SIMULATED ANNEALING
APPLIED TO
A FILE ALLOCATION PROBLEM

By

Monica Frederica Hess

B. Sc. (Mathematics) University of Guelph 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

MATHEMATICS

We accept this thesis as conforming

to the required standard

.....

.....

.....

.....

.....

.....

THE UNIVERSITY OF BRITISH COLUMBIA

October 1992

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Mathematics
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: October 15, 1992

Abstract

The simulated annealing is applied to an instance of the file allocation problem. The implementation of the algorithm is demonstrated using this example.

A study of the effects of using different penalty functions on this problem is performed. The results indicate that the choice of penalty function can have a significant impact on the performance of the algorithm. In particular, I conclude that the effect of a poor choice of penalty function can not be remedied by slower cooling.

Table of Contents

List of Tables

List of Figures

Acknowledgement

The generous help and encouragement of friends has been invaluable during the writing of this thesis and is much appreciated. Special thanks to Anthony Sayers for his valuable support, patience and for the early morning breakfast. In appreciation of their generosity and support, I dedicate this thesis to my parents Fred and Alida Hess.

Chapter 1

Introduction

The simulated annealing algorithm was first described by Kirkpatrick *et al.* (1983) [26] [27]. This general purpose algorithm has been used effectively in many difficult combinatorial optimization problems, especially in computer aided design. Although tailored heuristic algorithms have been shown to outperform the simulated annealing algorithm on many well-studied problems, the algorithm can be an effective alternative when no tailored algorithm is available.

Simulated annealing has the advantage of being relatively easy to implement, but the disadvantage of requiring much computer time. As such, the algorithm is not an ideal approach for all combinatorial problems. However, because of its adaptability, it remains a useful tool for finding high-quality solutions to many combinatorial problems without requiring the construction of complex tailored algorithms.

The primary goal of this thesis is to present an effective demonstration of the implementation of the simulated annealing algorithm. An outline of the procedures involved in implementing the algorithm to a particular problem is detailed by example.

For demonstration purposes an instance of the file allocation problem is used. The file allocation problem was originally studied by Chu *et al.* (1969) [9] and has been studied with many variations since then. The particular problem studied here is a standard multiple file allocation problem with the addition of storage capacity constraints. The addition of storage constraints increases the difficulty of the problem by not allowing a reduction into multiple single-file problems. This additional complexity makes the

problem an interesting candidate for solution by simulated annealing.

After presenting the problem in a simulated annealing format, further steps in the implementation are discussed with special attention is given to the annealing schedule.

The implementation of the storage constraints into the simulated annealing format involves what is known as a penalty function. The penalty function incorporates constraints into the objective function rather than requiring that the constraints be imposed explicitly. Variations on the penalty function have been used in earlier research of simulated annealing. Here a comparison of three different penalty functions is undertaken, and some interesting results are obtained. Observations conclude that the choice of penalty function can affect the performance of the algorithm significantly, so that even resorting to slower cooling cannot overcome the effect of a poor penalty function.

A complete introduction to the simulated annealing algorithm and previous research is presented in Chapter 2.

The file allocation problem is presented in chapter 3, with specific attention to the particular instance being studied. The steps necessary to adapt the problem to the simulated annealing format are demonstrated.

Details of the implementation are discussed in Chapter 4. In particular the annealing schedule is outlined and the various penalty functions being compared are given.

Chapter 5 begins by specifying further details of the implementation of the algorithm for this problem instance. A report and comparison of the solutions obtained follows, including an analysis of the annealing curves produced using each penalty function. The chapter ends with an evaluation of the process and discussion of the results.

A summary of the major observations of the research, and suggestions for further research appear in the final chapter, Chapter 6.

Chapter 2

Simulated Annealing

2.1 Combinatorial Optimization, NP-Completeness and Heuristics

An optimization problem consists of a set of instances, D, and aims to minimize a function, called the cost function, over a solution set. Specifically, for every instance of the problem the following must be well defined:

1. for each instance $I \in D$, a finite set, S, of feasible solutions called the set of feasible solutions or the solution set,

2. a function $c(x)$ called the cost function, that assigns a real value to each solution $x \in S$.

For a minimization problem a solution $x^\star \in S$ is a (globally) optimal solution if

$$c(x^\star) \leq c(x) \text{ for all } x \in S.$$

A problem instance is considered solved when an optimal solution, $x^\star$, and its associated cost value, $c(x^\star)$, have been determined.

Combinatorial optimization attempts to find the best or optimal solution to instances of problems involving discrete variables. Since the variables involved are discrete as opposed to continuous, the solution set contains a finite number of (or possibly countably infinite) elements or combinations, whereas the solution set for problems with continuous variables is uncountable. The different nature of the solution spaces in problems with discrete variables has a significant impact on the way the solution space might be searched.

This has resulted in a divergence in the approaches to finding solutions. The usual initial approach to combinatorial problems is to look for a clever search of the solution space which will reduce the number of feasible solutions that need to be evaluated.

In the study of a combinatorial problem the aim is to find the most efficient algorithm for solving any problem instance. The efficiency of the algorithm is a measure of the computer resources required for the algorithm to find the optimal solution. For an algorithm, the *time complexity function* describes the greatest possible time requirements for a given length of input, or *size* of the instance $|I|$. Combinatorial problems are classified according to their difficulty by distinguishing between polynomial and superpolynomial (and nonpolynomial) time requirements [14].

The class **NP** is the primary class of combinatorial problems. Membership in this class is determined by analysis of the *decision problem* or recognition version of a problem. In an instance of a minimization problem which can be stated as, "find the solution $x^\star \in S$ such that $c(x^\star) \leq c(x)$ for all $x \in S$," the associated decision problem is, "for a given number B **is there** an $x \in S$ such that $c(x) \leq B$." Thus, there is only a "yes" or "no" response to any decision problem. The class NP consists of all those problems for which a "yes" instance of the decision problem has a concise certificate, i.e. of length bounded by a polynomial in $|I|$, which can be checked in polynomial time for its validity [43].

Problems for which a polynomial time algorithm is known (ie. running in time L with $L \leq p(|I|)$ where p is a polynomial function) make up the class **P** of combinatorial problems. Being polynomially solvable makes any "yes" instance of the decision problem polynomially certifiable and so P $\subseteq$ NP. There are, however, problems in NP for which no polynomial algorithm has yet been found. Since these problems are numerous and remain without a polynomial time algorithm after a great deal of effort, the current wisdom is that these problems must be distinct from P and so P $\subset$ NP.

The class **NP-complete** consists of a large number of problems for which the existence of a polynomial time algorithm is unlikely. Membership in this class requires that a problem be in NP and that every NP-complete problem known must polynomially reduce to the problem at hand. Given this, if a polynomial time algorithm were to be found for an NP-complete problem, one would exist for every other NP-complete problem. NP-complete problems[1] include the Traveling Salesman Problem, Satisfiability and the Maximum Clique Problem [43].

Membership in the class NP-complete implies that the existence of a particularly efficient algorithm is unlikely, and effort in trying to find a polynomial time algorithm is not wisely spent. Unfortunately, the great majority of practical combinatorial problems are NP-complete [14]. Thus, when a problem is shown to be NP-complete, effort is might be spent in developing an algorithm which will return a near optimal solution running in polynomial or bounded time. Approximating algorithms which at least run fast in practice or algorithms which solve most instances exactly and fairly fast are alternative approaches. These algorithms are termed *heuristics*. The aim of heuristics, then, is to find a near optimal solution within the available computing time and using available resources.

One commonly used heuristic approach is iterative improvement or "local search" algorithm. This algorithms states: given a current solution, x_i, generate a new solution, x_j according to a prescribed generation mechanism and accept the new solution only if the new solution has lower cost (i.e. if $c(x_j) < c(x_i)$). The *generation mechanism* is a set of moves and defines a *neighbourhood*, N_i, for each solution, x_i, as the set of all solutions which can be reached from the current one in one move of the generation mechanism. The algorithm terminates when no solution in the neighbourhood of the current solution has

[1] For further information regarding classification of combinatorial problems and NP-completeness the interested reader is directed to [14] .

lower cost. Because of this the algorithm is guaranteed to terminate at a local minimum, but not necessarily a global one.

Iterative improvement algorithms can range from tailored to general in their implementation, depending on the types of moves used in the generation scheme. Tailored versions of the algorithm employ complex generation strategies which are more likely to result in a decrease in cost. These strategies take into account special features of the problem, and in this way allow for a shorter search on average. The most general form of the algorithm on the other hand depends on moves which are essentially randomly generated and do not take into account specific features of the problem being considered.

General algorithms remain theoretically sound due to their robustness, but in practice have drawbacks. These include weakness of the performance guarantees and a lack of useful bounds on the speed of the algorithm. Besides the tailoring of general algorithms to specific problems, another approach is the creation of *special purpose algorithms* developed solely for a particular class of problems [41].

2.2 Simulated Annealing from Statistical Mechanics

With new problems and variations on old problems constantly appearing, many of which are NP-complete, new heuristic approaches are continually being sought. Lin in [34] discussed using heuristic techniques for solving combinatorial problems and proposed guidelines for developing effective heuristics based on previously successful methods.

Analogy to a natural physical system of thermodynamics was used by Kirkpatrick Gelatt and Vecchi [27] to derive the simulated annealing heuristic. The annealing of solids which could be simulated by the algorithm of Metropolis *et al.* [37] was the basis of this new heuristic. The combinatorial problems this algorithm was first tested on were wire routing and component placement in VLSI design.

Annealing is the high temperature melting and subsequent slow cooling of matter, used in order to achieve the most stable form of the substance. The cooling must be slow enough so that the atoms can achieve rearrangements into highly stable, low energy configurations. This stable form will generally take on a crystalline structure as opposed to a less stable glass, which is achieved through quick cooling or quenching [26]. This most stable, low energy configuration of the atoms, is considered the *ground state* of the solid.

By describing the solid in its possible states in terms of the configurations of atoms, we can establish the analogy with combinatorial problems. The goal in annealing is to find the *lowest energy* configuration of the atoms, while the goal in combinatorial optimization is the find the lowest cost solution. Thus the cost function, which is to be minimized in the combinatorial problem, can take the place of the energy function while the set of feasible solutions takes the place of the set of possible configurations of the atoms. The temperature of the system has no analog in the combinatorial problem and becomes a control parameter in the process.

An algorithm to mimic the behavior of a thermodynamic system was developed by Metropolis *et al.* [37] using computer simulation. A system, such as atoms in a fluid, is known to have a Boltzmann distribution when at thermal equilibrium at temperature T. Starting with a random configuration of atoms i, the Metropolis algorithm simulates the movement of atoms by repeatedly generating small perturbations of the current configuration changes to a new configuration, j. The new configuration is accepted or rejected according to the Metropolis criterion:

If $\Delta E \leq 0$ then accept the move;

If $\Delta E > 0$ then accept the move with probability $P(\Delta E) = \exp(-\Delta E/kT)$;

where $\Delta E = E(j) - E(i)$ is the change in energy function, T is the temperature and k is

Boltzmann's constant. This simulation determines equilibrium of the system at a finite temperature by attempting a large number, M, of moves and hence generating a Markov chain of M events. Suppose the moves are generated by a scheme such that

$$\forall i, j \in R, G_{ij} = G_{ji} \tag{2.1}$$

where R is the space of possible configurations and G_{ij} is the probability of generating state j when in state i. With this restriction on move generation and using the Metropolis criterion the law of detailed balance is satisfied. Under these conditions it can be shown that as $t \to \infty$, the probability that the system is in a given state i, regardless of starting configuration, is the Boltzmann distribution [16].

To implement the algorithm for combinatorial problems each solution x_i is represented by a configuration i and the Metropolis criterion is applied to the cost function, $c(i)$, in place of the energy function, $E(i)$, and

$$\Delta c = c(j) - c(i) \tag{2.2}$$

in place of ΔE. For simplicity the Boltzmann constant is set to one. The annealing is achieved by repeated application of the simulation at a series of decreasing temperatures. The probabilistic aspect of the algorithm is easily implemented by comparing the value $P(\Delta c)$ to a (pseudo) random number drawn from a uniform distribution of the $(0, 1]$ interval. The process must rest long enough at each temperature to reach equilibrium for that temperature. When the temperature has reduced far enough so that no further changes in the configuration of the system are possible (or likely) the system is said to be *frozen*. The configuration at this point is taken to be a good approximation to the optimal solution. The resulting algorithm is shown in Figure 2.1.

In order to be able to apply simulated annealing to a combinatorial optimization problem, it must be possible to define the following [52]:

```
procedure SimulatedAnnealing;
begin { The Simulated Annealing Algorithm }
        i := i_0; { initial solution }
        T := T_0; { initial temperature }
        repeat
                repeat
                        j = generate(i);
                        Δc := c(j) − c(i);
                        if Δc <= 0 then accept := true
                        else
                                if exp(−Δc/T) > random[0,1) then accept := true;
                        if accept then i := j
                until "equilibrium criterion is met";
                T := decrement(T)
        until "the system is frozen";
end;
```

Figure 2.1: The Simulated Annealing Algorithm.

- a concise representation of feasible solutions as configurations,
- a cost function which can be evaluated for any configuration, and
- a generation mechanism for perturbing the present configuration.

Similarities between the simulated annealing algorithm and the iterative improvement algorithm are many. Both algorithms are based on evaluating possible moves from one feasible solution to another, where the moves are defined by a user declared generation mechanism. In both algorithms the generation mechanism will define a neighbourhood of feasible solutions N_i of the current solution i. The iterative improvement algorithm generates new states and accepts only those which correspond to a lowering of the cost function. From the viewpoint of statistical mechanics this is like the rapid freezing or *quenching* of material from a high relative temperature to low temperature. The result of this strategy is that the algorithm will terminate at the first local minimum that it

encounters. Simulated annealing can be thought of as an iterative improvement with a prescribed and controlled method for accepting increases in cost. This "hill climbing" ability means that if the algorithm comes across a local minimum there is a positive probability of it managing to move past, towards the global minimum.

2.3 Probabilistic Hill Climbing and Other Terminology

Simulated annealing algorithms can be thought of as a relaxation of the iterative improvement algorithm. The nature of this relaxation is specifically controlled by the analogy to statistical physics which motivated the algorithm.

Analysis of the algorithm must take place at the level of transitions from one configuration to another. The action can be represented in terms of transition probabilities $P_{ij}(T)$, from configuration i to j. In order for a transition from configuration i to j to take place the configuration j must first be generated and then accepted based on the criterion of the algorithm. Thus, it is possible to define the matrix of transition probabilities in terms of generation and acceptance probabilities.

The generation matrix $G(T)$ contains the probabilities $G_{ij}(T)$ of producing configuration j by perturbation from configuration i. In the original form of the algorithm [27] these probabilities are given by a uniform distribution over all configurations in N_i, that is,

$$G_{ij}(T) = \begin{cases} 1/|N_i| & \forall j \in N_i \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

The acceptance matrix $A(T)$ contains the probability $A_{ij}(T)$ of accepting configuration j once it has been generated from i. In the formulation of the algorithm in Figure 2.1 the values of matrix $A(T)$ are given by the Metropolis criterion. The transition probability matrix $P(T)$ containing the probabilities of moving to configuration j from present

```
procedure PHC;
begin { The Probabilistic Hill Climbing Algorithm }
        i := i_0; { initial solution }
        T := T_0; { initial temperature parameter }
        repeat
                repeat
                        j := generate(i);
                        if accept(c(j), c(i), T) then i := j
                until "inner loop criterion" is satisfied;
                T := decrement(T);
        until "terminating criterion" is satisfied;
end;

procedure accept;
begin { The accept procedure }
        Δc = c(j) − c(i);
        if f(Δc, T) > random[0, 1)
                then accept := true else accept := false
end;
```

Figure 2.2: The Probabilistic Hill Climbing Algorithm

configuration i is given by:

$$P_{ij}(T) = \begin{cases} G_{ij}(T)A_{ij}(T) & \forall j \neq i \\ 1 - \Sigma_{l=1, l\neq i}^{|N_i|} G_{ij}(T)A_{ij}(T) & j = i \end{cases} \tag{2.4}$$

The same basic algorithm with different generation and acceptance matrices has been considered by a number of authors. Probabilistic hill climbing was described as a general class of algorithms possessing the statistical properties of simulated annealing by Romeo and Sangiovanni-Vincentelli [47]. This generalized version of the simulated annealing algorithm is shown in Figure 2.2. The primary difference between these versions of the algorithm is the generality of the acceptance function.

Based on this, the simulated annealing algorithms form a subset of the probabilistic hill climbing algorithms. Experimental results in [47] led the authors to conclude that

"the use of different acceptance functions does not alter significantly the quality of the solution obtained." For this reason, Sechen and Sangiovanni-Vincentelli in developing the TimberWolf Placement and Routing Package at Berkley [50] used the acceptance function based on the Metropolis criterion.

Nahar *et al.* describe what they call the class of general adaptive heuristics [40]. This class of heuristic algorithms are general in the sense that they can be applied to a wide range of problems, and adaptive in the sense that parameter(s) may be modified in the algorithm. The set of parameters is not specified suggesting that it may include more than one. Neither is the **accept** function specified. Simulated annealing and probabilistic hill climbing algorithms are subsets of this class of algorithms.

The "adapting" of the parameters in this algorithm can be done by the user or the algorithm itself. Information that has been gained up to some point can be applied to the next stage of the search, reflecting a learning process.

The adapting of parameters has been adopted in simulated annealing algorithms in making decisions regarding the temperature parameter and the length of the Markov chains generated [55]. Note though, that adapting of parameters has been used in other algorithms which would not fit into the format given by Nahar *et al.* [17]. Given that the term "general" can be applied to many algorithms as well, the title "general adaptive heuristic" appears uninformative or even confusing and should be used with caution.

As with simulated annealing, a problem can be adapted to the probabilistic hill climbing or "general adaptive" algorithms if the cost function, configuration space and generation mechanism can be defined for the problem. All three algorithms maintain one solution at any time and can be considered relaxations of the iterative improvement algorithm[2]. I propose that any algorithm which fits into the probabilistic hill climbing

[2]This is in contrast with the algorithms such as the genetic algorithm [21] which maintains a whole set of solutions at all times.

format with a single control parameter and has an acceptance function based on the Metropolis criterion, should be referred to as simulated annealing, since the analogy to statistical physics holds when the Metropolis criterion is used. The algorithm used here fits into this category of simulated annealing algorithms.

2.4 Markov Chains and Convergence

A Markov chain is a sequence of trials, where the outcome of each trial depends only on the outcome of the previous one [52]. The transition probability matrix as defined in Equation 2.4 describes the transitions between states at one temperature or stage of the annealing procedure and, since this matrix is dependent only on the state which the system is presently in, the resulting sequence is a Markov chain. A Markov chain generated at a constant value of the control parameter T is called homogeneous. Thus, the overall algorithm can be described as a series of homogeneous Markov chains generated at decreasing values of the control parameter T. A variation of this algorithm is described by a single inhomogenous Markov chain if T is reduced at every new configuration, [52]. In practice the homogeneous version is the easiest to implement and most commonly used. I have used a homogeneous algorithm but will describe some convergence results for the inhomogeneous algorithm as well.

Using the model of a series of homogeneous Markov chains, Romeo and Sangiovanni-Vincentelli show that under certain conditions the probabilistic hill climbing algorithms will generate a globally optimal solution with probability one [47]. Requirements are placed on the rules used by the algorithm to generate and accept (or not) the new configurations and on the time spent at each temperature. The convergence is asymptotic in that it requires that an infinite number of iterations be performed at each temperature T. For the inhomogeneous algorithm the convergence has been shown to be asymptotic

as well [52]. In this case the temperature approaches zero infinitely slowly.

It is obvious that in any practical implementation of the algorithm the length of the run must not only be finite but short enough to run in the computer time given to this purpose. In this case the convergence guarantee can no longer hold.

Though the convergence guarantee is asymptotic, the theoretical results give some indication of important restrictions on the generation and acceptance functions. As well, it points out the interaction between the decrement function and the "inner loop criterion" when comparing the convergence of the homogeneous and inhomogeneous algorithms. In [50] Sechen and Sangiovanni-Vincentelli point out that,

> [The convergence results] serve to give confidence in the well posedness of the algorithm and to provide some insight on the reasons why simulated annealing has performed well in practical cases.

Properties of the (homogeneous) probabilistic hill climbing algorithm giving asymptotic convergence are based on the analysis of the process as a series of Markov chains. The individual Markov chains must be convergent to a stationary distribution. That is, the stationary distribution must exist and the process must converge to it [52]. Existence of the stationary distribution requires that the Markov chain be irreducible and aperiodic. A sufficient condition for irreducibility is that the solution space be strongly connected with respect to the chosen generation rule. Also, the acceptance rule must not destroy this connectivity by assigning a probability of zero to any perturbations. The Metropolis rule as acceptance function satisfies this property, so if a simulated annealing algorithm is used, one only needs to verify that the generation rule satisfies the connectivity requirement. The Markov chain will be aperiodic if the acceptance function, f, is such that there exists at least one pair of states i and j for which

$$0 < f(\Delta c_{ij}, T) < 1 \text{ for all } T > 0 \; . \tag{2.5}$$

Once again this condition is always satisfied by the Metropolis rule of the simulated annealing algorithm [47]. The convergence to the stationary distribution is ensured if the generation and acceptance matrices satisfy the global balance requirement [47]. Once again, when the acceptance function is based on the Metropolis criterion this simplifies greatly. It is sufficient if the generation matrix does not depend on T and satisfies the relation given by Equation 2.1.

Nahar, Sahni and Shragowitz point out [40], the convergence proof does not imply that the algorithm will even perform well when finite length runs are used. There is also no way of knowing when the optimal solution has been found if the algorithm should come across it. They state, "Whether a heuristic is good or not depends on its performance relative to other heuristics for the same problem and not on whether a convergence proof can be provided."

The simulated annealing algorithm also does not have a *performance guarantee.* That is, a bound on the distance between the true optimal solution and the best solution found by the algorithm. Such bounds can be useful and are considered an important aspect of heuristic algorithms. In practice there are instances where this bound has limited importance since it often does not reflect the performance of the algorithm when implemented. Since the bounds are worst-case bounds there are many instances where algorithms perform better in practice than their performance guarantees would suggest [14]. This is true for simulated annealing; it is an algorithm with no performance guarantee that has been used successfully for a range of practical problems.

In the practical implementation of the algorithm the theoretical results are an unobtainable ideal. The properties of the generation and acceptance matrices noted are, however, not difficult to include in an implementation of the sort done here. We shall see in the next section how controls and monitoring of the annealing process has also been influenced by the results discussed. The best implementation strategies, potential of the

algorithm, and problem suitability remain to be determined through testing.

2.5 Annealing Schedule

At the stage of implementing the algorithm decisions must be made about how the system will be allowed to evolve. These include how long the system will remain at each temperature and how the temperature will be decreased. Kirkpatrick *et al.* [27] define the *annealing schedule* as the "sequence of temperatures and the number of rearrangements of the $\{i\}$ attempted to reach equilibrium at each temperature." By specifying the sequence of temperatures as part of the annealing schedule the initial and final temperatures are implicitly included although these values were found primarily by trial and error in Kirkpatrick's initial implementation. White [55] and others have explicitly included the criterion for the initial temperature T_0, termed the *hot condition*, and the stopping temperature or *frozen condition*, as part of the annealing schedule.

The importance of the annealing schedule to the heuristic is assumed, based on the motivating analogy with statistical physics. The process of cooling a solid slowly to form low energy crystals requires the system be completely melted to start, and then be kept close to thermal equilibrium while the temperature is reduced, until it is deemed to have become frozen. Otherwise, higher energy configurations can become "locked into" the solid. This will result in the finding of local minima as occurs with the iterative improvement algorithm.

At the initial stage of the annealing process the system must be made "hot enough" that it is completely "melted." This corresponds to the configuration space being "randomized" in the sense that virtually every new configuration considered is accepted, so that the search moves randomly throughout the configuration space. The hot condition specifies the criterion used to determine when the initial temperature T_0 is high enough

that randomization will occur.

The stage between the melting of the system and when the system is determined to have become frozen shall be referred to as the *cooling stage.* This is the stage in which the algorithm spends the most time. In order to remain sufficiently close to thermal equilibrium there are two things the user can adjust. The first is to choose the *temperature decrement function* so that temperature is reduced by small decrements. Secondly, the *equilibrium criterion* specifies the length of time spent at each stage and can be lengthened so that the system has enough time to evolve into equilibrium. This reflects a physical interpretation of the asymptotic convergence result discussed earlier. Letting the system stay infinitely long at each temperature, as necessary for the asymptotic convergence of the homogeneous algorithm, ensures that the system will achieve equilibrium. Reducing the temperature by asymptotically small decrements, as necessary for the convergence of the inhomogeneous algorithm, ensures that the system can never stray far from equilibrium. In practice, a balance between these two approaches is usually relied upon during the cooling stage.

The frozen condition or *stopping criterion* indicates when the algorithm should be terminated. The system is said to be "frozen" when no further changes are possible or likely. Thus, the stopping criterion only needs to reflect that the likelihood of achieving further improvements by continuing is very small.

Therefore we see there are three stages and four associated criterion that determine the annealing schedule. The annealing schedule is sometimes added to the list of three items described in section 2.2 as "necessary to adapt simulated annealing to a combinatorial problem" [27]. I have separated the annealing schedule from this list since those three items fit into a more general category of representing the problem in an format appropriate to simulated annealing (or probabilistic hill climbing). The annealing schedule, on the other hand, involves decisions made at the stage of implementing the algorithm,

for which the user has a great deal of flexibility.

A further detailed discussion of types of annealing schedules and the specific annealing schedule chosen for use in the algorithm used here will be presented in Chapter 4.

2.6 Previous Results

The general nature of simulated annealing and its ease of implementation and design flexibility have resulted in the algorithm receiving a great deal of attention. It has been applied to a wide range of difficult optimization problems and has been found to perform well. A popular area of application has been in problems of computer-aided design (CAD) including cell-placement [50], logic minimization [31], and channel routing [4]. In general the research has shown the algorithm to be effective, achieving near-optimal results for NP-hard problems [52].

Sechen and Sangiovanni-Vincentelli [50] used the algorithm in the TimberWolf Package for placement of cells and gate-arrays and global routing. Significant improvements in layout (reduced necessary area and total wire length and congestion) are achieved using their package.

The major drawback of the simulated annealing algorithm in practice has been the excessive amount of computing resources needed for a single run. Certainly in comparisons with tailored algorithms the simulated annealing algorithm has performed poorly in this respect. In the first implementation and testing of the simulated annealing heuristic, Kirkpatrick *et al.* apply the algorithm to large scale problems in computer aided design (CAD) and also test it on the well known Travelling Salesman Problem (TSP). The results were encouraging in that the algorithm found high quality local minima for the problems. Later researchers continued with experiments on the TSP in order to compare simulated annealing with more refined or tailored heuristics generally finding that the

quality of solutions were comparable between the methods but that simulated annealing requires much longer computing time.

Due to the long running times, much effort has been spent in trying to find ways of reducing this time requirement. The most common method employed to speed up the algorithm has been determination of efficient annealing schedules with primary focus being on the long cooling stage, [22], [30], [1]. Other methods include approximating the cost value for newly generated solutions rather than using exact calculations [20], and using parallel processing [4], [56], [10].

Nahar *et al.* [40] tested the algorithm on the TSP and concluded that the algorithm is no match for well designed tailored heuristics. They also point out that the quality of a heuristic must be judged in comparison with other available heuristics both in terms of quality of solution produced and amount of computational resources required.

Golden and Skiscim [18] test the algorithm on both TSP and the p-median problem[3] and conclude that "simulated annealing is a clever, innovative approach, but there are probably more effective heuristics available for solving a given combinatorial optimization problem." They do go on to encourage further study of the approach however, since "it is perhaps unfair to expect a new approach such as simulated annealing to compete with the best of an almost endless array of TSP heuristics without extensive fine tuning."

From this it is possible to point out a useful niche for simulated annealing. When tailored algorithms are not available for a particular problem the simulated annealing algorithm appears quite reliable in finding high quality solutions. The algorithm can then prove to be useful in the absence of such refined algorithms when the only other alternative might be a random search. It may be possible to look for common features which characterize good solutions by analysing the solutions returned under the simulated

[3]The p-median problem is to best locate p facilities at various nodes of a network in such a way that the sum of the distances from each node in the network to its nearest facility is minimized.

annealing approach. This is suggested by Kirkpatrick [28] as well when he points out that using the algorithm "can give useful insights for devising effective heuristics."

Chapter 3

The File Allocation Problem

3.1 The File Allocation Problem

In this research the simulated annealing algorithm is applied to an instance of the File Allocation Problem with storage capacity constraints. The definition of the *File Allocation Problem (FAP)* (also called the file assignment or placement problem) is as follows.

> Consider a set of N nodes of a distributed computer network which use a common set of F computer files. An allocation i is a choice of the number of copies of each file $f \in F$ and the nodes $v \in N$ at which the copies are to be stored. Given usage rates and tariffs on usage of communication channels costs for each allocation can be determined. The solution to the FAP is that allocation i^* which minimizes communication costs for the network.

The particular version studied involves an imaginary company with offices across Canada. Computing facilities are available in twelve cities: Vancouver, Calgary, Regina, Winnipeg, Thunder Bay, Toronto, Ottawa, Montreal, Quebec City, Fredericton, Halifax, and St. John's. There are thirty different files accessed at each of these cities. Access rates are known and assumed stable for the two categories of transactions: query and update. Communication between cities is done over telephone lines and tariffs for telephone usage have been determined in order to calculate costs. Assuming that all transactions are carried out immediately, peak hour telephone rates t_{vw} between cities v and w will be in effect. The sizes of files are known l_f (length) and storage capacity available at each

facility is 30 megabytes (Mb). The query and update rates, files sizes and telephone rates between cities can all be found in the tables of Appendix A.

Formulating the problem as an integer (0 or 1) problem begins with the choice of decision variables. Let x_{fv} indicate whether a copy of the fth file is stored at the vth city.

$$x_{fv} = \begin{cases} 1 & \text{if a copy of } f \text{ is stored at } v \\ 0 & \text{if no copy of } f \text{ is stored at } v \end{cases} \tag{3.1}$$

where $f = 1, 2, \cdots, F$ and $v = 1, 2, \cdots, N$ with $F = 30$ and $N = 12$. That is to say that there are 30×12 decision variables which can be easily represented by a 30×12 matrix consisting of zeroes and ones. Each allocation i can be represented by one such matrix.

The cost of an allocation arises from communication between cities and is based on the access rates of files as well as the tariffs charged for using the communication facilities. Let the rate of query and update accesses from city v to file f be q_{vf} and u_{vf} respectively. A distinction is made between queries and updates because the transactions necessary are of different forms and the resulting costs due to each transaction determined differently. When a user node generates a query to a file it need only access the cheapest available copy of the file. The transaction then consists of sending a message to this copy and awaiting a response. Update messages, on the other hand must be sent to each file copy in the network in order to ensure integrity of information. Suppose the cost per transaction from node v to w is π_{vw} for updates and π'_{vw} for queries. Because of the necessity of waiting for a response in the case of queries these transactions have been assigned at twice the cost per transaction of updates, that is,

$$\pi'_{vw} = 2\pi_{vw}$$

Since telephone lines are used for communication these rates should be proportional to the telephone rates given in Appendix A, that is,

$$\pi_{vw} = \zeta t_{vw} \tag{3.2}$$

$$\pi'_{vw} = 2\zeta t_{vw} \tag{3.3}$$

where ζ represents the average length of a transaction and possibly a discount rate that the company is able to negotiate with the telecommunications service. Note that the rate is dependent on the node from which the transaction is generated since in many cases

$$t_{vw} \neq t_{wv}, \text{for } v, w = 1, 2, \cdots, 30.$$

Other authors distinguish between the cost of query and update communication between cities v and w also. Levin and Morgan [33] differentiate "on the basis that queries require faster response time than updating traffic and should be charged therefore with a higher rate." Casey [6] makes a note of a similar distinction where queries are again charged "at a higher communication rate" although equal rates are used in the experiments performed.

Since no storage costs are included in this instance (storage is included with the addition of the storage constraints), the cost for an allocation is simply the sum of the query costs and update costs. Also, costs due to each individual file are independent within an allocation. Let i_f be the sub-allocation for file f within allocation i. If $c(i_f)$ is the cost due to the sub-allocation of file f and $Q(i_f)$ and $U(i_f)$ the query and update costs respectively, with $f = 1, 2, \cdots, 30$, then the cost $c(i)$ for allocation i can be written

$$c(i) = \sum_{f=1}^{F} c(i_f) = \sum_{f=1}^{30} \{Q(i_f) + U(i_f)\} \tag{3.4}$$

Recalling that the usage rates q_{vf} and u_{vf} for query and update respectively are known for each file f, the calculation of the costs due to query and update are straightforward.

In the case of queries a message is sent to the minimum cost accessible file copy. The resulting cost due to queries is

$$Q(i_f) = \sum_{v=1}^{N} q_{vf} \min_{x_{fw}\neq 0}\{\pi'_{vw}\} \tag{3.5}$$

$$= 2\zeta \sum_{v=1}^{12} q_{vf} \min_{x_{fw}\neq 0}\{t_{vw}\}, \text{ for } f = 1, 2, \cdots, 30 \tag{3.6}$$

The update costs for file f with sub-allocation i_f is

$$U(i_f) = \sum_{v=1}^{N}\{u_{vf} \sum_{w=1}^{N} \pi_{vw} x_{fw}\} \tag{3.7}$$

$$= \zeta \sum_{v=1}^{12}\{u_{vf} \sum_{w=1}^{12} t_{vw} x_{fw}\}, \text{ for } f = 1, 2, \cdots, 30 \tag{3.8}$$

Combining this with Equation 3.4 the objective function becomes

$$\text{minimize } c(i) = \zeta \sum_{f=1}^{30}\{2 \sum_{v=1}^{12} q_{vf} \min_{x_{fw}\neq 0}\{t_{vw}\} + \sum_{v=1}^{12}\{u_{vf} \sum_{w=1}^{12} t_{vw} x_{fw}\}\} \tag{3.9}$$

And since the determining of the minimum allocation is of interest and not the precise cost calculation ζ can be ignored so that the actual objective function used is instead

$$\text{minimize } c(i) = \sum_{f=1}^{30}\{2 \sum_{v=1}^{12} q_{vf} \min_{x_{fw}\neq 0}\{t_{vw}\} + \sum_{v=1}^{12}\{u_{vf} \sum_{w=1}^{12} t_{vw} x_{fw}\}\} \tag{3.10}$$

Now it remains to present the constraints for this problem of which there are two in this case. First, at least one copy of each file must be maintained at all times. A mathematical representation for this is

$$\sum_{v=1}^{12} x_{fv} \geq 1, \text{ for } f = 1, 2, \cdots, 30 \tag{3.11}$$

Secondly, the storage capacity at each site can not be exceeded. The size l_f (length) of each file noted in Appendix A and the available space at each file was noted earlier as 30 Mb. This constraint can be written as

$$\sum_{f=1}^{30} x_{fv} l_f \leq 30, \forall v = 1, 2, \cdots, 12 \tag{3.12}$$

However, it shall be seen in Section 3.3, that the representations of these constraints as given above is not used in the simulated annealing algorithm.

Addition of the storage capacity constraint makes it impossible to reduce the multiple file problem to a collection of single file allocation problems. Without this constraint it would be possible to solve and combine the solutions to the individual file problems. Casey [6] studied the single file problem and developed theorems which are an aid to finding such solutions. In the size of problem given here it would not be difficult to exhaustively determine the minimum of each file problem of which there are $2^N - 1 = 2^{12} - 1 = 4095$ and combine these 30 solutions giving the optimal allocation. However, the storage constraints enlarge the search since all combinations of the each file solution must be considered, that is, $(2^N - 1)^F = (2^{12} - 1)^{30} > 10^{108}$ possible. Many of these will be infeasible with respect to storage capacity but the infeasibility can not be determined until an inspection is made, so that these infeasible solutions remain part of the search space in that sense.

In most research the issue of storage is include as either a cost in the objective function [6], or as a constraint [7]. In Chu's initial work on this problem [9] he, rather atypically, includes both storage costs and constraints in the formulation of the problem. Ceri *et al.* [7] justifies the inclusion of the storage constraint without a cost term by saying,

> this fact reflects the assumption of a global viewpoint in which storage costs are connected with the acquiring and maintenance costs of mass memory; adding a file to a node of the DDB [distributed database] has a null cost if the node capacity is not exceeded."

3.2 Previous Research of the File Allocation Problem

The FAP was first studied by Chu [9]. In addition to the storage capacity constraint Chu requires that the expected time to access each file be less than an allowable delay. Only one type of transaction, called a request, was considered and the resulting expected waiting time in the queues is modeled by a Poisson process. Also very different from the model presented above, the number of copies of each file is assumed to be fixed. Based on this model a linearization was performed reducing the problem to a linear integer programming problem, so that available linear integer programming techniques can be used to solve the problem. The number of variables increases so rapidly with this linearization however, that the remaining solution process is computationally difficult [7] making the method appropriate for only very small problem instances.

Many authors have focused on partitioning the multiple file allocation problem into multiple single file allocation subproblems [6], [33], [54]. Eswaran [12] uses this single file model, which he coined the *File Allocation Problem* (FAP), and shows that this is an NP-complete problem. The problem he presents incorporates both query and update transaction types but does not include reliability or availability requirements nor does it include storage constraints. This result is a disappointment in that, as Eswaran points out, it warns"people working in this area not to look for a solution that solves the file allocation problem in polynomial time."

Ramamoorthy [45] later observes that the typical version of the problem studied includes multiple copies and recommends that the single file version be instead termed the *Simple File Allocation Problem* (SFAP) and that FAP be the terminology reserved for more general versions of the problem, as it has been used here.

There have been a number of branches in the research areas due to varied extensions in the focus of the FAP. Because the problem itself is presented with such variation,

the heuristic methods which are then applied to solve these problems are likewise quite varied. Some examples are discussed here.

A number of authors [35], [13], [24], [44] have continued focusing on the design requirements of delay, reliability and availability of the files as in the original problem studied by Chu. Mahmoud and Riordan [35] also determine optimal capacities of communication channels along with file allocations. More recently Pirkul and Hou [44] stress these "availability and reliability issues" by requiring further precautions. They note that this can be very important for companies such as financial service companies with critical dependencies on their computer systems.

Levin and Morgan [32], [33], [38] research the extension of the problem from the level of *data sharing* where only data files are shared around the network, to the level of *program and data sharing.* At this level dependencies between programs and data files are included in the problem and the placement of both the programs and files must be determined. There may be limited sites at which the programs can be stored because of software compatibilities and routing variables determining paths for the transactions must be included.

Carresi and Gallo [5] continue researching at the level of program and data sharing. They formulate the problem as a linear integer programming problem with a "special structure" and use a Benders' decomposition approach to solve the linearized version.

Besides the level of data sharing presented, a framework for the categorization of the problems is presented based on the level of information available (complete or incomplete) and on whether the access request patterns were static or dynamic is presented in Levin's Ph. D. thesis. Based on these three criteria the problem studied here can be described as a data sharing problem assuming static access request patterns and with complete information.

Another area of focus has been cases of special topology of the network. Foster [13]

considers the case of a star topology network and Ranai *et al.* consider the multiple file FAP with performance constraints on a loop topology network. Again, the solution involves reducing the non-linear program to a linear form after some heuristic simplifications.

The heuristic techniques applied to the solution of these variations of the FAP are frequently developed by observing heuristics which work well on other similar problems. Ramamoorthy and Wah [45], for instance, show the isomorphism of the SFAP to the Single Commodity Warehouse Location Problem (SCWLP). With this they suggest that the add-drop algorithm, branch and bound technique, dynamic programming and other methods which have been used successfully on the SCWLP would be likely candidates for success use on the SFAP. In another instance Ceri *et al.* [7] show equivalence (via an intermediate equivalence) of the (multiple) FAP with the Knapsack Problem. In both of these cases the authors present greedy heuristics in their papers although some of the methods suggested have been implemented in other research: add-drop algorithm [35], branch and bound algorithm [25], and specialized greedy heuristics [54].

Chen and Akoka [8] develop what they call a "more complete" model which simultaneously considers the distribution of processing power, allocation of programs and databases, and the assignment of communication line capacities. They develop an algorithm of the bounded branch and bound type to solve the model they develop. Gavish and Pirkul [15] also focus on this by considering the placement of computer or processing capacity along with file allocation.

3.3 Using Simulated Annealing

In Chapter 2, it was noted that a combinatorial problem must satisfy three requirements in order to fit into a format appropriate for simulated annealing.

The representation of solutions to this file allocation problem as 12×30 matrices consisting of zeroes and ones has already been established. Each such matrix can be considered a configuration in the solution space. Since not all such matrices constitute feasible solutions and steps must be taken to deal with the constraints of the problem at this stage.

The requirement of a cost function is satisfied for the feasible solutions using the function of Equation 3.10 above and we shall see now how it is extended to incorporate the storage constraints. The generation mechanism is presented and discussed in the following section.

Constraints can be handled in at least two different ways by the simulated annealing algorithm. The first method is used by most other heuristic algorithms and consists of simply restricting the search to the set of solutions feasible with respect to that constraint. The second method, first used by Kirkpatrick *et al.* [26], involves incorporating the constraint into the objective or cost function as an infeasibility penalty. In their study of the chip placement problem moves or perturbations consisted of interchanging chips many of which were of unequal size. If chips of different sizes are chosen for interchange it is likely that an infeasible overlap would result. Trying to avoid such moves would result in a highly constrained move set and trying to construct "a program to produce legal changes in configuration is prohibitively difficult." Thus the constraint of the chips not overlapping was instead incorporated into the objective function. This additional penalty was of the form

$$\omega \sum_{a,b} overlap(a,b) \tag{3.13}$$

where $overlap(a,b)$ is the area of the overlap between blocks a and b and ω is a weight.

The logic of the penalty function is that the constraints "can be violated during the early stages of optimization (high temperature), rather than being imposed explicitly in

the algorithm that proposes changes in the configuration of the system. The Metropolis algorithm will drive the system toward satisfaction of the constraints as the temperature is reduced."

Each of the constraints in the FAP here must be considered independently. The "at least one copy" constraint is relatively easy to check for and a move strategy can be implemented which satisfies this constraint without restricting movement through the solution space. That is, a valid search can proceed which does not include these solutions.

The constraint of storage capacity obviously has much more impact on the solution space. It would be very complicated to develop a strategy which could avoid solutions which violate the storage capacity constraints without restricting the search of the solution space as in the case of the overlap constraint for the chip placement problem. In fact, such a strategy would need to be so involved as to make it nearly impossible to ensure that the solution space would remain connected with respect to that movement scheme. Thus, in the case of the storage capacity constraints, a penalty function representing the "infeasibility" will be added to the cost function.

Sechen and Sangiovanni-Vincentelli [50] also point out that employing an elaborate scheme to move away from infeasible solutions may destroy one of the advantages of the simulated annealing algorithm, that being its ability to slowly evolve into states with characteristics of a "good" solution. For instance, in the chip placement problem shifting the chips in some way to avoid an overlap may destroy the relationship between the shifted cells in the neighboring rows. In the case of adding and/or deleting files to obtain a solution satisfying the storage constraints would similarly destroy inter-file relationships established up to that point.

Variations on the penalty function are considered besides the form of Equation 3.13 above. The amount of exceeded storage capacity (if any) is calculated for each new

configuration considered and termed **OVER**(i). The resulting cost function is,

$$\hat{c}(i) = c(i) + \text{penalty}(\text{OVER}(i)) \tag{3.14}$$

where $c(i)$ is the communication cost for allocation i given by Equation 3.10.

This is in fact one of the strategies suggested by Lin in [34] of using heuristic methods in solving combinatorial problems. That is, he suggests that a "relaxation of the set of requirements" of the problem be considered, at least at the early stages.

More details of the actual penalty functions[1] tested will be given in Section 4.2.

3.4 Generation Mechanism

The generation mechanism defines the move set available for a problem. Based on these strategies the exploration of the solution space is carried out. As is the case for the iterative improvement algorithm the perturbations can be quite random or can incorporate complex strategies and insights gained about the makeup of the solution space developed through study of the problem. Using specialized moves, if they exist, was proposed by Kirkpatrick *et al.* [28] and has been implemented by others [22] [39] with results showing that this can improve the efficiency of the algorithm. This strategy can be considered a *hybridization* of the algorithm, such as the hybrid forms of the genetic algorithm created by Grefenstette [19].

In implementing the algorithm here no knowledge of advanced move strategies is assumed. The performance of the algorithm is evaluated using randomly generated moves.

The encoding of the problem presented determines the basic strategy of the generation mechanism[2]. Each allocation i can be represented as a group of file sub-allocations i_f, for

[1] In discussions of cost beyond this section "cost function" will refer to that function including both communication cost and infeasibility penalty and will be denoted simply $c(i)$.

[2] In practice the choice of encoding scheme may be influenced by the suitability of this scheme to appropriate generation strategies.

$f = 1, 2, \cdots, 30$. The splitting of the solution in this way is useful for the calculation of costs and it immediately suggests the first step to be taken in generating a perturbation. A random file number f is chosen and a perturbation of sub-allocation i_f is generated.

Each sub-allocation is an array of 12 integers (0 or 1) representing storage of the file at that city. A basic move is as follows:

> A site is chosen at random and the status of the file at that site is toggled. That is, if a copy exists there it is dropped (the array entry of 1 is changed to a 0) and if no copy is stored there one is added (the array entry of 0 is changed to a 1).

It was noted earlier that the constraint requiring "at least one copy" of each file should be checked during movement through the space. This constraint requires that not all entries in the sub-allocation array be set to zero at the same time. Hence, after a basic move is performed a test must be made if the entry chosen was changed from a one to a zero. If it is found that every entry of i_f is now zero, a randomly selected entry is changed to one.

It may be observed however, that starting from a random solution i it may take a large number of moves given only the "basic move" described above, to get to a near-optimal solution. An additional "compound move" is incorporated into the generation mechanism to aid in more rapid movement through the solution space. The choice of which type of move to make will be determined randomly in fixed ratio.

The compound move used here is based on the inversion move of the genetic algorithm[3] [21]. Inversion randomly cuts a string at two points in the string and creates a new structure by inverting the string between these two points [17]. In implementing this, two string (array) positions $s_1, s_2 \in 1, 2, \cdots, 12$ are chosen at random. The first position

[3]A comparison of simulated annealing algorithms and genetic algorithms was made by Ackley [2]

s_1 indicates where the inversion will start and the second s_2 where it will end. The string is treated as if it wraps around so that there is no need to require $s_1 < s_2$.

The idea of using compound moves was used by White [55] although the type of compound moves he presents are of a different type. Starting with a simple move, a move of length l is determined by repeated simple moves. The acceptance criterion is applied only when the entire move has been determined. By calculating moves of greater length White attempts to reduce the computing time for the algorithm since it requires "less computing effort to make 1 move of length l than l moves of length 1." In this way it may require fewer moves to reach equilibrium than it would simply relying on simple moves. Although the inversion moves used here are not determined in the same manner as those of White they can be considered compound moves since each inversion move could be achieved by repeated basic moves, if all moves are performed on a single file sub-allocation.

Based on the moves presented the solution space is strongly connected and so the homogeneous Markov chain at each temperature satisfies the requirement of existence of the stationary distribution. Also, the law of detailed balance is satisfied since both the basic move and the compound move are reversible and equally probable, satisfying Equation 2.1. This ensures that if the algorithm were allowed to evolve long enough at each temperature the Markov chain would converge to the stationary distribution.

The size of the neighborhood of each configuration is used to determine the length of stages of the annealing process. Using only the basic moves for this calculation (since the inversion type moves are viewed as compound basic moves) the size of a neighborhood is $M = 30 \times 12 = 360$ since one of the 30 files are chosen at random and, within the sub-allocation, one of the 12 entries is toggled.

The ratio of move types, basic to compound, was preset at 3:1 and was held fixed throughout. The approach used by White is to consider moves of greater length at higher

temperature. For instance, use moves of length l with $l = L$ at the start of the algorithm and $l \downarrow 1$ as T decreases. Using the inversion moves however, the "move length" has no equivalent interpretation and thus the ratio of moves was adopted. An alternative which would be interesting to evaluate in further research would be to begin with a small basic to compound move ratio and allow this ratio to increase as T decreases.

Chapter 4

Implementation

4.1 Annealing Schedule

Annealing schedules were discussed briefly in Section 2.5 and will be dealt with more thoroughly here. The annealing process was described as divided into three stages: melting, cooling and freezing. The schedule will describe how the system evolves through these three stages. The melting stage is determined by the *hot condition*, the cooling stage by the *temperature decrement* and the *equilibrium criterion* and the freezing stage by the *frozen condition.*

In the initial implementations [26] the issue of the annealing schedule was mentioned but little theoretical work was done at this point. An initial annealing schedule was designed and implemented including parameters which were fixed *a priori.* These were then tuned empirically to improve the result. Since that time each stage of the annealing algorithm has received research attention and the ideas put forward can be combined to generate a large selection of annealing schedules.

The distinction has been made between static and adaptive (or dynamic) annealing schedules. Static schedules have all parameters governing the stages of the algorithm determined for the entire run before the algorithm starts. Adaptive algorithms have some parameters or running patterns of the process determined during the run using information gathered about the cost distribution. Adaptive schedules can be classified depending on the number of stages during which parameters are being modified. A

schedule will be called *truly adaptive* if each of the three stages is adaptive. Schedules adaptive in only one or two of the stages will be called *semi-adaptive.*

Static schedules, though straightforward to implement, have the disadvantage of requiring tuning of the parameters through post-analysis for each new implementation. The stages of a semi-adaptive schedule which are static should likewise undergo tuning. Thus, there is an obvious advantage in using a truly adaptive schedule especially when simulated annealing is being applied to a new problem. The annealing schedule used in the tests done here was a truly adaptive schedule. The schedule is based on the annealing schedule presented by Huang *et al.* [22], with some modifications.

4.1.1 Hot Condition

The starting temperature of the system, T_0, must be chosen with care so that it is "hot enough" that the system becomes randomized or "melted." This implies that a very high proportion of the generated configurations must be accepted at this temperature. A static hot condition would specify T_0 *a priori* [40], while an adaptive one would determine T_0 based on some criterion indicating that the temperature is "hot enough."

Kirpatrick *et al.* [27] use a straightforward method of determining T_0 using a criterion based on the *acceptance ratio* χ. The acceptance ratio is the ratio between the number of accepted moves and the number of perturbations attempted over a test period. The user can specify a *target acceptance ratio* χ_0 for which the system will be considered "hot enough." Then, starting with an arbitrary temperature T, a number of moves (a few hundred is suggested) are attempted and the acceptance ratio $\chi(T)$ calculated for that value of T. If $\chi(T) < \chi_0$ the temperature is increased according to a predetermined rule, say doubled, and the test is repeated at the new temperature. This is continued until the target acceptance ratio has been met or passed. That is, T_0 will be the first T found such that $\chi(T) \geq \chi_0$. In [28] Kirkpatrick suggests χ_0 of around 80%.

Aarts and van Laarhoven present a refinement of this method [1]. Again the criterion is based on a specified acceptance ratio χ_0 but an intricate update rule increases the value of T at each step so that after m attempted perturbations a T_0 value is returned guaranteeing an acceptance ratio of χ_0.

The technique used here is given by Huang *et al.* [22]. Information about the distribution of the cost function is gathered and then applied to the calculation of T_0. This is based on the conclusion of White [55] that the system would be "hot enough" if $T_0 >> \sigma$ where σ is the standard deviation of the cost function. Huang *et al.* then determine T_0 by the equation

$$T_0 = k\sigma \tag{4.1}$$

Huang *et al.* find that a sufficiently high value of k is $k = 20$ based on assumptions of a normal distribution of the cost function. Though this assumption is not tested on the problems solved, the results using this k are good. In fact, $k = 20$ is a rather high value under the assumptions made, allowing room for error, and as Huang *et al.* note, the overhead in using a large value of k is small. The standard deviation σ is not known and so an estimate s is found by an initial exploration of the configuration space called the *heatup stage.* During this stage a large number of perturbations are performed with all being accepted, corresponding to a period of infinite temperature in the annealing process. The heatup stage also allows for random movement throughout the configuration space. The length of this search was not specified by Huang *et al.* In deciding what the length of this search should be I took into account the choices made for length of search at other temperatures (described below). The longest searches have lengths of a range of M to $4M$ where M is the number of states accessible from a single configuration, or the size of the neighbourhood. The value used is $M = 360$, the size of the neighbourhood of basic moves as described in Chapter 3. In the case of the heatup

stage each newly generated configuration is accepted and so the search need not be as long as $4M$. It is also desirable that the statistics generated be of good quality since the standard deviation is not updated again as the program proceeds but is used in the equilibrium criterion. Thus a range of $2M$ to $3M$ was felt to be sufficient. For notational convenience a perfect square was desirable and since $900 = 2.5 * M$ fell into the desired range this value was used.

Most implementations begin with a randomly generated initial solution. Nahar *et al.* [40] note that "for some problems, a good starting solution may yield better results than a random starting solution." The improvement in the final solution appears to be achieved primarily when the allowed running time for the simulated annealing algorithm is significantly shortened and the amount of "melting" of the system is restricted. For two reasons this was not an appropriate approach for solving the problem studied here. First, no other heuristics were being used and so what would constitute a good initial solution is not known at the start. Secondly, the algorithm is allowed to proceed long enough through the "heatup" stage in all runs. This involves random movement through the space, thus reducing the effect of any good initial solution. Hence, the initial solution was randomly generated for each run performed.

4.1.2 Cooling

Once the system begins the cooling stage the temperature decrement and the equilibrium criterion must be specified. As Lam [30] points out, the implementation of simulated annealing is based on the observation that physical annealing is successful if the system is kept close to thermal equilibrium. The temperature decrement and equilibrium criterion interact to keep the system near equilibrium. Because both of these decisions can impact on the closeness of the system to equilibrium during this stage, only one of the temperature decrement ratio or the equilibrium criterion need be adaptively adjusted for the

cooling stage to be considered adaptive.

For a typical adaptive schedule some check will be performed on the progress of the distribution of the cost function as T is decreased so that when the cost is changing most rapidly the process can be slowed. The "slowing" of the annealing is typically achieved through adjusting only one of the processes involved in the cooling stage. In the case of the temperature decrement the temperature is decreased in smaller steps [1], and in the case of the equilibrium criterion more time is spent at the key temperatures to allow the system to evolve longer at that temperature [28]. Some authors have used a combination of these two processes to slow the cooling [30].

Huang's method uses a running average of the cost function to adapt the equilibrium criterion during stages of rapid decrease in the cost function. The temperature decrement depends only on the temperature and is not affected by changes in the distribution of costs but the adaptations of the equilibrium criterion make this stage adaptive. The temperature decrement and the equilibrium criterion are described below.

Temperature Decrement

The usual method of decreasing the temperature involves a *decrement ratio* $\alpha(T)$ in the equation

$$T_{m+1} = \alpha(T_m) * T_m \tag{4.2}$$

where T_m is the old temperature and T_{m+1} is the new temperature and $\alpha(T_m)$ is either a constant or a function of T_m. In the implementation by Kirkpatrick *et al.* α was set at a constant value throughout, and $\alpha = .90$ was found to perform well [26].

In the TimberWolf package [50] Sechen and Sangiovanni-Vincentelli allowed α to be determined by the user throughout with the recommendation that a range of $0.80 \leq \alpha \leq 0.95$ be used with α depending on the changes in the cost function. When the

cost function is decreasing most rapidly α should be at its highest, slowing the cooling process. At the initial and final stages, when the changes in the cost function are smaller α should be at its lowest so the temperature decreases more quickly. The resulting cooling proceeds with a gradual increase in α early on in the algorithm until a largest value α value is reached and then a slow decrease in α towards the final stages of the algorithm.

The temperature decrement used by Huang *et al.* is of the form of Equation 4.2 with α an exponential function involving the standard deviation σ (where the estimate s of σ calculated during the "heatup stage" is used). Using a relation in statistical mechanics Huang *et al.* derive the equation

$$T_{m+1} = T_m * \exp(\frac{T_m \Delta\bar{c}}{\sigma^2}) \tag{4.3}$$

where $\Delta\bar{c}$ is the change in the average cost at temperature T_m. In order to stay near equilibrium the expected decrease in the average cost must be less than the standard deviation of the cost, that is, $\Delta\bar{c} = \lambda\sigma$ where $\lambda \leq 1$. This gives the decrement function

$$T_{m+1} = T_m * \exp(-\frac{\lambda T_m}{\sigma}) \tag{4.4}$$

A value of λ of 0.7 was used by Huang *et al.* and found to be adequate.

In order to avoid too great a decrease in temperature, especially when the curve is relatively flat at high temperature, a lower bound on the ratio of T_{m+1}/T_m of 0.5 was used by Huang *et al.* That is,

$$T_{m+1} \geq 0.5 T_m. \tag{4.5}$$

In implementing this schedule it was found that the asymptotic behavior of the sequence of temperatures near zero caused slow convergence of the algorithm. Based on the results on Kirpatrick *et al.* and Sechen *et al.* it was felt that an upper bound on the ratio of $T_{m+1}/T_m \leq 0.95$ would be sufficient so the limit of

$$\beta = T_m \tag{4.6}$$

was used. To test the quality of the solutions using this upper bound, runs were also performed with the upper bound of $T_{m+1}/T_m \leq 0.97$. Notable differences in the quality of the solutions with the ratio in this range would indicate that the upper bound on the ratio is causing too rapid a decrease in temperature. The temperature decrement is implemented in the **TEMPCONTROL** procedure in the attached program.

Equilibrium Criterion

In Chapter 2 the number of moves attempted at a particular temperature was referred to as the length of the Markov chain. The equilibrium criterion determines the length of the Markov chain for each temperature. The aim of the annealing schedule is to allow the system to evolve until it is "sufficiently close" to the stationary distribution referred to as *equilibrium* at temperature T. Because this criterion describes the length of each stage within the inner loop of the annealing algorithm it is also referred to as the "inner loop criterion" or simply the "length of the Markov chain".

In the initial implementation [26] the number of attempted moves or times through the inner loop was set at a constant value throughout. Later in [28] Kirkpatrick used a running average of the cost $\bar{c}$ to identify temperatures where the cost was decreasing rapidly. Large changes in $\bar{c}$ are used to indicate when the cooling was proceeding too quickly so that more time should be allowed at that temperature for the system to reach equilibrium. Kirkpatrick lengthened the time spent at such a temperature by "repeating a temperature," which is equivalent to doubling the length of the Markov chain. This is a method of adaptation of the equilibrium criterion within the cooling stage.

In Huang *et al.* schedule the equilibrium criterion is likewise the adaptive part of the cooling stage. Rather than simply lengthen the search at temperatures where rapid decreases in cost are occurring, a test of nearness to equilibrium is implemented. The search is allowed to proceed until near enough to equilibrium (within an upper bound).

The observation is made that the ratio of the number of accepted states with their costs within a certain range δ from the average cost to the total number of newly accepted states will reach a stable value χ near equilibrium. If the average cost is $\overline{c}$ then the number of accepted configurations with costs in the range of $(\overline{c}-\delta, \overline{c}+\delta)$ will be referred to as *within count*. Based on the assumption of a normal distribution the ratio between the *within count* states and all the accepted states is $\chi = erf(\delta/\sigma)$ where $erf(x)$ is the error function of probability theory [22]. The "typical" value of δ used by Huang *el al.* is 0.5σ, giving a χ value of $\chi = \text{erf}(0.5) = 0.38$. Equilibrium is said to have been reached when the count for this ratio is χ or greater.

The number of accepted states with costs outside the δ range, are referred to as the tolerance count. The point at which equilibrium has occurred is found by declaring a target *within count* and a maximum tolerance limit for the tolerance count statistic, in the proportions determined by χ. For χ as given above *within count* will be $0.38 * L$ and the maximum tolerance limit therefore $0.62 * L$, where L some constant indicating the total number of accepted states comprising the test. Equilibrium is then considered reached if the *within count* value reaches the target before the tolerance count exceeds the maximum tolerance limit. If the tolerance count exceeds this maximum value before equilibrium is reached both the *within count* and tolerance count are set at zero and the counting starts again. The number L of accepted states was set at $3 *$ (the size of the problem) by Huang *et al.* Here, the number of files $f = 30$ in problem was used instead of the size of the problem since this is representative of the "degrees of freedom" for the file assignment problem. This gives $L = 3 * 30$ and with rounding the target within count used was 34 and the maximum tolerance limit 56.

Ideally all the statistics associated with the determination of equilibrium would be updated continuously as the algorithm proceeds. The computing cost associated with

doing this is very high and in practical implementations the advantage gained is questionable. The average cost $\bar{c}$ must be updated however, since it is a record of changes to the cost distribution at different temperatures. Also, as the center of the *within count* interval, it has the most direct impact on the equilibrium test. An average of the last 100 accepted costs was maintained throughout as $\bar{c}$.

At the start of every new temperature Huang *et al.* require that a minimum number m of states be accepted to ensure that the $\bar{c}$ value has been updated before the test for equilibrium begins. They use m as the "size of the problem" and, as above, I have instead used the number of files, $m = f = 30$.

Two additional limits complete the determination of the length of the any Markov chain. Both of these are intended to control the performance of the algorithm at low temperature. Since the target *within count* and χ are not being dynamically adjusted with decreasing temperature, *within count* may never reach the target value at low temperatures. A generation limit is a bound on the number of attempted (generated) moves at one temperature. Two different generation limits are used to limit the length of a single chain. A maximum generation limit of M, where M the size of a typical neighbourhood, is the maximum number of moves which need to be generated if m are accepted at some point (again, $M = 360$ is used). For particularly low temperatures when the number of accepted configurations becomes a very small fraction of the attempted configurations an ultimate generation limit of $4M$ will indicate the end of the chain even if fewer than m configurations are accepted to that point.

Given this the length of a Markov chain is determined by which of the following three conditions occurs **first**.

1. Equilibrium based on the target *within count* and the maximum tolerance limit is reached, where the counting begins after a minimum of m configurations have been

accepted at the new temperature.

2. A minimum of m configurations have been accepted and the maximum generation limit M has been reached or exceeded.

3. The ultimate generation limit $4M$ has been reached.

In the attached program code the **COOL** and **EQUILIBRIUM** procedures demonstrate the implementation of the equilibrium criterion described.

4.1.3 Frozen Condition and Final Quenching

We say the system has become frozen when there is little or no possibility of further changes to the current cost distribution or current solution. The method used to determine when this likelihood has decreased is called the frozen condition.

The frozen condition used for the implementation here differs from the annealing schedule of Huang *et al.* and is the major divergence made from their schedule (the only other being the upper bound on the T_{m+1}/T_m ratio). The reason for this difference was primarily ease of implementation of the frozen condition used.

An effective adaptive frozen condition which is easy to implement is used by Sechen and Sangiovanni-Vincentelli in the TimberWolf package. In this case the algorithm is terminated when the last configuration of Markov chains at three consecutive temperatures remains unchanged. Because of the ease of implementing this criterion, this is the frozen condition used here. Note that this is still an adaptive frozen condition since it responds to the behavior of the algorithm. The **STOPCRITER** procedure in the attached program implements this frozen condition. The ease of implementing this condition as opposed to more elaborate schemes is a great bonus in an intricate simulated annealing program.

The method used by Huang *et al.* uses a comparison of two values to indicate when the search has found a region consisting of states of comparable quality. The difference between maximum and minimum costs among the accepted states at each temperature is compared with the maximum change in cost in any accepted move during that temperature and if these are the same annealing is halted. At this point a greedy algorithm is employed to find the best of solutions in this region. This is carried out by setting the temperature to zero so that only downward cost moves are accepted, and can be called the "quenching stage."

In the program developed here the final quenching stage is used to finish off the process. Though there is not much likelihood of improvements being found at this point, ending with a brief greedy search increases the probability that the program will terminate in a local minimum. Because of the low probability of improvements at this stage however, only M new configurations were generated and evaluated in this stage. The **QUENCH** procedure conducts this final search in the attached program code.

4.2 Penalty Function

In Chapter 3 the method of incorporating a constraint into the objective function of the problem through the use of a penalty function was introduced. The resultant objective function for configuration i is

$$\text{cost}(i) = \text{communication cost}(i) + \text{penalty}(i) \tag{4.7}$$

The storage capacity constraint for the file assignment problem is incorporated into this penalty function. The penalty is a function of the amount by which the solution i exceeds the storage capacity available, OVER(i). Three types of penalty functions were evaluated and compared in applying the simulated annealing algorithm to this file assignment problem.

The first penalty function used, called penalty A, was based on the original form of the penalty function of Kirkpatrick *et al.* This consists of a fixed weighting of the degree of infeasibility of the solution. In the cell placement problem this was the area of overlap, and for the file assignment problem it is OVER(i), giving the penalty function

$$A(i) = \omega_A * \text{OVER}(i) \tag{4.8}$$

where ω_A is the weight.

The second penalty function tested is based on the variation of the above function used by Sechen *et al.* in the TimberWolf Placement Package. In addition to the weight on the infeasibility a separate "offset parameter" is added to "ensure that when the parameter T approaches zero, the total amount of overlap approaches zero." This is achieved by keeping the penalty above some minimum no matter how small the amount of overlap. Thus, the Metropolis criterion will force even small infeasibilities to zero. Applied to the file assignment problem this penalty function, which shall be called B is

$$B(i) = \omega_B * \text{OVER}(i) + \psi \tag{4.9}$$

where ω_B is the weighting of the degree of infeasibility and ψ is the offset parameter.

In the third penalty function, which shall be called penalty C, ensuring that the final solution is feasible is again the driving objective. In this case, the infeasibility penalty is increased as the temperature decreases. Setting the penalty inversely proportional to temperature achieves this objective, giving the penalty function

$$C(i,T) = \omega_C * \frac{\text{OVER}(i)}{T} \tag{4.10}$$

where ω_C is again a weight and T is the temperature. The effect of progressively increasing the penalty function will be to allow greater movement through the configuration space in the early stages of the search. As the system is cooled and the penalty increases

the search will restrict itself increasingly to solutions which are feasible. If the movement through the space is too restrictive under penalties A and C this may improve the performance of the algorithm.

In the case of penalty C some care must be taken as T becomes small. For small T, the penalty function grows increasingly rapidly. The changes in the cost distribution at consecutive temperatures may cause the system to stray far from the stationary distribution. Since the objective of the annealing schedule is to maintain the system near equilibrium, this rapid growing stage of the penalty function should be corrected. This is easily achieved by using Equation 4.10 until T reaches a lower bound T_f after which a linear function like that of 4.8 is used. Under this altered form penalty C becomes

$$C(i,T) = \begin{cases} \omega_{C_1} * \frac{\text{OVER}(i)}{T} & T > T_f \\ \omega_{C_2} * \text{OVER}(i) & T \leq T_f \end{cases} \tag{4.11}$$

where ω_{C_1} and ω_{C_2} are weights satisfying the equation

$$\omega_{C_1} = \frac{\omega_{C_2}}{T_f} \tag{4.12}$$

in order that penalty C be a continuous function. Continuity ensures that the first T value with $T \leq T_f$ also does not cause a large increase in cost and hence, deviation from equilibrium.

The penalty function through the heatup stage also needs special consideration under penalty function C. The statistics gathered during this stage are used throughout the cooling stage, and should be representative. For penalty functions A and B the penalty weight does not change and so needs no adjustment in the heatup stage. For penalty C however, the penalty increases as the process continues. It was shown that the heatup stage is like annealing when $T = \infty$, which would imply a penalty of zero be assigned infeasible solutions under penalty C, based on the asymptotic limit of Equation 4.11. In

order for the gathered statistics to be more representative of the final cost distribution the penalty assigned during this stage shall instead be the linear part given in Eq. 4.11

$$C_\infty = \omega_{C_2} * \text{OVER}(i) \tag{4.13}$$

All of the penalty functions include weights and/or parameters which have not been specified. These remain to be determined through experimentation. It is hoped that some characteristics of or relationships between the three penalty functions can be determined through this experimentation.

Chapter 5

Experimental Results

5.1 Basis for Comparison

The details of the implementation have been determined. The determination of parameter values for the penalty functions will occur through experimentation. The results here will suggest strategies which may help in the process of determining effective parameter values.

The two central elements of the performance analysis are [52]:

1. Quality of the final solution.
2. Running time required by the algorithm.

Since the algorithm is not being changed dramatically, the running times are all quite similar. Therefore, comparisons between algorithms will focus on the quality of the final solution. The number of temperatures which the algorithm attempts is representative of the running time of the algorithm, and will be used when comparison of times is necessary.

In addition to returning a "final result", each run is programmed to keep track of the *best feasible solution* (BFS) it comes across in performing the search. In [28] Kirkpatrick suggests that "in systems with relatively few degrees of freedom we have sometimes found that the objective function is not very smooth, so that it is difficult to approach a low-lying local optimum gradually at low temperatures." To avoid this problem he suggests

recording the best solutions. Since simulated annealing slowly settles into regions of lower and lower cost values, it may come across a region containing a global minimum before the temperature is low enough that it will terminate there. In such a case, the result of the algorithm may be a different value than the BFS. This can be considered an adaptation of the algorithm to this kind of search space. In the cases where these results differ, both will be reported.

An initial run of the algorithm considered the problem without any storage constraints. That is, no penalty was added to the cost. It was felt this would give some indication of the performance of the algorithm, since the minimum is known to exist and the objective function would have a greater chance of being smooth in this region. Two runs were made, for each of two different values for the upper bound on the decrement ratio α, 0.95 and 0.97. All returned identical solutions, having a cost of 96021. Though the minimum was not otherwise determined, the consistency of the algorithm was both reassuring and promising.

The other information gained by doing these runs concerns the scale of the cost function, which will be the basis for the initial choice of the parameter values in the penalty functions. The objectives of the penalty function suggest that the parameters should be low to moderate in value. If the values are high, the search will be restricted, destroying the intent of the penalty function. Values that are too low will likely return infeasible results. Based on this information, the initial value of the parameters ranged between 200 and 750. After the initial runs the information obtained from these results would provoke choices for later trial parameter values.

The performance of the annealing schedule can also be evaluated by analysis of the *annealing curve*, a graph of the average cost at a temperature vs. the temperature. White [55] introduced the use of annealing curves to track the progress of the annealing and outlined the characteristics of the curve:

1. There is a minimum cost c_0 below which the system never goes.

2. There is a maximum average cost = c_∞, achieved at high temperatures where the system moves randomly through all states. Thus c_∞ is just the average cost (energy) of all states in the system.

3. A transition occurs at a low temperature, T_l, from an intermediate-cost regime to a low-cost regime, with average cost decreasing with $T > T_l$, and approximately constant and equal to c_0 for $T < T_l$.

4. A transition occurs at a high temperature, T_u, from the high-cost regime to the intermediate-cost regime.

The curve for the unconstrained problem in Figure 5.1 provides a good illustration of these features.

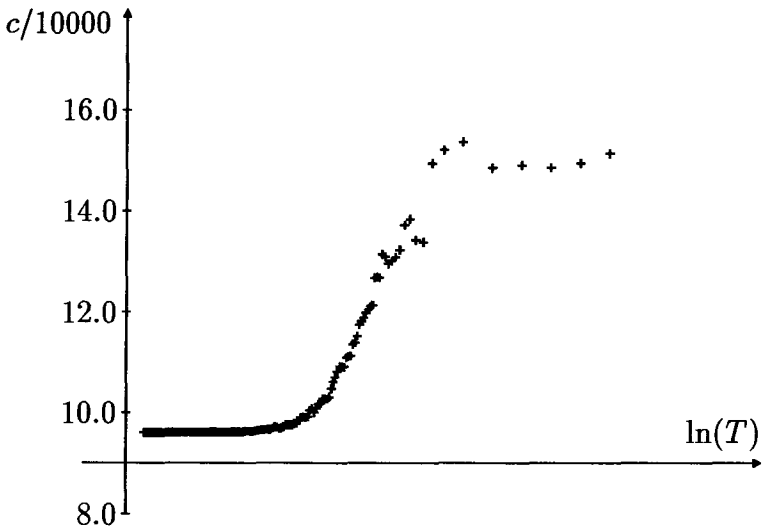

Figure 5.1: Annealing Curve for the Unconstrained Problem

The flat portion of the curve along the upper limit c_∞ indicates that the system was heated sufficiently. Another flattening along c_0 indicates that it was cooled sufficiently

gradually to allow for a settling into the lowest cost region possible.

Since the annealing schedule is not changed for the various runs (except for the upper bound on α), the comparison of annealing curves is used here instead to trace the effect of the different penalty functions. The ability of the annealing curve to follow the entire annealing process will be used to analyze the performance of the functions.

5.2 Comparison

The results of penalties A, B, and C are given in the tables of this section. If the solution returned is infeasible, this is noted, but the cost displayed does not indicate the extent of the infeasibility. That is, the portions of the cost due individually to communication and penalty, are not specified. Note that in the instances where the returned result is infeasible, there are cases where no BFS returned. This indicates that throughout the search no feasible solutions were found.

Runs for Type A and Type C penalty were made with $\alpha = 0.95$ and $\alpha = 0.97$. For Type B only $\alpha = 0.95$ runs were carried out. Effort was instead put into determining the influence of each of the two parameters, weight ω_B and offset parameter ϕ. Unfortunately this also reduces the already small number of trials for each penalty function. While for functions A and C the second set of runs helps to point out any patterns which may exist, the conclusions for the B functions are less certain. Drawing conclusions from the results in Table 5.2 may therefore be problematic.

The results of Table 5.2 are split into those with $T_f = 0.5$ and $T_f = 5.0$. The initial trial value was $T_f = 0.5$ but on first analysis the performance seemed erratic. The value of $T_f = 5.0$ was tried and seemed to perform better. It was not until analysis of the results later that the results for T_f were more clearly understood. The explanation of this accompanies the overall discussion of the penalty C results later in this section.better.

Penalty	Results	
	$\alpha = 0.95$	$\alpha = 0.97$
OVER $*$ 150	105964*	105221*
	106815	105547
OVER $*$ 200	106402*	105906*
	106572	106106
OVER $*$ 250	106498*	106263*
	106595	106363
OVER $*$ 300	107513	106075
OVER $*$ 350	107115	106818
OVER $*$ 400	107005	106424
OVER $*$ 450	107290**	106803
	107254	
OVER $*$ 500	107157**	106821
	107154	
OVER $*$ 550	107802**	107784
	107768	

Table 5.1: Results for Type A Penalty
* final result infeasible; feasible BFS noted if observed
** final result is feasible but a better BFS is noted

Because of the more thorough set of results for the $T_f = 5.0$ trials, these will be the intended reference in comparisons between penalty C and the other functions.

First, the impact of the choice of α will be considered. Comparing the tendencies along the columns of Table 5.2 it is clear that an increasing cost trend occurs for both α values. The values for $\alpha = 0.97$ are consistently superior. The best results for both cases occur in the first three lines, with the exception of one superior result in the second column. In these first six cases the result returned by the algorithm is an infeasible allocation, but a good quality solution had been found and stored as the BFS. In other words the tendency toward infeasibility at this end, and towards higher costs at the other, was a function of the penalty function alone. The higher α value improved the results an average of 562.

Penalty	Results
OVER $*$ 500 + 3000	108490
OVER $*$ 500 + 1000	107334
OVER $*$ 300 + 400	107218
OVER $*$ 300 + 300	106672**
	106661
OVER $*$ 300 + 100	106079
OVER $*$ 200 + 3000	107748
OVER $*$ 200 + 1000	107061
OVER $*$ 200 + 500	106866
OVER $*$ 200 + 300	105945
OVER $*$ 200 + 200	106686**
	106674
OVER $*$ 200 + 100	106405*
OVER $*$ 100 + 1000	106366*
OVER $*$ 100 + 500	105867*
	110212

Table 5.2: Results for Type B Penalty
* infeasible solution; feasible BFS noted if observed
** final result is feasible but a better BFS is noted

Continuing the investigation of α across the columns in Table 5.2, the penalty relationship appears even more significant here in that very similar solutions are obtained for a given penalty function regardless of the value of α. For instance, three penalty functions resulted in searches which found not a single feasible solution for either value of α. In comparing costs between columns the impact of α appears to not be significant. Of those seven pairs with feasible solutions in each column, four are lower and three higher for $\alpha = 0.97$.

The results of Table 5.2 imply that an improvement in the cost can be obtained for a given penalty function by increasing α, but the overwhelming evidence is that the penalty function is significantly more important. That is, cooling the system more slowly will not

overcome a poor choice of penalty function. However, the slower cooling may be a useful method of fine tuning, particularly for some penalty functions. Given this, the strategy to use upon implementation should be:

1. Using a fixed lower value of α, determine which penalty function(s) appears most promising.

2. Allow the runs to be repeated with higher α values, within an upper limit of available (computer) time.

Note also that any improvement in solution achieved by increasing α causes a substantial increase in the running time of the algorithm. For all $\alpha = 0.95$ runs, the number of temperatures before freezing has a range of 109 to 165, while the range for $\alpha = 0.97$ is 156-269 temperatures before freezing.

Now that the influence of α has been established, a more detailed inspection of the individual penalty functions is warranted. In evaluating the results, observe that the very best feasible result found has a cost of 105531. This will be called *low*. On the basis that *low* is the best value found over a number of trials, there is a positive probability that it is the optimal solution [34]. Thus, solutions will be judged on the basis of their closeness to this value.

Considering the results for Type A penalty in Table 5.2, the best group of solutions were the BFS solutions which accompanied an infeasible final result. These results occurred when the penalty was lowest. For higher penalty values the results were consistently feasible but of poorer quality. The implication of this is that higher penalty function manages to keep the search away from infeasible solutions but in so doing also restricts their access to the lower cost feasible solutions. That is, these two solution types must occupy a similar region of the configuration space. Encouraging the search of more infeasible solutions, by lowering the penalty, results in the discovery of yet lower

Penalty			Results	
T_f	$T > T_f$	$T \leq T_f$	$\alpha = 0.95$	$\alpha = 0.97$
0.5	$\text{OVER} * 100/T$	$\text{OVER} * 200$	107877*	107761*
0.5	$\text{OVER} * 250/T$	$\text{OVER} * 500$	105911** 105852	110838*
0.5	$\text{OVER} * 500/T$	$\text{OVER} * 1000$	106911	106080** 105697
0.5	$\text{OVER} * 750/T$	$\text{OVER} * 1500$	106198	107781** 105531***
5.0	$\text{OVER} * 500/T$	$\text{OVER} * 100$	104878*	104778*
5.0	$\text{OVER} * 750/T$	$\text{OVER} * 150$	105274*	105347*
5.0	$\text{OVER} * 1000/T$	$\text{OVER} * 200$	105427** 105647	105494* 105714
5.0	$\text{OVER} * 1250/T$	$\text{OVER} * 250$	105469* 105569	105672* 106175
5.0	$\text{OVER} * 1500/T$	$\text{OVER} * 300$	105758** 105624	105755* 105837
5.0	$\text{OVER} * 2000/T$	$\text{OVER} * 400$	105858* 105858	105597
5.0	$\text{OVER} * 2500/T$	$\text{OVER} * 500$	106010** 105757	105958** 105682

Table 5.3: Results for Type C Penalty
* infeasible solution; feasible BFS noted if observed
** final result is feasible but a better BFS is noted
*** best result found throughout

feasible solutions. The low cost of 105547 for penalty OVER$*$150 is very near the cost of *low*. Although no smaller ω_A values were considered, the proximity to a perceived limit suggests that values of $\omega_A < 150$ might be likely to produce infeasible results rather than further reductions in cost. This is speculative but we shall see in the comparison of the Type B and Type C penalty functions that if the penalty drops too low this can occur. To be certain, however, more testing should be done for weights in this lower range.

The results of the Type B penalty again follow the pattern of smaller penalty values resulting in better solutions. This applies to both the weight and the offset parameter. The solutions for higher parameter values were again consistently feasible (the stated goal of this penalty function). Comparing this to the $\alpha = 0.95$ column of Table 5.2, the results are similar. Upon inspection, other combinations of penalties which might improve the performance of this penalty function can be hypothesized. It was found that trying to tune two parameters at once made this penalty function difficult, especially since each parameter seemed to impact the performance significantly. In fact, more variations of this penalty function were attempted than any other function, with no obvious good strategy emerging. Given that the results are not significantly different from the results of the more straightforward Type A penalty function, it is does not appear to be a promising choice of penalty function.

The annealing curves[1] for both Type A and B penalties are very typical, conforming to the shape suggested by White. Examples are given in Figures 5.2 and 5.3. These represent results from penalty functions OVER$*$200 and OVER$*$300$+$300, respectively.

Now consider the results of Table 5.2. As noted earlier, the results obtained for $T_f = 0.5$ were considered suspicious at first. This was mainly because of the infeasible but high cost for the first penalty listed and the combination of high cost infeasible and good feasible solution for the next penalty. Upon closer inspection of the results,

[1]The graphs of annealing curves for the remaining cases can be found at the end of this Chapter

specifically the data obtained for the annealing curve, it was determined that the process was spending more time at higher temperatures and ending at higher temperatures then anticipated. The finishing temperature was typically in the range of 2 to 8. Based on this "freezing" temperature the penalties for these infeasible functions remained small. The final quenching stage recalculated the costs based on a $T = 0$ temperature, causing a sudden jump in cost for any infeasible solution. The feasible result in the midst of these other infeasible results must be considered an aberrant case.

Comparing the remainder of the results, this penalty function appears to be exceptionally promising. Even with a restriction to the $\alpha = 0.95$ instances the results are all close to the *low* value. The reasons for this good performance appears to be related again to the amount of infeasible space which is included in the search. A number of the final results were infeasible within this group, implying a concentrated search within a region containing many feasible and infeasible solutions. This was observed for penalty A as well, and appears again to be a strategy producing good results. In this case the results are more consistent, however. This is likely because the initially low penalty enabled the system to converge on a lower cost region, while the increase in the penalty caused the search to focus increasingly towards feasible solutions. Note the final two cases with $T_f = 0.5$ can be explained in a similar manner, with one change. The initial progress of the system should mirror the first two penalty functions listed for $T_f = 5.0$, diverging only during the quenching stage. The results of these two other penalty functions suggests that the solutions would be infeasible but close to feasible solutions. The high penalty then imposed during the quenching stage for $T_f = 0.5$ would drive the solutions to be feasible. For $T_f = 5.0$ however, the penalty remains small and therefore the solutions remain infeasible.

The most impressive feature of the results of penalty C is its consistency. This is likely since the changes in the penalty parameters have their primary impact later in the

search. The effect is therefore like starting all runs with the same penalty function and incrementally altering the penalties as the freezing point approaches. In any case, the consistent results indicate that the problem being studied is well suited to this type of penalty function.

The parameter T_f in penalty C has been ignored for the most part here. As was noted for Type B penalty, the effort of attempting to tune two parameters at once can be quite cumbersome. The good results of $T_f = 5.0$, combined with the information regarding the range of the freezing temperatures, suggests that a promising strategy is to pick T_f within this range. Further research would be needed to confirm this point.

The annealing curves for the Type C penalty functions are not typical of White's annealing curve. Because of the changing penalty, a minimum occurs below the c_0 boundary line. This reflects the search through a region consisting largely of infeasible solutions. At the upper end the curve is also rounded, rather than flat along the line c_∞. Initially the average cost increases with the increasing penalty, and so the curve rises to a maximum value before beginning its descent.

Larger values of the weight ω_{C_1} the initial portion of the graph $T > T_f$ produce flatter curves. Figure 5.4 is the annealing curve for penalty function with $\omega_{C_1} = 2000$ and $T_f = 0.5$ for $\alpha = 0.97$.

The obvious dip at the lower end of this curve illustrates the settling into a region of low cost infeasible solutions out of which the algorithm must climb. The increasing gradient at the end of this curve reflects the fact that the system became frozen before the process stabilized into a region of feasible solutions.

Figure 5.5 is the annealing curve for the Type C penalty function with $\omega_{C_1} = 2000$ and T_f for $\alpha = 0.95$.

In this case the "dip" is much less pronounced and the curve flattens out quickly. Observe how the curvature along the upper portion of the curve is similarly greater in

Figure 5.4 and nearly flat in Figure 5.5.

5.3 Summary of Results

The best results have been obtained throughout by penalty functions which manage to hover between infeasible and feasible solutions. The BFS was therefore particularly important in recording the desired feasible result, given that the algorithm results were frequently infeasible.

This boundary region between feasible and infeasible solutions appears to contain an abundance of good quality solutions, but to be of very complex structure. As Kirkpatrick suggests, the objective function is not smooth, making the minimum(a) difficult to approach.

The allocation results were compared for a number of cases using a simple file-compare procedure. In comparisons with the non-constrained result, cost 96021, it was found that most individual file results consisted of a subset of the "optimal" sub-allocation. This suggests why the searches which focussed on regions near the feasible/infeasible boundary were particularly effective. That is, the search towards good infeasible solutions produces features desirable in the good feasible solutions. The results among constrained problems showed particular similarities. One set of sub-allocations appeared to be consistently identical, while the remaining were remarkably similar. Further investigation of such similarities could be used to determine important features of high quality allocations. This demonstrates how the solutions obtained from simulated annealing can be used in the development of heuristics.

The Type C penalty function, within the range of the testing here, outperformed the other penalty functions. Along with being a particularly consistent function, it also appeared to be least (not at all?) affected by more rapid cooling for lower α values. Given

the already long running times of the simulated annealing algorithms, this reduction in the time required to find a quality solution is a significant advantage.

The comparisons made are dependent on the instance of the file allocation problem chosen for the study. Due to the characteristics of the results I inferred some features of the feasible solutions space, the feasible/infeasible boundary and the distribution of good quality solutions within this space. Further research may be able to determine more about the relationship between the best penalty function and the characteristics of problem instance.

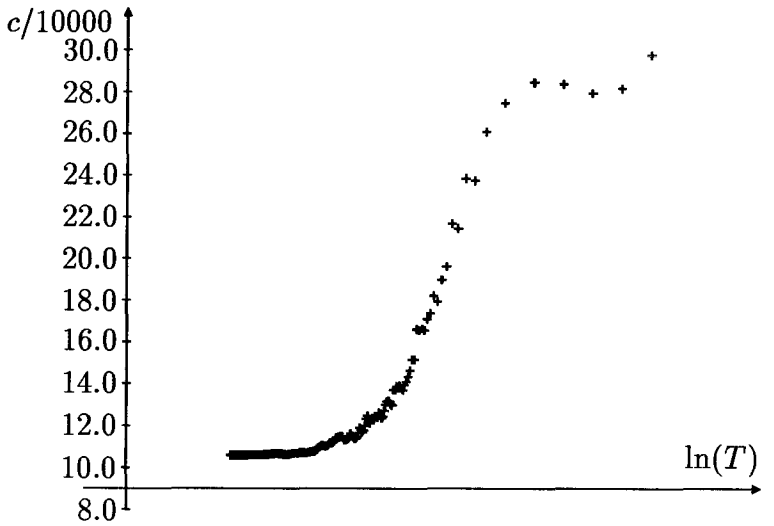

Figure 5.2: Typical Annealing Curve for Type A Penalty

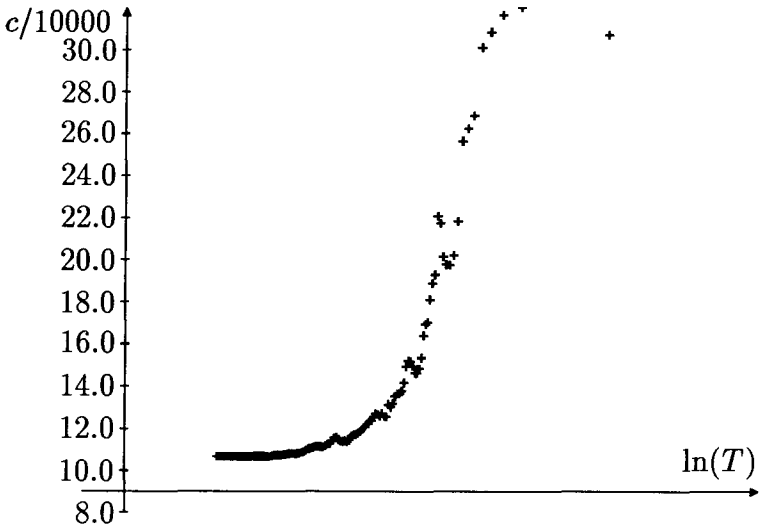

Figure 5.3: Typical Annealing Curve for Type B Penalty

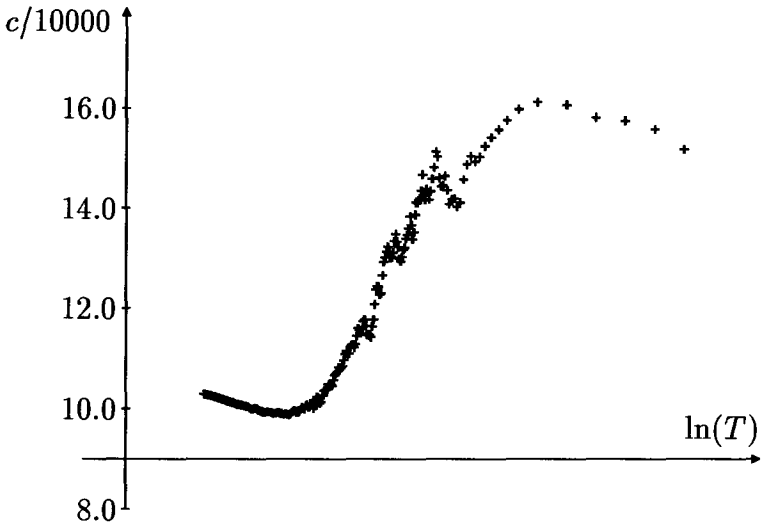

Figure 5.4: Annealing Curve for Penalty C with $\omega_{C_1} = 250$

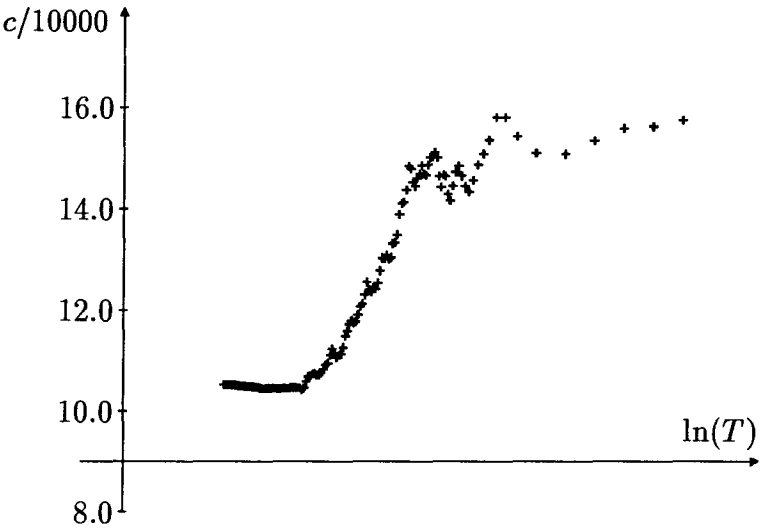

Figure 5.5: Annealing Curve for Penalty C with $\omega_{C_1} = 2000$

Chapter 6

Conclusions and Further Research

6.1 Conclusions

The implementation of the simulated annealing algorithm to a particular problem involves some basic steps. These have been clearly outlined in demonstrating the implementation of the simulated annealing algorithm.

In implementing the simulated annealing algorithm to solve a particular problem, three necessary features were required. These were: representation of solutions as configurations; a cost function which can be evaluated; and a generation mechanism which could perturb solutions. The basis for the cost function and an obvious representation are usually immediately available. The generation mechanism can be based on randomly generated moves or can incorporate complex move strategies which have been developed for the problem. As demonstrated here, the type of move produce by "random generation" can include simple and/or complex moves.

The choice of annealing schedule involves many decisions, but many well constructed schedules have been developed and can be exploited.

Complex constraints are typically handled by incorporating the constraint into the objective function in the form of an added penalty. This results in a higher cost for infeasible solutions which the algorithm will select against.

Different penalty functions were considered for the storage constraints of the problem being solved. The values of the parameters will be dependent on the scale of the

problem. If such information is available it can be the basis for the initial values of the parameters. The tuning of the parameters is achieved primarily through experimentation and comparison.

The various penalty functions seem to approach the optimal feasible solutions from different directions, suggesting that the best penalty function for a particular problem may be dependent on the relationship between the feasible and infeasible solutions. The most productive search region for this problem was along the boundary between feasible and infeasible solutions. Of the three penalties tested Type C, inversely proportional to the temperature function, performed the best. The performance of the algorithm and its annealing curve give some indication of the ability of this penalty to focus its search along the promising feasible/infeasible boundary.

The annealing curves may give some information about the process taking place but they do not provide sufficient detail to help in predicting performance. As used here, they give insight into interesting features of penalty strategies considered

Once a best penalty function is determined, it may be possible to improve on the results by slowing down the cooling process. This does not consistently improve solutions however, and comes at the cost of increased running time. Therefore, this strategy should be used with caution. Most importantly, slower cooling cannot remedy a poor choice of penalty function, so a good quality penalty function must be determined first.

6.2 Implications for Further Research

The compound "inversion" moves borrowed from the genetic algorithm incorporated White's idea of "longer moves" probabilistically. The effect of including this move strategy was not evaluated in any way. Some interesting comparisons might be: performance with and without this move strategy; different proportions of basic moves to compound

moves; an adaptive ratio of basic to compound moves, dependent on temperature.

The comparison of penalty functions done here has shown a surprising variation in the results for different penalty functions. Other parameter values remain to be attempted to confirm (hopefully) these results.

Since a specific problem was being considered throughout, the effect of changes to the problem, and thereby the geometry of the solution space, remain unknown. A cross-comparison using different types of functions would reveal the extent to which the best performing penalty is dependent on the problem being studied.

As can be seen from the implementation here, simulated annealing is a relatively easy to implement algorithm. Given its adaptability, it may prove a useful tool in finding good quality solutions to new or difficult problems. These solutions can be analyzed for similarities in the development of heuristics. Since tailored algorithms out-perform simulated annealing algorithms, primarily in terms of necessary computing time, this may be a useful niche for the algorithm to fill.

In versions of a particular problem where a new constraint is introduced which might make previously used heuristics impossible to apply, simulated annealing can act as a good intermediate algorithm which is little affected by the additional constraint, by using a penalty function. Using simulated annealing in this way may also give some insight as to the effect the additional constraint has on the standard problem, thus hinting at approaches which might be beneficial in developing a new heuristic. In this way, the algorithm would be used in the refinement of heuristic algorithms.

Bibliography

[1] E. Aarts and P. van Laarhoven, "A new polynomial- time cooling schedule," *Proc. IEEE Int. Conf. on Computer-Aided Design*, 1985, pp. 206–208.

[2] D. H. Ackley, "An empirical study of bit vector function optimization," *Genetic Algorithms and Simulated Annealing*, L. Davis editor, Morgan Kaufmann Publishers, Los Altos, Ca., U. S. A., 1987, pp. 170–204.

[3] K. Binder, "Theory and "technical" aspects of Monte Carlo simulation," *Monte Carlo Methods in Statistical Physics*, K. Binder editor, Springer-Verlag, New York, 1979.

[4] R. J. Brouwer and P. Banerjee, "A parallel simulated annealing algorithm for channel routing on a hypercube multiprocessor," *Proc. IEEE Int. Conf. on Computer Design*, 1988, pp. 4–7.

[5] P. Carraresi and G. Gallo, "Optimal location of files and programs in computer networks," *Mathematical Programming Study*, 20, October 1982, pp. 39–53.

[6] R. G. Casey, "Allocation of copies of a file in an information network," *Proc. AFIPS Conference 40*, Spring Joint Computing Conference, 1972, pp. 617–625.

[7] S. Ceri, G. Martella and G. Pelagatti, "Optimal file allocation for a distributed data base on a network of minicomputers," *Proc. Intl. Conf. on Databases*, S. M. Deen and P. Hammersley editors, University of Aberdeen, July 1980, pp. 216–237.

[8] P. Chen and J. Akoka, "Optimal design of distributed information systems," *IEEE Trans. Computers C-29*, 12, December 1980, pp. 1068–1080.

[9] W. W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Trans. Computers C-18*, 10, October 1969, pp. 885–889.

[10] F. Darema, S. Kirkpatrick and V. Norton, "Parallel techniques for chip placement by simulated annealing on shared memory systems," *Proc. IEEE Int. Conf. on Computer Design*, 1987, pp. 87–90.

[11] L. Davis (editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[12] K. Eswaran, "Placement of records in a file and file allocation in a computer network," *Information Processing 74*, 1974, pp. 304–307.

[13] D. Foster, L. Dowdy and J. Ames, "File assignment in a star network," *Proc. 1977 SIGMETRICS/CMG VIII Conf. Comput. Perform.* , Washington, DC, 1977, pp. 247–254.

[14] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York, W. H. Freeman and Company, 1979.

[15] B. Gavish and H. Pirkul, "Computer and database location in distributed computer systems," *IEEE Trans. Computers C-35*, 7, July 1986, pp. 583–590.

[16] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-6, 1984, pp. 721–741.

[17] D. E. Glover, "Solving a complex keyboard configuration problem through generalized adaptive search," *Genetic Algorithms and Simulated Annealing*, pp. 12–31, L. Davis editor, Morgan Kaufmann Pub. Los Altos, Cal. U. S. A., 1987.

[18] B. L. Golden and C. C. Skiscim, "Using simulated annealing to solve routing and location problems," *Naval Research Logistics Quarterly 33*, 1986, pp. 261–279.

[19] J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," *Genetic Algorithms and Simulated Annealing*, L. Davis editor, Morgan Kaufmann Pub. Los Altos, Cal. U. S. A., 1987, pp. 42–60.

[20] L. K. Grover, "A new simulated annealing algorithm for standard cell placement", *Proc. IEEE Int. Conf. on Computer-Aided Design*, 1986, pp. 378–380.

[21] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[22] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing", *Proc. IEEE Int. Conf. on Computer-Aided Design*, 1986, pp. 381–384.

[23] Y. Ioannidis and E. Wong, "Query optimization by simulated annealing," *Proceedings of ACM SIGMOD 1987 Annual Conference*, 1987, pp. 9–22.

[24] K. Irani and N. Khabbaz, "A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems," *IEEE Trans. Computers C-31*, 5, May 1982, pp. 419–434.

[25] C. Jenny, "Methodologies for placing files and processes in systems with decentralized intelligence," Research Report, IBM Zurich Research Laboratory, Switzerland, 1982.

[26] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by simulated annealing," IBM Computer Science/Engineering Technology Watson Res. Center, Yorktown Heights, NY, Tech. Report, 1982.

[27] S. Kirkpatrick, C. Gelatt and M. Vecchi, "Optimization by simulated annealing," *Science 220*, 4598, May 1983, pp. 671–680.

[28] S. Kirkpatrick, "Optimization by simulated annealing: quantitative studies," *Journal of Statistical Physics 34*, 5 & 6, 1984, pp. 975–986.

[29] S. Kravitz and R. Rutenbar, "Multiprocessor-based placement by simulated annealing," *Proc. 23rd IEEE Design Autotmation Conference*, 1986, pp. 567-573.

[30] J. Lam and J-M. Delosme, " Performance of a new annealing schedule," *Proc. 25th ACM/IEEE Design Automation Conference*, 1988, pp. 306–311.

[31] J. Lam and J-M. Delosme, "Logic minimization using simulated annealing," *Proc. IEEE Int. Conf. on Computer-Aided Design*, 1986, pp. 348–351.

[32] K. Levin, "Organizing distributed data bases in computer networks," Ph. D. Dissertation, U. of Pennsylvania, Philadelphia, Pa. 1974.

[33] K. Levin and H. Morgan, "Optimizing distributed data bases - a framework for research, *Proc. AFIPS Nat. Computer Conf.* 44, 1975, pp. 473–478.

[34] S. Lin, "Heuristic programming as an aid to network design," *Networks*, 5, 1975, pp. 33–43.

[35] S. Mahmoud and J. Riordon, "Optimal allocation of resources in distributed information networks," *ACM Trans. Database Systems*, 1, 1976, pp. 66–78.

[36] S. Mallela and L. Grover, "Clustering based simulated annealing for standard cell placement," *Proc. 25th ACM/IEEE Design Automation Conference*, 1988, pp. 312–317.

[37] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Jour. of Chemical Physics 21*, 6, June 1953, pp. 1087–1092.

[38] H. Morgan and K. Levin, "Optimal program and data location in computer networks," *Commun. ACM 20*, 5, May 1977, pp. 315–322.

[39] S. Nahar, S. Sahni, and E. Shragowitz, "Experiments with simulated annealing," *22nd Design Automation Conference*, 1985, pp. 748–752.

[40] S. Nahar, S. Sahni, and E. Shragowitz, "Simulated annealing and combinatorial optimization," *23rd Design Automation Conference*, 1986, pp. 293–299.

[41] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, U. S. A., 1988.

[42] R. H. Otten and L. P. van Ginneken, "Stop criteria in simulated annealing," *Proc. IEEE Int. Conf. on Computer Design*, 1988, pp. 549–552.

[43] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982.

[44] H. Pirkul and Y Hou, "Allocating primary and back-up copies of databases in distributed computer systems: a model and solution procedures," *Computer and Operations Research 16*, 3, 1989, pp. 235–245.

[45] C. Ramamoorthy and B. Wah, "The isomorphism of simple file allocation," *IEEE Trans. Computers C-32*, 3, March 1983, pp. 221–232.

[46] K. Ranai, C. Pursglove and P. Leng, "Optimal file allocation in a loop topology local area network," *Computers and Mathematics with Applications 13*, 7, 1987, pp. 601–610.

[47] F. Romeo and A. Sangiovanni-Vincentelli, "Probabilistic hill climbing algorithms: properties and applications," *1985 Chapel Hill Conference on Very Large Scale Integration*, Computer Sciences Press, Chapel Hill, N. C., 1985, pp. 393–417.

[48] F. Romeo, A. Sangiovanni-Vincentelli and C. Sechen, "Research on simulated annealing at Berkeley," *Proc. IEEE Int. Conf. on Computer Design*, October 1984, pp. 652–657.

[49] F. Romeo and A. Sangiovanni-Vincentelli, "A theoretical framework for simulated annealing," *Algorithmica 6*, 3, 1991, pp. 302–345.

[50] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE J. Solid State Circuits SC-20*, 2, April 1985, pp. 510–522.

[51] P. N. Strenski and S Kirkpatrick, " Analysis of finite length annealing schedules," *Algorithmica 6*, 3, 1991, pp. 346–366.

[52] P. van Laarhoven and E. Aarts, *Simulated annealing: theory and applications*, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1987.

[53] M. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. Computer Aided Design CAD-7*, October 1983, pp. 215–222.

[54] B. Wah, "An efficient heuristic for file placement on distributed databases," *COMPSACC 80*, October 1980, pp. 462–468.

[55] S. White, "Concept of scale in simulated annealing", *Proc. IEEE Int. Conf. on Computer Design*, October 1984, pp. 646–651.

[56] E. Witte, R. Chamberlain and M. Franklin, "Task assignment by parallel simulated annealing," *Proc. IEEE Int. Conf. on Computer Design*, 1990, pp. 74–77.

Appendix A

Problem Data

A firm has offices with computing facilities in 12 cities across Canada: (1) Vancouver, (2) Calgary, (3) Regina, (4) Winnipeg, (5) Thunder Bay, (6) Toronto, (7) Ottawa, (8) Montreal, (9) Quebec City, (10) Fredericton, (11) Halifax, (12) St. John's.

There are 30 different computer files which these offices must access for query and update transactions. Accesses are made to files remote locations by communicating over phone lines. Only 30 megabytes of storage space is available at each office.

The tables of this appendix contain the relevant information for this problem. Firstly, access rates from cities to files are given. Table A.1 contains the size of each file along with the update access rate for each of the thirty files from each city (where the numbering of the cities is as noted above). Table A.2 contains the query access rates to each file from each city.

Table A.3 contains the peak hour telephone rates in cents per minute. In reading Table A.3 note that the cities listed on the left column are the cities at which the calls originate and the cities listed along the top the destination of the calls. For instance a call from Regina (3) to Quebec City (9) will be charged at a rate of 68 cents per minute whereas a call from Quebec City to Regina will be charged at a rate of 47 cents per minute. The rates listed are those that would have been in effect January 1991.

File	Size (Mb)	Update access rate in each city											
		1	2	3	4	5	6	7	8	9	10	11	12
1	8	10	0	27	4	4	1	15	13	13	13	9	12
2	10	8	6	1	7	5	10	11	3	10	7	5	10
3	4	7	2	8	5	9	3	3	0	14	0	14	8
4	3	2	0	5	0	8	2	2	1	1	1	12	7
5	4	6	6	14	12	9	3	11	0	5	1	4	3
6	4	6	5	12	13	1	6	2	5	1	8	8	5
7	6	9	5	8	11	7	3	8	8	6	0	7	4
8	4	0	5	8	8	7	4	2	2	7	3	1	1
9	5	10	1	10	7	0	8	2	0	8	7	1	2
10	10	2	7	1	12	4	1	2	6	5	6	5	10
11	3	0	7	2	2	3	1	3	2	3	4	4	9
12	6	4	1	7	5	2	4	3	0	3	4	3	6
13	8	0	0	4	7	3	0	5	2	1	3	7	0
14	10	7	5	0	6	4	1	6	2	7	3	1	4
15	7	4	4	1	1	5	6	0	3	0	0	3	6
16	3	0	5	8	1	2	2	4	5	0	4	0	0
17	3	7	5	8	0	3	4	3	0	7	2	2	1
18	5	3	4	0	9	3	0	6	0	1	0	5	5
19	8	1	2	2	2	1	2	3	1	0	2	3	1
20	7	0	4	3	3	1	2	0	1	2	1	1	3
21	3	2	0	3	2	2	2	1	2	3	0	0	4
22	6	2	4	2	0	2	3	1	2	1	1	2	3
23	8	2	3	0	3	0	1	3	0	1	2	0	2
24	5	2	2	2	0	0	0	1	0	3	1	0	4
25	11	3	3	4	4	0	2	1	0	2	1	0	4
26	10	4	3	0	3	1	0	2	2	3	2	1	2
27	6	0	4	1	1	0	3	2	2	3	1	4	1
28	6	4	4	0	4	2	3	3	0	1	1	1	3
29	6	0	0	3	4	2	3	0	2	4	2	3	2
30	2	4	1	3	0	2	3	3	1	1	0	2	2

Table A.1: File size l_f and update frequency u_{vf}.

File	Query access rate in each city											
	1	2	3	4	5	6	7	8	9	10	11	12
1	3	0	1	2	0	0	1	0	2	1	0	3
2	2	5	1	0	0	5	2	0	4	1	0	6
3	1	1	3	3	1	0	2	0	0	0	5	4
4	0	0	2	0	1	0	0	0	1	0	2	3
5	0	0	9	0	0	1	2	0	2	0	2	1
6	0	0	6	10	1	0	1	2	0	0	0	0
7	1	0	1	10	1	0	1	3	3	0	0	0
8	0	0	1	5	2	0	0	0	0	3	0	0
9	0	0	6	2	0	0	1	0	0	6	0	0
10	2	5	1	11	1	0	0	0	1	6	0	1
11	0	0	1	1	0	0	1	0	0	0	3	1
12	3	0	3	2	2	0	0	0	0	3	0	3
13	0	0	1	1	2	0	0	0	0	3	2	0
14	4	4	0	1	2	0	0	1	0	0	0	1
15	4	0	0	0	1	5	0	1	0	0	0	1
16	0	0	2	0	0	1	3	3	0	2	0	0
17	0	2	0	0	1	0	0	0	4	1	0	0
18	0	0	0	0	0	0	1	0	0	0	4	0
19	0	0	0	0	0	0	1	0	0	0	0	0
20	0	3	2	1	0	1	0	0	0	0	0	0
21	1	0	0	1	1	0	0	1	1	0	0	0
22	0	2	2	0	0	0	0	0	0	0	0	3
23	0	0	0	0	0	0	0	0	0	0	0	2
24	1	1	0	0	0	0	0	0	1	0	0	0
25	1	0	0	0	0	0	1	0	0	0	0	0
26	2	1	0	1	1	0	1	2	0	0	0	1
27	0	2	0	0	0	1	0	1	0	0	0	1
28	0	0	0	1	0	1	1	0	1	0	0	0
29	0	0	1	0	0	1	0	0	1	1	0	1
30	0	0	1	0	1	0	0	0	0	0	0	0

Table A.2: Query frequency q_{vf}.

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	46	49	50	50	50	50	50	50	50	50	50
2	46	0	46	68	68	68	68	68	68	68	68	68
3	64	60	0	57	64	68	68	68	68	68	68	68
4	55	55	51	0	51	55	55	55	56	56	56	56
5	57	55	53	51	0	39	39	39	39	55	56	57
6	48	48	47	46	45	0	42	44	45	45	46	47
7	48	48	47	46	46	42	0	39	43	45	45	46
8	48	48	47	46	46	44	39	0	40	44	45	46
9	48	48	47	47	46	45	43	49	0	42	45	46
10	57	57	57	56	55	54	53	51	49	0	45	51
11	57	57	57	56	56	55	54	53	53	45	0	54
12	57	57	57	57	56	56	55	55	55	54	54	0

Table A.3: Telephone rates between cities t_{vw}.

Appendix B

Source Code

The following pages contain the source code which implements the simulated annealing algorithm. It is written in Turbo Pascal.

```
PROGRAM SIMANNLG;

     {An example of the simulated annealing procedure applied to a
      file allocation problem.  Written as part of Master's of
      Science requirements U.B.C.      Monica F. Hess, August 1992}

CONST
     MAXSITES = 12;          {Number of cities which are in the network
                              or sites where files are to be maintained.}
     MAXFILES = 30;          {Number of files to be stored within the
                              network.  Degrees of freedom for this problem}
     NUM      = 4095;        {Number of assignments per file = (2^12) - 1  }

TYPE
     SITES    = 1..MAXSITES;
     SOLUTION = ARRAY[SITES] OF INTEGER;{solution for a single file is
                                         a array by site of integers 0/1}
     FILECOST = ARRAY[1..NUM] OF REAL;  {costs of all possible solutions
                                         for a single file}
     ASSIGNMENT = ARRAY[1..MAXFILES]    {maintains an entire assignment in}
                  OF INTEGER;           {'code'... the decimal equivalent
                                         of the binary string}
     POINT    = ARRAY[1..2] OF REAL;    {points for annealing curve plot}

VAR
     ALLSOLNS : FILE OF SOLUTION;       {indexed by the decimal integer
                                         'code' this file gives the 0/1
                                         assignment solution by referal.}
     ALLSIZES : FILE OF INTEGER;        {File of Mb sizes for all files.}
     SIZEARR  : ARRAY[1..MAXFILES]      {Local copy of ALLSIZES array}
                OF INTEGER;
     ALLCOSTS : FILE OF FILECOST;       {FILECOST for each of the files.}
     SINGCOST : FILECOST;               {Local copy of the cost listing for
                                         a single file}
     CURVE    : TEXT;                   {Series of points to be plotted as
                                         an annealing curve after the run.
                                         Stored in ANCURV##.DTA.}
     RESULT   : TEXT;                   {Text file containing the final
                                         results: RESULT##.DTA}
     ASSIGNMT : ASSIGNMENT;             {The current file assignment as
                                         'coded' integer array}
     COMMCOST : REAL;                   {Cost of communication charges}
     PENALTY  : REAL;                   {Infeasibility penalty cost}
     BESTSOFAR: ASSIGNMENT;             {Best feasible solution encountered}
     LOWCOST  : REAL;                   {Cost of the BESTSOFAR assignment}
     LASTCOST : REAL;                   {Contains ending cost values for
                                         the previous temperature...used
                                         to check stopping condition}
     TEMP     : REAL;                   {Temperature control parameter}
     TABLE    : ARRAY[1..100] OF REAL;  {Table of last 100 costs, used in
                                         maintaining a running average.}
     TABVAL   : INTEGER;                {Index of the last entry entered
                                         into TABLE}
     SUM      : REAL;                   {Sum of the entries of TABLE}
     MEANCOST,                          {Average during the HEATUP stage;
                                         moving average during ANNEALING}
     STANDEV  : REAL;                   {Standard deviation of costs,
                                         calculated in the HEATUP stage}
     DELTA    : REAL;                   {Radius of 'withincount' interval,
                                         used in EQUILIBRIUM}
     BETA     : REAL;                   {Point at which the upper bound on
                                         temperature decrement ratio causes
                                         a change in the calculations in
                                         the TEMPCONTROL procedure}
     PLOT     : POINT;                  {(temperature, average cost) pairs
                                         for the annealing curve}
     I        : INTEGER;                {Counter}
```

```
     STOPLOOP : BOOLEAN;                {Flag for the end of the main
                                         annealing loop)
     UNCHANGED : INTEGER;               {Counter of the temperatures at
                                         which cost has been consecutively
                                         unchanged...STOPLOOP criterion)


{******************PROCEDURES and FUNCTIONS*********************)

{***************MENU***********)
{ 1. COSTCALC
  2. PENALTYCALC
  3. INITIALIZE
  4. CONVERT
  5. GENERATE
  6. ACCEPTPRO
  7. HEATUP
  8. EQUILIBRIUM
  9. STOPCRITER
  10. TEMPCONTROL
  11. ANNEAL
  12. QUENCH
  13. REPORTRESULT  )


{*** 1 ***)
FUNCTION COSTCALC(OLDASGNMT : ASSIGNMENT;
                  NEWASGNMT : ASSIGNMENT;
                    OLDCOST : REAL;
                   CHGINDEX : INTEGER): REAL;

     {Calculates the communication cost of a particular assignment
      using data from the ALLCOSTS file.  Only the file which has been
      altered in GENERATE must be indexed here.}

VAR
     INDEX1, INDEX2  : INTEGER;     {integer corresponding to file
                                     solution (binary converted to decimal))
BEGIN

     SEEK(ALLCOSTS, CHGINDEX);
     READ(ALLCOSTS, SINGCOST);
     INDEX1 := OLDASGNMT[CHGINDEX];
     INDEX2 := NEWASGNMT[CHGINDEX];
     COSTCALC := OLDCOST - SINGCOST[INDEX1] + SINGCOST[INDEX2]
END;


{*** 2 ***)
FUNCTION PENALTYCALC(AASGN : ASSIGNMENT;
                         T : REAL  ):REAL;

     {This procedure determines the nonfeasibility penalty to be
      added to the cost.  By calculating the space used at each site,
      the amount by which capacity is exceeded, OVER, is determined.}

     {SIZEARR called as a global variable)

CONST
     MBCAPACITY = 30;

VAR
     SPACE       : ARRAY[1..MAXSITES]     {Space occupied by all files at)
                     OF INTEGER;          {each of the sites.)
     OVER        : REAL;                  {tally of the Mb exceeded capacity)
```

```
     I,J,INDEX   : INTEGER;
     PRESSOLN    : SOLUTION;

BEGIN
     OVER := 0;
     FOR I := 1 TO MAXSITES DO
         SPACE[I] := 0;

     FOR I := 1 TO MAXFILES DO
         BEGIN
          INDEX := AASGN[I];
          SEEK(ALLSOLNS, INDEX);
          READ(ALLSOLNS,PRESSOLN);
          FOR J := 1 TO MAXSITES DO
               IF PRESSOLN[J] = 1 THEN
                   SPACE[J] := SPACE[J] + SIZEARR[J];
         END;
     FOR J := 1 TO MAXSITES DO
          IF SPACE[J] > MBCAPACITY THEN
               OVER := SPACE [J] - MBCAPACITY + OVER;

     IF OVER > 0 THEN                          {The actual penalty function}
          PENALTYCALC := OVER*250
     ELSE PENALTYCALC := 0;

END;


{*** 3 ***}
PROCEDURE INITIALIZE(VAR INITIAL : ASSIGNMENT;
                         VAR CCOST : REAL;
                       VAR PENCOST : REAL);

     {Randomly selects and evaluates an initial solution.}

VAR
     I        : INTEGER;
     INDEX    : INTEGER;

BEGIN
     FOR I := 1 TO MAXFILES DO
          INITIAL[I] := RANDOM(NUM) + 1;
     SEEK(ALLCOSTS,1);
     FOR I := 1 TO MAXFILES DO
         BEGIN
          READ(ALLCOSTS,SINGCOST);
          INDEX := INITIAL[I];
          CCOST := CCOST + SINGCOST[INDEX]
         END;

     PENCOST := PENALTYCALC(INITIAL,-1);   {Temp c      ..cates this is
                                    the first so.ut..  . the HEATUP phase}
END;


{*** 4 ***}
FUNCTION CONVERT(NEWSOLN: SOLUTION): INTEGER;

     {Converts the binary form of the solution from GENERATE into
      a decimal 'code' for use in referencing costs.}

VAR
     MID, I : INTEGER;
     EXTRA  : INTEGER;

BEGIN
```

```
      EXTRA := 0;
      MID := 1;
      FOR I := 12 DOWNTO 1 DO
          BEGIN
           IF NEWSOLN[I] = 1 THEN
                EXTRA := EXTRA + MID;
           MID := MID*2
          END;
      IF EXTRA = 0 THEN
          BEGIN {this should not occur and so acts as a warning signal}
           WRITELN('ERROR: A SOLUTION ARRAY OF ALL ZEROES ACCEPTED');
           REPEAT
                EXTRA := RANDOM(4095) + 1
           UNTIL (EXTRA > 0) AND (EXTRA <= 4095);
           WRITELN('SUBSTITUTE SOLUTION ',EXTRA, ' PUT IN PLACE')
          END;
      CONVERT := EXTRA
END;


{*** 5 ***}
PROCEDURE GENERATE(       ASGNMT : ASSIGNMENT;
                      VAR  PERTURB : ASSIGNMENT;
                      VAR CHGINDEX : INTEGER  );

      {Randomly generates a new solution, PERTURB, from the present one,
       ASGNMT.  One file is picked at random to be altered.  75% of the
       time the change will be a basic move and 25% an inversion move.}

VAR
      NUM               : REAL;
      OLD, I,
      SWITCH1, SWITCH2  : INTEGER;
      TEND              : BOOLEAN;
      NEWSOLN,
      OLDSOLN           : SOLUTION;
      OKAY              : BOOLEAN;
BEGIN
      PERTURB := ASGNMT;
      CHGINDEX := RANDOM(MAXFILES) + 1;     {pick an index for a file to be }
      OLD := ASGNMT[CHGINDEX];              {changed.}
      SEEK(ALLSOLNS,OLD);
      READ(ALLSOLNS,OLDSOLN);
      SWITCH1 := RANDOM(MAXSITES) + 1;      {site to change OR start postion
                                             for inversion}
      SWITCH2 := RANDOM(MAXSITES) + 1;      {end position for inversion}

      NUM := RANDOM;
      NEWSOLN := OLDSOLN;

      IF (NUM < 0.75) OR (SWITCH1 = SWITCH2) THEN
          BEGIN                                         {Basic move}
           IF OLDSOLN[SWITCH1] = 1 THEN
               BEGIN
                NEWSOLN[SWITCH1] := 0;
                OKAY := FALSE;                  {Check that all the entries for that
                                                  file have not gone to zero}
                FOR I := 1 TO MAXSITES DO
                    IF NEWSOLN[I] = 1 THEN OKAY := TRUE;
                IF OKAY = FALSE THEN NEWSOLN[SWITCH2] := 1
                                        {If entries have all become zero then
                                            change one to a 1}
               END

           ELSE NEWSOLN[SWITCH1] := 1       {If value is zero change to a one}
          END
```

```
      ELSE
          BEGIN                                    {Inversion move}
           IF SWITCH1 > SWITCH2 THEN TEND := FALSE
           ELSE TEND := TRUE;
           REPEAT
                NEWSOLN[SWITCH1] := OLDSOLN[SWITCH2];
                NEWSOLN[SWITCH2] := OLDSOLN[SWITCH1];
                SWITCH1 := SWITCH1 + 1;
                IF SWITCH1 = 13 THEN
                    BEGIN
                     SWITCH1 := 1;
                     TEND := TRUE
                    END;
                SWITCH2 := SWITCH2 - 1;
                IF SWITCH2 = 0 THEN
                    BEGIN
                     SWITCH2 := 12;
                     TEND := TRUE
                    END
           UNTIL TEND AND (SWITCH1 >= SWITCH2)
          END;

      PERTURB[CHGINDEX] := CONVERT(NEWSOLN)   {PERTURB is the 'code' decimal
                                        version of the solution obtained here}
END;


{*** 6 ***}
PROCEDURE ACCEPTPRO(              TMP : REAL;
                            VAR ACCPT : BOOLEAN;
                    OLDCOST, PERTCOST : REAL    );

      {Evaluation of the Metropolis criterion here determines whether
       the perturbed solution will be accepted or rejected}

VAR
      DIFFER    : REAL;         {to evaluate the difference between costs}
      RANDCOMP  : REAL;         {Random number used for comparison}
      RATIO     : REAL;
      LLOG      : REAL;

BEGIN
      RANDCOMP := RANDOM;
      ACCPT := FALSE;
      DIFFER := PERTCOST - OLDCOST;

      IF DIFFER <= 0 THEN ACCPT := TRUE
      ELSE
          BEGIN
           RATIO := -DIFFER/TMP;
           LLOG := LN(RANDCOMP);
           IF RATIO > LLOG THEN ACCPT := TRUE
          END;

      {The comparison made is equivalent to exp(-differ/temp) > random}

END;


{*** 7 ***}
PROCEDURE HEATUP(       VAR ASGNMT : ASSIGNMENT;
                 VAR CCOST, PENCOST : REAL;
                  VAR MNCOST, STDEV : REAL;
                           VAR BEST : ASSIGNMENT;
                            VAR LOW : REAL );
```

```
     {All new assignments are accepted and statistics on the cost
      distribution are gathered to be used in the calculation
      of the initial temperature and on equilibrium parameters.}

VAR
     PERTURB          : ASSIGNMENT;
     ACOST            : REAL;
     SQUARES          : REAL;
     MNSQRD           : REAL;
     I, CHANGEINDEX   : INTEGER;

BEGIN
     ACOST := CCOST + PENCOST;
     MNCOST := ACOST;
     SQUARES := SQR(ACOST/30);
     STDEV := 0;
     CHANGEINDEX := 0;
     FOR I := 1 TO 899 DO              { 899 + initial = 900 = 30^2 }

         BEGIN
          GENERATE(ASGNMT, PERTURB,CHANGEINDEX);
          CCOST := COSTCALC(ASGNMT,PERTURB,CCOST,CHANGEINDEX);
          PENCOST := PENALTYCALC(PERTURB, -1);  {Temp of -1 indicates Heatup}
          ASGNMT := PERTURB;

          IF (PENCOST = 0) AND (CCOST < LOW) THEN
              BEGIN                          { keep track of the BEST }
               BEST := ASGNMT;               { feasible solution seen }
               LOW := CCOST                   { i.e. LOWest cost.      }
              END;

          ACOST := CCOST + PENCOST;
          MNCOST := MNCOST + ACOST;          {adding up all the costs}
          SQUARES := SQUARES + SQR(ACOST/30)  {adding up all the squares}
         END;

     MNCOST := MNCOST/900;            {divide by 900 for the average}
     MNSQRD := SQR(MNCOST);
     STDEV := SQRT(SQUARES - MNSQRD);
                              {squares is sum(a/30)^2 = sum(a^2/900)
                                                      = sum(a^2)/900
                               then standard deviation is ...
                               (squares - meancost^2)^(1/2) = standev}

END;


{*** 8 ***}
PROCEDURE EQUILIBRIUM(     VAR STOPANNL : BOOLEAN;
                               DLTA : REAL;
                       VAR WTINCNT, TOLCNT : INTEGER;
                               MNCOST : REAL;
                           TESTCOST : REAL     );

     {Equilibrium criterion are tested during the ANNEAL procedure.
      STOP2 is set to true if equilibrium is reached and to false if
      more time should be spent at the temperature.}

CONST
     TARGWINCNT = 34;              {Target within count is approximately
                                     equal to 0.38*3*S where S is 30, dof}
     MAXTOL     = 56;              {Maximum tolerance limit is approximately
                                     equal to 0.62*3*S with S as above}

VAR
     UPPER, LOWER : REAL;
```

```
BEGIN
     {Equilibrium is considered maintained if the within count
      reaches the target value before the tolerance count exceeds
      the maximum tolerance limit.}

     STOPANNL := FALSE;
     UPPER := MNCOST + DLTA;              {Since the mean is changing these }
     LOWER := MNCOST - DLTA;              {  must be recomputed each time   }

     IF (TESTCOST < UPPER) AND (TESTCOST > LOWER) THEN
          WTINCNT := WTINCNT + 1
     ELSE TOLCNT := TOLCNT + 1;

     IF TOLCNT > MAXTOL THEN
         BEGIN
          WTINCNT := 0;
          TOLCNT := 0
         END
     ELSE IF WTINCNT >= TARGWINCNT THEN STOPANNL := TRUE
END;


{*** 9 ***}
PROCEDURE STOPCRITER( VAR STOPLP : BOOLEAN;
                      VAR LSTCOST : REAL;
                            CCOST : REAL;
                       VAR UNCHGD : INTEGER  );

     {Controls the number of passes through the main loop.  If the
      final cost remains unchanged for MINUNCD consecutive temps
      then STOPLP is set to true and the annealing will stop.}

CONST
      MINUNCD = 3;            {Number of final costs consecutively
                               unchanged which will determine stopping.}

BEGIN
      IF LSTCOST <> CCOST THEN
           UNCHGD := 0
      ELSE
           UNCHGD := UNCHGD + 1;
      LSTCOST := CCOST;
      IF UNCHGD >= MINUNCD THEN
           STOPLP := TRUE

END;


{*** 10 ***}
PROCEDURE TEMPCONTROL( VAR  T : REAL;
                         STDEV : REAL;
                           BTA : REAL );

     {Updates the temperature control parameter.}

VAR
     COMPL : REAL;                          {Lower bound 0.5}
BEGIN

     COMPL := 0.5 * T;
     IF T > BTA THEN                        {Decrement function of T}
          T := T*EXP(-0.7*T/STANDEV)
     ELSE T := 0.97 * T;                    {Bounded above by ALPHA * T}

     IF T < COMPL THEN T := COMPL           {and below by 0.5 * T}
```

```
     {ALPHA values must correspond with the value used to calculate
          BETA in the main program.  }
END;


{*** 11 ***}
PROCEDURE ANNEAL(       VAR ASGNMT : ASSIGNMENT;
                  VAR CCOST,PENCOST : REAL;
                          T, DEL : REAL;
                        VAR BEST : ASSIGNMENT;
                         VAR LOW : REAL        );

     {Perturbations are attempted until equilibrium is reached or an
      minimum number have been tried}
     {Variable TABLE is also needed in here but is sent as a global var
      along with TABVAL and SUM}

CONST
     M = 360;                    {360 = Number of configurations that
                                  can be reached in one simple move}
     MINACCEPT = 30;             {at least 30 moves must be accepted
                                  before testing for equilibrium begins}
VAR
     N,ACCEPTED,
     WITHIN,TOLCOUNT : INTEGER;
     CHANGEINDEX     : INTEGER;          {Index of altered file assignment}
     PERTURB         : ASSIGNMENT;
     NEWCOMM, NEWPEN,                    {Cost variables}
     NEWCOST,ACOST   : REAL;
     MEAN            : REAL;
     ACCEPT          : BOOLEAN;
     STOPANNEAL      : BOOLEAN;          {Indicates when equilibrium
                                          has been reached}

BEGIN
     ACCEPTED := 0;
     N := 0;
     PENCOST := PENALTYCALC(ASGNMT, T);      {penalty must be recalculated }
     ACOST := CCOST + PENCOST;               {when dependent on temperature}
     STOPANNEAL := FALSE;

     REPEAT                        {attempts before equilibrium is checked}
          N := N+1;
          GENERATE(ASGNMT, PERTURB,CHANGEINDEX);
          NEWCOMM := COSTCALC(ASGNMT,PERTURB,CCOST,CHANGEINDEX);
          NEWPEN := PENALTYCALC(PERTURB,T);
          NEWCOST := NEWCOMM + NEWPEN;

          IF (NEWPEN = 0) AND (NEWCOMM < LOW) THEN
              BEGIN                    {Keep track of best feasible soln}
               BEST := PERTURB;
               LOW := NEWCOMM
              END;

          ACCEPTPRO(T, ACCEPT, ACOST, NEWCOST);

          IF ACCEPT THEN
              BEGIN
               ACCEPTED := ACCEPTED + 1;
               ASGNMT := PERTURB;
               CCOST := NEWCOMM;
               PENCOST := NEWPEN;
               ACOST := NEWCOST;
               SUM := SUM - TABLE[TABVAL] + ACOST;
               TABLE[TABVAL] := ACOST;
```

```
                TABVAL := TABVAL + 1;
                IF TABVAL > 100 THEN TABVAL := 1;
              END;
        UNTIL (N >= 4*M) OR (ACCEPTED >= MINACCEPT);

        WITHIN := 0;
        TOLCOUNT := 0;
        MEAN := SUM/100;
        IF N >= 4*M THEN STOPANNEAL := TRUE;
        IF (N >= M) AND (ACCEPTED >= MINACCEPT) THEN STOPANNEAL := TRUE;

        WHILE NOT STOPANNEAL DO                    {Annealing with equilibrium
                                                    checking}
            BEGIN
             N := N+1;
             GENERATE(ASGNMT, PERTURB, CHANGEINDEX);
             NEWCOMM := COSTCALC(ASGNMT, PERTURB, CCOST,CHANGEINDEX);
             NEWPEN := PENALTYCALC(PERTURB,T);
             NEWCOST := NEWCOMM + NEWPEN;

             IF (NEWPEN = 0) AND (NEWCOMM < LOW) THEN
                  BEGIN                    {Keep track of best feasible soln}
                   BEST := PERTURB;
                   LOW := NEWCOMM
                  END;

             ACCEPTPRO(T, ACCEPT, ACOST, NEWCOST);
             IF (ACCEPT = TRUE) THEN
                 BEGIN
                  ACCEPTED := ACCEPTED + 1;
                  ASGNMT := PERTURB;
                  CCOST := NEWCOMM;
                  PENCOST := NEWPEN;
                  ACOST := NEWCOST;
                  SUM := SUM - TABLE[TABVAL] + ACOST;
                  TABLE[TABVAL] := ACOST;
                  TABVAL := TABVAL + 1;
                  IF TABVAL > 100 THEN TABVAL := 1;
                  MEAN := SUM/100;
                  EQUILIBRIUM(STOPANNEAL, DEL, WITHIN, TOLCOUNT, MEAN, ACOST)
                 END;
             IF (N >= M) AND (ACCEPTED >= MINACCEPT) THEN STOPANNEAL := TRUE;
             IF N >= 4*M THEN STOPANNEAL := TRUE
            END
    END;


    {*** 12 ***}
    PROCEDURE QUENCH(          VAR ASGNMT : ASSIGNMENT;
                      VAR CCOST, PENCOST : REAL;
                                VAR BEST : ASSIGNMENT;
                                 VAR LOW : REAL       );

         {Improvements in the cost function are attempted for M moves.}

    VAR
         I, CHANGEINDEX : INTEGER;
         NEWCOMM,NEWPEN,
         NEWCOST, ACOST : REAL;
         PERTURB        : ASSIGNMENT;

    BEGIN

         PENCOST := PENALTYCALC(ASGNMT, 0);
         ACOST := CCOST + PENCOST;
         FOR I := 1 TO 360 DO                  { M = 360}
```

```
          BEGIN
           GENERATE(ASGNMT, PERTURB, CHANGEINDEX);
           NEWCOMM := COSTCALC(ASGNMT,PERTURB,CCOST,CHANGEINDEX);
           NEWPEN := PENALTYCALC(PERTURB, 0);
           NEWCOST := NEWCOMM + NEWPEN;

           IF (NEWPEN = 0) AND (NEWCOMM < LOW) THEN
               BEGIN                          {Keep track of the best feasible soln}
                BEST := PERTURB;
                LOW := NEWCOMM
               END;


           IF NEWCOST < ACOST THEN            {Accept the new configuration}
               BEGIN                          { only if lower in cost }
                ASGNMT := PERTURB;
                CCOST := NEWCOMM;
                PENCOST := NEWPEN;
                ACOST := NEWCOST
               END
          END;

END;


{*** 13 ***}
PROCEDURE REPORTRESULT(        ASGNMT : ASSIGNMENT;
                        CCOST, PENCOST : REAL       );

     {Creates a text file to report both final solution and BESTSOFAR}

VAR
     I, J       : INTEGER;
     ONESOLN    : SOLUTION;

BEGIN
     FOR I := 1 TO 30 DO
         BEGIN
          WRITELN(RESULT);
          WRITELN(RESULT,'"Coded" solution for this file is  ',ASGNMT[I]);
          WRITELN(RESULT,'Copy of file ',I,' stored in city #j: (1=YES, 0=NO)');

          SEEK(ALLSOLNS,ASGNMT[I]);
          READ(ALLSOLNS,ONESOLN);
          WRITE(RESULT, '  J   ');
          FOR J := 1 TO 12 DO
              WRITE(RESULT, J:5);
          WRITELN(RESULT);
          WRITE(RESULT, ' 0/1  ');
          FOR J := 1 TO 12 DO
              WRITE(RESULT, ONESOLN[J]:5);
          WRITELN(RESULT); WRITELN(RESULT);

         END;
     WRITELN(RESULT, 'The cost of this assignment is:  ',CCOST:16:2);
     WRITELN(RESULT, 'Penalty due to infeasibility is:  ',PENCOST:16:2);
     WRITELN(RESULT);
     WRITELN(RESULT, 'TOTAL cost thus is:  ',(CCOST + PENCOST):16:2)
END;


{*************************************************************************}
{************************** MAIN PROGRAM *********************************}
{*************************************************************************}
```

```
BEGIN
     ASSIGN(ALLSOLNS, 'SOLNFILE.DTA');
     RESET(ALLSOLNS);

     ASSIGN(ALLSIZES, 'SIZES.DTA');          {Enter the data on file size
                                              into sizearr for easy access}
     RESET(ALLSIZES);
     SEEK(ALLSIZES,1);
     FOR I := 1 TO MAXFILES DO
          READ(ALLSIZES, SIZEARR[I]);
     CLOSE(ALLSIZES);

     ASSIGN(ALLCOSTS, 'TELECOST.DTA');
     RESET(ALLCOSTS);

     ASSIGN(CURVE, 'ANCURV89.DTA');          {a text file}
     REWRITE(CURVE);                        {OPEN the file and clear}

     RANDOMIZE;
     COMMCOST := 0;
     PENALTY := 0;

     FOR I := 1 TO 30 DO          { Initialize the array for BESTSOFAR and }
          BESTSOFAR[I] := 0;      { assign LOW an extremely high cost based }
     LOWCOST := 500000;           { on previous experience or calculations }

     INITIALIZE(ASSIGNMT,COMMCOST,PENALTY);
     WRITELN('****GOING INTO HEATUP STAGE****');

     {************* STAGE ONE ************}

     HEATUP(ASSIGNMT,COMMCOST,PENALTY,MEANCOST,STANDEV,BESTSOFAR,LOWCOST);
     PLOT[1] := -1; PLOT[2] := MEANCOST;        {First entry in CURVE will }
     WRITELN(CURVE,PLOT[1]:5:2,PLOT[2]:16:2);   {store MEANCOST from heatup}

     DELTA := 0.5*STANDEV;            {Initialize the range of within count
                                       interval for equilibrium testing }
     TEMP := 20*STANDEV;              {Starting temperature }
     PLOT[1] := TEMP;
     BETA := (LN(0.97))/(-0.7)*STANDEV;  {Limit of temperature decrement
                                      function... Argument of LN is ALPHA}
     UNCHANGED := 0;
     LASTCOST := COMMCOST;
     STOPLOOP := FALSE;
     SUM := 0;
     TABVAL := 1;
     FOR I := 1 TO 100 DO                {Initialize table contents}
          TABLE[I] := 0;

                                         {*****************************}
     WHILE NOT STOPLOOP DO               {**** MAIN ANNEALING LOOP ****}
                                         {********* STAGE TWO *********}
         BEGIN
          WRITELN('ANNEALING...');
          ANNEAL(ASSIGNMT, COMMCOST, PENALTY, TEMP, DELTA,BESTSOFAR,LOWCOST);
          MEANCOST := SUM/100;
          PLOT[2] := MEANCOST;
          WRITELN(CURVE, PLOT[1]:16:2,PLOT[2]:16:2);
          WRITELN('WORKING...Average cost is now = ',MEANCOST:10:6);
          STOPCRITER(STOPLOOP, LASTCOST, COMMCOST, UNCHANGED);
          IF STOPLOOP = FALSE THEN
              BEGIN
               WRITELN('Must anneal further still...');
               TEMPCONTROL(TEMP,STANDEV,BETA);
               PLOT[1] := TEMP;
```

```
            WRITELN('TEMP is now = ', TEMP:16:10)
          END
       ELSE WRITELN('Through with annealing stage.');
      END;

    {************ STAGE THREE ***********}

    WRITELN('*****BEGINNING QUENCHING STAGE*****');

    QUENCH(ASSIGNMT,COMMCOST,PENALTY,BESTSOFAR,LOWCOST);

    {******* WRAP UP *******}

    CLOSE(ALLCOSTS);                              {Close files}
    CLOSE(CURVE);

    ASSIGN(RESULT, 'RESULT89.DTA');
    REWRITE(RESULT);
    REPORTRESULT(ASSIGNMT, COMMCOST, PENALTY);    {Report solution}
    WRITELN(RESULT);
    WRITELN(RESULT, 'COMPARE THIS TO THE BESTSOFAR FOUND IN THE SEARCH:');
    WRITELN(RESULT);
    REPORTRESULT(BESTSOFAR, LOWCOST, 0);          {Report BESTSOFAR}
    CLOSE(RESULT);

    CLOSE(ALLSOLNS)
END.
```