151.68				
13000				1161.48 1146.48 1141.06			
15000					1154.60 1140.49 1152.08		
17000						1149.30 1130.23 1160.12	
19000							1130.23 1147.52 1160.87

Table 4.16: Delta Values for the Four Large Problems Using Method I at a 40% Ratio of MaxAccepted to MaxAttempted.

Problem 5
MaxAttempted = 75000
MaxAccepted = 30000
3549.63
3517.35
3517.35

Problem 7
MaxAttempted = 35000
MaxAccepted = 14000
1151.68
1139.17
1151.56

Problem 6
MaxAttempted = 75000
MaxAccepted = 30000
3240.59
3277.85
3279.14

Problem 8
MaxAttempted = 50000
MaxAccepted = 20000
747.57
748.96
782.79

Table 4.17: Delta Values for the Four Large Problems Using Method I at a 50% Ratio of MaxAccepted to MaxAttempted.

Problem 5
MaxAttempted = 75000
MaxAccepted = 37500
3525.29
3558.11
3524.04

Problem 7
MaxAttempted = 35000
MaxAccepted = 17500
1140.49
1140.49
1146.48

Problem 6
MaxAttempted = 75000
MaxAccepted = 35000
3278.57
3251.82
3251.82

Problem 8
MaxAttempted = 50000
MaxAccepted = 25000
812.99
755.51
820.40

The two MaxAccepted/MaxAttempted ratios produced similar results. In both cases, the data is substantially better than that found in the preliminary runs, and it is also substantially better than that found using Smith's modified exhaustive search. It is also interesting to note the fairly tight range of the data values, which seems to indicate convergence. The processing time for these problems followed the same pattern as for the small problems, as the low WEST ratio/low F-ratio combination problem required the most processing time (approximately 26 hours), while the high WEST ratio/high F-ratio problem required the least (approximately 8 hours).

4.2.2.2 SA Method II

Since for the small problems, a cooling rate of 0.99 produced the best results, experiments were run only for this cooling rate. Both initial temperature methods and both values of epsilon were used, and the results are provided in Table 4.18. The data found using this method is very similar to that found using Method I with the 40% ratio. However, the processing time and number of feasible solutions searched were significantly smaller for this method than for SA Method I. The actual time values are provided in Appendix D, but they range from 7 hours for the low WEST ratio/low F-ratio problem to 45 minutes for the high WEST ratio/high F-ratio problem. It is also interesting to note that for this method, the lower value of initial temperature performed slightly better.

Table 4.18: Delta Values for the Four Large Problems Using SA Method II.

Problem 5

	To: 1.5	To: 10
0.00050	3517.35	3517.35
	3517.35	3775.00
	3522.13	3517.35
	3520.51	3520.51
	3517.35	3517.35
0.00005	3517.35	3671.79
	3517.35	3557.57
	3522.13	3531.46
	3584.80	3584.80
	3755.21	3755.21

Problem 7

	To: 1.5	To: 10
0.00050	1156.21	1193.44
	1130.23	1151.68
	1130.23	1157.72
	1147.76	1156.21
	1173.32	1157.72
0.00005	1130.23	1157.72
	1147.76	1157.72
	1161.68	1130.23
	1142.08	1173.25
	1200.08	1204.18

Problem 6

	To: 1.5	To: 10
0.00050	3295.61	3256.48
	3282.85	3835.57
	3240.59	3291.24
	3279.14	3291.71
	3220.59	3312.34
0.00005	3279.14	3279.14
	3300.17	3251.82
	3274.08	3283.15
	3279.14	3240.59
	3291.71	3254.79

Problem 8

	To: 1.5	To: 10
0.00050	800.29	795.89
	774.84	891.46
	759.80	762.74
	790.34	723.59
	756.51	733.99
0.00005	756.73	777.84
	730.94	823.90
	744.22	753.62
	792.84	742.54
	734.49	735.76

4.3 SUMMARY OF RESULTS

4.3.1 Small Problems

To facilitate comparison of the three line balancing methodologies (exhaustive search, SA Method I, and SA Method II), a summary table, Table 4.19, is provided for the four small problems. This table provides information on the best parameter set for each method, the percentage of optimal solutions found when the best parameter set was used, and an approximation of processing time for each method.

4.3.2 Large Problems

Table 4.20 summarizes the results of the experiments with the large problems. This table is similar to the one for the small problems. However, since the true optimal solution is not known, the best solution found using each method is provided. Also, the experimental parameters used, and the approximate processing times involved are recorded.

Table 4.19: Summary of Results for the Small Problems

Problem 1

	Exhaustive Search	SA Method I	SA Method II
Best Parameters	N/A	MaxAttempted:MaxAccepted = 2:1 MaxAttempted > 2300	alpha = 0.99 Either To Either epsilon
% Optimal	100%	97%	80%
Approximate Time	*268 mins	10 mins	6 mins

Problem 2

	Exhaustive Search	SA Method I	SA Method II
Best Parameters	N/A	MaxAttempted:MaxAccepted = 2:1 MaxAttempted > 2300	alpha = 0.99 Either To Either epsilon
% Optimal	100%	94%	80%
Approximate Time	*1976 mins	4 mins	2.5 mins

Table 4.19: Summary of Results for the Small Problems, continued

Problem 3

	Exhaustive Search	SA Method I	SA Method II
Best Parameters	N/A	100 < MaxAttempted < 500 50 < MaxAccepted < 450	alpha = 0.99 Either To Either epsilon
% Optimal	100%	98%	95%
Approximate Time	* 45 secs	40 secs	1 min

Problem 4

	Exhaustive Search	SA Method I	SA Method II
Best Parameters	N/A	MaxAttempted:MaxAccepted = 2:1 MaxAttempted > 700	alpha = 0.99 Either To Either epsilon
% Optimal	100%	98%	75%
Approximate Time	*136 mins	2.5 mins	40 secs

* These problems were run with very loose constraints. Times would be less if tighter constraints on cycle time were given.

Table 4.20: Summary of Results for the Large Problems

Problem 5

	Smith's Heuristic	SA Method I	SA Method II
Parameters Used	Stepsize = 5 stations	MaxAttempted = 75000 MaxAttempted:MaxAccepted Ratios of 40% and 50%	alpha = 0.99 Both To Both epsilons
Best Solution Found	4449.47	3517.35	3517.35
Approximate Time	31 mins	26 hours	7 hours

Problem 6

	Smith's Heuristic	SA Method I	SA Method II
Parameters Used	Stepsize = 5 stations	MaxAttempted = 75000 MaxAttempted:MaxAccepted Ratios of 40% and just under 50%	alpha = 0.99 Both To Both epsilons
Best Solution Found	4664.59	3240.59	3220.59
Approximate Time	5 mins	20 hours	4 hours

Table 4.20: Summary of Results for the Large Problems, continued

Problem 7

	Smith's Heuristic	SA Method I	SA Method II
Parameters Used	Stepsize = 2 stations	MaxAttempted = 35000 MaxAttempted:MaxAccepted Ratios of 40% and 50%	alpha = 0.99 Both To Both epsilons
Best Solution Found	1288.04	1139.17	1130.23
Approximate Time	42 mins	10 hours	1.5 hours

Problem 8

	Smith's Heuristic	SA Method I	SA Method II
Parameters Used	Stepsize = 2 stations	MaxAttempted = 50000 MaxAttempted:MaxAccepted Ratios of 40% and 50%	alpha = 0.99 Both To Both epsilons
Best Solution Found	1217.57	747.57	723.59
Approximate Time	240 mins	8 hours	1 hour

CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH

5.1 CONCLUSIONS

The objectives of this research were to determine the feasibility of applying the simulated annealing technique to the mixed model assembly line balancing problem, to assess its effectiveness, and to provide a set of general guidelines for applying the SA to any mixed model problem. These objectives have been met and are described in the following sections.

5.1.1 Assessment of SA Applied to Mixed Model Line Balancing

This research has proven simulated annealing to be a feasible technique for solving the deterministic mixed model assembly line balancing problem. Although the previous applications of simulated annealing have involved problems with no constraints, or very simple constraints, the technique worked quite well for this highly-constrained problem. The technique was tested against a set of problems which represent the range of characteristics of line balancing problems, and it performed well in all cases. Simulated annealing is not only feasible, but it has also been shown to be very effective in solving this problem. It is sensitive to the parameters used, yet as long as the parameters are reasonable, very good solutions are generated. It provides a solution to the problem in a fraction of the time required for an exhaustive search.

5.1.2 Guidelines for When to Use SA

Simulated annealing is the best approach for large problems. Without question, simulated annealing is preferred to exhaustive searches. As seen in this research, exhaustive searches require a prohibitively large amount of processing time. As problem size increases, processing time increases exponentially. However, simulated annealing provides an intelligent means for searching the solution space, and provides very good solutions even for large problems in just a few hours. Although Smith's (1990) modified exhaustive search is also quite time-effective, simulated annealing found significantly better solutions.

For small problems, however, it may be more desirable to run an exhaustive search, depending on the nature of the problem and the objective of the assembly line designer. Exhaustive searches guarantee the optimal solution, which may be worth the additional time, since in the small problem case the additional time will not be much. The actual time required is a function of the particular problem instance. If the work element times and the cycle time are such that the elements fit nicely into the stations without much variation from the ideal cycle time, then the exhaustive search can be given tight constraints which will still allow feasible solutions to be generated. However, if this is not the case, and the exhaustive search must peruse a large portion of the solution space, then simulated annealing may be more appropriate.

For the 19-element problem used in this research, in every case except the low WEST ratio/high F-ratio case, the entire solution space was explored (via a 100% exhaustive search) in under 4½ hours. However, the low WEST ratio/high F-ratio

problem required 1½ days of CPU time, even when constrained to a cycle time limit of 50% of the ideal cycle time. It happens that these particular problems, due to their nature, could have been constrained such that they would have required less time. However, for problems which cannot be sufficiently constrained, simulated annealing is the better choice. If an exhaustive search is deemed necessary, simulated annealing could be used as a very helpful complement, by quickly generating solutions which will indicate how much the exhaustive search can be constrained.

In practice, assembly line design requires consideration of many factors. Thus, designers often wish to experiment with different design parameters. During the early stages of design, the designer will probably be willing to accept good solutions instead of optimal solutions if they require less time. In these circumstances, simulated annealing may be preferred, even in the small problem case. Simulated annealing provides the user with a great deal of flexibility, since by changing the values of the parameters, the user can regulate how much emphasis is placed on optimality versus speed.

Regardless of the problem size, simulated annealing is best suited to problems where the objective is to minimize delta and the number of stations is not fixed. It is quite easy with SA to set boundaries on the number of stations desired, but to guarantee a fixed number of stations would require that the initial solution have that exact number of stations and that the algorithm not allow the addition or removal of stations. This is possible, but somewhat more taxing on the SA algorithm since there will be much less flexibility for elements to move among the stations.

5.1.3 Guidelines for Applying Simulated Annealing to the General Case of Mixed Model Problems

The results of this thesis are sufficient to provide a user of simulated annealing with a general set of guidelines for selecting SA parameters which are appropriate for his/her particular problem. The experimentation indicated that SA Method I may be more appropriate for small problems and Method II more appropriate for large problems. For the small problems, a higher percentage of optimal solutions was obtained through Method I. For the large problems, however, similar results were obtained for both methods, yet Method II required less processing time.

The experiments with Method I showed that the simulated annealing parameters are a function of a given problem's characteristics. A general rule is that the ratio of MaxAccepted to MaxAttempted should be 1:2 or slightly less. For a problem of roughly 20 elements or less, values for MaxAttempted of 50n to 150n should be sufficient to provide very good results. For a problem of approximately 50 elements, the values of MaxAttempted should range from 500n to 1500n. For the high WEST ratio/low F-ratio problem, the parameters can be on the low side of the range, for the high WEST ratio/high F-ratio problem, the parameters should be at least in the center of the range, and for both low WEST ratio problems, parameters must be on the high side of the range. In all cases, however, a higher value of the cooling rate (α = 0.9) produced superior results. Also, the higher initial temperature usually provided better results.

The experimental results indicated that as the parameters are increased (proportionally), the solution generally improves. This is logical, since the percentage of

the solution space searched increases. Ultimately, it is the user who must decide whether the additional processing time is justified.

Method II requires fewer parameter decisions than Method I. The highest cooling rate, $\alpha = 0.99$, performed best for this method. An epsilon value of 0.0005 is sufficient; the smaller epsilon did not provide noticably different results. Also, the initial temperature does not seem to be an important factor; however, in the case of the large problems, the smaller initial temperature method exhibited a slight tendency towards better results.

5.2 FUTURE RESEARCH

This research has established the feasibility and effectiveness of using SA for the mixed model line balancing problem, and has laid substantial groundwork for future users of the technique. However, there is a great deal of additional work which could be done. First, it is likely that these two formulations of simulated annealing could be programmed more efficiently for reduced computational requirements. Also, these formulations could be tested and verified more thoroughly by using different parameters and experimenting with additional problems.

Many modifications of these formulations could be explored; for example, it may be interesting to try lowering MaxAccepted dynamically, since as the system approaches equilibrium, the number of acceptable solutions will decrease. Having MaxAccepted change during the annealing may prevent premature freezing. Another avenue for exploration would be to compare, for large problems, the results of several short SA runs

(using small parameters) versus one time-intensive run (using large parameters). Because of the Monte Carlo nature of SA, the multiple runs approach may have a higher probability of identifying the optimal solution.

In addition to experimenting with different simulated annealing strategies, other methods for formulating the mixed model line balancing problem for SA may be considered. For example, there may be an alternate way to handle the constraints, or a more efficient way to generate perturbations. Perhaps it would be more efficient to perform two SA runs per problem--one which does not allow the addition of stations (thus tends to minimizes stations) and one which does allow addition of new stations--and compare the results. This may be a more effective way to explore the solution space.

Another area for research involves the extension of this application to include the realistic problems found in industry, which involve, for example, parallel stations, fixed stations, et cetera. The literature applauds COMSOAL for being one of the only methods which provides this type of flexibility. COMSOAL is a Monte Carlo method, and thus it would seem that simulated annealing should be able to accommodate the same variations that COMSOAL addresses.

One of these extensions would be to allow the use of a more complex, multi-objective function, which, for example, may seek to minimize delta while also minimizing the number of stations. Also, since assembly line designers may have a fixed number of stations that must be used, it would be desirable to have SA address this concern. This may require a more sophisticated initial solution routine which provides the desired

number of stations. Another extension would be to use simulated annealing for addressing the other critical area of mixed model assembly line design, model sequencing.

REFERENCES

Aarts, Emile and Jan Korst, 1989, *Simulated Annealing and Boltzmann Machines*, John Wiley and Sons.

Abramson, D., 1991, "Constructing School Timetables Using Simulated Annealing: Sequential and Parallel Algorithms," *Management Science*, Vol. 37, No. 1, January, pg. 98-113.

Arcus, Albert L., 1966, "COMSOAL: A Computer Method of Sequencing Operations for Assembly Lines," *The International Journal of Production Research*, Vol. 4, No. 4, pp. 259-277.

Baybars, Ilker, 1986, "A Survey of Exact Algorithms for the Simple Line Balancing Problem," *Management Science*, Vol. 32, No. 8, August, pp. 909-932.

Betts, J. and K. I. Mahmoud, 1989, "A Method for Assembly Line Balancing," *Engineering Costs and Production Economics*, Vol. 18, pp. 55-64.

Bowman, E. H., 1960, "Assembly-Line Balancing By Linear Programming," *Operations Research*, Vol. 8, No. 3.

Bryton, Benjamin, 1954, Balancing of a Continuous Production Line. Unpublished M.S. Thesis, Northwestern University.

Bulgak, Akif A., and Jerry L. Sanders, 1991, "Modeling and Design Optimization of Asynchronous Flexible Assembly Systems with Statistical Process Control and Repair," *The International Journal of Flexible Manufacturing Systems*, Vol. 3, p. 251-274.

Buzacott, John A. and J. George Shanthikumar, 1993, *Stochastic Models of Manufacturing Systems*, Prentice Hall.

Chakravarty, A. K. and A. Shtub, 1985, "Balancing mixed model lines with in-process inventories," *Management Science*, Vol. 31, No. 9.

Chow, We-Min, 1990, *Assembly Line Design: Methodology and Applications*, Marcel Dekker, Inc.

Dar-El, E. M., 1973, "MALB -- A Heuristic Technique for Balancing Large Single-Model Assembly Lines," *AIIE Transactions*, December, pg. 343-356.

Dar-El, E. M., and A. Nadivi, 1981, "A Mixed-Model Sequencing Application," *International Journal of Production Research*, Vol. 19, No. 1, pp. 69-84.

Dar-El, E. M., and Y. Rubinovitch, 1979, "MUST--A Multiple Solutions Technique for Balancing Single Model Assembly Lines," *Management Science*, Vol. 25, No. 11, November, pg. 1105-1114.

Das, H., P. T. Cummings, and M. D. LeVan, 1990, "Scheduling of Serial Multiproduct Batch Processes via Simulated Annealing," *Computers in Chemical Engineering*, Vol. 14, No. 12, pp. 1351-1362.

Derin, Haluk, 1986, "Experimentations with Simulated Annealing in Allocation Optimization Problems," *Proceedings of the Annual Conference on Information Sciences and Systems*, Vol. 20, pg. 165-171.

Feller, William, 1968, *Introduction to Probability Theory and Its Applications*, Vol. 1, 3rd Edition, John Wiley and Sons, p. 38.

Geman, S. and D. Geman, 1984, Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images, *IEEE Proceedings, Pattern Analysis and Machine Intelligence*, PAMI-6 , pp. 721-741.

Ghosh, Soumen and Roger J. Gagnon, 1989, "A Comprehensive Lierature Review and Analysis of the Design, Balancing, and Scheduling of Assembly Systems," *International Journal of Production Research*, Vol. 27, No. 4, pp. 637-670.

Glauber, R. J., 1963, "Time-Dependent Statistics of the Ising Model," *Journal of Mathematics and Physics*, Vol. 4, No. 294.

Gutjahr, Allan L. and George L. Nemhauser, 1964, "An Algorithm for the Line Balancing Problem," *Management Science*, Vol. 11, No. 2, November, pp. 308-315.

Harhalakis, G., J. M. Proth, and X. L. Xie, 1990, "Manufacturing Cell Design Using Simulated Annealing: An Industrial Application," *Journal of Intelligent Manufacturing*, Vol. 1, p. 185-191.

Held, Michael, Richard M. Karp, and Richard Shareshian, 1963, "Assembly-Line Balancing--Dynamic Programming with Precedence Constraints," *Operations Research*, Vol. 11, No. 3.

Helgeson, W. B. and D. P. Birnie, 1961, "Assembly Line Balancing Using the Ranked Positional Weight Technique," *The Journal of Industrial Engineering*, November-December, pp. 394-398.

Jackson, James R., 1956, "A Computing Procedure for a Line Balancing Problem," *Management Science*, Vol. 2, No. 3.

Johnson, R. V., 1981, "Assembly Line Balancing Algorithms: Computational Comparisons," *International Journal of Production Research*, Vol. 19, No. 3.

Johnson, Roger V., 1983, "A Branch and Bound Algorithm for Assembly Line Balancing Problems with Formulation Irregularities," *Management Science*, Vol. 29, No. 11, November, pp. 1309-1324.

Kalpakjian, Serope, 1989, *Manufacturing Engineering and Technology*, Addison-Wesley Publishing Company.

Kao, E. P. C., and Queyranne, M., 1982, "On Dynamic Programming Methods for Assembly Line Balancing," *Operations Research*, Vol. 30, No. 2.

Kilbridge, Maurice D. and Leon Wester, 1961, "A Heuristic Method of Assembly Line Balancing," *The Journal of Industrial Engineering*, Vol. 12, No. 4, July-August, pp. 292-298.

Kilbridge, Maurice D. and Leon Wester, 1962, "A Review of Analytical Systems of Line Balancing," *Operations Research*, Vol. 10, No. 5, Sept-Oct, pp. 626-638.

Kirkpatrick, S., C. D. Gelatt, Jr., M. P. Vecchi, 1983, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, May 13, pp. 671-680.

Kottas, J. F. and H. S. Lau, 1973, "A Cost-Oriented Approach to Stochastic Line Balancing," *AIIE Transactions*, Vol. 5, No. 2.

Kouvelis, P., Chaing, W., and J. Fitzsimmons, 1992, "Simulated Annealing for Machine Layout Problems in the Presence of Zoning Constraints," *European Journal of Operational Research*, Vol. 57, pp. 203-223.

Kovach, Jerry, 1969, "CALB: Systematic Assembly Management," *Assembly Engineering*, September, pp. 20-25.

Ku, Hong-ming and Iftekhar Karimi, 1991, "An Evaluation of Simulated Annealing for Batch Process Scheduling," *Industrial Engineering Chemical Research*, Vol. 30, No. 1, pg. 163-169.

Macakill, J. L. C., 1972, "Production-Line Balances for Mixed-Model Lines," *Management Science*, Vol. 19, No. 4, December, pp. 423-434.

Mansoor, E. M., 1964, "Assembly Line Balancing -- An Improvement on the Ranked Positional Weight Technique," *The Journal of Industrial Engineering*, March-April, pp. 73-77.

Manz, Eileen M., Jorge Haddock, and John Mittenthal, 1989, "Optimization of an Automated Manufacturing System Simulation Model using Simulated Annealing," *Proceedings of the 1989 Winter Simulation Conference*, pg. 390-394.

Moodie, C. L. and H. H. Young, 1965, "A Heuristic Method of Assembly Line Balancing for Assumptions of Constant or Variable Work Element Times," *The Journal of Industrial Engineering*, January-February, pp. 23-29.

Ogbu, F. A. and D. K. Smith, 1990, "The Application of the Simulated Annealing Algorithm to the Solution of the n/m/C_{max} Flowshop Problem," *Computers and Operations Research*, Vol. 17, No. 3, pp. 243-253.

Pantouvanos, John P., 1992, "A Computerized Methodology for Balancing and Sequencing Mixed Model Stochastic Assembly Lines," Master's Thesis.

Patterson, J. H., and J. J. Albracht, 1975, "Assembly-Line Balancing: Zero-One Programming with Fibonacci Search," *Operations Research*, No. 23, pp. 166-172.

Roberts, S., and C. Villa, 1970, "A Multiproduct Assembly Line Balancing Problem," *AIIE Transactions*, Vol. 2, No. 4.

Salveson, M. E., 1955, "The Assembly Line Balancing Problem," *Journal of Industrial Engineering*, Vol. 6, No. 3.

Schofield, Norman A., 1979, "Assembly Line Balancing and the Application of Computer Techniques," *Computers and Industrial Engineering*, Vol. 3, pp. 53-69.

Schrage, L. and K. R. Baker, 1978, "Dynamic Programming Solution of Sequencing Problems with Precedence Constraints," *Operations Research*, Vol. 26, May-June.

Smith, P. R., 1990, "A Computerized Search Methodology for the Design of Mixed Model Assembly Systems," *Master's Thesis*, Virginia Polytechnic Institute and State University.

Talbot, F. B. and J. H. Patterson, 1984, "An Integer Programming Algorithm with Network Cuts for Solving the Assembly Line Balancing Problem," *Management Science*, Vol. 30, No. 1.

Thangavelu, S. R. and C. M. Shetty, 1971, "Assembly Line Balancing by Zero-One Programming," *AIIE Transactions*, Vol. 3, No. 1.

Thomopoulos, Nick T., 1967, "Line Balancing-Sequencing for Mixed-Model Assembly," *Management Science*, Vol. 14, No. 2, October, pg. B-59--B-75.

Thomopoulos, Nick T., 1970, "Mixed Model Line Balancing with Smoothed Station Assignments," *Management Science*, Vol. 16, No. 9, May, pp. 593-603.

Tonge, F. M., 1961, *A Heuristic Program for Assembly Line Balancing*, Englewood Cliffs, New Jersey: Prentice Hall.

Tonge, F. M., 1960, Summary of a Heuristic Line Balancing Procedure," *Management Science*, Vol. 7, No. 1.

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

Vrat, P. and A. Virani, 1976, "A Cost Model for Optimal Mix of Balanced Stochastic Assembly-Line and the Modular Assembly System for a Customer Oriented Production System," *International Journal of Production Research*, Vol. 14, No. 4.

Wester, L. and M. D. Kilbridge, 1962, "Heuristic Line Balancing: A Case," *Journal of Industrial Engineering*, Vol. 13, No. 3.

White, W. W., 1961, "Comments on a Paper by Bowman," *Operations Research*, Vol. 9, March-April.

Wilhelm, Mickey R., and Thomas L. Ward, 1987, "Solving Quadratic Assignment Problems by Simulated Annealing," *IIE Transactions*, March, pg. 107-119.

APPENDIX A: COMPUTER CODES

The computer codes for both SA Method I and SA Method II are given below. Both were written using Borland Turbo C++ Version 3.0.

Simulated Annealing Method I

```
#include <conio.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <memory.h>
#include <math.h>
#include <time.h>
#include <string.h>
#define MODSIZE 20
#define ELEMSIZE 64
#define NUMINTS 2
#define NUMRUNS 1
#define NUMREPS 1

void Swap ();
void Transfer ();
void New ();
void Calculate_ObjFunc ();
void InitialSolution ();
void SA();
void DisplayResults ();
void GetData ();

FILE *outfptr;
FILE *delfptr;

float  TTc,                          //Ideal cycle time
       TTcUL,                        //Upper limit on cycle time
       ModelTe[MODSIZE],             //Total time to build one unit of a model
       Ts[ELEMSIZE][MODSIZE],        //Station Time [station][model]
                                     //resulting from given balance.
       TTsjk[ELEMSIZE][MODSIZE],     //Ts[j][k] (above) times Q[k]
       TTsk[MODSIZE],                //ModelTe[k] times Q[k] divided by NumStations
       Te [ELEMSIZE][MODSIZE],       //Element time [element i][model k]
       TTe[ELEMSIZE],                //Total element time [element i] to produce
                                     //entire model mix.
       cycletime,                    //Used to calculate Tc at each station
```

```
        temperature,                    //Current Temperature
        BestObj,
        NewObj,
        OldObj,
        COOLING_RATE,
        To;                             //Initial Temperature

int     runcount, repcount,
        MaxStns,                        //Maximum Number of Stations allowed
        Station1,                       //These 4 variables used in swapping,
        Station2,                       //transferring, and making new stns.
        Element1,
        Element2,
        NumElems,                       //Number of elements in problem
        NumMods,                        //Number of models in problem
        Q[MODSIZE],
        NumStations=0,                  //Number of stations required for solution
        tempNumStations=0,              //Number of stations for temporary soln.
        ElementCount [ELEMSIZE],        //Keeps track of how many elements assigned
                                        //to each station
        tempElemCount [ELEMSIZE],       //Same, but for temporary solution
        InfeasFlag = 0,
        tempsol [ELEMSIZE][ELEMSIZE],   //Temporary solution (once acceptance criterion
        solution [ELEMSIZE][ELEMSIZE],  //is checked, may become solution)
        best_solution [ELEMSIZE][ELEMSIZE], //Stores best soln so far
        best_NumStns = 0,
        best_ElemCount[ELEMSIZE];

long unsigned int MAX_ATTEMPTED, MAX_ACCEPTED;

long int prec[ELEMSIZE][NUMINTS];       //This array holds precendence constraints
unsigned long  TotalAttempted;          //If NumElems > 32, NUMINTS should be set
                                        //to NumElems%32.
clock_t start, end;

void Calculations ()                    //This function is called after data
{                                       //is read in;  it does preliminary
        int     k, elem, SumQ;          //calculations that are needed for
                                        //determining the objective function value
        SumQ = 0;
        for (k=0; k<NumElems; k++) TTe[k] = 0;
        for (k=0; k<NumMods; k++)
                {
                ModelTe[k] = 0;
                SumQ+=Q[k];
                }
        for (elem=0; elem<NumElems; elem++)
                {
                for (k=0; k<NumMods; k++)
                        {
                        TTe[elem]+= Te[elem][k]*Q[k];
```

```
                        ModelTe[k] += Te[elem][k];
                        }
                }
}


void GetData()                                  //This function reads in data for a
{                                               //problem.  Note that it expects a
        int     OkayFlag = 0;                   //file in binary format.
        FILE    *dfptr;
        char    ch,
                filename[81];

        do
        {
        clrscr();
        gotoxy (5,8);
        printf("Enter the name of the data file to retrieve:  ");
        gets(filename);
        if (filename[0] != '\r')
        {
                if ((dfptr = fopen(filename, "rb")) == NULL)
                {
                printf("The file %s does not exist", filename);
                printf("Please hit any key . . .");
                }
                else OkayFlag = 1;
        }
        }  while (!OkayFlag);
        printf ("READING FILE: %s\n", filename);
        clrscr();
        fscanf(dfptr,"%f %f %d %d",&TTc, &TTcUL, &NumElems, &NumMods);
        fread(Q, sizeof(Q), 1, dfptr);
        fread(TTe, sizeof(TTe),1, dfptr);
        fread (Te, sizeof (Te), 1, dfptr);
        fread(prec, sizeof(prec), 1, dfptr);
        fclose(dfptr);
        Calculations ();
}


void InitialSolution ()                         //This function
{                                               //determines an initial
        long    pc[ELEMSIZE][NUMINTS];          //solution
        int     pos = 0,
                PrecFlag=1,
                j = 0,
                k = 0,
                m,
                count = 0,
```

```
        elem = 0;
    float   sum = 0.0;

    pos = 0; count = 0;
    tempNumStations = 0;
    for (j= 0; j<ELEMSIZE; j++)
        for (k=0; k<MODSIZE; k++)
          {
          tempsol [j][k] = 0;
          tempElemCount [j] = 0;
          }

    TTc = 0.0;
    memmove(pc, prec, sizeof (prec));
    do
    {
    for (elem = 0; elem<NumElems; elem ++)
        {
        PrecFlag = 1;
        for (m=0; m<NUMINTS; m++) {if (pc[elem][m]) PrecFlag = 0;}
        if (PrecFlag)
            {
            sum = TTc + TTe[elem];
            if (sum > TTcUL)
                {
                pos=0;
                tempNumStations++;
                TTc = 0;
                }
            tempsol[tempNumStations][pos] = elem;
            tempElemCount [tempNumStations] = pos;
            TTc += TTe[elem];
            for (j=0; j<NumElems;j++)
               pc[j][elem/32] &= !((unsigned long)1 << (elem%32) -1);
            for (m=0; m<NUMINTS; m++) pc[elem][m] = ~pc[elem][m];
            pos++;
            count++;
            }
        }
    } while (count < NumElems);
}


void InitialTemp ()             //This function takes initial
{                               //samples and determines To.
    float   diff = 0.0,
            MaxDiff = 0.0,
            Old,
            n,
            p;
```

```
        Old = NewObj;  //store obj. func. value for the initial soln.
        memmove (solution, tempsol, sizeof (tempsol));  //need a 'solution' to start with
        memmove (ElementCount, tempElemCount, sizeof (tempElemCount));
        NumStations = tempNumStations;
        for (n=0; n<100; n++)                          //generate 300 attempts
        {
                do
                {
                InfeasFlag = 0;
                memmove (tempsol, solution, sizeof (solution));
                memmove (tempElemCount, ElementCount, sizeof (ElementCount));
                tempNumStations = NumStations;
                p = (float)random(101)/100.0;
                if (p < 0.48) Swap ();
                else if (p >= 0.48 && p < 0.96) Transfer ();
                else New ();
                } while (InfeasFlag);
        Calculate_ObjFunc();
        diff = fabs (NewObj-Old);
        MaxDiff = (diff > MaxDiff) ? diff : MaxDiff;
        }
        temperature = To * MaxDiff;
}


void Calculate_ObjFunc ()
{

        int     i,
                j,
                k,
                pos;

        for (j=0; j<=tempNumStations; j++)
                for (k=0; k<NumMods; k++)
                        Ts[j][k] = 0;

        for (j=0; j<=tempNumStations; j++)
                for (k=0; k<NumMods; k++)
                        for (pos=0; pos<= tempElemCount [j]; pos++)
                        Ts[j][k]+=Te[tempsol[j][pos]][k];

        NewObj=0;
        for (j=0; j<=tempNumStations; j++)
        {
                for (k=0; k<NumMods; k++)
                {
                        TTsk[k] = Q[k] * ModelTe[k]/(tempNumStations+1);
                        TTsjk[j][k] = Ts[j][k]*Q[k];
                        NewObj+= fabs(TTsk[k] - TTsjk[j][k]);
```

```
                }
        }
}

void Swap ()
{
        int     j,
                bit,
                pos,
                Station1,
                Station2,
                Element1,
                Element2,
                temp;
        Station1 = random (tempNumStations +1);
        do
        Station2 = random (tempNumStations +1);  //necessary to choose two
        while (Station1 == Station2);          //different stations
        if (Station1 > Station2)
                {
                temp = Station2;             //make Station1 the upper
                Station2 = Station1;            //station for convenience
                Station1 = temp;
                }
        Element1 = random (tempElemCount[Station1] + 1);  //really a
        Element2 = random (tempElemCount[Station2] + 1);  //position of element;
                                                          //not an element
        //swap the elements//
        temp = tempsol[Station2][Element2];
        tempsol[Station2][Element2] = tempsol[Station1][Element1];
        tempsol[Station1][Element1] = temp;

        //check precedence constraints//

        for (j=Station1; j<Station2; j++)                  //one direction
                for (pos=0; pos<=tempElemCount[j]; pos++)
                    {
                    bit = tempsol[Station2][Element2];
                    if (prec[tempsol[j][pos]][bit/32] &
                      (long)1<<bit%32) InfeasFlag = 1;
                    }

        for (j=Station1+1; j<=Station2; j++)               //other direction
                for (pos=0; pos<=tempElemCount[j]; pos++)
                    {
                    bit = tempsol[j][pos];
                    if (prec[tempsol[Station1][Element1]][bit/32] &
                      (long)1<<bit%32) InfeasFlag = 1;
                    }

        //check cycle time constraint//
```

```
        cycletime = 0.0;
        for (pos=0; pos<=tempElemCount[Station1]; pos++)
                cycletime += TTe[tempsol[Station1][pos]];
        if (cycletime > TTcUL) InfeasFlag = 1;

        cycletime = 0.0;
        for (pos=0; pos<=tempElemCount[Station2]; pos++)
              cycletime += TTe[tempsol[Station2][pos]];
        if (cycletime > TTcUL) InfeasFlag = 1;
}


void Transfer ()                    //moves one element (allows
{       int     j,                    //for changes in the number of
                pos,                  //elements per station)
                bit,
                Station1,
                Station2,
                Element1,
                ReduceFlag=0;           //tells prec. constraint check
                                        //that a station was removed; because
                                        //stn. was removed, Station1 and Station2
                                        //are not necessarily the same as before
                                        //since stations were shifted as necessary.


        ReduceFlag = 0;
        Station1 = random (NumStations +1);   //Station1 is where element will
                                              //be taken from; Station2 is
                                              //the new station.
        do
        Station2 = random (NumStations +1); //necessary to choose two
        while (Station1 == Station2);         //different stations

        Element1 = random (tempElemCount[Station1] + 1); //find elem to move
                                                         //again, a position!
        tempsol[Station2][tempElemCount[Station2] + 1] =  //move element
                        tempsol[Station1][Element1]; //to the last position
                                                     //in selected station

        tempElemCount[Station2] ++;         //modify counter for number of
        tempElemCount[Station1] --;         //elements in each station

        if (tempElemCount [Station1] >= 0)    //if there is at least one element
                                              //remaining in the station, tidy up
                {
                memmove (*(tempsol+Station1)+Element1, *(tempsol+Station1)+(Element1+1),
                        (tempElemCount[Station1] - Element1 +2)*2); //close gap in stn
                }
```

```
else
        {                               //have emptied a stn; remove it:
                                        //shift solution matrix:
        memmove (*(tempsol+Station1), *(tempsol+Station1+1),
                2*ELEMSIZE*(tempNumStations-Station1+1));
                                        //2 bytes per integer, ELEMSIZE
                                        //integers per 1-D array
                                        //times the number of remaining
                                        //stations.
        memmove (tempElemCount+Station1,
                tempElemCount+Station1+1,
                2*(tempNumStations - Station1 + 1));
                                        //accordingly, must also
                                        //shift ElementCount array
        ReduceFlag = 1;
        tempNumStations--;              //reduce NumStations
        }

//check precedence constraints//
if (Station1 < Station2)                //that is, an element is
{                                       //moved further down...
for (j=Station1; j<Station2-ReduceFlag; j++)
        for (pos=0; pos<=tempElemCount[j]; pos++)
           {
           bit = tempsol[Station2-ReduceFlag][tempElemCount[Station2-ReduceFlag]];
           if (prec[tempsol[j][pos]][bit/32] &
             (long)1<<bit%32) InfeasFlag = 1;
           }
}
else                                    //else, if element is moved up,..
{
for (j=Station2+1; j<=Station1-ReduceFlag; j++)
        for (pos=0; pos<=tempElemCount[j]; pos++)
           {
           bit = tempsol[j][pos];
           if (prec[tempsol[Station2][tempElemCount[Station2]]][bit/32] &
             (long)1<<bit%32) InfeasFlag =1;
           }
}


//check cycle time constraint//

cycletime = 0.0;

if (Station1<Station2)   //element moved down
   {
   for (pos=0; pos<=tempElemCount[Station2-ReduceFlag]; pos++)
        cycletime += TTe[tempsol[Station2-ReduceFlag][pos]];
   if (cycletime > TTcUL) InfeasFlag =1;
   }
```

```
        else                    //element moved up
            {
            for (pos=0; pos<=tempElemCount[Station2]; pos ++)
                cycletime += TTe[tempsol[Station2][pos]];
            if (cycletime > TTcUL) InfeasFlag = 1;
            }
}



void New ()
{
int     j, pos, i, bit,
        Station1,
        Element1,
        Station2,
        store,
        flag = 0,
        ReduceFlag = 0;

ReduceFlag = 0;  flag = 0;
Station1 = random (tempNumStations +1);
Element1 = random (tempElemCount[Station1] + 1);

do Station2 = random (tempNumStations+1);
while (Station2 == Station1);

store = tempsol [Station1][Element1];
if (Station2 == tempNumStations)  //if putting new station at end
    {
    tempNumStations ++;
    tempsol [tempNumStations][0] = store;
    tempElemCount [tempNumStations] = 0;
    }
else                                    //if new station somewhere in middle
    {
    memmove (*(tempsol + Station2+2), *(tempsol+Station2+1),
            2*ELEMSIZE*(tempNumStations-(Station2+1)+1));
    for (i=0; i<ELEMSIZE; i++) tempsol [Station2+1][i] = 0;
    tempsol[Station2+1][0] = store;

    memmove (tempElemCount+Station2+2,
                        tempElemCount+Station2+1,
                        2*(tempNumStations - (Station2+1)+1));
    tempElemCount[Station2+1] = 0;
    tempNumStations++;
    }
    if (Station2 < Station1)    //stn was added above stn removed
                        flag = 1;
    tempElemCount [Station1+flag] --;
    if (tempElemCount [Station1+flag] >= 0)     //if there is at least 1 element remaining
```

```
                                        //in the station, tidy up.

            memmove (*(tempsol+Station1+flag)+Element1,
                    *(tempsol+Station1+flag)+(Element1+1),
                    2*(tempElemCount[Station1+flag] - Element1 +2));   //close gap
        else
            {                               //have emptied a stn; remove it:
                                            //shift solution matrix
            memmove (*(tempsol+Station1+flag), *(tempsol+Station1+1+flag),
                    2*ELEMSIZE*(tempNumStations-Station1+1));
                                            //2 bytes per integer, ELEMSIZE
                                            //integers per 1-D array
                                            //times the number of remaining
                                            //stations.
            memmove (tempElemCount+Station1+flag,
                     tempElemCount+Station1+1+flag,
                     2*(tempNumStations - Station1 + 1));
                                            //accordingly, must also
                                            //shift ElementCount array
            ReduceFlag = 1;
            tempNumStations--;              //reduce NumStations
            }

        //check precedence constraints
        if (Station1 < Station2)        //element moved down
        {
        for (j=Station1; j<=Station2-ReduceFlag; j++)
            for (pos = 0; pos <=tempElemCount[j]; pos++)
            {
            bit = tempsol[Station2+1-ReduceFlag][tempElemCount[Station2+1-ReduceFlag]];
            if (prec[tempsol[j][pos]][bit/32] &
              (long)1<<bit%32)  InfeasFlag = 1;
            }
        }
        else                        //element moved up
        {
        for (j=Station2+2; j<=Station1+1-ReduceFlag; j++)
            for (pos=0; pos <=tempElemCount[j]; pos++)
            {
            bit = tempsol[j][pos];
            if (prec[tempsol[Station2+1][tempElemCount[Station2+1]]][bit/32]
               & (long)1<<bit%32)  InfeasFlag = 1;
            }
        }
}
```

```
void Perturb ()
{
        float     p;

        InfeasFlag = 0;
        memmove (tempsol, solution, sizeof (solution));
                                        //tempsol allows the elements to be
                                        //moved around without permanently
                                        //losing the last good balance.
                                        //if tempsol is good, it will be
                                        //saved in SA.

        memmove (tempElemCount, ElementCount, sizeof (ElementCount));
        tempNumStations = NumStations;
        p = (float)random (101)/100.0;
        if ((p > 0.96) && (tempNumStations < MaxStns)) New ();
        else if (p < 0.48) Swap();
        else Transfer ();
}



void SA()
{
        long unsigned int       NumAttempted = 0,
                                NumAccepted = 0;
        int       j, k, z, x,
                  ColdCount = 0;
        float   rnum,
                  prob;

        TotalAttempted=0;
        BestObj = 99999;
        InitialSolution();          //Find Initial Temperature
        Calculate_ObjFunc();
        InitialTemp();


        tempNumStations = 0;                    //reinitialize tempNumStations
        for (j=0; j<ELEMSIZE; j++)              //reinitialize tempsol
                for (k=0; k<ELEMSIZE; k++)
                  tempsol[j][k] = 0;
        for (j=0; j<ELEMSIZE; j++)  tempElemCount [j] = 0; //reinitialize
                                                //tempElementCount
        InitialSolution();
        memmove (solution, tempsol, sizeof (tempsol));
        memmove (ElementCount, tempElemCount, sizeof (tempElemCount));
        NumStations = tempNumStations;
        MaxStns = NumStations;
        Calculate_ObjFunc();
        OldObj = NewObj;
```

```
do                              //while not frozen//
{
        do                      //for each temperature//
        {
        do                      //keep trying until you find
        Perturb ();             //a FEASIBLE solution
        while (InfeasFlag);
        NumAttempted++;                 //count FEASIBLE attempts
        Calculate_ObjFunc();
        if (NewObj < OldObj)            //better solution//
                {
                NumAccepted++;
                OldObj = NewObj;
                if (OldObj < BestObj)
                        {BestObj = OldObj;
                        memmove (best_solution, tempsol, sizeof (tempsol));
                        memmove (best_ElemCount, tempElemCount, sizeof
                                        (tempElemCount));
                        best_NumStns = tempNumStations;}
                memmove (solution, tempsol, sizeof (tempsol));
                memmove (ElementCount, tempElemCount, sizeof (tempElemCount));
                NumStations = tempNumStations;

                }
        else                    //worse solution//
                {
                prob = exp(-(NewObj - OldObj)/temperature);
                rnum = (float)random(101)/100.0;
                if (prob < 0.001) prob = 0.0;
                if ((1-prob) < rnum) //find prob
                        {                       //accept anyway
                        NumAccepted++;
                        OldObj = NewObj;
                        if (OldObj < BestObj)
                                {BestObj = OldObj;
                                 memmove (best_solution, tempsol, sizeof (tempsol));
                                 memmove (best_ElemCount, tempElemCount, sizeof
                                        (tempElemCount));
                                 best_NumStns = tempNumStations;}
                        memmove (solution, tempsol, sizeof (tempsol));
                        memmove (ElementCount, tempElemCount, sizeof
                                                (tempElemCount));
                        NumStations = tempNumStations;
                        }
                }
        } while ((NumAccepted < MAX_ACCEPTED) && (NumAttempted <
MAX_ATTEMPTED));
        if ((ColdCount != 0) && (NumAccepted >= MAX_ACCEPTED)) ColdCount = 0;
        if (NumAccepted < MAX_ACCEPTED) ColdCount++; //if stopped at that
                                                //temp due to not enough
```

```
                                                    //good solutions, then may be cold
        temperature = temperature * COOLING_RATE;
        TotalAttempted+=NumAttempted;
        NumAttempted = 0;
        NumAccepted = 0;
        } while ((ColdCount < 3) && (temperature > 1));
}

void DisplayResults ()
{
        int     i, j, k, mins;

        fprintf (outfptr,"\nRun # %d: ", runcount);
        fprintf (outfptr, "%lu  %lu  %.2f  %.2f", MAX_ATTEMPTED, MAX_ACCEPTED, To,
        COOLING_RATE);
        if (repcount == 0) fprintf (delfptr, "%lu  %lu  %.2f  %.2f\n", MAX_ATTEMPTED,
MAX_ACCEPTED,       To, COOLING_RATE);
        for (i=0; i<=best_NumStns; i++)
                {
                fprintf (outfptr,"\nStation #%d :", i+1);
                for (j=0; j<=best_ElemCount[i]; j++)
                fprintf (outfptr,"\t%d ", best_solution[i][j]+1);
                }

        fprintf (outfptr,"\nDelta:  %7.3f", BestObj);
        fprintf (delfptr, "  %7.3f\n", BestObj);
        mins = (end-start)/CLK_TCK/60;
        fprintf (outfptr,"      Time:  %d min  %.2f secs.\n",
              mins, fmod((end-start)/CLK_TCK, 60.));
        fprintf (outfptr,"Feasible solns. attempted: %lu.\n", TotalAttempted);
        fprintf (outfptr, "final temp = %f \n", temperature);
}



void main (void)
{
        int     i, j;
        struct rundata
          {
          long unsigned int m_attempt, m_accept;
          float t, r;
          } run[NUMRUNS];
        FILE *fptr;
        randomize ();
        GetData();
        fptr = fopen("input.txt", "rt");
        for (i=0; i<NUMRUNS; i++) fscanf (fptr, "%lu %lu %f %f",
           &run[i].m_attempt, &run[i].m_accept, &run[i].t, &run[i].r);
        fclose (fptr);
        outfptr= fopen ("lgrun1z1.out", "wt");
```

```
        delfptr= fopen ("lgrun1z1.del", "wt");
        for (runcount=0; runcount<NUMRUNS; runcount++)
            {
                printf ("RunCount = %d\n", runcount);
                for (repcount=0; repcount<NUMREPS; repcount++)
                {
                printf ("repcount = %d\n", repcount);
                MAX_ATTEMPTED=run[runcount].m_attempt;
                MAX_ACCEPTED=run[runcount].m_accept;
                To = run[runcount].t;
                COOLING_RATE=run[runcount].r;
                start=clock();
                SA();
                end=clock();
                DisplayResults();
                }
            }
        fclose (outfptr);
        fclose (delfptr);
}
```

Simulated Annealing Method II

```
#include <conio.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <memory.h>
#include <math.h>
#include <time.h>
#include <string.h>
#define MODSIZE 20
#define ELEMSIZE 64
#define NUMINTS 2
#define NUMRUNS 2
#define NUMREPS 2

void Swap ();
void Transfer ();
void New ();
void Calculate_ObjFunc ();
void InitialSolution ();
void SA();
void DisplayResults ();
void GetData ();

FILE *outfptr;
FILE *delfptr, *tfptr, *sfptr;

float   TTc,
        TTcUL,
        ModelTe[MODSIZE],               //Total time to build one unit of a model
        Ts[ELEMSIZE][MODSIZE],          //Station Time [station][model]
                                        //resulting from given balance.
        TTsjk[ELEMSIZE][MODSIZE],       //Ts[j][k] (above) times Q[k]
        TTsk[MODSIZE],                  //ModelTe[k] times Q[k] divided by NumStations
        Te [ELEMSIZE][MODSIZE],         //Element time [element i][model k]//
        TTe[ELEMSIZE],                  //Total element time [element i] to produce
                                        //entire model mix.
        cycletime,
        temperature,
        BestObj,
        NewObj,
        OldObj,
        COOLING_RATE,
        To,
        epsilon,
        newOFbar,
        OFbar[3],
        OFsum,
        SOFold,
        SOFnew,
```

```
        oldtemp;

int     runcount, repcount, MaxStns,
        Trials_per_Temp,
        callcount,
        Station1,
        Station2,
        Element1,
        Element2,
        NumElems,
        NumMods,
        Q[MODSIZE],
        NumStations=0,      //Number of stations required for solution
        tempNumStations=0,
        ElementCount [ELEMSIZE],    //Keeps track of how many elements assigned to
                                    //each station
        tempElemCount [ELEMSIZE],
        InfeasFlag = 0,
        tempsol [ELEMSIZE][ELEMSIZE],   //temporary solution (once acceptance criterion
        solution [ELEMSIZE][ELEMSIZE],  //is checked, may become solution)
        best_solution [ELEMSIZE][ELEMSIZE],
        best_NumStns = 0,
        best_ElemCount[ELEMSIZE];

long int prec[ELEMSIZE][NUMINTS];
unsigned long  TotalAttempted;
clock_t start, end;



void Calculations ()
{
        int     k, elem, SumQ;

        SumQ = 0;
        for (k=0; k<NumElems; k++) TTe[k] = 0;
        for (k=0; k<NumMods; k++)
                {
                ModelTe[k] = 0;
                SumQ+=Q[k];
                }
        for (elem=0; elem<NumElems; elem++)
                {
                for (k=0; k<NumMods; k++)
                        {
                        TTe[elem]+= Te[elem][k]*Q[k];
                        ModelTe[k] += Te[elem][k];
                        }
                }
}
```

```
void GetData()
{
        int     OkayFlag = 0;
        FILE    *dfptr;
        char    ch,
                filename[81];
        do
        {
        clrscr();
        gotoxy (5,8);
        printf("Enter the name of the data file to retrieve:  ");
        gets(filename);
        if (filename[0] != '\r')
        {
                if ((dfptr = fopen(filename, "rb")) == NULL)
                {
                printf("The file %s does not exist", filename);
                printf("Please hit any key . . .");
                }
                else OkayFlag = 1;
        }
        }  while (!OkayFlag);
        printf ("READING FILE: %s\n", filename);
        clrscr();
        fscanf(dfptr,"%f %f %d %d",&TTc, &TTcUL, &NumElems, &NumMods);
        fread(Q, sizeof(Q), 1, dfptr);
        fread(TTe, sizeof(TTe),1, dfptr);
        fread (Te, sizeof (Te), 1, dfptr);
        fread(prec, sizeof(prec), 1, dfptr);
        fclose(dfptr);
        Calculations ();
}


void InitialSolution ()
{
        long    pc[ELEMSIZE][NUMINTS];
        int     pos = 0,
                PrecFlag=1,
                j = 0,
                k = 0,
                m,
                count = 0,
                elem = 0;
        float   sum = 0.0;

        pos = 0; count = 0;
        tempNumStations = 0;
        for (j= 0; j<ELEMSIZE; j++)
                for (k=0; k<MODSIZE; k++)
                  {
```

```
                tempsol [j][k] = 0;
                tempElemCount [j] = 0;
                }

        TTc = 0.0;
        memmove(pc, prec, sizeof (prec));
        do
        {
        for (elem = 0; elem<NumElems; elem ++)
                {
                PrecFlag = 1;
                for (m=0; m<NUMINTS; m++) {if (pc[elem][m]) PrecFlag = 0;}
                if (PrecFlag)
                        {
                        sum = TTc + TTe[elem];
                        if (sum > TTcUL)
                                {
                                pos=0;
                                tempNumStations++;
                                TTc = 0;
                                }
                        tempsol[tempNumStations][pos] = elem;
                        tempElemCount [tempNumStations] = pos;
                        TTc += TTe[elem];
                        for (j=0; j<NumElems;j++) pc[j][elem/32] &= !((unsigned long)1 <<
                                (elem%32) -1);
                        for (m=0; m<NUMINTS; m++) pc[elem][m] = ~pc[elem][m];
                        pos++;
                        count++;
                        }
                }
        } while (count < NumElems);
}



void InitialTemp ()
{
        float   diff = 0.0,
                MaxDiff = 0.0,
                Old, n, p;
        Old = NewObj;  //store obj. func. value for the initial soln.
        memmove (solution, tempsol, sizeof (tempsol));  //need a 'solution' to start with
        memmove (ElementCount, tempElemCount, sizeof (tempElemCount));
        NumStations = tempNumStations;
        for (n=0; n<100; n++)                   //generate 300 attempts
        {
                do
                {
                InfeasFlag = 0;
```

```
                memmove (tempsol, solution, sizeof (solution));
                memmove (tempElemCount, ElementCount, sizeof (ElementCount));
                tempNumStations = NumStations;
                p = (float)random(101)/100.0;
                if (p < 0.48) Swap ();
                else if (p >= 0.48&& p < 0.96) Transfer ();
                else New ();
                } while (InfeasFlag);
        Calculate_ObjFunc();
        diff = fabs (NewObj-Old);
        MaxDiff = (diff > MaxDiff) ? diff : MaxDiff;
        }
        temperature = To * MaxDiff;
}


void Calculate_ObjFunc ()
{
        int     i, j, k, pos;

        for (j=0; j<=tempNumStations; j++)
                for (k=0; k<NumMods; k++)
                        Ts[j][k] = 0;

        for (j=0; j<=tempNumStations; j++)
                for (k=0; k<NumMods; k++)
                        for (pos=0; pos<= tempElemCount [j]; pos++)
                        Ts[j][k]+=Te[tempsol[j][pos]][k];

        NewObj=0;
        for (j=0; j<=tempNumStations; j++)
        {
                for (k=0; k<NumMods; k++)
                {
                        TTsk[k] = Q[k] * ModelTe[k]/(tempNumStations+1);
                        TTsjk[j][k] = Ts[j][k]*Q[k];
                        NewObj+= fabs(TTsk[k] - TTsjk[j][k]);
                }
        }
}


void Swap ()
{
        int     j, bit, pos, temp,
                Station1, Station2,
                Element1, Element2;
```

```
Station1 = random (tempNumStations +1);
do
Station2 = random (tempNumStations +1);  //necessary to choose two
while (Station1 == Station2);          //different stations
if (Station1 > Station2)
        {
        temp = Station2;             //make Station1 the upper
        Station2 = Station1;            //station for convenience
        Station1 = temp;
        }
Element1 = random (tempElemCount[Station1] + 1);  //really a
Element2 = random (tempElemCount[Station2] + 1);  //position of element;
                                                  //not an element
//swap the elements//
temp = tempsol[Station2][Element2];
tempsol[Station2][Element2] = tempsol[Station1][Element1];
tempsol[Station1][Element1] = temp;

//check precedence constraints//

for (j=Station1; j<Station2; j++)                 //one direction
        for (pos=0; pos<=tempElemCount[j]; pos++)
            {
            bit = tempsol[Station2][Element2];
            if (prec[tempsol[j][pos]][bit/32] &
              (long)1<<bit%32) InfeasFlag = 1;
            }

for (j=Station1+1; j<=Station2; j++)              //other direction
        for (pos=0; pos<=tempElemCount[j]; pos++)
            {
            bit = tempsol[j][pos];
            if (prec[tempsol[Station1][Element1]][bit/32] &
              (long)1<<bit%32) InfeasFlag = 1;
            }


//check cycle time constraint//

cycletime = 0.0;
for (pos=0; pos<=tempElemCount[Station1]; pos++)
        cycletime += TTe[tempsol[Station1][pos]];
if (cycletime > TTcUL) InfeasFlag = 1;

cycletime = 0.0;
for (pos=0; pos<=tempElemCount[Station2]; pos++)
      cycletime += TTe[tempsol[Station2][pos]];
if (cycletime > TTcUL) InfeasFlag = 1;
}
```

```
void Transfer ()                    //moves one element (allows
{       int         j,              //for changes in the number of
                    pos,            //elements per station)
                    bit,
                    Station1,
                    Station2,
                    Element1,
                    ReduceFlag=0;           //tells prec. constraint check
                                            //that a station was removed; because
                                            //stn. was removed, Station1 and Station2
                                            //are not necessarily the same as before
                                            //since stations were shifted as necessary.
        ReduceFlag = 0;
        Station1 = random (NumStations +1);  //Station1 is where element will
                                             //be taken from; Station2 is
                                             //the new station.
        do
        Station2 = random (NumStations +1); //necessary to choose two
        while (Station1 == Station2);       //different stations

        Element1 = random (tempElemCount[Station1] + 1); //find elem to move
                                                         //again, a position!
        tempsol[Station2][tempElemCount[Station2] + 1] = //move element
                        tempsol[Station1][Element1]; //to the last position
                                                     //in selected station

        tempElemCount[Station2] ++;       //modify counter for number of
        tempElemCount[Station1] --;       //elements in each station

        if (tempElemCount [Station1] >= 0)   //if there is at least one element
                                             //remaining in the station, tidy up
                {
                memmove (*(tempsol+Station1)+Element1, *(tempsol+Station1)+(Element1+1),
                        (tempElemCount[Station1] - Element1 +2)*2); //close gap in stn
                }
        else
                {                            //have emptied a stn; remove it:
                                             //shift solution matrix:
                memmove (*(tempsol+Station1), *(tempsol+Station1+1),
                        2*ELEMSIZE*(tempNumStations-Station1+1));
                                                //2 bytes per integer, ELEMSIZE
                                                //integers per 1-D array
                                                //times the number of remaining
                                                //stations.
                memmove (tempElemCount+Station1,
                         tempElemCount+Station1+1,
                         2*(tempNumStations - Station1 + 1));
                                                //accordingly, must also
                                                //shift ElementCount array
                ReduceFlag = 1;
                tempNumStations--;              //reduce NumStations
```

```
            }

        //check precedence constraints//
        if (Station1 < Station2)              //that is, an element is
        {                                     //moved further down...
        for (j=Station1; j<Station2-ReduceFlag; j++)
                for (pos=0; pos<=tempElemCount[j]; pos++)
                    {
                    bit = tempsol[Station2-ReduceFlag][tempElemCount[Station2-ReduceFlag]];
                    if (prec[tempsol[j][pos]][bit/32] &
                       (long)1<<bit%32) InfeasFlag = 1;
                    }
        }
        else                                  //else, if element is moved up,..
        {
        for (j=Station2+1; j<=Station1-ReduceFlag; j++)
                for (pos=0; pos<=tempElemCount[j]; pos++)
                    {
                    bit = tempsol[j][pos];
                    if (prec[tempsol[Station2][tempElemCount[Station2]]][bit/32] &
                       (long)1<<bit%32) InfeasFlag =1;
                    }
        }

        //check cycle time constraint//
        cycletime = 0.0;
        if (Station1<Station2)   //element moved down
           {
           for (pos=0; pos<=tempElemCount[Station2-ReduceFlag]; pos++)
                cycletime += TTe[tempsol[Station2-ReduceFlag][pos]];
           if (cycletime > TTcUL) InfeasFlag =1;
           }
        else                     //element moved up
           {
           for (pos=0; pos<=tempElemCount[Station2]; pos ++)
                cycletime += TTe[tempsol[Station2][pos]];
           if (cycletime > TTcUL) InfeasFlag = 1;
           }
}


void New ()
{
int     j, pos, i, bit,  store,
        Station1, Station2,
        Element1,
        flag = 0,  ReduceFlag = 0;
Station1 = random (tempNumStations +1);
Element1 = random (tempElemCount[Station1] + 1);
```

```
do Station2 = random (tempNumStations+1);
while (Station2 == Station1);

store = tempsol [Station1][Element1];
if (Station2 == tempNumStations)  //if putting new station at end
     {
     tempNumStations ++;
     tempsol [tempNumStations][0] = store;
     tempElemCount [tempNumStations] = 0;
     }
else                                   //if new station somewhere in middle
     {
     memmove (*(tempsol + Station2+2), *(tempsol+Station2+1),
               2*ELEMSIZE*(tempNumStations-(Station2+1)+1));
     for (i=0; i<ELEMSIZE; i++) tempsol [Station2+1][i] = 0;
     tempsol[Station2+1][0] = store;
     memmove (tempElemCount+Station2+2,
                         tempElemCount+Station2+1,
                         2*(tempNumStations - (Station2+1)+1));
     tempElemCount[Station2+1] = 0;
     tempNumStations++;
     }
     if (Station2 < Station1)    //stn was added above stn removed
                    flag = 1;
tempElemCount [Station1+flag] --;
if (tempElemCount [Station1+flag] >= 0)     //if there is at least 1 element remaining
                                            //in the station, tidy up.
               memmove (*(tempsol+Station1+flag)+Element1,
                         *(tempsol+Station1+flag)+(Element1+1),
                         2*(tempElemCount[Station1+flag] - Element1 +2));   //close gap
        else
               {                                //have emptied a stn; remove it:
                                                //shift solution matrix
               memmove (*(tempsol+Station1+flag), *(tempsol+Station1+1+flag),
                         2*ELEMSIZE*(tempNumStations-Station1+1));
                                                //2 bytes per integer, ELEMSIZE
                                                //integers per 1-D array
                                                //times the number of remaining
                                                //stations.
               memmove (tempElemCount+Station1+flag,
                         tempElemCount+Station1+1+flag,
                         2*(tempNumStations - Station1 + 1));
                                                //accordingly, must also
                                                //shift ElementCount array
               ReduceFlag = 1;
               tempNumStations--;               //reduce NumStations
               }
        //check precedence constraints
        if (Station1 < Station2)       //element moved down
        {
        for (j=Station1; j<=Station2-ReduceFlag; j++)
```

```
                for (pos = 0; pos <=tempElemCount[j]; pos++)
                {
                bit = tempsol[Station2+1-ReduceFlag][tempElemCount[Station2+1-ReduceFlag]];
                if (prec[tempsol[j][pos]][bit/32] &
                   (long)1<<bit%32)  InfeasFlag = 1;
                }
        }
        else                    //element moved up
        {
        for (j=Station2+2; j<=Station1+1-ReduceFlag; j++)
                for (pos=0; pos <=tempElemCount[j]; pos++)
                {
                bit = tempsol[j][pos];
                if (prec[tempsol[Station2+1][tempElemCount[Station2+1]]][bit/32]
                    & (long)1<<bit%32)  InfeasFlag = 1;
                }
        }
}



void Perturb ()
{
        float     p;
        InfeasFlag = 0;
        memmove (tempsol, solution, sizeof (solution));
                                        //tempsol allows the elements to be
                                        //moved around without permanently
                                        //losing the last good balance.
                                        //if tempsol is good, it will be
                                        //saved in SA.
        memmove (tempElemCount, ElementCount, sizeof (ElementCount));
        tempNumStations = NumStations;
        p = (float)random (101)/100.0;
        if ((p > 0.96) && (tempNumStations < MaxStns)) New ();
        else if (p < 0.48) Swap();
        else Transfer ();
}



void SA()
{
        int       NumAttempted = 0,
                  NumAccepted = 0,
                  i,j, k, z, x,
                  temp_chg_count = 0;
        float   rnum,
                  prob,
                  FirstOFbar;;
```

```
TotalAttempted=0;
BestObj = 99999;
InitialSolution();          //Find Initial Temperature
Calculate_ObjFunc();
InitialTemp();
tempNumStations = 0;                    //reinitialize tempNumStations
for (j=0; j<ELEMSIZE; j++)              //reinitialize tempsol
        for (k=0; k<ELEMSIZE; k++)
          tempsol[j][k] = 0;
for (j=0; j<ELEMSIZE; j++)  tempElemCount [j] = 0; //reinitialize
                                                   //tempElementCount
InitialSolution();
Trials_per_Temp = ( ((tempNumStations)*NumElems+ 1)/2);
memmove (solution, tempsol, sizeof (tempsol));
memmove (ElementCount, tempElemCount, sizeof (tempElemCount));
NumStations = tempNumStations;
MaxStns = NumStations;
Calculate_ObjFunc();
OldObj = NewObj;

do                              //while not frozen//
{
        for (i=0; i<Trials_per_Temp; i++) //for each temperature//
        {
        do                      //keep trying until you find
        Perturb ();             //a FEASIBLE solution
        while (InfeasFlag);
        NumAttempted++;             //count FEASIBLE attempts
        Calculate_ObjFunc();
        if (NewObj < OldObj)        //better solution//
                {
                NumAccepted++;
                OFsum += OldObj;   //sum the accepted obj. funcs.
                OldObj = NewObj;
                if (OldObj < BestObj)
                        {BestObj = OldObj;
                        memmove (best_solution, tempsol, sizeof (tempsol));
                        memmove (best_ElemCount, tempElemCount, sizeof
                                (tempElemCount));
                        best_NumStns = tempNumStations;}
                memmove (solution, tempsol, sizeof (tempsol));
                memmove (ElementCount, tempElemCount, sizeof (tempElemCount));
                NumStations = tempNumStations;
                }
        else                        //worse solution//
                {
                prob = exp(-(NewObj - OldObj)/temperature);
                rnum = (float)random(101)/100.0;
                if (prob < 0.001) prob = 0.0;
                if ((1-prob) < rnum) //find prob
                        {                   //accept anyway
```

```
                                NumAccepted++;
                                OFsum += OldObj;   //sum the accepted obj. funcs.
                                OldObj = NewObj;
                                if (OldObj < BestObj)
                                        {BestObj = OldObj;
                                         memmove (best_solution, tempsol, sizeof (tempsol));
                                         memmove (best_ElemCount, tempElemCount, sizeof
                                                (tempElemCount));
                                         best_NumStns = tempNumStations;}
                                memmove (solution, tempsol, sizeof (tempsol));
                                memmove (ElementCount, tempElemCount, sizeof
                                        (tempElemCount));
                                NumStations = tempNumStations;
                                }
                        }
                }
        if (NumAccepted > 0) newOFbar = OFsum/NumAccepted; //Average value of o.f. at this temp
        else newOFbar = OFbar[2];
        if (temp_chg_count <3)
                {
                OFbar[temp_chg_count] = newOFbar;
                FirstOFbar = OFbar[0];
                }
        else            //move OFbars--displacing the oldest
                {
                SOFold = (OFbar[0] + OFbar[1] + OFbar[2])/3.;
                OFbar[0] = OFbar[1];
                OFbar[1] = OFbar[2];
                OFbar[2] = newOFbar;
                SOFnew = (OFbar[0] + OFbar[1] + OFbar[2])/3.;
                }
        oldtemp=temperature;
        temperature = temperature * COOLING_RATE;
        temp_chg_count ++;
        TotalAttempted += NumAttempted;
        NumAttempted = 0;
        NumAccepted = 0; OFsum = 0;
        }  while (((temp_chg_count<4) || ( (fabs(SOFnew-SOFold))/
                (oldtemp-temperature)*temperature/FirstOFbar > epsilon))
                && (temperature > 1.0));
    }


void DisplayResults ()
{

        int     i, j, k, mins;

        fprintf (outfptr,"\nRun # %d: ", runcount);
        fprintf (outfptr,"%f %.2f %.2f", epsilon, To, COOLING_RATE);
```

```
        if (repcount == 0)
           {fprintf (delfptr,"%f %.2f %.2f\n", epsilon, To, COOLING_RATE);
            fprintf (tfptr,"%f %.2f %.2f\n", epsilon, To, COOLING_RATE);
            fprintf (sfptr,"%f %.2f %.2f\n", epsilon, To, COOLING_RATE);
           }
        for (i=0; i<=best_NumStns; i++)
                {
                fprintf (outfptr,"\nStation #%d :", i+1);
                for (j=0; j<=best_ElemCount[i]; j++)
                fprintf (outfptr,"\t%d ", best_solution[i][j]+1);
                }
        fprintf (outfptr,"\nDelta:  %7.3f", BestObj);
        fprintf (delfptr, "  %7.3f\n", BestObj);
        mins = (end-start)/CLK_TCK/60;
        fprintf (outfptr,"      Time:  %d min  %.2f secs.\n",
              mins, fmod((end-start)/CLK_TCK, 60.));
        fprintf (tfptr," %d:%.0f\n",
                mins, fmod((end-start)/CLK_TCK, 60.));
        fprintf (outfptr,"Feasible solns. attempted: %lu.\n", TotalAttempted);
        fprintf (sfptr, "%lu\n", TotalAttempted);
        fprintf (outfptr, "final temp = %f \n", temperature);
}


void main (void)
{
        int     i,
                j;
        struct rundata
           {
           float e, t, r;
           } run[NUMRUNS];

        FILE *fptr;
        randomize ();
        GetData();
        fptr = fopen("das.txt", "rt");
        for (i=0; i<NUMRUNS; i++) fscanf (fptr, "%f %f %f",
           &run[i].e, &run[i].t, &run[i].r);
        fclose (fptr);
        outfptr= fopen ("dasl3y.out", "wt");
        delfptr= fopen ("dasl3y.del", "wt");
        tfptr = fopen ("dasl3y.t", "wt");
        sfptr = fopen ("dasl3y.s", "wt");
        for (runcount=0; runcount<NUMRUNS; runcount++)
            {
                printf ("RunCount = %d\n", runcount);
                for (repcount=0; repcount<NUMREPS; repcount++)
                {
                printf ("repcount = %d\n", repcount);
```

```
                epsilon = run[runcount].e;
                To = run[runcount].t;
                COOLING_RATE = run[runcount].r;
                start=clock();
                SA();
                end=clock();
                DisplayResults();
                }
            }
        fclose (outfptr);
        fclose (delfptr);
        fclose (tfptr);
        fclose (sfptr);
}
```

APPENDIX B: RESULTS FROM EXHAUSTIVE SEARCHES AND SMITH'S (1990) MODIFIED EXHAUSTIVE SEARCH

Part 1: Exhaustive Searches

Exhaustive searches were successfully completed for the four small problems. With the exception of Problem 2, these were complete exhaustive searches--with no lower bound on cycle time. The results of these runs are provided below.

Problem 1

Station 1: 2, 6
Station 2: 5, 7, 10
Station 3: 1, 12
Station 4: 3, 9, 11
Station 5: 4, 8, 13, 14
Station 6: 15
Station 7: 16, 17
Station 8: 18, 19

Delta: 254.0
Time: 268 minutes, 9.73 seconds
Number of Feasible Solutions Found: 288,434

Problem 2
(Note: This problem was run with
a lower limit on cycle time of 50%)

Station 1: 2, 4
Station 2: 1, 5, 8
Station 3: 7, 11
Station 4: 3, 6, 10, 13, 16, 17
Station 5: 12, 14, 19
Station 6: 9, 18
Station 7: 15

Delta: 161.7
Time: 32 hours, 56 minutes, 15.44 seconds
Number of Feasible Solutions Found: 112,542,808

Problem 3

Station 1: 1, 2, 3, 4, 8
Station 2: 5, 6, 7, 9, 10, 12
Station 3: 11, 13, 14, 15
Station 4: 16, 17, 18, 19

Delta: 164.0
Time: 0 minutes, 45.05 seconds
Number of Feasible Solutions Found: 2,080

Problem 4

Station 1: 2, 3, 4, 5, 9, 11
Station 2: 1, 7, 8, 13, 14, 16, 17
Station 3: 6, 10, 12, 15, 18, 19

Delta: 32.0
Time: 136 minutes, 3.52 seconds
Number of Feasible Solutions Found: 3,437,924

Part 2: Smith's Modified Exhaustive Search

Smith's (1990) modified exhaustive search was run for the four large problems. The parameters for the runs are given in Section 4.1.2, and thus only the station assignments are provided here.

Problem 5

Station 1: 1, 4, 6
Station 2: 9, 13
Station 3: 3, 8
Station 4: 12, 16
Station 5: 20, 22, 23
Station 6: 5
Station 7: 7, 10, 14
Station 8: 17, 18
Station 9: 21, 24
Station 10: 25, 28
Station 11: 27, 30
Station 12: 32
Station 13: 2
Station 14: 11, 26
Station 15: 15, 34
Station 16: 19, 29
Station 17: 31, 33
Station 18: 35, 36
Station 19: 37, 38
Station 20: 39
Station 21: 40, 41
Station 22: 42
Station 23: 43, 44
Station 24: 45
Station 25: 46
Station 26: 47, 48
Station 27: 49, 50

Delta: 4449.47
Time: 31 minutes, 14 seconds

Problem 6

Station 1: 3, 7
Station 2: 5, 12
Station 3: 8, 11, 20
Station 4: 4, 9, 23
Station 5: 6, 25
Station 6: 1, 13
Station 7: 10, 14, 15
Station 8: 2, 18
Station 9: 17, 21
Station 10: 16, 28
Station 11: 22, 29
Station 12: 24, 27
Station 13: 26, 31
Station 14: 33, 35
Station 15: 19, 39
Station 16: 37
Station 17: 40
Station 18: 30
Station 19: 32
Station 20: 34, 41
Station 21: 42
Station 22: 43
Station 23: 36, 44
Station 24: 46
Station 25: 38, 49
Station 26: 45
Station 27: 47, 48, 50

Delta: 4664.59
Time: 4 minutes, 57 seconds

Problem 7

Station 1: 1, 2, 5, 6
Station 2: 3, 4, 8, 10, 12, 14, 18, 21
Station 3: 7, 11, 16, 20, 22, 23
Station 4: 9, 13, 24, 25
Station 5: 15, 17, 26, 28, 29
Station 6: 19, 30, 31, 33, 35
Station 7: 27, 32, 37, 39
Station 8: 34, 36, 38, 40, 41
Station 9: 42, 43, 44, 45
Station 10: 46, 47, 48, 49, 50

Delta: 1288.04
Time: 42 minutes, 27 seconds

Problem 8

Station 1: 1, 2, 5, 6
Station 2: 3, 4, 7, 9, 10, 12, 14
Station 3: 8, 13, 20, 23, 25
Station 4: 17, 18, 21, 27, 29, 33
Station 5: 11, 16, 22, 24
Station 6: 15, 26, 28, 30, 31, 35
Station 7: 32, 34, 39, 41
Station 8: 19, 36, 37, 38, 40
Station 9: 42, 43, 45
Station 10: 44, 46, 47, 48, 49, 50

Delta: 1217.57
Time: 3 hours, 55 minutes, 8 seconds

APPENDIX C: RESULTS FROM SIMULATED ANNEALING

Small Problems

Data on processing time and number of feasible solutions from SA Method I is provided in Tables C.1 - C.6. Data on processing time and number of feasible solutions from SA Method II is provided in Tables C.7 - C.14.

Large Problems

Data on processing time and number of feasible solutions from SA Method I is provided in Tables C.15 - C.18, while data on processing time and number of feasible solutions from SA Method II is provided in Tables C.19 - C.20. Finally, the station assignments for the best solutions found for each of the four large problems are given.

Table C.1: Time Data: Problem 4, SA Method I

MaxAttempted	MaxAccepted																			
	50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
100	0:11																			
	0:5																			
	0:10																			
	0:5																			
	0:19																			
300	0:9	0:12	0:18	0:24	0:15															
	0:10	0:24	0:27	0:18	0:15															
	0:10	0:30	0:28	0:21	0:15															
	0:15	0:12	0:28	0:21	0:17															
	0:9	0:23	0:27	0:18	0:17															
500	0:25	0:47	0:43	0:36	0:26	0:31	0:33	0:31	0:22											
	0:27	0:15	0:17	0:21	0:36	0:31	0:33	0:32	0:23											
	0:25	0:15	0:19	0:24	0:43	0:26	0:38	0:30	0:20											
	0:38	0:15	0:35	0:24	0:27	0:26	0:29	0:30	0:17											
	0:29	0:17	0:20	0:22	0:25	0:43	0:35	0:29	0:20											
700	0:26	0:58	0:24	1:1	1:2	0:38	0:40	0:40	0:44	0:48	0:45									
	0:18	0:51	0:59	0:27	0:37	1:2	1:11	0:43	1:4	1:2	0:48									
	0:27	0:55	1:5	0:28	1:8	0:34	1:9	0:49	0:46	0:50	0:43									
	0:39	0:20	0:56	0:31	0:59	0:37	0:42	0:48	0:60	0:41	0:49									
	0:16	0:56	0:23	0:31	1:8	0:58	1:7	1:6	0:45	0:47	0:42									
900	0:21	0:54	1:20	1:23	0:38	0:43	1:28	0:45	0:50	0:56	1:16	1:22	1:2	0:54	0:51					
	0:19	1:3	1:25	1:6	0:44	0:39	0:51	1:18	0:46	1:31	1:22	1:12	0:60	0:57	0:49					
	0:27	1:9	1:24	0:33	0:37	1:30	0:46	1:30	0:51	1:33	1:20	1:15	1:12	1:2	0:48					
	0:17	0:24	0:26	1:7	2:2	0:45	0:43	0:45	1:22	0:57	1:19	0:55	0:57	0:57	0:53					
	0:21	0:60	1:37	0:32	0:37	1:20	0:50	1:33	1:31	0:53	1:21	1:3	1:1	0:54	0:52					

Table C.1: Time Data: Problem 4, SA Method I, continued

MaxAttempted	MaxAccepted 50	100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
1100	0:45	0:31	1:25	1:51	1:42	1:21	1:24	0:54	1:43	1:42	1:53	1:43	1:40	1:25	1:10					
	0:22	0:57	0:33	1:58	0:37	1:33	1:52	1:32	1:50	0:60	1:0	1:12	1:35	1:17	1:29					
	0:21	1:17	0:31	1:48	0:44	0:42	2:8	2:7	1:49	1:45	1:45	1:43	1:45	1:11	1:29					
	0:22	1:0	1:25	0:42	1:37	0:48	0:48	0:52	1:40	0:60	1:52	1:49	1:44	1:46	1:21					
	0:31	0:26	0:29	1:52	0:41	0:46	1:34	1:31	0:60	0:57	1:44	1:42	1:45	1:20	1:29					
1300	0:26	0:29	1:23	0:45	0:48	3:30	1:57	1:60	2:12	1:56	1:59	1:10	2:4	1:55	1:59	1:57	1:54			
	0:25	0:29	0:32	2:0	0:51	2:7	1:46	1:53	1:59	2:11	2:15	1:14	2:13	1:55	2:2	1:25	1:49			
	0:31	0:29	1:22	0:38	1:11	1:54	1:54	0:57	0:60	2:8	2:3	1:58	2:14	1:26	1:56	1:59	2:12			
	0:36	0:32	0:35	0:39	2:17	0:54	0:49	1:47	1:6	1:4	2:16	2:3	1:16	1:16	1:18	1:48	1:21			
	0:28	0:30	1:25	0:42	2:17	1:55	1:35	2:4	2:3	1:55	1:55	1:23	1:15	1:60	2:12	1:19	1:46			
1500	0:28	0:54	0:37	1:45	0:43	0:52	0:57	1:1	1:9	1:21	1:7	2:20	2:7	2:28	1:17	1:30	2:19	2:8	1:41	1:45
	0:32	0:59	1:25	0:41	2:25	0:50	0:55	1:5	2:13	2:14	2:4	1:17	1:16	2:28	1:28	2:32	1:32	2:13	2:18	2:10
	0:47	0:31	0:38	0:44	0:51	2:49	0:55	2:3	1:59	2:7	2:25	2:16	2:14	1:33	1:26	2:23	2:19	2:9	1:35	1:49
	0:39	0:57	1:7	1:14	0:46	2:53	1:57	2:12	2:10	2:23	1:17	1:27	2:20	2:20	2:26	2:19	2:16	1:29	1:36	2:5
	0:26	0:32	0:34	1:53	0:57	1:50	2:13	1:5	2:15	1:13	2:15	2:35	1:19	1:23	2:39	1:32	2:29	1:51	2:24	1:38
1700	0:43	0:58	1:51	1:55	0:55	0:53	3:10	2:20	1:4	1:9	1:23	2:37	1:27	2:48	1:30	1:46	1:42	2:40	2:33	1:54
	0:27	1:13	0:44	1:57	0:51	0:57	1:49	2:6	2:36	2:30	2:34	1:22	2:30	2:55	1:30	2:31	1:54	2:26	1:42	2:50
	0:28	0:39	0:40	1:50	2:16	2:47	2:19	2:38	2:27	1:11	2:22	2:27	1:30	1:23	2:39	2:42	2:36	1:44	2:49	1:49
	0:27	1:25	0:41	1:57	2:24	2:47	0:59	1:4	2:27	2:31	1:23	3:5	2:36	2:35	1:37	2:46	2:48	2:36	2:37	2:34
	0:27	0:37	1:21	0:47	0:49	2:47	3:28	1:4	2:22	1:12	2:29	1:25	1:24	2:18	2:37	2:51	2:55	1:45	1:39	2:33
1900	0:32	0:46	0:53	0:52	2:19	2:52	1:7	2:28	3:3	1:24	2:33	2:39	2:55	1:26	1:31	3:5	2:42	1:55	2:50	2:59
	0:31	1:3	1:25	0:52	2:20	2:23	1:11	1:7	1:10	3:3	2:57	2:55	2:59	2:55	2:51	2:58	3:1	1:43	1:46	2:57
	0:33	0:55	0:51	1:50	0:54	2:49	3:13	3:16	1:11	1:16	1:21	2:48	1:25	1:33	2:53	2:49	1:47	2:44	2:7	2:50
	0:31	0:60	1:22	0:48	2:18	0:56	1:7	3:12	2:25	1:18	1:27	1:21	1:34	2:39	2:53	2:55	3:7	2:41	3:11	1:50
	0:38	0:39	1:26	0:44	2:19	1:10	3:17	1:5	1:15	2:36	2:43	2:44	1:30	2:30	1:40	2:46	2:54	2:48	3:4	1:52

Table C.2: Number of Feasible Solutions: Problem 4, SA Method I

MaxAttempted	MaxAccepted 100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
100																			
300	6546	7695	8565	6855															
	7900	7845	7743	6534															
	7978	8727	7768	6832															
	6492	9413	8278	7132															
	7467	8510	8945	7427															
500	11776	13101	13500	12239	13732	13546	13031	10101											
	8779	9679	11250	15429	13231	13060	13001	10118											
	8288	9992	11040	14533	13192	13154	12020	9099											
	8169	11257	11252	12608	12257	13022	12052	8081											
	8854	10636	11190	12608	13980	13064	11529	9080											
700	16541	12592	17284	16382	17558	17705	17740	18926	18574	17187									
	16447	16194	14038	15170	20486	20505	17592	21570	19935	17317									
	16359	18622	13737	18753	16225	20465	18256	18150	17901	16607									
	10596	15172	15771	18695	16129	17833	18396	21578	17176	17365									
	16164	12317	14850	20103	19533	20504	20672	18711	17882	15914									
900	16100	23855	24461	19873	20152	25305	21516	22020	23101	25914	26804	23190	21991	19676					
	17982	24138	18355	18033	19660	21780	24538	21896	27175	27578	25087	23224	22042	20579					
	19532	24440	18351	17691	24541	21611	26487	21227	27787	25895	26820	24959	22862	18831					
	12373	14331	21106	31499	19084	20779	21499	25397	23081	26548	23254	24011	21943	20579					
	17021	22544	16415	18782	23404	19821	27146	26217	23044	26621	23448	24119	21910	20527					

Table C.2: Number of Feasible Solutions: Problem 4, SA Method I, continued

MaxAttempted	MaxAccepted 100	150	200	250	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000
1100	18312	24710	32269	24863	23576	26700	26323	30936	30635	32287	33596	32806	31673	28103					
	16287	17246	32761	20301	28358	28912	29447	32053	26855	26933	27864	32938	29404	31076					
	19790	15872	32275	20650	21314	32516	32403	30922	30738	33560	34783	34104	29563	30159					
	17120	24567	20128	26596	23693	23234	24997	29847	27838	32305	33720	33958	33687	28992					
	15409	16712	33080	22748	21605	27678	30325	27612	26573	30939	32532	32573	29394	30210					
1300	17573	24328	22501	25141	46572	34683	34128	37815	33414	35207	30744	37956	38000	37897	37580	36815			
	18028	18442	28729	23609	35048	31838	32786	35086	37372	37640	31866	39376	37969	37974	33801	36819			
	17737	23866	21593	23142	32592	31272	26797	27340	36123	36313	38270	38030	33442	39330	37494	40755			
	18987	19857	20613	40313	24300	26202	32685	27290	29612	36633	36603	32937	32515	34598	37271	34417			
	17535	24830	23673	40452	25782	27903	32284	34119	36041	36462	32068	32996	36912	40527	33871	35679			
1500	16904	21214	33164	23924	26449	26841	31393	32017	33096	31535	41422	42887	44389	35112	38617	42812	42029	40144	40543
	16610	24646	22624	40902	26404	27113	29052	38526	37908	38129	34055	34933	44168	36601	44835	39335	43729	44297	41702
	20063	22626	23391	26235	49023	27201	36318	35669	38817	42924	41618	40382	36951	36584	44893	46280	43747	38676	42035
	14275	21256	23017	24457	48803	32782	36497	37248	40640	34597	33629	41163	42750	44188	43295	42848	39839	40055	41756
	19302	20408	32908	28520	25844	38495	30066	37365	33322	39632	43258	36745	37372	46813	38755	44257	39518	45698	38910
1700	14658	32949	32610	28137	26886	56794	40098	32436	34279	36394	46926	40400	50133	40500	44214	41111	50849	51668	44144
	18818	26817	28032	23870	29467	33805	37334	42671	43180	43625	37441	46598	48022	40570	49441	44793	49101	42309	52028
	24358	22883	32345	40097	48997	40015	43211	41035	35686	40322	44809	38786	38009	49644	48771	48307	43543	53317	44073
	18297	24655	33535	41374	48965	30622	32851	40950	43231	36682	47220	45145	47874	40239	50695	51283	50789	49856	50861
	23789	24759	26464	26654	49083	56268	32568	42580	35516	45157	39885	38927	45395	47672	50702	51700	43389	43843	50089
1900	28617	30910	27066	40995	48702	33453	40839	52498	38399	47092	48752	49875	39913	42953	55155	52274	47421	53805	54460
	14496	24429	27956	41037	37859	37219	35857	36868	51326	49632	51904	50573	52677	51774	53578	54117	46240	48422	60961
	16279	30972	31766	27595	50035	57746	56677	34519	36101	37199	47860	41117	43984	54196	57925	47542	53247	47825	56495
	16379	24717	27615	40735	28872	35442	55405	41989	36599	38714	40160	42675	51291	51429	52824	55773	57355	57481	46474
	22986	22118	24613	40395	29508	57376	34544	36299	45731	46182	47726	43039	48560	44140	52845	54480	52277	55868	46617

Table C.3: Time Data: Problem 1, SA Method I

MaxAccepted	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
1100	5:52	3:0	4:57	3:46	3:40	4:1	4:27	4:46	4:4	3:47																			
	4:20	3:11	3:6	4:20	4:13	3:42	4:20	4:0	3:57	3:31																			
	2:57	5:35	4:46	3:27	4:25	4:49	4:6	4:8	3:47	4:1																			
	4:3	5:40	4:22	5:50	3:42	4:7	3:49	4:9	3:35	3:41																			
	2:43	5:50	4:35	3:33	3:58	4:10	4:60	4:21	3:53	4:10																			
1300	3:2	6:33	3:30	4:10	4:27	4:53	4:37	5:6	5:16	4:55	4:56	4:40																	
	2:50	6:11	6:19	4:35	3:59	4:44	4:55	5:38	4:54	4:26	4:53	4:45																	
M	3:10	4:55	3:31	5:56	5:13	3:59	4:50	5:1	4:58	4:23	4:28	4:29																	
a	3:10	4:15	3:15	4:28	6:17	6:21	4:37	4:49	5:22	4:45	4:52	4:37																	
x	4:58	3:22	6:41	3:33	5:52	4:8	4:48	5:31	4:52	4:50	4:32	4:45																	
A 1500	7:60	4:2	10:38	7:24	7:10	4:41	4:43	5:10	6:1	5:2	5:48	5:27	5:38	5:12	5:33														
t	7:48	3:57	4:9	6:56	7:37	7:20	6:26	4:51	5:9	5:33	5:12	5:17	5:30	5:35	5:21														
t	7:49	7:9	10:20	5:60	7:10	4:38	6:41	5:44	5:5	6:11	5:22	5:14	5:40	5:39	5:18														
e	5:25	5:32	7:32	7:40	7:44	7:6	4:21	5:52	5:15	6:10	5:23	5:12	5:39	5:22	5:17														
m	5:30	3:58	9:58	5:17	4:24	4:21	4:43	6:23	5:15	6:6	5:4	5:47	5:34	5:46	5:25														
p 1700	8:10	9:17	9:21	12:13	5:22	6:16	8:1	7:38	5:14	6:1	6:41	6:12	6:26	5:56	6:16	6:26	6:28												
t	5:38	8:60	10:36	4:11	4:25	6:8	8:3	8:22	7:20	7:3	5:39	5:48	6:17	5:57	6:6	7:23	6:9												
e	2:55	9:19	6:13	11:48	10:22	9:6	8:40	6:42	7:27	7:17	7:8	5:46	6:7	6:57	6:9	6:24	6:31												
d	7:54	9:3	10:24	5:40	4:50	4:48	5:3	6:17	7:39	7:1	5:38	6:30	6:22	6:3	6:16	6:27	6:34												
	5:19	3:50	8:3	8:13	4:45	9:4	5:49	5:10	5:18	5:59	7:35	6:32	5:57	5:59	6:3	6:31	6:36												
1900	9:20	9:13	10:19	6:46	7:3	11:45	5:15	5:21	9:11	6:31	6:14	6:7	6:53	6:50	7:15	7:39	7:7	7:30	6:52										
	9:40	9:30	9:43	4:17	9:1	5:0	8:51	5:41	5:55	9:17	9:8	7:3	7:0	6:45	6:26	7:34	7:21	6:52	6:34										
	3:38	3:53	6:7	4:29	5:12	9:19	6:58	8:43	8:50	6:12	6:17	7:1	6:27	6:20	6:23	7:1	7:18	7:29	7:12										
	7:53	6:21	10:49	4:49	9:50	8:49	7:17	5:26	9:28	7:42	5:52	7:3	7:39	7:16	7:48	8:13	7:31	7:26	7:29										
	7:58	6:16	10:9	4:46	6:19	5:22	6:5	8:21	5:43	7:26	6:15	6:6	6:40	6:26	8:2	7:37	7:17	7:26	7:44										

Table C.3: Time Data: Problem 1, SA Method I, continued

MaxTemp	MaxAccepted																													
	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
2100	7:11	8:18	4:48	4:53	4:45	15:34	7:57	12:35	10:34	8:25	6:52	7:4	10:7	8:24	8:10	8:8	8:12	8:40	7:60	8:22	7:58								
	3:51	8:17	4:59	8:5	5:14	5:54	5:50	10:47	7:29	8:1	7:40	9:31	8:5	9:20	8:9	9:13	8:17	8:37	8:12	8:27	7:43								
	10:0	10:6	11:15	12:39	11:36	15:21	13:42	11:60	6:56	6:30	8:44	6:60	9:15	7:39	8:34	9:10	8:28	8:9	8:14	8:18	8:17								
	3:59	4:29	4:46	12:43	7:9	15:33	6:20	13:44	6:52	6:11	7:4	7:7	8:51	8:2	8:18	8:29	9:2	7:58	8:4	7:27	9:9								
	3:50	4:4	5:8	12:33	7:49	5:56	13:2	10:8	8:30	6:26	7:7	8:7	9:37	7:43	9:10	7:27	7:55	8:27	7:43	8:15	8:28								
2300	10:11	9:58	4:60	8:2	11:41	7:30	7:57	7:56	10:52	6:6	7:2	7:7	11:16	9:24	7:32	8:41	8:26	8:42	9:1	8:42	9:13	9:6	9:48						
	8:17	8:24	5:26	8:13	8:10	5:59	5:56	6:39	7:34	7:0	6:40	10:24	10:34	10:2	9:37	9:43	8:11	10:2	9:20	8:31	9:32	9:11	9:0						
	4:1	4:17	5:4	13:5	13:35	5:23	5:5	8:59	11:8	6:9	10:12	7:55	8:41	7:14	7:18	9:58	9:38	9:10	9:2	9:19	9:19	8:46	9:5						
	10:22	4:29	11:32	8:25	7:49	13:56	5:41	7:53	11:41	11:54	12:6	7:14	6:56	7:29	8:25	7:20	7:56	9:42	9:21	9:43	9:5	8:39	9:2						
	6:45	4:34	7:57	5:9	11:29	5:1	9:18	6:7	6:24	12:3	11:17	6:45	11:32	7:18	9:32	9:5	7:38	8:35	9:17	9:4	8:47	9:30	9:25						
2500	8:25	11:43	5:29	12:45	12:59	6:27	12:33	18:51	13:11	7:19	14:27	7:51	7:12	11:55	9:16	8:38	8:59	8:59	10:10	8:60	9:4	9:15	9:28	9:39	10:19				
	3:46	4:42	10:57	8:47	8:28	10:13	17:4	18:3	7:10	7:49	11:14	13:41	7:19	9:20	8:38	11:59	8:57	10:22	8:60	9:8	9:19	10:24	9:33	9:45	10:21				
	10:53	5:1	13:54	12:35	12:54	14:50	13:31	12:6	7:8	7:54	7:13	7:54	8:17	8:52	13:31	9:34	11:11	8:49	9:38	9:31	9:17	9:40	8:41	10:2	9:42				
	4:33	5:11	13:39	8:58	9:1	15:16	6:40	7:3	9:23	7:58	13:14	7:26	11:34	7:48	8:53	9:19	12:43	9:22	10:6	9:21	9:4	8:34	9:21	10:16	9:41				
	10:35	8:58	4:31	5:37	5:48	15:29	17:17	6:57	7:21	13:34	12:30	13:52	7:59	8:24	12:31	8:48	10:10	9:56	9:2	8:31	8:48	9:0	9:41	9:30	9:43				
2700	5:37	12:10	5:52	9:29	10:21	14:23	13:59	10:40	13:55	14:4	7:55	8:31	13:53	13:56	8:44	11:58	12:48	13:58	11:10	10:11	9:26	10:15	9:59	9:58	10:3	10:1	10:58		
	4:39	4:39	10:56	6:22	14:37	6:20	13:54	6:31	19:24	18:46	8:25	15:39	12:44	8:38	9:11	9:59	12:22	11:20	10:15	10:11	10:50	9:24	8:59	10:21	9:43	11:15	10:33		
	8:22	12:9	5:57	10:49	5:36	7:2	16:56	18:28	7:49	14:14	8:6	13:22	11:1	12:8	13:23	8:56	12:31	9:53	10:0	9:32	10:55	9:57	10:0	9:31	10:24	10:42	9:60		
	4:49	12:38	5:33	10:14	5:36	14:45	6:59	14:5	7:59	12:12	13:4	8:4	8:43	13:1	13:24	9:19	9:17	9:50	10:2	9:56	10:56	10:3	10:32	9:39	9:51	10:33	10:31		
	10:44	12:18	13:53	5:18	14:45	14:52	17:5	18:29	8:8	14:30	14:6	13:39	8:32	14:24	14:25	10:58	9:57	11:4	10:48	11:26	12:15	9:26	9:25	10:4	9:10	10:19	10:44		
2900	11:23	9:58	13:24	5:45	5:48	7:18	6:27	16:34	7:25	10:27	8:2	18:18	10:44	10:45	9:59	14:10	11:54	14:9	11:40	14:54	9:57	12:01	13:8	11:25	13:45	11:28	14:21	11:57	10:55
	5:4	9:47	5:1	14:33	9:0	15:23	16:46	18:32	18:26	7:7	8:25	12:30	15:21	12:18	9:15	16:9	10:15	12:15	12:29	13:31	11:4	12:52	13:5	12:31	13:38	13:33	12:46	11:23	11:17
	5:29	12:8	5:34	15:29	6:2	17:22	7:7	9:60	16:16	15:11	20:30	16:58	13:41	9:28	13:53	11:54	14:40	11:16	11:40	12:36	10:1	9:43	13:38	12:42	12:37	11:8	12:45	12:30	11:22
	5:30	12:33	13:52	6:33	14:40	15:34	15:52	16:9	7:28	21:36	7:59	8:59	16:37	9:23	9:23	16:40	13:2	11:47	12:20	12:55	10:11	10:44	11:11	11:39	13:39	11:21	11:49	11:14	10:34
	8:32	12:21	13:15	14:7	15:6	15:46	15:29	15:24	15:47	8:4	8:36	14:6	13:53	16:15	8:48	10:32	10:20	12:51	13:44	13:49	10:54	13:55	14:35	12:58	12:4	12:20	12:35	11:26	10:59

Table C.4: Number of Feasible Solutions: Problem 1, SA Method I

MaxTemp	MaxAccepted 300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
1100	30886	23478	30991	28303	28397	28591	30811	33222	30891	30386																			
	25396	24582	26176	29580	27465	28769	30929	29924	30699	28001																			
	23408	31488	31139	25977	29301	31951	32076	30944	30578	31616																			
	25393	32779	27979	34012	28441	28856	29844	32070	29696	29350																			
	21792	32653	28130	26090	28571	29665	33089	31940	31857	31550																			
1300	26420	35844	28271	30156	30397	33848	34215	35404	37402	35901	36613	35969																	
	23677	36781	38494	30700	31564	32529	31680	37786	34884	35754	36590	35903																	
M	26873	29628	28840	34296	35362	32738	32975	35219	36316	34216	33930	34648																	
a	26175	29109	26777	29210	38498	38925	32992	32836	36257	34689	36713	35921																	
x	29818	28543	36875	29980	35730	32689	34177	36388	34864	35949	35363	36167																	
A 1500	41999	30835	55438	42523	42477	36413	37497	39865	40980	39225	39880	42210	42703	40459	42365														
t	39402	28525	31633	40326	43681	44001	42292	37241	39884	39106	41347	39347	39736	41806	40924														
t	39762	38839	54420	36547	42190	34732	40052	38165	41142	42133	41484	40588	41426	41635	42034														
e	31903	32471	42213	42812	44465	42440	35966	38492	37085	43697	40031	40844	41405	41797	42152														
m	31530	32098	54836	34481	34684	34953	37409	43051	39565	42108	38352	42112	41241	41965	40796														
p 1700	41932	47605	48031	62315	37548	40082	47508	47023	41869	44417	48196	45763	46521	47210	47618	47890	46506												
t	34632	47694	54636	34083	37290	39787	47869	50669	47244	46919	45329	46009	48323	45423	46174	50091	44877												
e	26133	47361	37915	60734	57080	49765	49654	43575	48690	47626	46724	44631	46672	49085	46100	46397	48106												
d	40739	49324	54345	34442	37944	38186	38027	40638	50232	45432	45213	47605	46536	45529	47671	46279	46677												
	30803	30849	44878	49358	36427	52628	39544	42712	42071	41593	48700	46332	43443	45665	46143	46518	46534												
1900	46591	48816	54387	39884	42432	64955	42108	45094	53438	44976	49649	46306	50977	49584	51946	50514	50888	52729	52655										
	48302	50809	50536	35517	49202	41199	51723	43787	46241	55347	56678	52235	51142	49332	49848	52089	52539	50857	48815										
	31971	33692	38523	35337	41137	51697	42685	55401	53606	46824	48100	50133	48826	48068	50108	50470	52611	52775	52633										
	41585	38003	55552	39797	52808	50274	44918	43734	54175	48420	47591	48256	52530	51627	55913	55978	54336	53051	52691										
	41675	35279	52769	39289	42591	42315	43827	50840	44304	48217	47501	48570	50694	49577	55672	54247	52351	52742	53195										

Table C.4: Number of Feasible Solutions: Problem 1, SA Method I, continued

MaxAccepted	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
2100	37156	44165	37581	39893	41101	76715	46839	65561	64254	53521	48732	52965	63876	56054	54558	56996	57475	57437	57512	57331	57383								
	30058	43835	40734	45196	43363	43324	46689	57965	46661	49555	50092	59181	55094	57972	56458	56914	55246	59638	57318	59475	57558								
	50461	48058	55153	59973	60643	74905	70314	65059	48801	47122	52109	50856	57601	55988	56638	60730	59087	57219	59502	59448	59480								
	31201	37065	38418	61310	45167	76312	43854	69154	50334	47473	50614	54749	58503	55614	58341	59014	59309	55299	57310	53134	61529								
	29150	35133	42085	62648	45758	41396	66423	55798	52065	48248	50022	52893	59551	54388	56465	54997	58802	59494	57457	57561	61330								
2300	47973	47627	39102	43046	63890	46963	49691	52541	67504	51941	53253	54040	69424	62778	57109	59911	62150	62181	62004	59883	62396	64379	66413						
	40935	44869	42502	48124	47378	48517	47209	52899	53377	51681	52973	69348	65985	66066	61162	64757	60381	64625	62315	62520	64446	62210	64409						
M	34215	34282	41003	62105	70639	44480	42992	54430	64532	49878	64915	58281	56538	58728	56862	64560	66231	62363	60053	62196	64347	59724	64209						
a	48909	36093	55372	45191	46673	75286	45070	47133	71009	68754	68888	53457	56050	55376	59347	57806	55620	62504	64375	64807	62084	62014	64032						
x	38124	35881	43501	44203	61129	41517	54889	50931	51371	71907	66953	53714	69541	56705	62271	64075	60074	58278	62508	62568	62479	64392	64317						
A 2500	41226	56734	44744	62552	69796	53618	63649	89820	67534	55093	78257	58490	62303	69244	62355	64827	65363	65338	68012	67560	67573	69216	67113	66710	71187				
t	31611	36738	54263	50645	49650	53821	82891	89312	54417	55079	63292	73015	59663	61911	59495	72192	62872	70379	65011	65330	69992	72445	69480	69289	71058				
t	49626	42606	64093	60476	69154	75852	66457	63784	52812	54477	57125	56389	64750	60137	74105	64638	72879	62842	65712	68076	67461	72530	64259	69376	69027				
e	37776	43553	63427	48377	53676	75564	47308	50158	55912	57740	72064	56208	69992	59628	62342	62311	74986	65081	67809	68190	65149	64784	69553	71755	71054				
m	50959	46123	36271	41431	47416	77462	82410	50652	52926	70918	67429	72968	61804	61786	71213	62625	60809	65300	65428	65300	67391	69764	69447	66454	69032				
p 2700	43188	57127	48722	52458	54101	70766	71320	59257	73177	73986	59192	59248	75394	76580	63178	76064	78754	78158	71227	68349	65819	78426	72888	72326	72012	74171	76284		
t	39265	38788	55041	48587	70644	46812	71520	52099	95261	93996	58756	80542	71951	62749	63174	65763	78673	71081	71061	73616	70776	73248	70022	75082	72105	76861	73569		
e	40731	56037	49779	56830	43188	49609	82336	88875	57751	75442	61969	76088	65216	68802	80454	65922	75354	68490	66646	68247	76215	70879	72904	72099	74727	76820	73263		
d	41659	57514	44034	53975	47173	70782	51381	73716	55345	66154	78859	56308	65113	73822	74282	65920	63641	70046	66501	68565	73639	76102	78204	72633	71895	74032	73566		
	49573	57157	64668	41575	71910	70291	82243	90716	59507	73987	75457	75720	62439	79634	77958	70519	68466	68712	71515	73608	76126	73257	70296	74732	71925	74088	70874		
2900	58744	48537	65360	48450	52276	55757	51537	94731	58455	63452	62070	93832	66504	67331	68535	79580	71544	79531	76815	84651	74226	81180	73249	83658	84550	77368	82694	79272	74877
	45031	47593	41217	69614	52573	75486	83282	90094	95987	56352	58976	67782	89326	75319	65927	83040	67454	71813	76867	78584	74173	76565	75492	80773	82106	86439	82213	78993	81048
	44955	57009	45353	72727	53764	82406	57062	59138	81459	77664	1E+05	88369	79152	65931	82053	68970	80815	71600	75210	79103	73937	76413	78688	78248	79270	80246	79683	82114	81359
	49204	57682	65388	51904	75634	76858	80392	79007	59487	1E+05	62449	61056	92448	68205	65532	82347	77394	76699	76655	76693	73807	75652	78208	80597	81111	80251	79529	79141	78099
	41067	56623	64646	68407	75927	76246	77229	75686	79830	58945	58640	78225	79334	87562	61524	64621	69404	71378	79563	79333	76590	84005	78765	75664	84649	80226	79673	79059	78364

Table C.5: Time Data: Problem 2, SA Method I

MaxTested	MaxAccepted																																
	300	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
1100	0:60	1:11	1:7	1:2	1:7	1:6	1:4	0:58	0:54																			
	0:58	1:8	1:5	1:6	0:60	1:4	0:56	0:60	0:57																			
	1:1	1:1	1:10	0:58	1:1	1:7	1:3	0:55	0:57																			
	0:55	1:3	1:8	1:2	1:10	1:2	0:58	0:49	0:52																			
	1:5	1:2	1:9	1:8	1:0	0:60	1:1	1:1	0:51																			
1300	1:6	1:9	1:27	1:12	1:12	1:15	1:12	1:18	1:19	1:8	0:59																	
	1:7	1:18	1:15	1:7	1:18	1:17	1:16	1:14	1:17	1:12	1:8																	
M	1:4	1:6	1:1	1:10	1:15	1:19	1:29	1:10	1:12	1:15	1:3																	
a	1:2	1:19	1:22	1:13	1:9	1:9	1:17	1:12	1:20	1:9	1:1																	
x	1:12	1:12	1:11	1:6	1:10	1:13	1:15	1:14	1:4	1:10	1:6																	
A 1500	1:12	1:27	1:22	1:26	1:27	1:23	1:50	1:36	1:34	1:25	1:18	1:26	1:14	1:16														
t	1:21	1:27	1:24	1:28	1:23	1:26	1:20	1:30	1:26	1:23	1:18	1:18	1:10	1:11														
t	1:22	1:24	1:17	1:28	1:29	1:44	1:34	1:40	1:21	1:21	1:23	1:28	1:20	1:9														
e	1:21	1:24	1:32	1:30	1:36	1:16	1:23	1:28	1:21	1:20	1:22	1:32	1:8	1:16														
m	1:6	1:35	1:15	1:31	1:28	1:28	1:33	1:29	1:30	1:33	1:28	1:19	1:12	1:13														
p 1700	1:18	1:42	1:43	1:50	1:43	1:36	1:48	1:35	1:43	1:39	1:32	1:44	1:52	1:34	2:15	2:13												
t	1:25	1:21	1:29	1:48	1:33	1:47	1:59	1:44	1:38	1:48	1:43	1:44	1:25	1:36	2:21	2:13												
e	1:34	1:36	1:29	1:27	1:39	1:32	1:30	1:32	1:45	1:34	1:32	1:38	1:32	1:33	2:23	2:22												
d	1:20	1:35	1:33	1:32	1:39	1:27	1:42	1:31	1:34	1:43	1:33	1:30	1:28	1:27	2:30	2:22												
	1:15	1:34	1:25	1:36	1:30	1:42	1:51	1:45	1:34	1:35	1:31	1:30	1:31	1:31	2:10	2:20												
1900	1:29	1:40	1:34	1:41	1:54	1:58	1:59	1:41	1:52	2:4	1:53	2:4	1:60	1:41	2:51	2:39	2:38	2:36										
	1:26	1:45	1:43	1:52	1:46	1:44	1:38	1:36	1:48	1:48	2:1	1:44	1:54	1:45	2:49	2:34	2:30	2:27										
	1:37	1:38	1:35	1:41	1:47	1:37	1:46	2:5	1:58	1:47	1:56	1:53	1:54	1:55	2:52	2:41	2:41	2:32										
	1:28	1:48	1:43	1:39	1:44	1:49	1:45	1:57	1:39	1:51	1:52	1:54	1:53	1:52	2:44	2:46	2:34	2:35										
	1:39	1:38	1:54	1:41	1:49	1:47	1:47	1:45	2:0	1:48	1:45	1:53	1:48	1:51	2:51	2:35	2:28	2:39										

Table C.5: Time Data: Problem 2, SA Method I, continued

MaxAccepted	300	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
2100	1:48	1:56	2:15	2:20	2:26	2:25	2:33	2:46	2:37	2:56	2:52	2:49	3:2	2:52	2:57	3:0	3:8	2:49	2:54	2:55								
	1:55	1:56	2:16	2:27	2:30	2:25	2:30	2:30	3:14	2:36	2:57	2:60	3:2	3:8	2:51	2:59	2:53	2:56	2:39	2:45								
	1:55	2:2	2:4	2:22	2:29	2:27	2:33	2:40	2:40	2:46	2:57	2:53	2:44	2:50	2:56	3:4	3:5	3:1	2:55	2:54								
	1:39	2:14	2:18	2:21	2:19	2:29	2:39	2:27	2:47	2:47	2:53	2:51	2:54	3:30	3:44	3:13	3:5	2:55	2:48	2:55								
	1:49	2:3	1:59	2:29	2:17	2:29	2:27	2:48	2:38	2:48	2:43	2:46	2:57	2:54	2:52	2:58	3:11	2:48	2:55	2:52								
2300	1:57	2:15	2:23	2:38	2:33	2:54	2:40	2:54	2:40	2:51	3:7	3:4	3:12	3:12	3:19	3:21	3:22	3:17	3:15	3:9	3:9	3:16						
	2:17	2:10	2:25	2:13	2:27	2:34	2:47	2:41	2:49	2:51	2:48	3:8	3:8	3:20	3:1	3:10	3:16	3:9	3:14	3:3	3:2	3:7						
M	2:14	2:17	2:16	2:21	2:36	2:30	2:46	2:59	2:52	2:53	2:56	3:7	2:60	3:3	3:16	3:20	3:15	3:15	3:14	2:60	3:1	3:15						
a	2:2	2:41	2:20	2:32	2:16	2:39	2:40	2:53	2:48	3:9	2:52	3:1	3:1	3:13	3:7	3:8	3:13	3:8	3:17	3:4	3:17	3:0						
x	2:4	2:25	2:46	2:24	2:39	2:40	2:48	2:36	2:53	2:54	3:0	2:59	2:51	3:5	3:15	3:6	3:13	3:15	3:16	2:57	2:60	3:11						
A 2500	2:31	2:36	2:54	2:53	3:4	3:1	3:7	3:4	3:26	3:20	3:46	3:02	3:48	3:23	3:49	3:51	3:49	3:50	3:26	3:39	3:38	3:19	3:14	3:11				
t	2:14	2:51	2:38	2:60	2:47	2:57	3:20	3:7	3:19	3:16	3:27	3:21	3:22	3:23	3:39	3:50	3:36	3:38	3:31	3:36	3:38	3:33	3:22	3:23				
t	2:30	2:33	2:46	2:35	2:51	3:4	3:6	3:29	3:34	3:17	3:39	3:20	3:46	3:44	3:46	3:47	3:43	3:50	3:41	3:25	3:47	3:20	3:25	3:12				
e	2:38	2:29	2:34	2:42	2:51	3:1	3:7	3:6	3:21	3:28	3:32	3:11	3:28	3:36	3:48	3:45	3:56	3:53	3:25	3:48	3:43	3:31	3:30	3:29				
m	2:30	2:50	2:45	2:45	3:1	3:10	3:8	3:18	3:21	3:22	3:20	3:18	3:39	3:47	3:35	3:43	3:40	3:48	3:23	3:33	3:35	3:15	3:29	3:23				
p 2700	2:22	2:36	2:47	2:46	3:6	3:16	3:13	3:25	3:20	3:33	3:24	3:34	3:55	3:58	3:56	3:45	3:48	4:15	4:8	4:19	3:43	3:36	3:43	3:54	3:32	3:49		
t	2:28	2:47	3:2	3:7	3:7	2:60	3:32	3:21	3:37	3:10	3:44	3:42	3:49	4:16	3:59	3:56	3:54	4:7	4:12	4:11	3:43	3:44	3:26	3:33	3:32	3:45		
e	3:27	2:53	2:29	3:4	3:2	2:55	3:23	3:28	3:33	3:20	3:31	3:38	3:40	3:58	3:57	3:59	3:55	3:55	4:2	4:0	3:45	3:43	3:42	3:38	3:36	3:44		
d	3:57	2:47	2:46	2:44	3:9	3:7	3:5	3:28	3:11	3:47	3:41	3:59	3:32	3:54	3:41	3:48	4:3	3:57	4:13	4:2	3:34	3:50	3:44	3:48	3:44	3:39		
	2:26	2:47	2:54	2:55	3:16	3:17	3:12	3:18	3:39	3:32	3:27	3:27	3:55	3:44	4:5	3:43	3:54	4:7	3:53	3:58	3:37	3:45	3:35	3:40	3:46	3:46		
2900	2:31	2:42	2:30	2:47	2:45	2:55	3:2	2:57	3:15	3:17	3:21	3:40	3:14	3:46	3:47	3:58	3:49	4:17	3:57	4:08	4:4	4:10	3:44	3:50	3:45	3:54	4:2	3:48
	2:25	2:36	2:37	2:55	2:39	2:47	2:55	2:59	2:56	3:31	3:24	3:33	3:20	3:22	4:2	3:59	3:45	3:45	3:43	4:01	4:12	3:55	3:55	4:7	4:7	3:54	3:42	3:56
	2:40	2:30	2:28	2:47	3:3	2:49	2:53	2:54	3:11	3:17	3:8	3:24	3:24	3:24	3:29	3:48	3:54	3:49	4:4	4:08	4:11	4:11	3:44	3:51	3:59	4:15	3:49	4:8
	3:10	2:27	2:44	3:01	2:38	2:60	2:57	2:57	3:15	3:20	3:38	3:23	3:25	3:43	3:53	3:49	3:59	3:57	3:48	3:57	4:2	4:7	3:54	4:8	3:57	3:56	4:1	3:57
	2:20	2:48	2:32	2:59	2:41	2:49	2:59	3:20	3:14	3:20	3:26	3:39	3:33	3:23	3:45	3:54	4:4	4:1	3:55	3:56	4:11	4:9	3:53	4:1	4:0	3:56	3:51	3:59

Table C.6: Number of Feasible Solutions: Problem 2, SA Method I

MaxAccepted	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
1100	22711	24326	25914	28018	29222	29259	32065	30678	30938	30525																			
	22475	26400	28247	26764	29708	27677	29577	28567	29933	30144																			
	24535	24460	26957	26538	29356	28599	29673	27593	31549	28922																			
	24862	26035	26893	30343	28452	30815	29998	29627	30863	30546																			
	22191	25955	27481	28765	29171	29725	29679	32356	30697	30513																			
1300	28339	26500	28846	32903	32148	33711	32961	35311	37438	35782	36480	37423																	
	23528	29964	32236	30618	33593	31682	35398	36624	35037	35942	36719	35905																	
M	26463	28291	29587	34281	31415	35140	36669	36101	35916	36033	35005	37433																	
a	28249	29010	26754	28732	31340	35131	34207	34666	33340	35514	34784	36166																	
x	24946	30298	29339	32748	32652	33884	34076	36507	34375	35491	35105	36874																	
A 1500	30600	33639	30565	31654	35422	35786	38848	38415	39312	42035	39837	40624	39480	43226	43854														
t	27089	31751	31427	31845	34115	37151	34654	42526	39685	41997	41313	40498	41566	43429	40796														
t	28087	31401	32530	33162	37035	35530	39402	38635	44046	43362	41699	40517	42685	41848	40239														
e	29949	32177	34795	29382	37289	35800	38580	39190	39823	40654	41415	40792	41573	39812	42187														
m	29079	32524	30161	36544	36384	36127	39885	41365	39667	38760	40041	40230	43030	43471	43734														
p 1700	29426	32042	37507	38957	40819	39807	40978	42045	46707	47625	46580	49475	42927	47268	47287	46008	46604												
t	28330	35139	34013	42719	40057	39807	41288	43777	41503	44384	46518	46095	46666	47223	46318	48010	45128												
e	40653	33085	34962	38884	35203	43258	41359	40742	46878	49372	48393	47553	46242	47186	45486	47666	48394												
d	33112	31136	38205	36178	38233	40943	39568	44158	43769	45934	45167	47600	48123	45478	47230	51472	48505												
	30125	33715	34096	36774	36415	38314	39041	40024	41775	45944	45140	47184	48449	48855	47822	44448	47725												
1900	30181	39463	38159	36612	41312	42383	46436	42703	51436	48828	45542	50635	52552	50941	53713	54418	53703	53146	52357										
	34885	35016	39054	42886	39713	42163	43796	42689	44134	45061	49178	47741	51005	49699	51647	54269	52448	51171	50463										
	35553	34538	35891	40622	43361	44470	42269	44339	49529	48293	51466	50053	50283	51345	52031	54194	54409	54887	52773										
	34144	34478	36987	40872	38132	45453	45273	47802	46515	46632	50664	50439	49108	53373	51717	52479	56353	52192	53015										
	32919	40004	37609	41195	38085	43089	42138	48148	47148	50244	49193	47918	54360	51241	51530	54307	52696	50150	54900										

Table C.6: Number of Feasible Solutions: Problem 2, Method I, continued

MaxAccepted	300	350	400	450	500	550	600	650	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	1350	1400	1450	1500	1550	1600	1650	1700
2100	33528	40126	36536	43021	44684	44875	45867	48878	52697	50100	56440	55029	53506	58001	54635	54208	57167	59492	57406	59575	59779								
	36341	37273	36224	42161	46601	45392	45957	46861	47968	54399	49602	56816	57210	58192	60627	54357	56448	55380	59804	54689	56982								
	36038	38091	37125	40187	43978	47800	46026	47503	51559	50772	52091	56363	55483	51608	54489	55015	59179	59585	61759	59734	59247								
	32656	53076	43188	42361	43160	44403	46232	50416	46735	52962	53485	54903	54922	55643	54385	56527	61203	59358	59601	57060	59735								
	33806	38267	38106	37573	48616	42621	47486	45805	53713	49695	53611	51222	53383	55991	56366	54721	57263	60939	57499	59751	58790								
2300	36276	40018	41527	45859	49174	46525	55026	51332	55045	50289	54253	57940	57895	60680	60981	64121	61508	64499	62434	62279	64020	64386	66685						
	42882	41644	42028	45270	41971	46194	48849	52246	50855	53917	54574	52986	60475	58702	63209	57429	59682	62345	59647	62069	61867	61587	63155						
M	41134	41861	45047	44709	44354	48805	47058	53322	57351	55515	55016	55682	60026	56397	58708	62023	62485	62204	61701	62306	62346	62168	66499						
a	39676	39524	48879	43907	48590	43098	50282	49710	53709	52630	59317	54038	58093	58480	61318	59361	60116	62015	59590	61916	61628	61333	61117						
x	39871	48059	45550	51352	44917	51338	49586	54120	49501	54468	56099	57792	56570	54312	59135	61762	59679	61676	62025	62467	60070	59479	64388						
A 2500	41069	46340	43434	48996	51657	54138	53408	53227	53351	60952	57524	65894	57156	66223	59400	65214	67868	67308	67565	64736	69882	69936	66819	66125	65863				
t	38634	41582	46490	45424	54422	46847	49509	57637	53973	57052	56110	60627	63732	59199	59849	64494	67312	62757	65085	67085	67383	69145	72088	69584	69202				
t	42480	41745	45286	45928	47987	48328	54130	52081	60665	62349	58311	61824	63341	66405	64929	66702	67181	64601	67617	70069	64746	72243	67240	69589	65674				
e	43630	41257	41620	44010	49494	48595	52245	53078	54762	58805	58792	61177	60716	59389	62274	67214	66955	69685	68279	64735	72710	70038	71511	70978	71493				
m	42106	45558	49697	46267	50312	51856	54517	55255	56481	58798	58360	58893	62843	63948	64468	62566	64941	64966	67330	64747	67560	69377	66588	71667	68280				
p 2700	42148	44407	43614	48857	48294	55676	58123	54277	60546	56339	60810	59102	61709	68074	68732	68196	65748	65911	73953	71113	75970	75277	72792	75201	77535	70929	78634		
t	40935	67171	44301	50083	51957	54847	50405	62025	58292	62914	55602	66037	64742	66933	73336	70020	68364	67916	72731	73813	73250	72624	75140	69296	71357	71216	76371		
e	53365	39983	47091	42525	53890	52198	49532	59807	60806	60732	57628	61313	63938	64457	70169	68356	68613	68405	68809	70855	70793	75703	75293	74431	73774	73469	76566		
d	60970	45350	49417	49831	48396	55458	53289	53263	60145	55207	65785	65147	69797	62240	67903	63256	66162	70717	68438	73662	70440	73030	77969	75005	76679	75926	73332		
	42587	46871	48684	50386	52242	54627	56315	56586	56256	64336	60715	60740	59729	67592	65500	71332	65510	68639	71076	67971	70403	73209	75328	72103	74737	77215	76292		
2900	48211	52554	51707	46747	51193	56260	59105	62482	57889	66005	65629	67925	70778	65254	76346	71538	76717	72113	80586	74253	79664	76525	79146	75264	77432	76711	78928	82072	77310
	48001	49431	50514	54832	56270	53069	54962	60227	59693	58629	70369	68022	66102	66960	68155	76851	76475	70053	71450	70619	76246	79571	75482	78565	83880	83106	79831	75431	78543
	48639	67521	50056	50489	51453	58387	56037	56504	57385	60840	66743	62561	64670	68423	68134	65969	71874	73978	71353	77334	79273	79355	79298	74960	78278	80666	85452	78175	84199
	58247	48376	48174	52636	58618	52399	59074	60124	58321	64475	68236	72480	64416	68505	74002	74144	71244	74644	74534	71564	76472	76581	78641	78707	83707	79482	79847	81822	80983
	46298	48911	52142	50432	55212	55090	57270	60586	63787	64281	67187	69668	69741	71179	68316	71386	73401	77043	76446	73990	75618	79621	78826	78468	80692	80637	79708	79178	81357

Table C.7: Time Data: Problem 1, SA Method II

To =1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	0:28	1:31	2:13	6:25
	0:27	0:49	2:54	4:44
	0:26	1:20	2:13	5:37
	0:48	0:48	2:2	5:27
	0:45	0:59	3:28	5:52
	0:28	1:18	3:10	3:47
	0:28	0:33	3:37	4:13
	0:25	1:18	1:60	8:31
	0:27	0:54	3:9	5:26
	0:50	1:3	2:54	6:9
0.00005	0:23	0:38	2:10	8:9
	0:34	1:25	4:19	5:31
	0:48	0:53	2:13	4:31
	0:25	1:2	3:3	4:28
	0:51	1:32	2:59	4:54
	0:23	0:46	3:15	6:8
	0:28	1:15	3:39	5:40
	0:29	0:57	2:59	3:48
	0:38	1:9	3:16	6:19
	0:24	1:1	2:12	4:60

To =10

	α			
	0.9	0.95	0.98	0.99
0.00050	0:56	1:35	3:2	10:10
	0:37	1:26	4:43	9:18
	0:36	0:57	2:53	6:26
	0:54	1:6	3:4	4:11
	0:43	1:6	3:52	5:51
	1:3	1:8	3:1	7:56
	0:34	1:17	2:53	5:55
	0:53	1:15	4:14	8:27
	0:48	1:55	3:47	7:10
	0:38	1:30	3:37	7:48
0.00005	0:36	1:13	3:35	5:13
	1:7	1:57	4:20	9:23
	0:35	1:30	3:58	6:12
	0:55	2:13	3:51	6:25
	0:36	1:17	3:5	10:8
	0:38	2:17	4:13	7:52
	0:57	1:43	3:19	8:25
	0:59	1:18	3:8	6:47
	0:57	1:11	3:15	7:27
	1:3	2:4	3:4	8:6

Table C.8: Number of Feasible Solutions: Problem 1, SA Method II

To =1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	3724	6764	17328	32984
	2964	6080	16112	24928
	3192	5852	15580	26600
	3572	6384	15504	27208
	3420	7144	15048	30020
	3724	6308	14440	27056
	3192	3724	16492	27740
	2432	6156	14136	37924
	2204	6232	15276	27284
	3572	7068	15732	28956
0.00005	3268	5092	16340	35416
	3040	6080	19836	30172
	3344	6156	17176	26904
	3496	7068	14288	30628
	3800	6916	15200	27588
	3192	5852	15808	30780
	3116	5928	16796	27132
	3800	7600	14364	21964
	2964	5320	16264	31844
	3268	7068	15352	26372

To =10

	α			
	0.9	0.95	0.98	0.99
0.00050	4636	9196	21736	50692
	4484	10564	24700	49780
	4332	6156	21660	45676
	4636	8588	22040	27588
	5168	9044	22420	40964
	5092	8816	22192	42028
	4560	9044	20748	42864
	4864	9424	21812	44612
	3952	9348	20596	43092
	4560	8740	19760	43016
0.00005	4256	9424	22572	37088
	5168	9956	21584	50084
	4180	10260	21204	42408
	4636	10640	22420	36176
	4864	10412	22724	50920
	5092	9120	23864	43168
	4940	9044	23408	45524
	4636	9120	22496	45068
	4636	9120	23940	41268
	5168	10640	21432	42104

Table C.9: Time Data: Problem 2, SA Method II

To = 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	0:06	0:10	0:23	0:42
	0:06	0:09	0:21	0:44
	0:06	0:09	0:22	0:42
	0:06	0:11	0:25	0:44
	0:06	0:08	0:24	0:39
	0:5	0:9	0:20	0:34
	0:5	0:10	0:17	0:40
	0:6	0:9	0:20	0:34
	0:5	0:8	0:20	0:38
	0:2	0:9	0:23	0:40
0.00005	0:06	0:09	0:22	0:42
	0:06	0:07	0:23	0:35
	0:06	0:11	0:21	0:02
	0:06	0:09	0:23	0:46
	0:05	0:10	0:20	0:41
	0:5	0:9	0:21	0:41
	0:5	0:10	0:23	0:35
	0:6	0:10	0:20	0:34
	0:6	0:9	0:19	0:33
	0:5	0:8	0:22	0:39

To = 10

	α			
	0.9	0.95	0.98	0.99
0.00050	0:09	0:18	0:39	1:17
	0:09	0:17	0:37	1:08
	0:09	0:16	0:37	0:17
	0:08	0:16	0:38	1:13
	0:09	0:17	0:40	1:10
	0:8	0:16	0:34	1:2
	0:7	0:16	0:32	1:3
	0:8	0:15	0:37	1:2
	0:8	0:16	0:23	1:6
	0:8	0:3	0:34	1:2
0.00005	0:08	0:15	0:39	1:04
	0:08	0:17	0:39	1:14
	0:09	0:16	0:34	1:11
	0:09	0:16	0:34	1:09
	0:09	0:16	0:35	1:04
	0:8	0:15	0:34	1:1
	0:8	0:15	0:35	1:5
	0:7	0:16	0:35	1:5
	0:8	0:14	0:32	1:5
	0:8	0:15	0:35	1:4

Table C.10: Number of Feasible Solutions: Problem 2, SA Method II

To = 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	1073	1885	4553	8990
	1073	1798	4234	9396
	1131	1798	4582	9483
	1189	2146	5568	9280
	1102	1334	5133	8642
	1131	1943	4669	7801
	1073	2175	3828	9860
	1160	1972	4640	7656
	1160	1769	4553	8932
	145	1769	5655	9251
0.00005	1073	1798	4727	9019
	1160	1363	4843	7453
	1102	2233	4466	145
	1073	1914	4988	9715
	870	1943	4292	8874
	1044	2030	4843	9947
	928	2204	5481	7917
	1160	2117	4524	7830
	1073	2001	4466	7482
	1073	1624	5365	9164

To = 10

	α			
	0.9	0.95	0.98	0.99
0.00050	1711	3509	7540	15225
	1711	3132	7424	13427
	1479	3132	7192	2697
	1537	2871	7540	14413
	1653	3248	8062	13833
	1711	3335	7366	13485
	1421	3335	6989	13746
	1740	3045	8091	13253
	1421	3219	4611	14558
	1508	464	7163	13630
0.00005	1421	3045	7830	12470
	1450	3190	7627	14761
	1624	3074	6554	13891
	1711	3132	6438	13862
	1624	3045	6873	12093
	1624	3248	7337	13688
	1682	3161	7424	14326
	1247	3248	7453	14413
	1479	3074	7018	14268
	1392	3103	7395	13804

Table C.11: Time Data: Problem 3, SA Method II

To = 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	0:12	0:20	0:50	1:47
	0:12	0:25	0:55	1:45
	0:14	0:22	0:56	1:53
	0:11	0:22	0:55	1:48
	0:14	0:19	1:10	1:41
	0:13	0:25	0:56	1:41
	0:11	0:21	0:55	1:39
	0:13	0:21	1:16	1:35
	0:11	0:24	0:51	1:35
	0:1	0:24	0:55	1:58
0.00005	0:13	0:21	0:58	1:05
	0:14	0:22	0:56	1:49
	0:12	0:23	0:56	1:44
	0:12	0:23	0:52	1:52
	0:12	0:23	0:55	1:47
	0:13	0:22	0:55	1:49
	0:11	0:23	0:47	1:44
	0:11	0:22	0:48	1:50
	0:12	0:26	0:52	1:60
	0:12	0:23	0:54	2:22

To = 10

	α			
	0.9	0.95	0.98	0.99
0.00050	0:17	0:19	1:18	2:29
	0:15	0:31	1:13	2:28
	0:11	0:30	1:11	2:29
	0:15	0:33	1:18	0:15
	0:15	0:38	1:29	2:22
	0:15	0:22	1:18	2:35
	0:18	0:32	1:12	2:30
	0:16	0:29	1:13	2:29
	0:15	0:32	1:14	2:49
	0:17	0:28	1:12	2:31
0.00005	0:19	0:30	1:15	2:37
	0:18	0:32	1:17	2:36
	0:17	0:31	1:32	2:23
	0:17	0:33	1:25	2:50
	0:16	0:33	1:17	2:30
	0:15	0:31	1:18	2:23
	0:15	0:28	1:12	2:38
	0:16	0:37	1:13	2:23
	0:18	0:28	1:14	2:37
	0:19	0:34	1:12	2:29

Table C.12: Number of Feasible Solutions: Problem 3, SA Method II

To = 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	3496	5700	14212	31008
	3344	7068	14820	30324
	3572	6156	16188	33516
	2964	6232	15352	30628
	3648	5472	18468	29108
	3724	6916	15732	28044
	3268	6004	15580	28728
	3648	6080	19608	26676
	3496	7220	14364	27512
	380	6460	16112	31236
0.00005	3800	6080	15504	18772
	3800	6460	16036	32072
	3496	6688	15352	28272
	3572	6384	13984	31312
	3344	6688	14668	30628
	3724	5776	15428	30704
	2888	6612	13224	30476
	2964	6080	13148	31920
	3268	6688	14136	31996
	3192	6232	14668	37696

To = 10

	α			
	0.9	0.95	0.98	0.99
0.00050	4788	5624	23104	41344
	4484	8664	22040	44156
	3040	8892	20976	42712
	4560	9272	22344	4636
	4636	10564	24776	40964
	4408	6384	22800	45220
	5168	9500	21812	44536
	4636	8740	21432	44992
	4408	8968	22268	49552
	4636	8132	21584	44156
0.00005	5016	8512	22192	42712
	5092	8968	22040	46132
	4712	8816	25232	41040
	4864	9272	24244	47272
	4484	9424	22952	43092
	4256	9728	23484	41952
	4332	8208	22572	44688
	4712	10260	22648	41116
	5092	8284	22116	47500
	5168	9348	23864	42560

Table C.13: Time Data: Problem 4, SA Method II

To = 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	0:03	0:08	0:23	0:25
	0:02	0:05	0:09	0:43
	0:05	0:09	0:11	0:36
	0:02	0:05	0:22	0:40
	0:05	0:05	0:21	0:17
	0:4	0:9	0:13	0:20
	0:3	0:10	0:14	0:20
	0:6	0:9	0:10	0:19
	0:6	0:5	0:11	0:21
	0:4	0:6	0:19	0:47
0.00005	0:02	0:04	0:11	0:19
	0:06	0:05	0:19	0:30
	0:02	0:05	0:03	0:22
	0:03	0:04	0:08	0:41
	0:03	0:05	0:10	0:35
	0:2	0:4	0:11	0:41
	0:3	0:4	0:10	0:34
	0:3	0:4	0:10	0:40
	0:3	0:13	0:9	0:22
	0:3	0:4	0:20	0:33

To = 10

	α			
	0.9	0.95	0.98	0.99
0.00050	0:03	0:07	0:19	0:34
	0:04	0:08	0:26	0:22
	0:04	0:07	0:26	0:29
	0:04	0:08	0:30	0:34
	0:04	0:08	0:18	0:39
	0:4	0:8	0:30	0:53
	0:3	0:8	0:18	0:51
	0:4	0:13	0:17	0:35
	0:3	0:11	0:17	0:52
	0:3	0:8	0:18	0:33
0.00005	0:03	0:13	0:18	0:35
	0:04	0:11	0:16	0:35
	0:07	0:07	0:24	0:39
	0:06	0:07	0:27	0:47
	0:03	0:09	0:16	0:56
	0:3	0:7	0:20	0:33
	0:7	0:13	0:25	1:1
	0:4	0:6	0:15	0:56
	0:4	0:7	0:30	0:36
	0:7	0:13	0:18	0:57

Table C.14: Number of Feasible Solutions: Problem 4, SA Method II

To = 1.5

	α			
	0.9	0.95	0.98	0.99
0.00050	1189	2204	5220	10237
	928	2320	4843	10324
	1044	1798	5858	8816
	986	2233	4930	10614
	1102	2523	5626	8874
	1160	2436	5742	8932
	1421	2494	5539	10498
	1334	2320	4611	9135
	1247	2436	5771	10324
	986	1711	4698	10237
0.00005	1189	2320	5133	10179
	1102	2262	5220	6554
	1189	2320	841	10005
	1421	2320	4263	9512
	1421	2233	5481	8642
	1160	2059	5568	9802
	1189	2262	5017	8613
	1189	2291	2088	9222
	1450	2320	4814	10295
	1276	2291	4582	6641

To = 10

	α			
	0.9	0.95	0.98	0.99
0.00050	1595	3538	7685	15660
	1914	3451	7743	8120
	1740	3364	7772	11803
	1682	3567	8062	12934
	1711	2262	8294	15167
	1711	3306	8062	15573
	1537	3103	8265	15225
	1827	2958	8062	15399
	1624	3016	7743	15805
	1508	3161	8178	14964
0.00005	1595	3422	7453	15051
	1653	2726	772	10092
	1711	3190	7163	15660
	1827	3132	7772	14036
	1740	3596	6815	14993
	1624	3480	8178	16037
	1624	2958	7627	17139
	1798	3045	7134	15892
	1769	3190	8120	15370
	1711	3625	8178	15283

Table C.15: Time Data for the Four Large Problems Using SA Method I with a MaxAccepted:MaxAttempted Ratio of 40%

Problem 5
MaxAttempted = 75000
MaxAccepted = 30000
1888:13
1763:04
2002:56

Problem 7
MaxAttempted = 35000
MaxAccepted = 14000
640:59
634:06
644:06

Problem 6
MaxAttempted = 75000
MaxAccepted = 30000
1688:13
1492:59
1185:51

Problem 8
MaxAttempted = 50000
MaxAccepted = 20000
469:04
484:19
480:43

Table C.16: Time Data for the Four Large Problems Using SA Method I with a MaxAccepted:MaxAttempted Ratio of 50%

Problem 5
MaxAttempted = 75000
MaxAccepted = 37500
1605:19
1790:33
1673:40

Problem 7
MaxAttempted = 35000
MaxAccepted = 17500
679:07
685:09
710:19

Problem 6
MaxAttempted = 75000
MaxAccepted = 35000
1439:10
1399:56
1428:04

Problem 8
MaxAttempted = 50000
MaxAccepted = 25000
545:40
635:58
558:42

Table C.17: Number of Feasible Solutions for the Four Large Problems Using SA Method I with a MaxAccepted to MaxAttempted Ratio of 40%.

Problem 5
MaxAttempted = 75000
MaxAccepted = 30000
3872783
2025212
2008550

Problem 7
MaxAttempted = 35000
MaxAccepted = 14000
967596
959277
958534

Problem 6
MaxAttempted = 75000
MaxAccepted = 30000
2590924
2384293
2005067

Problem 8
MaxAttempted = 50000
MaxAccepted = 20000
1244432
1281188
1279310

Table C.18: Number of Feasible Solutions for the Four Large Problems Using SA Method I with a MaxAccepted to MaxAttempted Ratio of 50%.

Problem 5
MaxAttempted = 75000
MaxAccepted = 37500
2041002
2270122
2103910

Problem 7
MaxAttempted = 35000
MaxAccepted = 17500
989503
988449
1023143

Problem 6
MaxAttempted = 75000
MaxAccepted = 35000
2357959
2628580
2288014

Problem 8
MaxAttempted = 50000
MaxAccepted = 25000
1374226
1392604
1354656

Table C.19: Time Data for the Four Large Problems Using SA Method II

Problem 5

	To: 1.5	To: 10
0.00050	445:30	412:42
	412:38	391:41
	265:27	488:11
	344:54	411:11
	317:50	331:26
0.00005	386.49	593:1
	545:25	461:13
	538:21	506:58
	294:18	391:35
	490:55	508:17

Problem 7

	To: 1.5	To: 10
0.00050	86:15	104:16
	77:21	89:57
	75.42	115:56
	92:42	115:42
	84:58	103:44
0.00005	73:40	115:53
	83:18	117:35
	75:53	104:33
	97:40	95:24
	85:13	109:36

Problem 6

	To: 1.5	To: 10
0.00050	235:48	243:50
	223:32	106:4
	191:22	299:59
	224:58	287:56
	219:16	281:25
0.00005	263:3	341:59
	283:2	389:06
	298:35	457:42
	306:7	348:48
	282:35	392:55

Problem 8

	To: 1.5	To: 10
0.00050	40:20	53:54
	41:19	41:32
	40:23	63:04
	45:15	65:45
	45:28	66:35
0.00005	44:26	61:11
	41:56	53:57
	47:42	62:13
	52:33	68:27
	51:59	69:15

Table C.20: Number of Feasible Solutions for the Four Large Problems Using SA Method II

Problem 5

	To: 1.5	To: 10
0.00050	410600	484800
	379200	508800
	263200	556800
	353600	511200
	338400	428800
0.00005	355200	636800
	488000	568800
	488000	559200
	340800	491200
	438400	616000

Problem 7

	To: 1.5	To: 10
0.00050	128250	146925
	119025	134100
	116325	170775
	122175	170775
	110250	153000
0.00005	112050	172125
	122400	173025
	113850	159075
	128475	141300
	120375	162450

Problem 6

	To: 1.5	To: 10
0.00050	345600	416800
	328800	215200
	295200	493600
	352800	488000
	334400	480800
0.00005	401600	559200
	404000	583200
	432000	619200
	468000	564800
	429600	621600

Problem 8

	To: 1.5	To: 10
0.00050	105075	139275
	110250	104175
	105750	159750
	104625	151875
	108225	150975
0.00005	115425	161100
	109125	135675
	125550	162000
	123300	157725
	125100	158625

The following are the station assignments corresponding to the best solutions found by Simulated Annealing.

Problem 5

Station 1: 5
Station 2: 2
Station 3: 3, 7
Station 4: 4, 10
Station 5: 9, 13
Station 6: 1, 6, 8
Station 7: 12, 17
Station 8: 14, 16
Station 9: 20, 22, 23
Station 10: 18, 24
Station 11: 25, 27
Station 12: 21, 26, 28
Station 13: 11, 29
Station 14: 30
Station 15: 32
Station 16: 31, 34
Station 17: 15, 33
Station 18: 35, 36
Station 19: 37, 38
Station 20: 19, 39
Station 21: 40, 41
Station 22: 42
Station 23: 43, 44
Station 24: 45
Station 25: 46
Station 26: 47, 48
Station 27: 49, 50

Delta: 3517.35

Problem 6

Station 1: 3, 6
Station 2: 2
Station 3: 5, 12
Station 4: 23, 4
Station 5: 25
Station 6: 10, 29
Station 7: 8, 9
Station 8: 16
Station 9: 1, 22
Station 10: 7, 33
Station 11: 21, 37
Station 12: 24, 28
Station 13: 30
Station 14: 11, 26
Station 15: 13, 31
Station 16: 17
Station 17: 32
Station 18: 34, 35
Station 19: 15, 39
Station 20: 36, 41
Station 21: 38, 40
Station 22: 27, 42
Station 23: 43, 44
Station 24: 46
Station 25: 19, 49
Station 26: 45
Station 27: 47, 48

Delta = 3220.59

Problem 7

Station 1: 3, 4, 5, 8, 12
Station 2: 1, 7, 9, 11, 16
Station 3: 2, 6, 10, 13, 20
Station 4: 14, 18, 22, 23, 24, 25
Station 5: 17, 21, 26, 27, 28, 29
Station 6: 15, 30, 31, 33, 35
Station 7: 19, 32, 37, 39
Station 8: 34, 36, 38, 40, 41
Station 9: 42, 43, 44, 45
Station 10: 46, 47, 48, 49, 50

Delta: 1130.23

Problem 8

Station 1: 3, 4, 7, 9, 12, 13
Station 2: 2, 5, 6, 8
Station 3: 1, 11, 15, 16, 20
Station 4: 10, 14, 23, 25, 29
Station 5: 17, 18, 21, 28, 33, 37
Station 6: 19, 22, 30, 40
Station 7: 24, 32, 42
Station 8: 26, 31, 34, 35, 36
Station 9: 27, 38, 43, 45, 47, 48
Station 10: 39, 41, 44, 46, 49, 50

Delta: 723.59

VITA

Sherry L. Edwards was born on July 19, 1967 to William J. and Jean H. Edwards, who currently reside in Richmond, Virginia. Sherry graduated with honors from John Randolph Tucker High School in Richmond, Virginia in 1985. She then obtained a Bachelor of Science degree in Systems Engineering from University of Virginia in 1989. While at University of Virginia, Sherry was selected as a member of Tau Beta Pi, the national engineering honor society. Following her undergraduate education, Sherry worked for two years as a defense consultant with Analytic Services, Incorporated. She left this position to pursue a Master of Science degree in Industrial and Systems Engineering, with a concentration in Manufacturing Systems Engineering, at Virginia Polytechnic Institute and State University. Upon graduation, Sherry will begin work for General Electric Drive Systems in Salem, Virginia.

Sherry L. Edwards

Sherry L. Edwards