01 Oct 1992

Relaxing Synchronization in Distributed Simulated Annealing

Chul-Eui Hong

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

RELAXING SYNCHRONIZATION IN DISTRIBUTED SIMULATED ANNEALING

CSC 92-18

Chul-Eui Hong* and Bruce McMillin

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401

*This report is substantially the Ph.D. dissertation of the first author, completed Fall 1992.

Abstract

Simulated annealing is an attractive, but expensive, heuristic for approximating the solution to combinatorial optimization problems. Since simulated annealing is a general purpose method, it can be applied to the broad range of NP-complete problems such as the traveling salesman problem, graph theory, and cell placement with a careful control of the cooling schedule.

Attempts to parallelize simulated annealing, particularly on distributed memory multicomputers, are hampered by the algorithm's requirement of a globally consistent system state. In a multicomputer, maintaining the global state S involves explicit message traffic and is a critical performance bottleneck. One way to mitigate this bottleneck is to amortize the overhead of these state updates over as many parallel state changes as possible. By using this technique, errors in the actual cost $C(S)$ of a particular state S will be introduced into the annealing process.

This dissertation places analytically derived bounds on the cost error in order to assure convergence to the correct result. The resulting parallel Simulated Annealing algorithm dynamically changes the frequency of global updates as a function of the annealing control parameter, i.e. temperature. Implementation results on an Intel iPSC/2 are reported.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figures Page

LIST OF TABLES

I. INTRODUCTION

For many important practical or theoretical problems, the objective is to choose a "best" solution out of a large number of candidate solutions or solution space. Such problems are typically known as combinatorial optimization problems. A combinatorial optimization problem is formalized as a pair (S, C), where S is the finite - or possibly countably infinite - set of configurations (also called configuration or search space) and C a cost function, $C: S \rightarrow R$, which assigns a real number to each configuration. For convenience, it is assumed that C is defined so that the lower the value of C, the better (with respect to the optimization criteria) the corresponding configuration. The problem now is to find a configuration for which C takes its minimum value, i.e. an (optimal) configuration i_{opt} satisfying

$$C_{opt} = \min C(i) \quad \text{for } \forall \ i$$

where C_{opt} denotes the optimum (minimum) cost.

Most combinatorial optimization problems are NP-hard [GaJo79]. Such problems involving a combinatorial effort which is bounded by a polynomial function of the size of the problem are unlikely to be well solvable. Therefore approximation algorithms or heuristics are used with no guarantee that the solution found by the algorithm is optimal.

A. EXAMPLES OF COMBINATORIAL PROBLEMS

In this section, stock cutting as well as VLSI placement are defined as model problems.

1. VLSI Placement. [ShMa91] defines the VLSI placement problem as follows: Given an electrical circuit consisting of modules with predefined input and output terminals and interconnected in a predefined way, construct a layout indicating the positions of the modules so that the estimated wire length and layout area are minimized. The inputs

to the problem are the module description, consisting of the shapes, sizes, terminal locations and the netlists describing the interconnections between terminals of the modules. The output is a list of locations for all modules. Figure 1 provides an example of placement, where the circuit schematic of (a) is placed in the standard cell layout style in (b). Optimal chip area usage is required to fit more modules into a given chip area. Minimal wire length is needed to reduce the capacitive delays associated with longer nets and to speed up the operation of the chip.

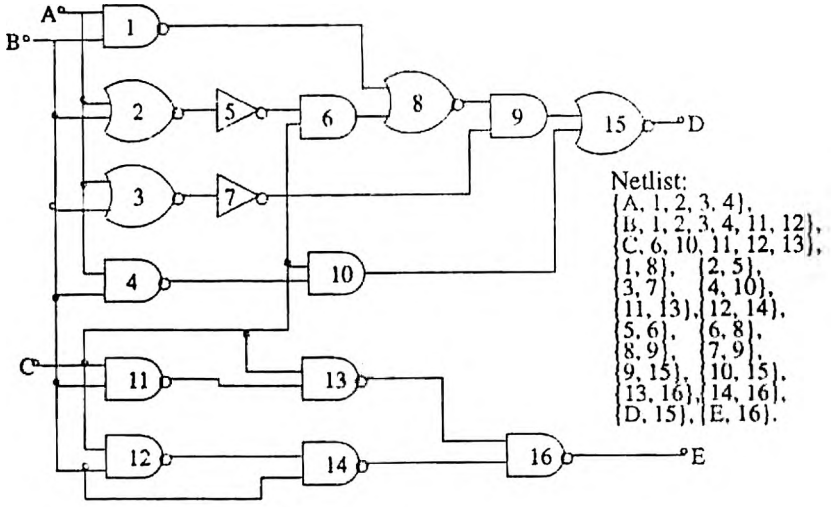
The cost function C is chosen to be the weighted sum of three components

$$C = \alpha f_w + \beta f_a + \gamma f_o$$

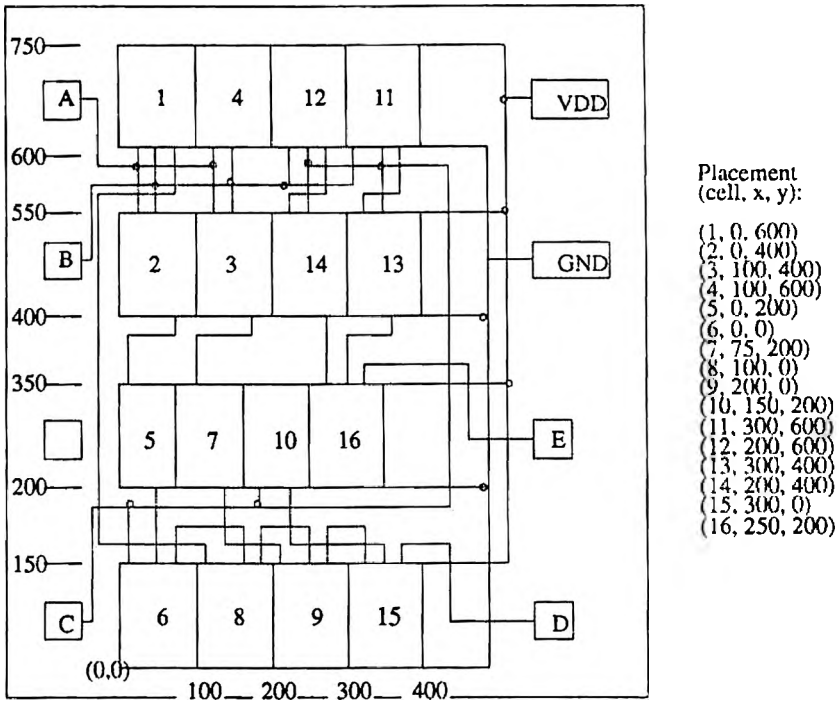
where f_w is the weighted total wire length; the length of a net is esimated by computing the half-perimeter of the bounding box including all the pins connected to the net. f_a is a function of the total area of the chip; the total area of the chip is calculated as the area of its bounding box, i.e., the smallest rectangle including all the cells. f_o is the total overlapping area between cells; this component has to be zero at the end of the algorithm, to achieve a feasible placement. α, β, and γ are nonnegative weights.

2. Stock Cutting. One common combinatorial optimization problem that arises frequently in applications is the stock cutting problem. Rectangular and/or irregular patterns are allocated onto a large stock sheet of finite dimensions in such a way that the resulting scrap will be minimized. This problem is common to many applications in aerospace, shipbuilding, VLSI design, steel construction, shoe manufacturing, clothing and furniture. This problem is commonly known as the stock cutting problem or the 2D bin packing problem.

Because of the nature of the composite stock sheet, most applications do not allow the patterns to be rotated by any random angle. In other words, patterns may rotate only for a limited number of rotation angles. The dissertation addresses using parallel



(a) Input: Netlist



(b) Output: module coordinates

Figure 1. Cell Placement: Problem Definition [ShMa91]

computing techniques to reduce the run time. Also a cost function is made up of the affinity relation between patterns, the distance from the origin, and overlap penalty between patterns [LMPD92]. Consider the cost function C as being

$$C = -\alpha \sum \frac{a_{i,j}}{d_{i,j}} + \beta \sum d_{i_0} + \gamma \sum O_{i,j}$$

where α, β, and γ are positive real numbers indicating the contribution of each of the components in the cost function. $a_{i,j}$ is the affinity relation between pattern i and j. $d_{i,j}$ is the distance between pattern i and j. d_{i_0} represents the distance of pattern i from the origin. $O_{i,j}$ is the overlap between pattern i and j.

A sample pattern placement is shown in Figure 2.

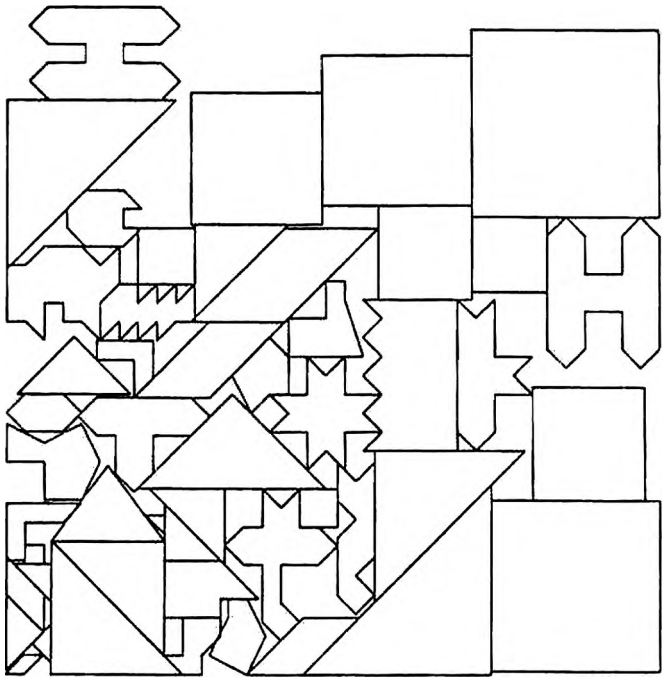


Figure 2. A Sample Placement for 50 Patterns

There are similarities in defining a cost function of VLSI placement and stock cutting problems. Two problems try to minimize the chip area for VLSI placement and stock sheet area for stock cutting. The affinity relation term of stock cutting corresponds to the wire length term of VLSI placement, the cluster term to the chip area term, and the overlap penalty term between patterns to the overlap term between cells respectively. So, the cost functions of stock cutting and VLSI placement problems are defined similarly.

B. COMBINATORIAL OPTIMIZATION METHODS

In this Section, various kinds of combinatorial optimization methods are discussed which can be used to solve the VLSI placement or stock cutting problem.

1. Linear Programming. Linear programming methods have been extensively researched. Work on the stock cutting problem has been done by Gilmore and Gomery [GiGo61, GiGo63, GiGo65], Geoffrion and Marsten [GeMa72], and Haessker[Haes80]. In general these methods involve a solution of the problem through the development of mathematical models. These consist of an objective function that is to be minimized or maximized and constraint functions indicating the limitations on the allowed values of the variables of the objective function. Both the objective and constraint functions are linear functions of variables. Any model may be transferred to the following standard form:

minimize $c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$

subject to

$$a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n = b_1$$

$$a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n = b_2$$

$$\cdots$$

$$a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n = b_m$$

The stock cutting problem may result in a mathematical model consisting of hundreds and even thousands of variables and constraints. The above mentioned work has concentrated on finding special structural characteristics of the model and developing techniques that exploit these structures. These methods have successfully been applied to a broad class of stock cutting problems. There are, however, many real situations for which these methods are not appropriate due to their structure or size. In many cases, this is caused by special restrictions. For such problems, other methods—often heuristic ones— are used.

2. Dynamic Programming. Dynamic programming is one often used heuristic method. Dynamic programming is an algorithm design method that takes a model of the problem and converts it into a series of single stage problems. This transformation is intuitively based on the principle that an optimal set of decisions has the property that whatever the first decision is, the remaining decisions must be optimal with respect to the outcome which results from the first decision. The difficulty is in the time required to determine the optimal decisions. Otherwise, the problem degrades into an enumeration of the decisions and then determining which is the best. This has exponential complexity. Studies of dynamic programming approaches to the stock cutting problem have been done by Beasley [Beas85a, Beas85b] and Sarker [Sark88].

3. Tree-Search. Another class of heuristics often used is the tree-search method. This method enumerates all possible solutions in a tree-like organization. Many different organizations may exist for the solution space. Heuristics exist for finding the solution to the problem by traversing the tree. These heuristics will start out on one path and will terminate when either an optimal solution is believed to have been found or the path is known to result in an unsatisfactory solution. It is difficult to determine which path to start on and once on a particular path, determining whether the path is worth traversing, i.e. if lower costs are possible or whether to proceed on a different path. Work has been

done by Christofides and Whitlock [ChWh77], Hinxman [Hinx80], and Beasley [Beas85a, Beas85b].

4. Iterative Improvement. Another heuristic used is the iterative improvement method [PaSt82, LiKe73]. Application of the method of iterative improvement requires the definition of a solution space, a cost function, and a set of moves that can be used to modify a solution. Define a solution $\mathbf{s}_i = (s_{i0}, s_{i1}, \cdots, s_{im})$ on m variables. A solution $\mathbf{s}'$ is a neighbor (neighboring solution) of a solution $\mathbf{s}$ if $\mathbf{s}'$ can be obtained from $\mathbf{s}$ via one of the moves. In this method, one starts with an initial solution represented by $\mathbf{s}_0 = (s_{00}, s_{01}, \cdots, s_{0m})$. At iteration i, if the current solution is $\mathbf{s}_i$ then its neighbors are examined until a neighboring solution $\mathbf{s}_{i+1}$ is found with a new lower cost. In that case, $\mathbf{s}_{i+1}$ is the new solution and the process is continued to examine the neighbors of the new solution. The algorithm terminates when it arrives at a solution which has no neighboring solutions with a lower cost.

This process tends to minimize the cost but can get trapped in a poor solution, i.e. it may be at a local minimum, but not a global minimum. Figure 3 shows how this may happen. If $\mathbf{s}_0$ is the initial configuration selected, then the iterative improvement method will choose configuration A as the optimum. However, if $\mathbf{s}_0'$ is the chosen initial configuration, then the iterative improvement method will choose configuration B as the optimum. Solution A is a local minimum. All neighboring solutions have a higher cost than A, hence iterative improvement is trapped at A. However, B is the configuration that minimizes the cost; it is the global solution. Thus, iterative improvement is sensitive to the choice of the initial configuration.

To avoid some of the aforementioned disadvantages, one might think of a number of alternative approaches. The iterative improvement algorithm can be used for a large number of initial configurations, N. In this case, for $N \rightarrow \infty$, a global optimum can be

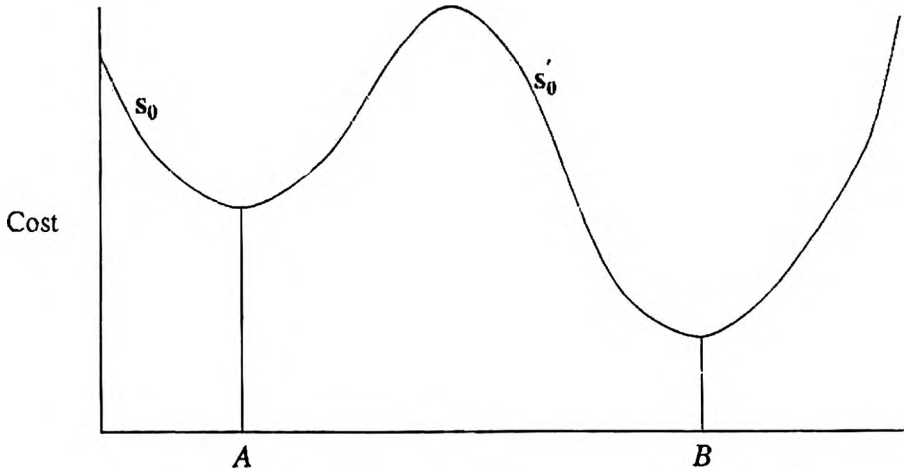


Figure 3. Local Minimum Problem

found with probability 1. The search can be refined by information gained from previous runs. The second approach is that acceptance of transitions which correspond to an increase in the cost function is in a limited way (in an iterative improvement algorithm only transitions corresponding to a decrease in cost are accepted). The second approach is probabilistic, in nature. Next we survey some probabilstic algorithms such as genetic algorithms, Tabu search, and simulated annealing.

5. Genetic Algorithms. Genetic algorithms (GA) are a class of machine-learning techniques that gain their name from a similarity to certain processes that occur in the interactions of natural, biological genes [Morr91, Wayn91]. A GA is based on feedback received from its repeated attempts at a solution. Each attempt a GA makes towards a solution is called a *gene*; a sequence of information that can somehow be interpreted in the problem space to yield a possible solution. Deciding how to encode genes to represent possible solutions in a particular problem space leads to the next requirement: a

suitable objective function. An objective function must be able to interpret the data contained within a gene and decide how good a solution it represents.

Each pass through the set of optimization steps is called a *generation.* The optimization steps in each generation are called *reproduction, crossover,* and *mutation.* In reproduction, the first part of a generation, genes from the previous generation are duplicated and form the new population. The fitness of a gene, conferred by the objective function, is used to decide how likely that gene is to reproduce. Genes that are more fit are more likely to be duplicated; less fit genes have a poorer chance. However, reproduction is ruled by chance, so it is possible for less fit genes to be reproduced. It is important that all sorts of genes, both fit and unfit, are maintained in the population. After reproduction, the genes undergo crossover, where pairs of genes are selected at random. The gene pairs then exchange parts of their sequences. The result is two new genes. In the crossover step, the genes exchange subsequences that contain good information about solutions. Hopefully, this exchanging will result in genes that are better than their ancestors. The final step in a generation is mutation. This optimization entails randomly altering a very small percentage of the genetic sequences present in the population. Mutation may introduce new concepts into the population.

In summary, a GA is actually a series of steps. Initially, a random population of genes is created. Then, an attempt is made to optimize the fitness of the genes by running through generations of optimization steps (reproduction, crossover, and mutation).

6. Tabu Search. The Tabu Search (TS) is an optimization technique based on selected concepts from artificial intelligence [Glov90]. TS is founded on three primary themes: (1) The use of flexible attribute-based memory structures, designed to permit evaluation criteria and historical search information to be exploited more thoroughly than by rigid memory structures (as in branch and bound) or by memoryless systems (as in simulated annealing and other randomized approaches). (2) An associated mechanism of

control—i.e, for employing the memory structures—based on the interplay between conditions that constrain and free the search process (embodied in *tabu restrictions* and *aspiration criteria*). (3) The incorporation of memory functions of different time spans to implement strategies for *intensifying* and *diversifying* the search. Intensifying strategies refer to procedures for reinforcing move combinations and solution features historically found good, while diversification strategies refer to driving the search into new regions.

This TS method is an iterative technique which explores a set of problem solutions denoted by S by repeatedly making moves from one solution s to another solution s' located in the neighborhood $N(s)$ of s. This is a metaheuristic, since at each step it uses a heuristic to move from one solution to the next, guiding the search in S. To avoid being trapped in a local minimum, a guidance procedure must be able to accept a move from s to s' even if the value of the objective function $f(s') > f(s)$. But when a solution s' worse than s may be accepted, cycling may occur, causing the process again to be trapped by returning repeatedly to the same solution. The Tabu Search approach seeks to counter the danger of entrapment by incorporating a memory structure that forbids or penalizes certain moves that would return to a recently visited solution. The notion of using memory to forbid certain moves (i.e., to render them tabu) can be formalized in general by saying that the solution neighborhood depends on the time stream, hence on the iteration number k. That is, instead of $N(s)$ a neighborhood is denoted $N(s, k)$. For instance, suppose memory is employed that recalls solution transitions over some time horizon. $N(s, k)$ is created by deleting from $N(s)$ each solution that was an immediate predecessor of s in one of these transitions. The form of the procedure that uses these modified (tabu) neighborhoods is shown in Figure 4.

It is to be emphasized again that a nontrivial, strategically generated sample of the solution neighborhood is examined at each step, and a best element from the sample is selected (subject to avoiding moves that are classified tabu). The goal is therefore to

Procedure Tabu_Search

Choose an initial solution s in S

$s^* := s$ and $k := 1$

While the stopping condition is not met do

$k := k + 1$

Generate $V^* \subseteq N(s, k)$

Choose the best s' in V^*

$s := s'$

if $f(s') < f(s^*)$ then $s^* := s'$

end while

Figure 4. Procedure of Tabu Search

make improving moves to the fullest extent allowed by the structure of $N(s, k)$, balancing trade-offs between solution quality and computational effort in examining larger samples. A crucial aspect of the procedure involves the choice of an appropriate definition of $N(s, k)$. Due to the exploitation of memory, $N(s, k)$ depends upon the trajectory followed in moving from one solution to the next. As a starting point, consider a form of memory embodied in a Tabu List **T** that records the |**T**| solutions most recently visited, yielding $N(s, k) = N(s) - \mathbf{T}$. Such a recency based memory approach will prevent cycles of length less than or equal |**T**| from occurring in the trajectory. Memory is also used in TS in a kind of learning process: having visited several situations, it is deemed worthwhile to observe whether the good solutions visited so far have some common properties. This generates an intensification scheme for the search. Intensification by itself is insufficient to yield the best outcomes for general classes of optimization problems. The

complementary notion of diversification must be invoked to allow the most effective search over the set S.

7. Simulated Annealing. Simulated annealing was independently introduced by Kirkpatrick et. al. [KiGe83] and Cerny [Cern85]. Application of the simulated annealing heuristic requires (1) formulation of an appropriate cost or energy function, (2) formulation of an appropriate cooling schedule, (3) formalization of a *move* or state perturbation, and (4) parallelization of simulated annealing for speedups.

Simulated annealing is a method of optimization designed to avoid the pitfalls inherent in other optimization methods, such as the iterative improvement approach. It seeks the global or near global minimum of a function without getting trapped in a local minimum. Simulated annealing is well suited to optimize functions of several hundred variables or more, especially when the functions are not smooth, i.e. have many local minimums. Simulated annealing has been used in solving chip placement, image processing, and the traveling salesman problem.

Definition 1-1: When the new cost is greater than the current cost, this proposed move is called a *hill climbing move.*

Simulated annealing is a stochastic algorithm for solving discrete optimization problems. Theoretical studies have shown that a global optimum can be reached with unit probability in infinite time provided a set of conditions are satisfied [Haje85]. One of the major obstacles towards successful application of simulated annealing to combinatorial optimization problems is its massive requirement of computation time, arising from the probabilistic hill climbing nature of the method. Various approaches have been proposed to speed up the simulated annealing process.

1. Careful control of the cooling schedules.
2. Improved move generation.
3. Parallel implementation.

From the mentioned combinatorial optimization methods, linear programming, dynamic programming, and tree search methods are not suitable for large problem sizes. Genetic algorithms, Tabu search, and simulated annealing are popular heuristic algorithms. Among these heuristic algorithms, simulated annealing has a sound theoretical basis for analysis of the convergence to the optimal results. This will aid in the analysis done in this dissertation.

The dissertation is organized as follows: In Section II, the theory of simulated annealing is presented, and convergence properties are proved. Section III discusses the various types of efficient cooling schedules and improved move generations. In Section IV, a taxonomy of parallel simulated annealing techniques is discussed. Section V discusses the previous cost-error-tolerant schemes and the shortcomings of the traditional cost error measurement method. In Section VI, a new cost-error-tolerant scheme is proposed by relaxing synchronization to improve parallel speedups. Section VII presents the implemental details of the parallel space-decomposition algorithm. Section VIII discusses the experimental results of relaxing synchronization. Finally Section IX concludes with future research areas.

II. THEORY OF SIMULATED ANNEALING

The simulated annealing algorithm is based on the analogy between simulation of the annealing of solids and the problem of solving large combinatorial optimization problems (Table I) [KiGe83]. The ground states (global optimum) of a complex physical system can be reached by heating the system up to some high temperature (melting point) and then cooling it slowly, maintaining an equilibrium condition. Thus all possible states are considered allowing to visit the increased energy states, i.e. hill climbing moves. At each temperature value T, the solid is allowed to reach thermal equilibrium, characterized by a probability of being in a state with energy E given by the Boltzmann distribution [LaAa88]

$$\Pr[\mathrm{E} = E] = \frac{1}{Z(T)} \cdot \exp\left(-\frac{E}{\kappa_B T}\right) \tag{2-1}$$

where $Z(T)$ is the normalization factor depending on the temperature T and κ_B is the Boltzmann constant. The factor $\exp\left(-\frac{E}{\kappa_B T}\right)$ is known as the Boltzmann factor.

A. BASIC SEQUENTIAL ALGORITHM

Metropolis et. al. [MeRo53] propose a Monte Carlo method, which simulates this evolution to thermal equilibrium of a solid at a fixed value of the temperature T. In simulated annealing, the initial temperature is set sufficiently high so that all moves are accepted. With a small perturbation of the current state space, a new state is reached. Let ΔC be the difference of the energies (cost) of current state and new state, i.e. the cost of the new state minus the cost of the current state. The probability that a proposed move is accepted or rejected in simulated annealing is determined by the Metropolis criterion:

$$\Pr[\Delta C \text{ is accepted}] = \min\left(1, \exp(-\frac{\Delta C}{T})\right) \tag{2-2}$$

Physical Systems	Optimization Problems
State(Structure)	Configuration
Energy	Cost
Phase Transition	Move Generation
Ground State	Optimal Solution
Quick Cooling(Quenching)	Iterative Improvement
Slow Cooling(Annealing)	Simulated Annealing

Table I. Analogy between Physical Systems and Optimization Problems.

If a proposed move is accepted, then the new state becomes the current state; if the proposed move is rejected, the current state remains unchanged. The above procedure continues until the system reaches thermal equilibrium, i.e. the probability distribution of the states approaches the Boltzmann distribution. The Pascal-like pseudo code for simulated annealing is shown in Figure 5.

[LuMe86] shows that simulated annealing performs better than repeated application of the iterative improvement method. Quantitatively, the upper time bound for iterative improvement is O(S), however, the upper time bound of the simulated annealing method is O(N$\ln|S|$), where **N** is size of neighborhood of any state and S is the total size of the

PROCEDURE SIMULATED ANNEALING

```
begin
  INITIALIZE;
  k:= 0;
  repeat
    repeat
      PERTURB(config. i → config. j, ΔC_ij);      /* evaluation of the cost change */
      if ΔC_ij ≤ 0 then accept
      else if exp(−ΔC_ij/T_k) > random [0,1) then accept;
      if accept then
        UPDATE(configuration j);
    until equilibrium is approached sufficiently closely;

    T_{k+1} := f(T_k);
    k:= k+1;
  until stop criterion == true (system is 'frozen');
end.
```

Figure 5. The Metropolis Procedure

configuration space. ln S is a bound for the number of Markov Chains, which is derived in [LuMe86]. Since, generally, N is polynomial and S is exponential in the size of the input of the problem, execution of simulated annealing takes polynomial time while the iterative improvement algorithm takes exponential time.

B. ASYMPTOTIC CONVERGENCE IN MATHEMATICAL MODEL

The simulated annealing method accepts deteriorations of the system, i.e. configurations that correspond to an increase in the cost function, according to the Metropolis criteria (2-2). This method prevents from getting stuck in local minima, thus enables the final result to arrive at a global minimum.

The sequence of transitions of simulated annealing can be represented by a Markov Chain, since the outcome of a transition depends only on the outcome of the previous one. There are two types of formulation for the simulated annealing algorithm [LaAa88]. One is the homogeneous algorithm, where each Markov Chain is generated at a fixed value of the control parameter (temperature), and the control parameter is decreased between subsequent Markov Chains. Another is the inhomogeneous algorithm, where the value of the control parameter is decreased in between subsequent transitions.

Definition 2-1: Let the transition matrix $P(T)$ be defined as

$$P_{i,j}(T) = \begin{cases} G_{i,j}(T)A_{i,j}(T) & \forall j \neq i \\ 1 - \sum_{l=1,l\neq i}^{|N|} G_{il}(T)A_{il}(T) & j = i, \end{cases} \tag{2-3}$$

with generation probability $G_{i,j}(T)$ for generating configuration j from configuration i, and acceptance probability $A_{i,j}(T)$ for accepting configuration j, once it has been generated from i, and $|N|$ is the size of neighborhood configuration space.

Definition 2-2: (Irreducibility) Given any two states i and j, j is reachable from i.

$$\forall i, j: \; p_{i,j}^{[k]}(T) > 0, \quad \text{for some finite } k$$

Definition 2-3: (Reversibility) The generation probabilities from i to j and from j to i are the same for any state i and j.

$$\forall i, j: G_{i,j} = G_{j,i}$$

which means the Markov process is aperiodic.

Definition 2-4: If the temperature is fixed at T, then the Markov Chain has the stationary transition probability distribution

$$\pi_T(i) = \frac{\pi_\infty(i) \exp\left(-\frac{C(i)}{T}\right)}{Z_T} \tag{2-4}$$

where $Z_T = \sum_{\forall i} \pi_\infty(i) \exp\left(-\frac{C(i)}{T}\right)$ is known as the partition function for the system. π_∞ is the stationary transition probability distribution when the temperature (T) is positive infinity.

Theorem 2-1 [Lam88]: In the time homogeneous Markov Chain, if the irreducibility and reversibility conditions are satisfied and the Markov Chain length is infinite at a given temperature T, then a global optimum can be found with probability one.

Proof: In the time homogeneous Markov Chain, if both the irreducibility condition and the reversibility condition exist, then the Markov ergodic convergence theorem [Sen81] applies. That is, $\lim_{k\to\infty} \Pr[s_k \in i_{opt}] = \sum_{i \in i_{opt}} \pi_T(i)$ where i_{opt} denotes the set of minimum value states, since when the Markov Chain goes to infinity at a fixed temperature, it reaches the stationary condition (Definition 2-4). This implies that

$$\lim_{T\to 0}\left[\lim_{\substack{k\to\infty \\ T_k=T}} \Pr[s_k \in i_{opt}]\right] = 1$$

because at a very low temperature there is no move which deteriorates the state of the system, i.e.,

$$\exp\left(-\frac{\Delta C}{T}\right) \approx 0 \quad , \qquad \Delta C > 0$$

So by letting T tend to 0 as k tends to infinity, the final equilibrium probability distribution is obtained

$$\pi(i) = \begin{cases} |i_{opt}|^{-1} & \text{if } i \in i_{opt} \\ 0 & \text{otherwise} \end{cases} \qquad \square$$

State i is said to be local minimum if no state i' with $C(i') < C(i)$ is reachable from state i at cost $C(i)$.

Definition 2-5 [Haje85]: Define the depth of a local minimum i to be $+\infty$, if state i is a global minimum. Otherwise, the depth of i is the smallest energy E, $E>0$, such that some state i' with $C(i') < C(i)$ can be reached from i at height $C(i) + E$.

So the depth refers to the minimum hill climbing energy to get out of the local minimum (Definition 1-1).

Theorem 2-2: [Haje85] In the time inhomogeneous Markov Chain, in addition to the time homogeneous conditions(irreducibility, reversibility and infinite Markovian transitions), if the temperature is decreased as an inverse logarithm, then the probability of the convergence to the global optimum is one.

Proof: Since the time homogeneous case is proven in Theorem 2-1, only the temperature decrement condition need be proven.

a) For any state i that is not a local minimum

$$\lim_{k\to\infty} \Pr[s_k = i] = 0$$

b) Suppose that the states in B are local minima of depth d. Then

$$\lim_{k\to\infty} \Pr[s_k \in B] = 0$$

if and only if

$$\sum_{k=1}^{\infty} \exp\left(-\frac{d}{T_k}\right) = +\infty.$$

c) (Consequence of (a) and (b)) Let d^* be the maximum of the depths of all states which are local, not global, minima. Let i_{opt} denote the set of global minima. Then

$$\lim_{k\to\infty} \Pr[s_k \in i_{opt}] = 1$$

if and only if

$$\sum_{k=1}^{\infty} \exp\left(-\frac{d^*}{T_k}\right) = +\infty.$$

If T_k assumes the parametric form

$$T_k = \frac{c}{log(k+1)}$$

then condition (c) is true if and only if $c \geq d^*$. So convergence to the global minimum is guaranteed with probability one. □

Since the decrement of temperature is represented as

$$T_k = \frac{\lambda}{\log(k+1)} \qquad \text{for } \lambda \geq d^*, \tag{2-5}$$

this Markov Chain schedule is a logarithmic schedule [Lam88]. Although the

logarithmic schedule can be shown to converge asymptotically to globally optimal solutions, its computation time is high. This is because the logarithmic schedule has to guarantee convergence even in the worst case of stepping into a deep local minimum(d^*), no matter how unlikely such an event is to occur.

An alternate annealing schedule can be based on approximate equilibrium criteria [Whit84]. This annealing schedule no longer guarantees convergence to the global optimal solution, but gives a better performance in practice.

C. STATISTICAL PHYSICS IN SIMULATED ANNEALING

Contrary to the annealing schedule based on Markov Chains, [Whit84] developed another annealing schedule based on the quasi-equilibrium criteria. There are important scales in equilibrium dynamics such as expected cost, variance and specific heat, etc. The behavior of simulated annealing can be characterized with these scales.

Since the sequential annealing schedule is applied to the parallel version, the following statistics are important to analyze the behavior of annealing and to speedups.

Definition 2-6: Let the expected cost in equilibrium be

$$< C(T) > = \sum_{i \in S} C(i) q_i(T) \tag{2-6}$$

where $q(T)$ is the stationary distribution at temperature T,

$$\Pr[\text{ configuration} = i\ ] = q_i(T) = \frac{1}{Q(T)} \cdot \exp\left(-\frac{C(i)}{T}\right)$$

where $Q(T)$ is a normalization constant depending on the temperature.

Definition 2-7: The variance of the cost in equilibrium is

$$\sigma^2(T) = < (C(T) - < C(T) >)^2 > = < C^2(T) > - < C(T) >^2 \tag{2-7}$$

Definition 2-8: The specific heat is

$$\frac{\partial < C(T) >}{\partial T} = \frac{\sigma^2}{T^2} \tag{2-8}$$

A large value of specific heat signals a change in the state of the order of a system, and can be used in the optimization context to indicate that freezing has begun and hence that very slow cooling is required. Equations 2-6 through 2-7 are well known in statistical physics and they serve an important role in the analysis of the mechanics of large physical ensembles at equilibrium. They will be used in deriving the optimal cooling schedules in Section III.

Definition 2-9: [GrHa85] (Ergodicity) Markov process $X(t)$ is ergodic if, with probability one, all its "measures" can be defined or well approximated from a single realization, $x_0(t)$, of the process.

Since statistical measures of the process are usually expressed as time averages, this is often stated as $X(t)$ is ergodic if time averages equal ensemble averages, that is, expected values. The mechanics of a physical multi-particle system is compatible with a statistical ensemble. The time average of a mechanical quantity of the system under macroscopic equilibrium is equal to the corresponding ensemble average (Definition 2-9). So a number of useful macroscopic quantities can be derived given the equilibrium distribution of the system.

In discussing annealing schedules, it is useful to graph $<C(T)>$, the average energy at a fixed temperature vs. the log scale of temperature (Figure 6). This graph will be referred to as the annealing curve. It tracks the progress of the annealing in a way that is tied to the fundamental quantities in the system, and is relatively independent of the details of how the annealing schedule is carried out. Many annealing problems have been

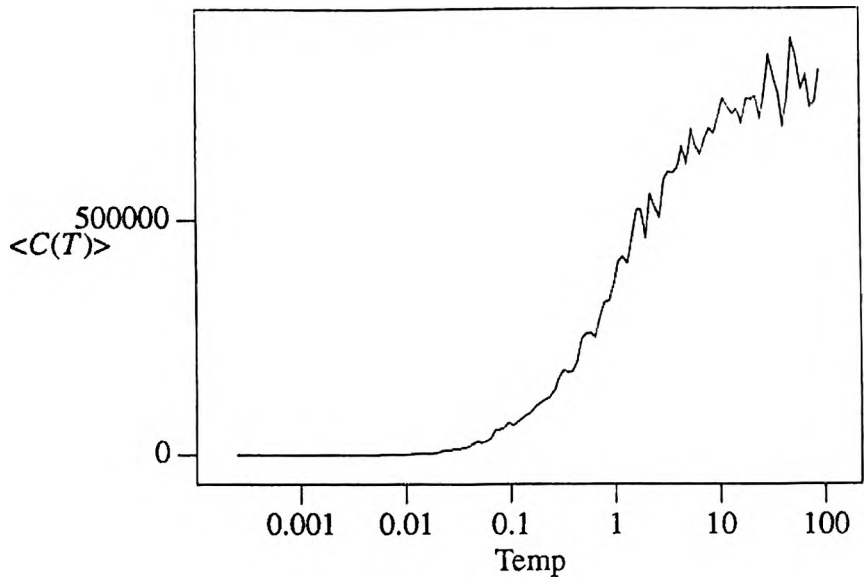


Figure 6. Annealing Curve

studied showing similar annealing curves. The overall features include:

1) There is a minimum cost, C_0, below which the system never goes. This must be the case; the system has some finite global minimum.

2) There is a maximum average cost, C_∞. There certainly must be a maximum cost; the system is finite, and some configuration provides a finite upper bound on the cost. Beyond this, in the infinite temperature limit, any attempted move will be accepted, regardless of whether it increases or decreases the cost. Hence, at very large temperatures, the system moves randomly through its states, and the observed average cost is just the average of the costs of all of the possible states of the system.

3) The annealing curve makes a transition at a low temperature, T_f, from an intermediate cost region to a low cost region, with $< C(T) >$ decreasing with $T > T_f$, and $< C(T) > \approx C_0$ for $T < T_f$.

4) The annealing curve makes a transition at a high temperature, T_0, from the high cost region to the intermediate cost region. This transition fluctuates heavily.

III. SEQUENTIAL SIMULATED ANNEALING

In Section II, it was shown that the simulated annealing algorithm converges to a globally minimal configuration asymptotically with probability one. However, in any real implementation of the algorithm, asymptotic convergence can only be approximated.

Simulated annealing is not an algorithm *per se*, in the sense of a prescribed sequence of operations to solve a set of problem. Rather, it is a paradigm for constructing algorithms to solve optimization problems of a particular character. Design of good annealing algorithms is nontrivial. The design of the herein annealing algorithm is comprised of five parts [KrRu87]:

1) **Configuration Space:** The set of allowed configurations of the system must facilitate easy representation of each state and easy generation of perturbations.

2) **Move Set:** The set of feasible moves must be rich enough so that all reasonable solutions can be found by applying a sequence of moves from this set. In addition, these moves must be relatively inexpensive to compute since many moves will be performed.

3) **Cost Metric:** The metric must be incrementally computable so that the time to evaluate each move is minimal. For placement problems, this metric must be physically meaningful. That is, the placements with the smallest area have the least cost. Since the Metropolis criterion depends on the cost change, cost metric must properly represent the difference of cost between the new and current states.

4) **Cooling Schedule:** The manner in which the temperature T is lowered during annealing (the temperature schedule) is crucial. Starting too cold, stopping too hot, or cooling too quickly all produce suboptimal solutions. Starting too hot, stopping too cold, or cooling too slowly wastes CPU cycles. Analytical and adaptive heuristic techniques have been proposed for controlling this schedule. The number of

moves evaluated at each temperature to approximate equilibrium must also be determined.

5) **Data Structures:** The ability to propose and evaluate moves efficiently hinges on a good representation for the basic objects in the problem. For placement, this means structures for modules and nets arranged so that connectivity and spatial location are quickly assessed.

Although the simulated annealing framework is conceptually straightforward, design of a successful annealing-based algorithm involves considerable engineering judgement in the process of designing the five components just described.

In deciding the optimal cooling schedule, the following parameters must be specified.

1. Initial value of the control parameter (temperature): T_0
2. Final value of the control parameter: T_f
3. Length of the Markov chain at any control parameter T_k: L_k
4. Decrement of the temperature: $T_{k+1} = f(T_k)$

Particular choice for these four parameters is referred to as a *cooling schedule*. The choice of these parameters is very important to balance the quality of the final result and the computation time. Additionally, for efficient operation, a controlled move generation strategy is needed.

A. INITIAL VALUE OF THE CONTROL PARAMETER: T_0

The initial value of temperature, T_0, is determined in such a way that virtually all transition moves are accepted. This means that for $T_0 \rightarrow \infty$, the stationary distribution is given by the uniform distribution of the set of configurations S.

[JoAr87] determined T_0 by calculating the average in cost, $\overline{\Delta C}^{(+)}$ which counts only the positive cost change, for a number of random transitions and solved T_0 from $\chi_0 = \exp\left(-\frac{\overline{\Delta C}^{(+)}}{T_0}\right)$, where χ_0 is initial acceptance ratio (usually 0.8 to 0.9). So

$$T_0 = \frac{\overline{\Delta C}^{(+)}}{\ln(\chi_0^{-1})} \tag{3-1}$$

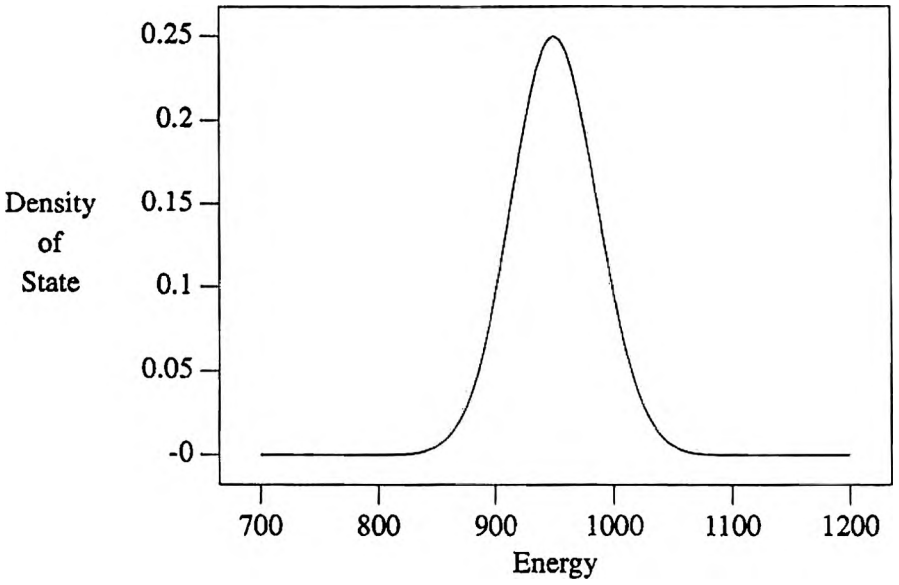


Figure 7. Density of States for a Typical Problem [Whit84]

A more elaborate approach by [Whit84] was based on the configuration density function, $\omega(C)$, which is the number of possible states of the system, per unit cost(energy), at cost C.

$$\omega(C) = \frac{1}{|S|}\{i \in S |\ C \le C(i) \ge C + dC\} \tag{3-2}$$

Using the acceptance probability, the probability density at a given temperature T is given by

$$\Omega(C,T) = \frac{\omega(C)\exp\left(-\frac{C - C_{opt}}{T}\right)}{Z(T)} \tag{3-3}$$

where $Z(T) = \int_{-\infty}^{\infty} \omega(C')\exp\left(-\frac{C' - C_{opt}}{T}\right)dC'$. C_{opt} is the cost of the optimal state.

So the expected cost in equilibrium

$$< C(T) > = \int_{-\infty}^{\infty} c'\Omega(c',T)dc' \tag{3-4}$$

Assuming that $\omega(C)$ is Gaussian near the average energy $\bar{C}$ and a standard deviation σ_0 at T_0.

$$\omega(C) \propto \exp\left(-\frac{(C - \bar{C})^2}{2\sigma_0^2}\right) \tag{3-5}$$

When $\omega(C)$ is given by equation (3-5), equation (3-4) becomes [Reif65]

$$< C(T) > = C_{opt} + \sqrt{2}\cdot\sigma_0\cdot\left(y + \frac{e^{-y^2}}{\sqrt{\pi}(1+\mathrm{erf}(y))}\right) \tag{3-6}$$

where $y \equiv \frac{1}{\sqrt{2}}\left(\frac{\bar{C} - C_{opt}}{\sigma_0} - \frac{\sigma_0}{T}\right)$

For large T, $|y| \gg 1$ and $y > 0$. So equation (3-5) yields [Whit84]

$$< C(T) > \approx \bar{C} - \frac{\sigma_0^2}{T} \tag{3-7}$$

The characteristic high temperature T_0 can be identified with the temperature at which $< C(T_0) >$ is just within thermal noise (standard deviation) of $\bar{C}$. This is the temperature

at which $(\bar{C} - < C(T_0) >) \leq \sigma_0$, that is,

$$T_0 \geq \sigma_0$$

So the system is "hot enought" at T_0.

Huang et. al. [HuRo86] proposed the initial temperature similar to White's

$$T_0 = -\frac{3\sigma_0}{\ln P} \tag{3-8}$$

where P is the acceptance ratio at high temperature. P is usually set to 90%. So at high temperature, $3\sigma_0$ worse cost is accepted with probability P. To calculate σ_0, an initial exploration of the configuration space is performed. During the exploration, all the generated states are accepted, i.e., the temperature is assumed infinite. In this manner. the standard deviation (σ_0) is calculated at the initial temperature.

B. FINAL TEMPERATURE: T_f

[Sech87] proposed that the stopping criterion be implemented by recording the value of the cost function at the end of each temperature during the annealing process. The annealing process ends when the value of the cost function has remained unchanged for a sufficient number of Markov chains.

[Whit84] considers a state with cost C_1 that is just above a nondegenerate local minimum with cost C_0. If $\exp\left(-\frac{C_1 - C_0}{T_f}\right) \leq \frac{1}{L}$, then there is no hill-climbing process, where L is the length of Markov chain. This means that the probability of accepting a transition from the state of cost C_0 to the state of C_1 is smaller than $1/M$. So

$$T_f \leq \frac{C_1 - C_0}{\ln M}. \tag{3-9}$$

Finally, [HuRo86] proposed that when the equilibrium is established, the difference of maximum and minimum costs among the accepted states at that temperature be compared with the maximum change in cost in any accepted move during the constant temperature. If they are the same, apparently all the accessible states have comparable costs and there is no need to continue the simulated annealing process. The temperature is then set to zero and the algorithm becomes a standard greedy random selection algorithm.

C. LENGTH OF MARKOV CHAIN: L_k

A sufficiently long chain length is required to reach quasi-equilibrium at each temperature. L_k is the length of the Markov Chain at a given temperature. The simple choice for the fixed L_k is that, for each value T_k, a minimum amount of transitions should be accepted. However, as T_k approaches 0, transitions are accepted with decreasing probability. Thus L_k is ceiled by some constant. Usually L_k is some integer multiple of the maximum number of neighboring states.

The Markov Chain length L_k as a function of temperature T_k is defined by Huang. Huang et. al. [HuRo86] proposed to detect thermal equilibrium by keeping track of two counters for the number of accepted moves. They record energy values within and without the interval $[\bar{C}(T) - \alpha\sigma,\ \bar{C}(T) + \alpha\sigma)$, where $\bar{C}(T)$ is the measured average energy at temperature T. If the number of accepted moves within the interval reaches its target first, the system is considered in thermal equilibrium; if the number of accepted moves without interval reaches its target first, both counters are reset to zero and the counting is initiated again.

The target values for the within and without interval counters are set to 3erf(0.5)N and 3(1-erf(0.5))N, respectively, where N is a parameter measuring the size of the optimization problem. In order to guarantee the validity of $\bar{C}(T)$, this method invokes this detection mechanism only after a total of N moves has been accepted. It is possible that

the system may never arrive at thermal equilibrium using this scheme. Therefore, the system is also considered in thermal equilibrium if the number of steps is greater than the maximum generation limit, M, where M is the number of states that may be reached in one move.

D. DECREMENT OF TEMPERATURE: $T_{k+1} = f(T_k)$

Small temperature decrements are used to avoid the necessity of long Markov chains for re-establishing equilibrium at each new value of the control parameter (temperature). A frequently used decrement rule is given by

$$T_{k+1} = \alpha \cdot T_k , \quad k = 0, 1, 2, \cdots \tag{3-10}$$

where α is a constant smaller than but approximately 1. Kirkpatrick et. al. [KiGe83] proposed this rule with $\alpha = 0.95$. However, the efficient decrement ratio is dependent on the cost distribution. In [Sech87], various decrement ratios based on the annealing curve are used. Moreover when the annealing is done at a high temperature, α can be low.

The variable decrement of temperature is based on the assumption that the stationary distributions for succeeding values of the control parameter should be nearly the same,

i.e. $$\forall i \in S: \frac{1}{1+\delta} < \frac{q_i(T_k)}{q_i(T_{k+1})} < 1+\delta, \quad k = 0, 1, 2, \cdots \tag{3-11}$$

for some small real number δ.

The equation (3-11) is satisfied if

$$\forall i \in S: \exp\left(-\frac{C(i) - C_{opt}}{T_k}\right) < (1+\delta)\exp\left(\frac{C(i) - C_{opt}}{T_{k+1}}\right) \tag{3-12}$$

Assuming that the value of the cost function is normally distributed for a given value of T, then the $\Delta C_{i_{opt}i}$ is normally distributed with mean $\mu(T_k) = \bar{C}(T_k) - C_{opt}$ and variance

$\sigma^2(T_k) = \bar{C}^2(T_k) - (\bar{C}(T_k))^2$ [AaLa85]. It is shown that

$$T_{k+1} > T_k\left(1 + \frac{\ln(1+\delta)T_k}{3\sigma(T_k)}\right)^{-1} \tag{3-13}$$

In [HuRo86], *a cooling schedule* is derived using the annealing curve [Whit84] — a curve of the equilibrium average energy, $< C(T) >$, versus the logarithm of the temperature, $\log(T)$ — to guide the temperature change. The idea is to control the temperature so that $< C(T) >$ decreases in a uniform manner. From the slope of the annealing curve,

$$\frac{\partial < C(T) >}{\partial \ln(T)} = T \cdot \frac{\partial < C(T) >}{\partial T} \tag{3-14}$$

From the well known equation (3-8):

$$\frac{\partial < C(T) >}{\partial T} = \frac{\sigma^2}{T^2}$$

it follows that

$$\frac{\partial < C(T) >}{\partial \ln(T)} = \frac{\sigma^2}{T}$$

The slope itself can be approximated by $\Delta C(T) / (ln(T_{k+1}) - \ln(T_k))$, where $\Delta C(T)$ is the change in the average cost at a different temperature. Hence,

$$\frac{\Delta C(T)}{\ln(T_{k+1}) - \ln(T_k)} = \frac{\sigma^2}{T_k}$$

which leads to

$$T_{k+1} = T_k \cdot \exp\left(\frac{T_k \cdot \Delta C(T)}{\sigma^2}\right)$$

To maintain quasi-equilibrium, the expected decrease in the average cost must be less than the standard deviation of the cost. $\Delta C(T) = -\lambda\sigma$ where $\lambda \leq 1$. A typical value of λ is 0.7. Finally,

$$T_{k+1} = T_k \cdot \exp(-\lambda \frac{T_k}{\sigma}) \tag{3-15}$$

In the actual implementation, the ratio of T_{k+1}/T_k is lower bounded by a small number (typically 0.5) to prevent a drastic decrease in temperature caused by the flat annealing curve at high temperatures.

E. CONTROLLED MOVE GENERATION STRATEGY

With a controlled move generation strategy, the simulated annealing algorithm can approach equilibrium faster and the acceptance ratio can be increased. This results in faster annealing.

Greene and Supowit [GrSu84] proposed to bias the generation of transitions by using a list of the effects of each possible transition on the cost function. Suppose that configuration i is given and let $W_{ij}(T)$ for each of the R possible transitions be given by

$$W_{ij}(T) = \min\{1, \exp(-(C(j) - C(i))/T)\}, \tag{3-16}$$

where T is the current value of the control parameter. Instead of the traditional generation, where G_{ij} is given by

$$G_{ij} = R^{-1} \tag{3-17}$$

$G_{ij}(T)$ is defined as

$$G_{ij}(T) = \frac{W_{ij}(T)}{\sum_{k=1}^{R} W_{ik}(T)}, \qquad \forall \in R_i$$

and $A_{ij} = 1$ (for all i and j). i.e. all transitions are accepted once they are generated. Thus, if the length of a Markov chain is determined by a minimal number of accepted transitions L, then each Markov chain will have length L. It is called the rejectionless method.

[Whit84] proposed another method. During execution of the simulated annealing algorithm, the value of the control parameter is gradually decreased. The lower the value of T, the lower the probability for acceptance of a transition, which corresponds to a large increase in the cost function. For high values of T, where virtually all transitions are accepted, it would therefore be helpful to bias the generation of transitions in favor of large transitions. This suggests a strategy for the move set being adapted to the temperature scale.

At the highest temperatures, only the largest cost change moves (i.e. the ones with the largest $< |\Delta C| >$) are made. Smaller cost change moves change the cost by trivial amounts, and are not useful in helping the system come to equilibrium. As the temperature decreases, use smaller cost change moves (i.e. ones with smaller $< |\Delta C| >$), appropriate to that temperature. Larger cost change moves change the cost by such a larger amount, that they are accepted only with an extremely low probability. Finally White claims that for each value of T, the change of the cost must be fairly continuous as it proceeds to equilibrium at that temperature. Thus each move in the move class should change the cost by somewhat less than the average thermal fluctuations at equilibrium. This means that, at any temperature T, move classes should be selected which give

$$< |\Delta C| > \approx T, \tag{3-18}$$

to within a standard deviation in the distribution of $|\Delta C|$'s and such that

$$< |\Delta C| > \; < \sqrt{< C^2(T) > - (< C(T) >)^2} \; . \tag{3-19}$$

[Sech88] proposed the range-limiter window, in order to generate moves which have a reasonable probability of acceptance in the placement problem. When an object i is selected for displacement, the range-limiter window is centered at (x_0, y_0), corresponding to the center of object i. The randomly selected new location for object i must lie within the range-limiter window. At the beginning of the annealing process, the window size is

set to be large enough to contain all the objects and it shrinks slowly as the temperature decreases. In fact, the height and width of the window are proportional to the logarithm of the temperature.

$$W_x(T) = W_x^0 \cdot \frac{\rho^{\log_{10}(T)}}{\lambda}$$

$$W_y(T) = W_y^0 \cdot \frac{\rho^{\log_{10}(T)}}{\lambda} \tag{3-20}$$

W_x^0 represents the window span in the x-direction at the initial temperature(T_0). W_y^0 represents the window span in the y-direction at the initial temperature. The value of λ was chosen such that for the initial temperature, the term on the right most side of the above equations is normalized to one. That is,

$$\lambda = \rho^{\log_{10} T_0} \tag{3-21}$$

where ρ is in the range $1 \leq \rho \leq 4$.

Despite this tuning, simulated annealing is still time-consuming. In the next section the speedup of simulated annealing through parallelism will be explored.

IV. PARALLEL SIMULATED ANNEALING

The development of distributed multiprocessors depends on the recent appearance of powerful microprocessors and inexpensive memory. It also has lead to the development of commercial, microprocessor based, shared memory multiprocessors. In a shared memory system, processors are connected via a fast bus in small scale systems, while large multiprocessors contain hundreds or thousands of processors connected to memory modules via an interconnection network.

The most striking difference between distributed memory multiprocessors and shared memory multiprocessors is network latency. If memory is truly "shared", it must be possible to access any portion of memory within a few instruction cycles. This implies that the multiprocessor interconnection network must be faster and correspondingly more expensive than a multicomputer interconnection network. This high performance, high cost interconnection network permits parallel tasks to interact much more frequently using the shared memory than is possible by passing messages on a distributed multiprocessors network [ReFu87]. However, shared memory is limited in its ability to provide massive computing speedups.

Definition 4-1: The *speedup* achieved by a parallel algorithm running on p processors is the ratio between the time taken by that parallel computer executing the fastest serial algorithm and the time taken by the same parallel computer executing the parallel algorithm using p processors.

To reduce execution time, researchers have parallelized simulated annealing. Since simulated annealing is an inherently sequential process, it is hard to implement this algorithm in parallel without changing the final result. So great care has been devoted to state generation, control of the cost error, and dividing the state space among the processors.

Consider following questions [Gree89]:

1. How is the state space divided among the processors?
2. Does the state generator for the parallel algorithm produce the same neighborhood as the sequential algorithm? How are states generated?
3. Can moves made by one processor cause cost-function calculation errors in another processor? Are there any mechanisms to control these errors?
4. What is the speedup? How does the final cost vary with the number of processors? How fast is the algorithm, when compared to an optimized sequential program?

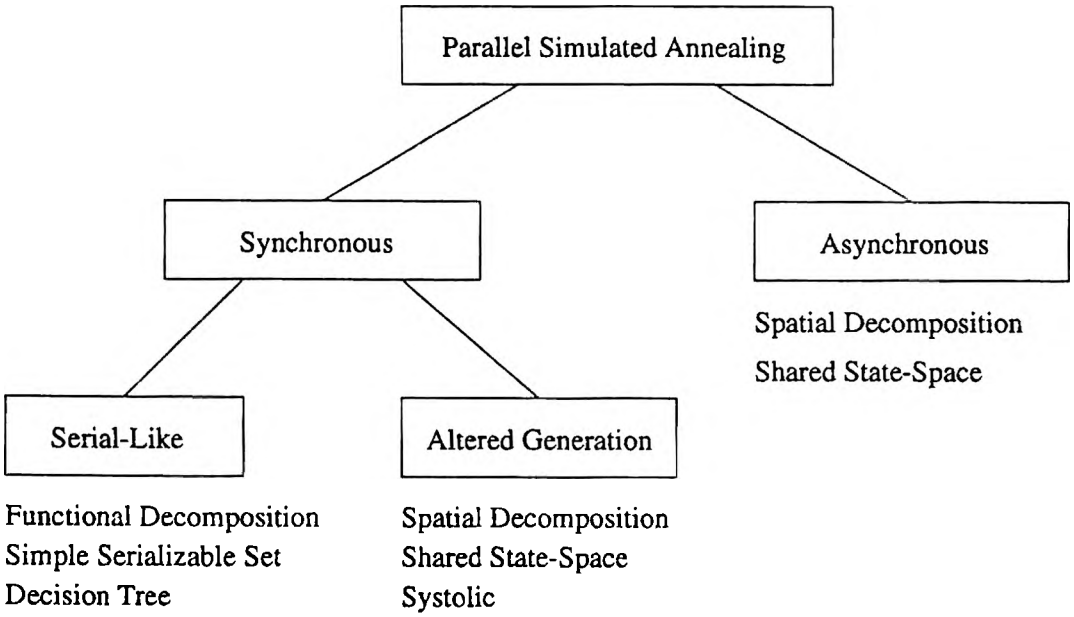


Figure 8. Parallel Simulated Annealing Taxonomy [Gree89]

Parallel simulated annealing can be divided into synchronous and asynchronous [Gree89]. In synchronous, there are functional decomposition, simple serializable set, decision tree, altered spatial decomposition and systolic methods. In asynchronous, there

are spatial decomposition and shared state space methods. In synchronous algorithms, cost calculations are correct. However careful consideration is still required for generating the move set. But synchronous algorithms have a scalability problem; so only limited speedup is possible. In asynchronous algorithms, cost calculations are not correct, so an error in the cost must be tolerated to retain the convergence properties of the synchronous algorithms.

A. DISTRIBUTED VS. SHARED MEMORY IN SIMULATED ANNEALING

Different schemes of state maintenance and algorithm implementation are used between distributed and shared memory. In parallel applications of simulated annealing, it is important to note the problem of interacting parallel moves, i.e. in the case of the placement problem, if two processors propose moves simultaneously to the same location which is empty, the object will be overlapped. This problem is most easily solved on shared memory machines. In such architectures, each processor can easily and quickly access the global state of the layout configuration. Therefore, interacting moves can be identified, or the problem may be repartitioned to minimize future erroneous moves.

On the other hand, in a distributed memory architecture such as a hypercube, there is no globally available, centrally located system state. Such state information is distributed among the processors. So updating the entire global state involves explicit message traffic, potentially between distant nodes, and is a critical bottleneck. To mitigate this bottleneck, it becomes necessary to amortize the cost of these state updates over as many parallel moves evaluations as possible, and to optimize the speed of each required update. [JaRu87] used two kinds of update schemes. One is a global state updating, while the other is a partial state updating, where a processor distributes its local state information to only a selected subset of the other processors. By careful design of the topological arrangement of the processors, all processors will eventually receive this update information, although it may be delayed.

B. SERIAL-LIKE ALGORITHMS

Serial-like convergence algorithms maintain the convergence properties of sequential annealing. Three synchronous parallel algorithms preserve the convergence properties of sequential simulated annealing: functional decomposition, simple serializable set, and decision tree decomposition.

1. Functional Decomposition (FD). Functional decomposition algorithms exploit parallelism in calculating the cost function. [KrRu87] used a functional decomposition algorithm for VLSI circuit placement in parallel on shared memory. [KrRu87] divided individual moves into subtasks and distributed them across cooperating processors. The work of a move consists of selecting a feasible perturbation in the move set, evaluating the cost change, deciding to accept or reject, and updating a global data-base. Since shared memory is employed, no configurations have to be communicated.

These tasks are fine-grain parallelism and the processors have to synchronize at least once per move evaluation. Parallelism is strictly limited since only a limited number of connected cells and nets are perturbed. One can extract a maximum speedup of $1+2j+k$, where j is the average cells affected per move, and k is the average wires affected per move. The disadvantage is that as the number of processes cooperating to evaluate each move increases, synchronization overhead also increases, reducing the expected speedups from greater parallelism. [KrRu87] conjectured that the parallelism is not more than 10. The advantage is that FD algorithms are temperature independent, so when the acceptance rate is high at high temperatures, a speedup is possible.

2. Simple Serializable Set (SSS). The SSS algorithm computes several complete moves in parallel. If a collection of moves affect independent state variables, distinct processors can independently compute the cost change (ΔC) without communicating.

Begin

Choose some random initial configuration s';

Repeat

$s \leftarrow s'$;

Parallel Block Begin

$s' \leftarrow$ some random neighboring configuration(s);

$C_0 \leftarrow$ block-length-penalty(s);

$C_{1,0} \leftarrow$ overlap for affected cell c_0 before move;

$\cdots C_{1,J} \leftarrow$ overlap for affected cell c_J before move;

$C_{2,0} \leftarrow$ overlap for affected cell c_0 after move;

$\cdots C_{2,J} \leftarrow$ overlap for affected cell c_J after move;

$C_{3,0} \leftarrow$ length change for affected wire w_0;

$\cdots C_{3,k} \leftarrow$ length change for affected wire w_k;

End Parallel Block

$\Delta C \leftarrow C_0 + (C_{1,0} + \cdots + C_{1,J}) - (C_{2,0} + \cdots + C_{2,J}) + (C_{3,0} + \cdots + C_{3,k})$;

if accept($\Delta C, T$) then

Parallel Block Begin

update overlap values;

update blocks and cells;

update wire w_0;

$\cdots$ update wire w_k;

End Parallel Block

recompute T;

until stop criteria;

End;

Figure 9. Algorithm FD for VLSI Placement

The moves can be concluded in any order, and the result will be same. The simplest is a collection of rejected moves because the order is irrelevant and the outcome is always the starting state. The sequence formed by taking all rejected moves, and appending one accepted move is always serializable. At low temperatures, the acceptance rate is often very low. So in [KrRu87], the SSS algorithm uses this property in low temperature regions.

[RoDr90] used the SSS algorithm over the entire temperature region, $T_0 \rightarrow T_\infty$. They implemented the parallel algorithm on a distributed memory transputer array. In the low temperature region, if $a(T) < 1/N$ where $a(T)$ is the acceptance ratio at temperature T and N is the number of processors, less than one move out of N will be accepted. Thus, the processors attempt moves on their own variables, asynchronously, in parallel, until one of the N processors accepts a move. When an accepted move is found, the processors are synchronized, their memories are updated with the new configuration, and the next evaluation step take place. For the performance evaluation, if there are no acceptance moves, $M = 0$, then count N moves as progress towards equilibrium. If $M > 0$, then the $N - M$ rejected moves and one accepted move are counted as progress to equilibrium. So the expected speedup

$$E[|S^{ss}|] = \Pr(M = 0) \cdot N + \sum_{m=1}^{P} \Pr(M = m) \cdot (N - m + 1)$$

$$= N \cdot (1 - a(T)) + 1 - (1 - a(T))^N \qquad (4\text{-}1)$$

where $|S^{ss}|$ means the size of the serializable subset.

In the high temperature region($a(T) > 1/N$), each processor is allowed to evaluate one move only and waits until all the other processors complete their evaluation. Then, one of the accepted moves is selected at random, the processors memories are updated with the new configuration, and the next evaluation step takes place. For the performance evaluation, number the processors in an arbitrary, but definite order, from 1 to N. Then

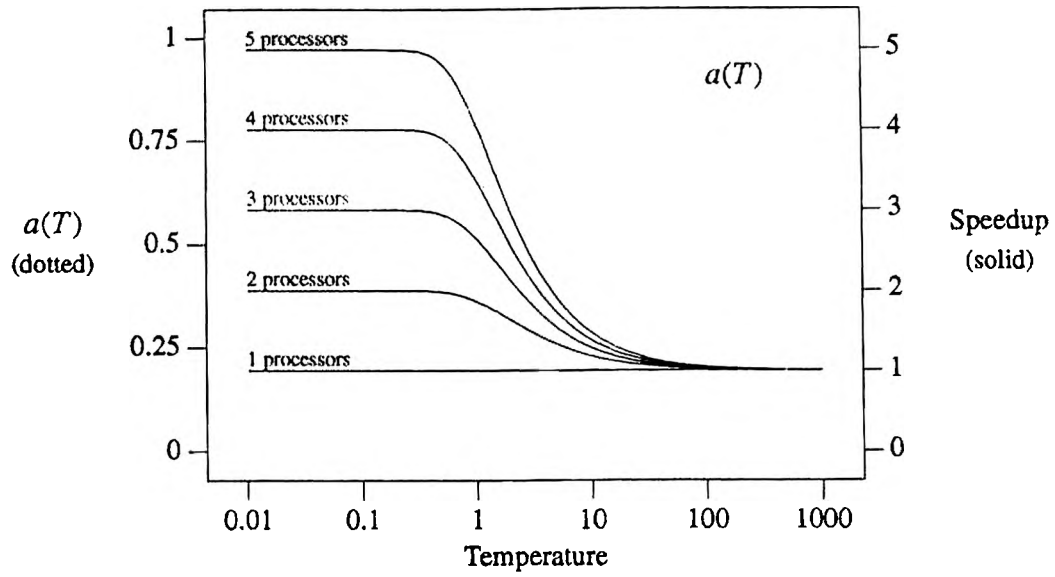


Figure 10. Performance Model for SSS [KrRu87]

denote by n, the number of the first processor in the list which accepts a move and count n moves as progress to equilibrium. The average speedup

$$E(n) = (N+1)/(N-r+1) \tag{4-2}$$

where r moves out of P are rejected.

Algorithm SSS has limitations. Some annealing schedules [Lam88] maintain an acceptance rate at relatively high values (0.44) throughout the temperature range by adjusting the generation function. An important observation is that FD and SSS algorithms are not mutually exclusive. With sufficient parallel resources, one could evaluate several moves in parallel and also divide each of these parallel moves into cooperating subtasks.

3. Decision Tree Decomposition (DTD). [ChEd88] proposed the DTD algorithm. Each move consists of move, evaluate, decide and update. Because decisions are binary(accept the move or reject the move), the SA algorithm can be viewed in terms of selecting a particular path through a binary decision tree. Figure 11 shows a binary tree in which each of the nodes can be viewed as performing a move/evaluate/decide task. The two children of each node correspond to the two possible decision results. The paths correspond to available communication paths between the processors.

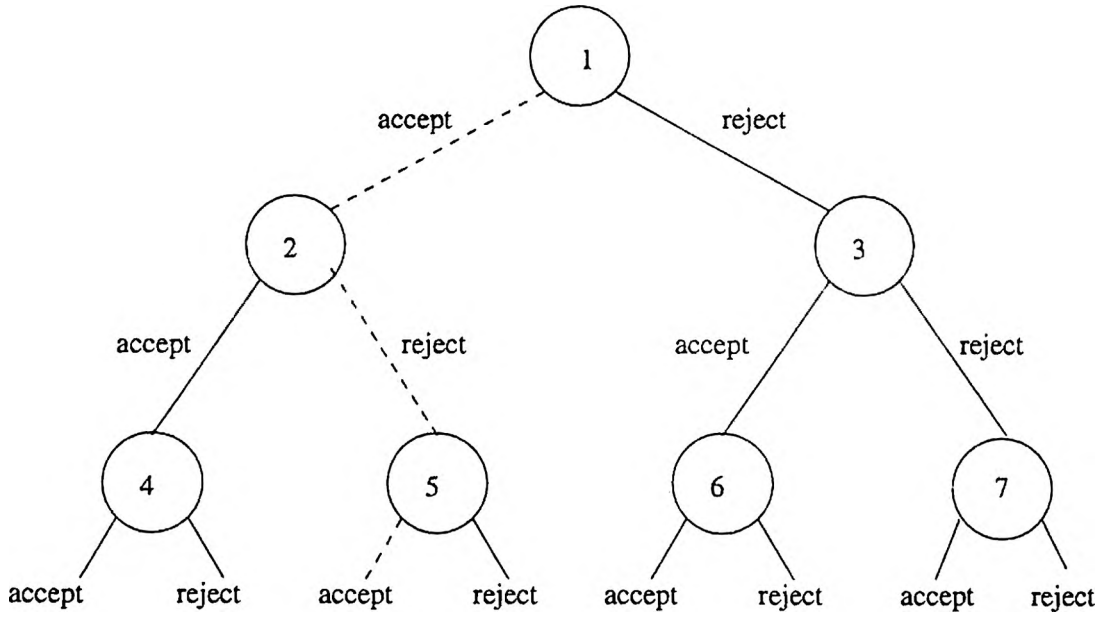


Figure 11. Decision Tree Decomposition [ChEd88]

A vertex generates a move in time t_m, evaluates the cost in time t_e, and decides whether to accept in time t_d.

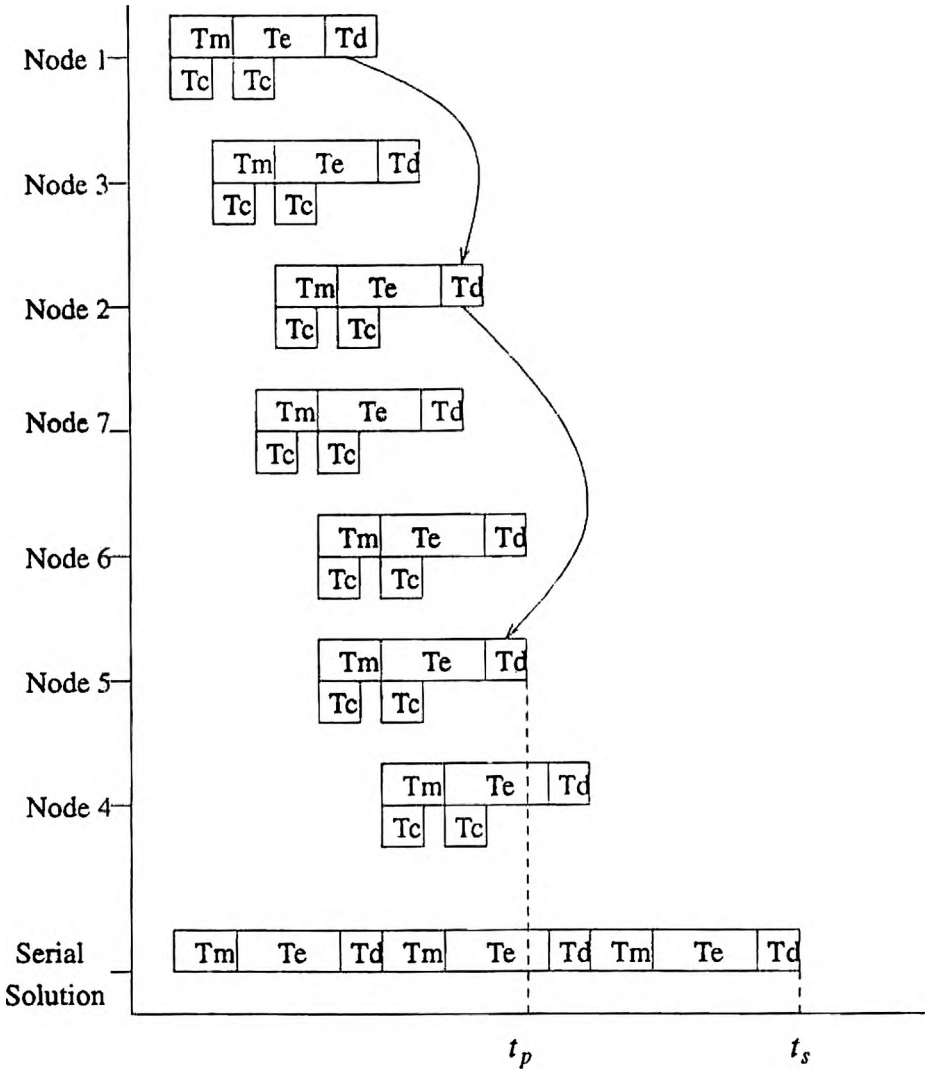


Figure 12. Timing Diagram [ChEd88]

1. Each processor along a reject path can start performing its move/evaluate/decide task as soon as its parent node has indicated a start of the process(i.e. after a T_c delay). That is, since it is on the reject path, no state change information must be communicated. This can be seen in Figure 12 in the start of the move task(T_m) on Node 3 relative to its parent on Node 1.

2. Each processor along an accept path can start performing its move/evaluate/decide task as soon as the move taken by its parent has been communicated. That is, since the new state of the parent has been communicated, a new move can be undertaken by the child processor without inducing a state conflict. This can be seen in Figure 12 in the start of the move task on Node 2 relative to its parent on Node 1.

At high temperatures, the acceptance rate is nearly 1, so the tree will be heavy on the accept side. As the temperature decreases, the tree will become heavy on the reject side. In the complete binary tree, speedup is expected $\log_2 N$(the depth of tree) for N processors. However, by skewing the tree, a rough estimate of the speedup possible over all temperatures can be obtained by assuming a linear, symmetric acceptance probability/temperature curve. At each end of the temperature schedule, the tree depth is N, and at the mid-point of the temperature the tree depth is $\log_2 N$. So the average speedup will be $(N + log_2 N)/2$.

In numerical simulations, where $T_d = T_c$, $T_m = 2 \cdot T_d$, $T_e = K \cdot T_m$, the speedup fall flat. In VLSI placement since $t_m \gg t_e$, the speedup is less than 2.5 on 30 processors. Interestingly, at low temperatures, the DTD algorithm is same to SSS algorithm. However, at high temperatures small evaluation/decision time savings can be achieved using the DTD algorithm.

C. ALTERED GENERATION ALGORITHMS

State generation can be modified to reduce inter-processor communication. These altered generation methods change the pattern of state space exploration, and thus change the solution quality and execution time.

1. Spatial Decomposition. In spatial decomposition techniques, state variables are distributed among the processors, and variable updates are transmitted between processors as new states are accepted if the proposed move does not conflict with another move.

Since parallel moves are done synchronously, cost calculations are always correct. Spatial decomposition techniques are typically implemented on message-passing multiprocessors.

a) *Cooperating Processors (CP).* Processors must carefully coordinate move generation to avoid conflicting moves for the correct cost calculation. A cooperating processor algorithm disjointly partitions state variables over the processors. A processor that generates a new state notifies other affected processors. Then those processors synchronously evaluate and update the state.

[BrBa88] used this algorithm to minimize the number of routing channels for a VLSI circuit on the Intel iPSC/2. The basic idea of the algorithm is to start with the number of tracks equal to the channel density, assign sets of adjacent tracks to processors arranged in a linear reflected gray code fashion, selectively move nets in parallel between tracks of nodes which are paired up, and broadcast information to other nodes to update their data structures. Processors proceed in a lockstep communication pattern. At each step, all processors are divided into master-slave pairs. The master processor randomly decides among four move classes:

Intra-displace: each node of a pair performs a displacement move within its own set of subnets and tracks.

Inter-displace: master node displaces a subnet from its track domain to a track within the domain of the slave node.

Intra-exchange: each node of a pair performs an exchange move within its own set of subnets and tracks.

Inter-exchange: master and slave nodes each select a subnet to exchange each other.

Since the move generation is changed, the speedup cannot be calculated directly. Experiments indicate superlinear speedups, from 2.7 on 2 processors to 17.7 on 16

processors. This results are from a near optimal initial state and at high temperatures, the Markov Chain length is too small to reach the equilibrium. So the near optimal initial state preserved to the final result. Since [BrBa88] restrict the move only to the neighbor processors, using a greater Markov Chain length than that of the sequential algorithm, the equilibrium condition is reached.

b) *Independent Processors (IP).* In the IP algorithm, processors must not generate moves that affect other processors' state variables, that is, each processor generates state changes which affects only its own variables. To guarantee that the annealing algorithm searches the entire state space, state variable redistributions must be done periodically.

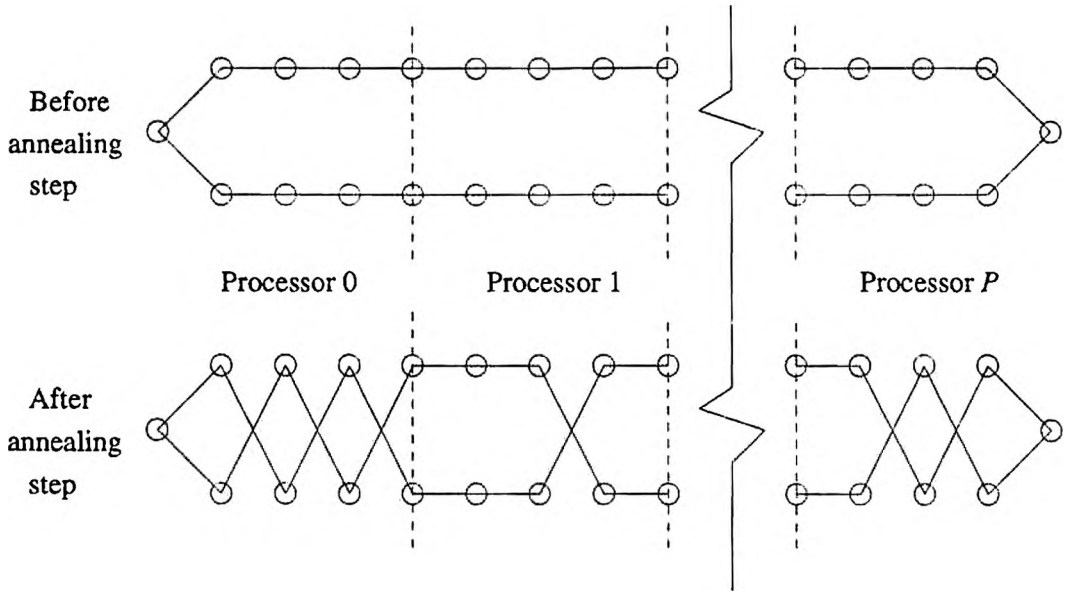


Figure 13. Rubber Band TSP Algorithm

[AlCa89] used the IP algorithm for the traveling salesman problems. The initial random configuration is stretched out like a rubber band, and the number of cities are equally distributed among the processors. Each processor anneals its own paths by swapping corresponding endpoints. After a fixed a number of tries in each processor, the total path length is computed, and a new temperature is chosen. After state variables are redistributed, above procedure is repeated.

In experiments, the 30 processors versus 2 processors speedup ranged from about 8 for a 243 city TSP, to 9.5 for a 1203 city TSP. Speedup versus single processor is not reported. The IP algorithm is not applied in general. For example, in VLSI design, most cells are connected. So applying IP algorithm always causes cost error.

2. Shared State Space. Shared state-space algorithms make simultaneous, independent moves on a shared-memory state-space. No cost-function errors can occur. [DaKi87] used this algorithm for optimizing gate-array placement by locking both affected cells and wires before move generation. However, changes in the state generation function caused poor convergence. In a shared memory simulation of the IBM RP3, the speedup is 7.1 for 16 processors, with poor results.

3. Systolic. [AaBo86a, AaBo86b] and [MeRo53] presented the systolic algorithm which relies on the property that simulated annealing brings a thermodynamic system toward the Boltzmann distribution.

The systolic annealing algorithm divides each Markov chain into a number of subchains. The quasi-equilibrium is preserved by adjusting the intermediate results obtained for the subsequent Markov chain after each of its subchains. Suppose there are P processors and the chain length is N. We divide one chain into P subchains of length $SL = \lfloor N/P \rfloor$, $T_{k,m}$ the temperature for the mth subchain of the Markov chain M_k, and $s_{k,m,i}$ the ith configuration vector of the subchain $M_{k,m}$. At any **PICK** node on processor

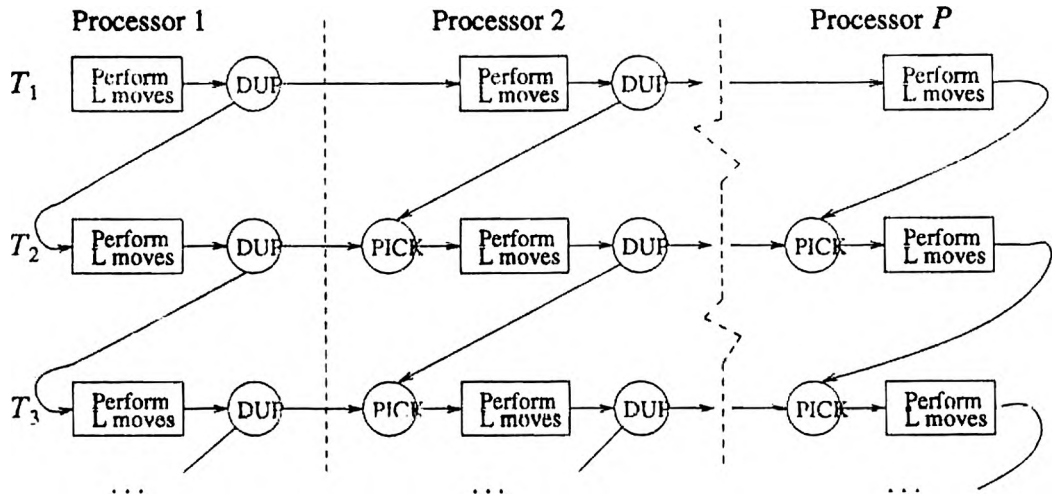


Figure 14. Systolic Algorithms [AaBo86b]

p, we must decide between state $s_{n-1,p,SL}$ computed by processor p at temperature T_{n-1}, and state $s_{n,p-1,SL}$ computed by processor $p-1$ at temperature T_n. We make the choice according to the Boltzmann distribution. The relative probability of picking $s_{n-1,p,SL}$ is

$$\rho_0 = \frac{1}{Z(T_{n-1})} \exp\left(-\frac{C(s_{n-1,p,SL}) - C_{opt}}{T_{n-1}}\right) \tag{4-3}$$

and the relative probability of picking $s_{n,p-1,SL}$ is

$$\rho_1 = \frac{1}{Z(T_n)} \exp\left(-\frac{C(s_{n,p-1,SL}) - C_{opt}}{T_n}\right) \tag{4-4}$$

$Z(T)$ is the partition function over the state space

$$Z(T) = \sum_{s \in S} \exp\left(\frac{C(s)}{T}\right)$$

where S is the entire state space and C_{opt} is the optimal cost.

The **PICK** node then selects $s_{n-1,p,SL}$ and $s_{n,p-1,SL}$ with probabilities

$$p(s_{n-1,p,SL}) = \frac{\rho_0}{\rho_0 + \rho_1}, \qquad p(s_{n,p-1,SL}) = \frac{\rho_1}{\rho_0 + \rho_1} \tag{4-5}$$

The temperature decrement is defined as

$$T_{k+1,m} = \frac{T_{k,m}}{1 + \dfrac{\ln(1+\delta)}{3\sigma(T_{k,m})} T_{k,m}} \tag{4-6}$$

where $\sigma(T_{k,m})$ is the standard deviation of the costs of subchain $M_{k,m}$.

The important observation is that the partition function, Z, is not easily calculated and the optimal cost is generally unknown. The important requirement for convergence is that the annealing process must reach quasi-equilibrium at the end of the first subchain at each temperature. However, if the number of processors increases, the length of the subchain SL decreases. This leads to an incorrect standard deviation, so the temperature decrement is not correct.

[KiKi90] suggested the stepwise-overlapped parallel annealing algorithm(SOPA) for these problems. The Markov chains M_{k-p} and M_k are both assigned to the same processor. Thus before starting Markov chain M_k, the standard deviation $\sigma(T_{k-p})$ of the Markov chain M_{k-p} is known. The new temperature for the Markov chain M_k is calculated as follows:

$$T_k = \frac{T_{k-1}}{1 + \dfrac{\ln(1+\delta)}{3\sigma(T_{k-p})} T_{k-1}} \tag{4-7}$$

For the Markov chain M_k of $1 < k \leq P$, $\sigma(T_{k-p})$ is not available, so we use the standard deviation of the initial temperature, σ_0, instead. $\sigma(T_{k-p})$ may be rather outdated information with which to calculate T_k, but it is better than erroneous estimation of the standard deviation from the first subchain of M_{k-1}. SOPA introduced complex moves. Since both

the optimal cost and partition function are generally unknown, the configuration choice in **PICK** node is done by the metropolis criteria. So the probabilistic choices of complex moves(picking the previous Markov chain configuration) are considered as simple moves.

The numerical results show that the quality of the final solutions improved as the number of processors increased. With 8 processors operating on a 15×15 uniform grid of cities, the systolic algorithm obtained a mean path-length of 230, at a speedup of about 6.2, while the sequential algorithm obtained an average of about 228.5. An important thing is that we can get speedups using SSS parallel algorithms only in the low temperature region with a low acceptance rate. However, the speedup of the Systolic algorithm is independent of the temperature, so the Systolic algorithm can be used over the entire temperature range.

[AaBo86] developed the cluster algorithm, which is similar to the SSS algorithm. The basic idea underlying the cluster algorithm is to use all available processors for the generation of one Markov chain at a given temperature. In the cluster algorithm, a Markov chain is generated by using all available processors, in such a way that system perturbations, cost calculations, and acceptance decisions are done in parallel. System updates can only be done after each other and while an update is carried out, all processors are halted until the system is adapted to the new configuration. To achieve this, the procedures **WAIT**(s) and **SIGNAL**(s) are introduced, acting on the same semaphore s where s is the state variable.

With the cluster algorithm, speedups are achieved only in the low temperature region, as for the SSS algorithm. To increase the efficiency in the upper temperature region, similarly to the systolic algorithm, the concept of subchains is introduced. This modified systolic algorithm (the division algorithm) can be described as follows. Let N be the number of processors, L the length of a Markov chain, and $l = \lceil L/N \rceil$ the length of

a sub-chain. In the division algorithm, independent processors copy the current state, then complete a stream of moves at the same temperature. The **PICK** operation chooses among the results, like the complex moves in [KiKi90]. The division algorithm has a linear speedup. However, in the lower temperature region, the deviation from the temperature region becomes prohibitive, so this algorithm is not suited for the lower region. This is due to the fact that the sub-chain length is too short to enable the process to restore quasi-equilibrium.

D. ASYNCHRONOUS ALGORITHMS

Asynchronous algorithms use a method related to chaotic relaxation, since processors operate on outdated information. Since simulated annealing randomly selects hill-climbing moves, it can tolerate some errors. Under the right conditions, annealing algorithms can evaluate the cost using old state information, but still converge to a reasonable solution. So it is important to find an upper bound on the cost error at a particular temperature to maximize a speedup in the parallel implementation.

1. Asynchronous Spatial Decomposition. Asynchronous spatial decomposition methods partition state variables across different processors. However, in asynchronous algorithms, each processor also maintains read-only copies of state variables from other partitions. When a processor evaluates a new state, it uses only local copies of state variables.

a) *Clustered Decomposition (CD).* [CaRo87] presented a clustered decomposition algorithm, which divided state variables (macro cells) equally among the processors, while putting dependent variables (adjacent or connected macro-cells) on the same processor. The macro-cell placement problem has been implemented on the Sequent Balance 8000, a multiprocessor system with a shared memory architecture. Utilization is greater than 80 percent using up to 8 processors. This algorithm used overlap method.

No advantage can be obtained by restricting the configuration space to the set of feasible solutions. And more, simply checking that a cell is moved to an empty spot is at least as time consuming as computing the overlap for the cell.

In addition, the Timber-Wolf package has shown that the quality of the final solution, as well as the speed with which convergence is reached, improved if intermediate overlaps are allowed. This algorithm used a clustering method. If cells are assigned which are neighbors to the same processor, the probability of generating a move that will bring two cells to overlap because of an error in the cost calculation should be low. This algorithm partitions the moves among the processors by partitioning the set of cells to be placed on the P processors with nearly equal cells. For each processor p, we compute the center of gravity X_p of the cluster C_p

$$X_p = \frac{1}{\sum_{c \in C_p} A(c)} \sum_{c \in C_p} x(c)A(c) \tag{4-8}$$

And the moment of inertia

$$\Gamma_p = \sum_{c \in C_p} \|x(c) - X_p\|^2 A(c) \tag{4-9}$$

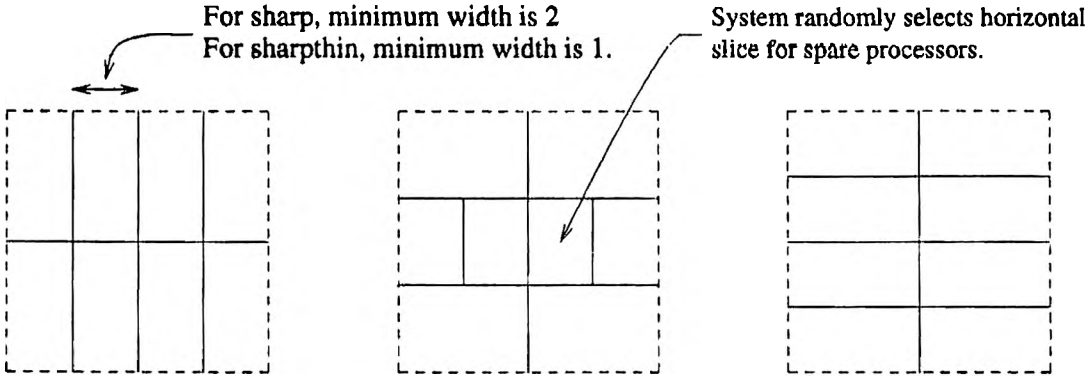
where $x(c)$ is the center of cell c and $A(c)$ its area.

The cluster cost is then defined as

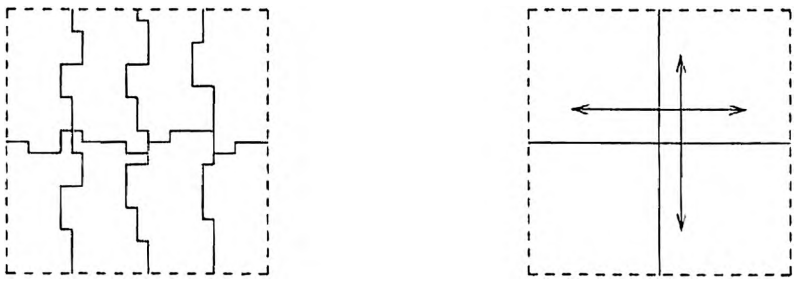
$$\text{cluster cost} = w_{cc} \times \sum_{i=1}^{p} \Gamma_i \tag{4-10}$$

where w_{cc} is a nonnegative user defined weight. If a cell is passed by a processor to another one that owns cells which are close to this cell, then the cluster cost decreases because the moment of inertia of the system also decreases. The cluster method is proposed as a mechanism to reduce the risk of error at low temperature.

b) *Rectangular Decomposition (RD).* [GrDa89] presented a simpler approach, rectangular decomposition, which divides the grid of a VLSI placement problem into disjoint rectangles, then shifts the boundaries after each stream. At low temperatures, interdependent state variables typically share a rectangle. This algorithm is implemented on the simulated RP3 environment.



Sharp and Sharpthin configurations with 8 processors.



Fuzzy (edges are ±1 from sharp) Proportional (only origin moves)

Figure 15. Rectangular Decomposition Schemes [GrDa89]

Four different variants were tried, proportional, sharp random, sharpthin random and fuzzy random (Figure 15). Proportional rectangles method divides the layout into equally sized rectangles, roughly proportional to the overall layout size. Then we randomly select the origin for this grid. The sharp random rectangles method divides the grid into rectangles with a minimum width and height of 2. The sharpthin random rectangles method is similar to the sharp random method, except that the minimum width and height is 1. The fuzzy random rectangles method is also similar to the sharp random method, except that the borders are fuzzy. All rectangular decomposition schemes produced small errors and converged close to the minimum. Among the 4 methods, as the number of processors increased, the sharp random rectangles method produced the best result. However, the speedup was not reported.

2. Asynchronous Shared State-Space. [DaKi87] investigated modifications of the standard annealing method for circuit placement using simulated shared memory. Each processor randomly picks two chips, sets a flag so that no other processor can move either of these two chips, and attempts to exchange their positions.

Three methods were implemented. Method A locks both the chips considered for interchange and all the nets that involve these two chips. In Method B1, only the chips considered for a move are locked, and the correct histograms of wires, and therefore the cost, are recalculated after all trials at a given temperature are completed. However, Method B2 does not correct the histograms and cost at each temperature.

In comparing the speedups for Method A and both Methods B as the number of processors increased, the efficiency of Method A dropped faster than that of either of the Method B's. Method B2 yielded superlinear speedups in 2 and 4 processors while the results diverged due to cumulative error. Method B1 gave a comparable quality of results. Method B1 employed a maximum number of chips with efficiencies about 80%.

E. CONCLUSION

Table II summarizes the various parallel algorithms. In general, speedup is greater using distributed memory than using shared memory. Also the distributed memory scheme does not have the scale problem.

Speedups of the Parallel Algorithms		
Algorithm	Memory Type	Speedups
FD [KrRu87]	shared	not more than 10
SSS [RoDr90]	distributed	$1/a(T)$, where $a(T)$ is the acceptance rate
DTD [ChEd88]	distributed	$(N + N \log_2 N)/2$ for N processors
CP [BrBa88]	distributed	17.7 on 16 processors (super-linear sppedups)
IP [Gree90]	?	9.5 for 1203 city TSP (30 vs. 2 processors)
Shared state space	shared	7.1 on 16 processors [DaKi87]
Systolic [KiKi90]	shared	6.2 on 8 processors
CD [CaRo87]	shared	80% utilization on 8 processors
RD [GrDa90]	distributed	?

Table II. Comparison of Parallel Algorithms

V. GENERAL CONCEPTS OF COST ERROR

To make annealing run faster, parallel processing is a logical candidate. The main difficulty is the maintenance of the global state S of the annealing process. This is further complicated by the desirability of using distributed memory multiprocessors or multicomputers which have no single global picture of the state.

The simulated annealing algorithm can be looked upon as a random iterative improvement algorithm, with a certain probability of making mistakes by accepting hill climbing moves that increase the cost to get out of the local minima. Since simulated annealing randomly selects hill climbing moves, it can tolerate some cost errors. Thus, an approximate calculation, instead of an exact calculation, which uses old state information from other nodes can be used to evaluate the cost function. This modified procedure is an asynchronous algorithm, whereas, a straightforward implementation of parallel simulated annealing is strictly synchronous (and sequential!).

Under the proper conditions, annealing algorithms can evaluate the cost using old state information, and still converge to a reasonable solution. So it is important to find an upper bound on the cost error at a particular temperature to maximize speedups in the parallel implementation. Herein these two algorithms will be differentiated as Sequential Simulated Annealing (even for a parallel version since the sequence of state updates is the same as for a sequential version) for the former and Error-Present Simulated Annealing for the latter.

Cost error tolerance plays a useful role in multiprocessing. When processors independently operate on different parts of the problem, they need not synchronously update other processors. A processor can save several changes, and then send a single block to the other processors. Asynchronous algorithms require a minimum of synchronization. However, at low temperatures, the cost error may degrade the final result unless corrected

by a later move. So, simulated annealing does not have an unlimited tolerance for cost error.

Section V.A explains how a cost error can occur in multicomputers. Section V.B defines the traditional error measurement method and discusses previous work on cost-error-tolerant schemes. In Section V.C, three interesting phenomena of the cost error-present algorithm and shortcomings of the traditional error measurement method are discussed. The analysis is applied to the composite stock cutting problem using a simulated annealing algorithm which features asynchronous parallel spatial decomposition on the stock sheet.

A. OCCURRENCE OF COST ERROR

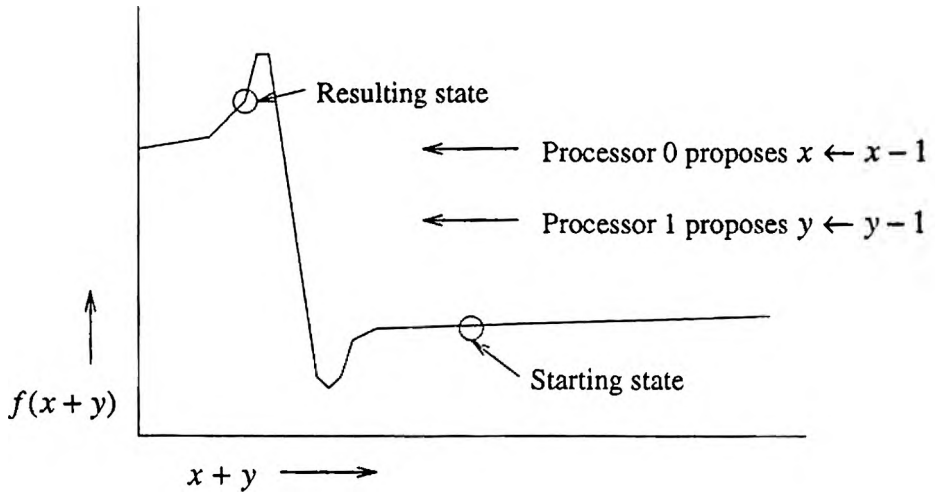


Figure 16. Errors Can Cause Annealing Failure [Gree89]

Consider a system with two state variables x and y and also some state $s = < x, y > \in S$ (Figure 16). Let the cost function be $f(x+y)$. Now put x and y on two

separate processors. Each processor proposes a move: processor 0 generates $x \leftarrow x - 1$, while processor 1 generates $y \leftarrow y - 1$. In both cases, the cost change, $\Delta C < 0$, so each move will be accepted. However, the cost function error causes the state to jump to a high local minimum. At low temperatures, the annealing algorithm probably will not escape this local minimum trap.

B. PREVIOUS WORK ON ERROR TOLERANCE

[JaDa88] describes the characteristics of cost errors at different temperatures. The error in the cost function is defined to be the difference between the real change in cost from initial to final states and the estimated change in cost, which is equal to the sum of the changes in cost(ΔC_i for processor i) at each processor.

Definition 5-1: The *cost error* (ΔE) is defined as the difference between the actual (real) cost change and the estimated (measured) cost change. That is, due to the local copy of the out-dated information, the actual cost change calculation may be different from the estimated cost change.

$$\Delta E = \Delta C_a - \Delta C_e = (C_{af} - C_{ai}) - \sum_{i=1}^{P} \Delta C_i \qquad (5\text{-}1)$$

where ΔC_a is the actual cost change, ΔC_e is the estimated cost change. C_{af} is the actual final cost and C_{ai} is the actual initial cost. ΔC_i is the estimated cost change in processor i, and P is the total number of processors.

This cost error measurement scheme will be referred as the *traditional error measurement scheme*. There are shortcomings in this traditional error measurement scheme. These are discussed in the next sub-section V.C.

Definition 5-2: An *optimistic error* occurs when the cost error (ΔE) is positive from Definition 5-1, i.e. the estimated cost change (ΔC_e) is less than the actual cost change (ΔC_a), $\Delta C_a > \Delta C_e > 0$. Since the Metropolis criterion (equation 2-2) is used for the cost

changes, $e^{-\frac{\Delta C_e}{T}} > e^{-\frac{\Delta C_a}{T}}$, the acceptance ratio is increased in the case of an optimistic error. In other words, the candidate move is accepted while this move may be rejected in sequential (error-free) simulated annealing since $e^{-\frac{\Delta C_e}{T}} > e^{-\frac{\Delta C_a}{T}}$. This kind of error is called an *optimistic error* and this move an *optimistic move*.

Definition 5-3: A *pessimistic error* occurs when the cost error (ΔE) is negative, i.e. the actual cost change is less than the estimated cost change, $\Delta C_e > \Delta C_a > 0$. This being the reverse case of an optimistic error, the acceptance ratio is decreased in the case of a pessimistic error.

Definition 5-4: The *stream length, s,* is defined as the number of continuous moves before the global update where all local informations are broadcast and updated.

[JaDa88] observed that the average cost error in the high temperature region increases with an increase in the stream length. However, the average cost error reduces and finally drops to 0 in the low temperature region because in the low temperature region, the acceptance ratio of moves is small and consequently, there are very few interacting moves causing cost errors.

[Grov86] presents a cost-error-tolerant scheme based on the analogy with statistical mechanics to show that cost errors which are much smaller than the temperature do not change the results of the algorithm. In statistical mechanics, all macroscopic properties of a material can be derived from the partition function z, which is defined as the sum of the Boltzman factors over all possible states, $z = \sum_{i \in S} \exp(-\frac{C(i)}{T})$. With this method, the maximum stream length in a fixed temperature can be probabilistically predicted based on the expected magnitude of a cost error.

[BaJo90] suggest an adaptive stream length control. The goal is to find an upper bound on the maximum permissible cost error at a particular temperature. By adjusting

the stream length dynamically, the average cost error can be limited to a specific range. This method is based on the move acceptance curve in which the acceptance ratio P is given by:

$$P = Prob[\text{move accepted} \mid \Delta C > 0] \cdot Prob[\Delta C > 0]$$
$$+ Prob[\text{move accepted} \mid \Delta C \leq 0] \cdot Prob[\Delta C \leq 0] \quad (5\text{-}2)$$

where ΔC is the proposed cost change. By considering the induced cost error and the Metropolis criteria, the acceptance ratio with cost error can be rewritten as

$$P_E = e^{-(\Delta C \pm E)/T} \cdot Prob[\Delta C > 0] + Prob[\Delta C < 0] \quad (5\text{-}3)$$

where E is the total amount of cost error at a fixed temperature.

If the acceptance ratio with cost error (P_E) is held to within 5 percent of a normal distribution, a pessimistic cost error bound B_+ and an optimistic cost error bound B_- are approximated as follows:

$$B_+ \leq -T \cdot \ln(1 - 0.05) \approx T/20$$
$$B_- \leq \ T \cdot \ln(1 + 0.05) \approx T/21 \quad (5\text{-}4)$$

If the average cost error after a stream length is higher than (T/21), the stream length is reduced commensurate with that excess. If average cost error is lower than (T/42), the stream length is increased slowly. A 5 percent deviation in composite acceptance is set experimentally to maintain convergence.

C. ANALYSIS OF COST ERROR

There are two different types of cost errors: *Temporary errors* and *cumulative errors* [Dura89]. *Temporary errors* occur when two processors simultaneously consider interacting moves. For example, in the stock cutting problem, if two processors attempt to move an object simultaneously to the same location which is empty, the objects will overlap. If the processors investigate the overlap of the moved objects after each move, the

system gets a single, consistent and correct state. *Cumulative errors* develop when local state information used to compute the cost becomes increasingly out of date as the annealing process continues. In the stock cutting problem, the affinity relation cost becomes incorrect as the stream length increases, because as the stream length increases, the local information gets out-dated. [Dura89] observes that a temporary error has only a minor affect on the convergence of simulated annealing, while a cumulative error appears to have a strong affect on convergence.

When a cost error affects the annealing process, there are some interesting phenomena. First, from Definition 5-1, [JaDa88] and [CaRo87] say that the cost error is mostly negative and the absolute value of the cost error is large at high temperatures, but goes to zero as the temperature decreases. The ratio of accepted versus attempted moves tends to be very small at low temperatures, even if the range limiter tries to keep it large. With very few moves accepted, the probability of accepting parallel moves is also very small. Furthermore, even if moves generated in parallel are actually accepted, they are range-limited, so that the error cannot be arbitrarily large at low temperatures.

In the stock cutting problem, Figure 17 indicates that the cost error measured by the traditional scheme (Definition 5-1) is not mostly negative. However, the cost error only for the accepted hill climbing moves is mostly negative. Figure 18 indicates that the absolute value of the average cost error of pessimistic moves and that of optimistic moves are almost the same in the high temperature region. However, the absolute value of the average cost error of pessimistic moves is a little smaller than that of optimistic moves in the critical and the low temperature regions. Figure 19 shows that the number of accepted pessimistic moves is greater than that of accepted optimistic moves. From above figures (Figure 18 and 19), it is expected that the total pessimistic cost error is greater than the total optimistic cost error. This corresponds to Figure 17.

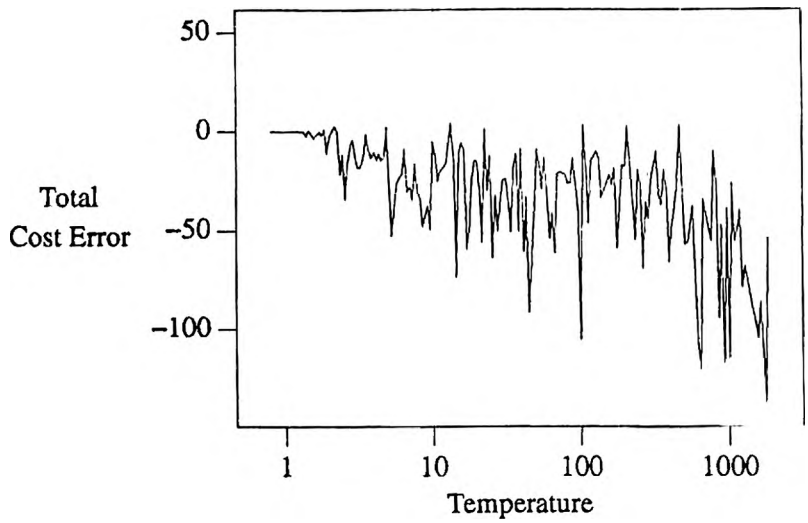


Figure 17. Total Cost Error for All Accepted Hill Climbing Moves (Stream length is 125)
($\cdots$: ΔE for all accepted moves, —: ΔE only for the accepted hill climbing moves)

Secondly, when a cost error is present, [RoBl86] states that the average acceptance ratio increases in the low temperature region as the number of processors increases. This is due to the misinformation causing some moves that would not have been made in sequential simulated annealing. Further moves are then necessary to make up for these "wrong" moves, thus increasing the acceptance ratio. In the stock cutting problem, Figure 20 depicts the acceptance ratio of hill climbing moves. The acceptance ratios of the optimistic and pessimistic hill climbing moves are almost same. So from Figure 19 and 20, it can be expected that the pessimistic hill climbing moves occur more frequently than the optimistic moves. This can be explained by noting that the hill climbing move tends to be estimated higher than the actual cost by using the out-of-date local information.

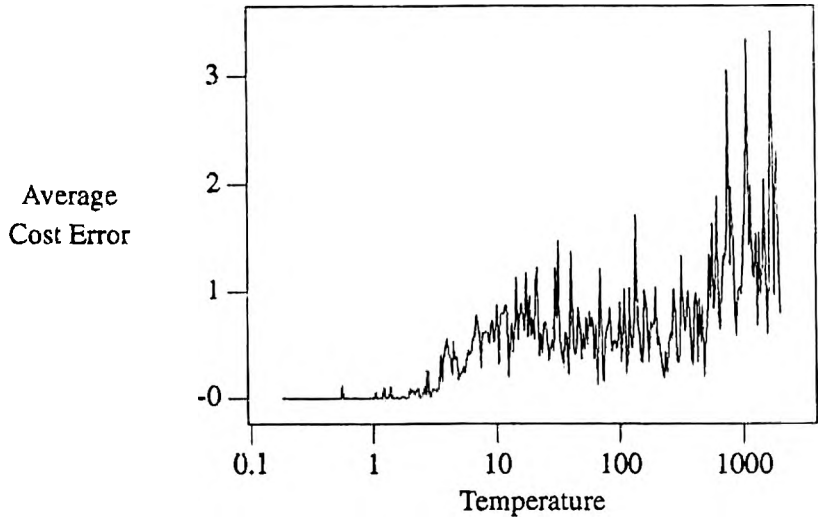


Figure 18. Average Cost Error for One Accepted Hill Climbing Move (Stream length is 125)
($\cdots$: $| < \Delta E > |$ for optimistic moves, — : $| < \Delta E > |$ for pessimistic moves)

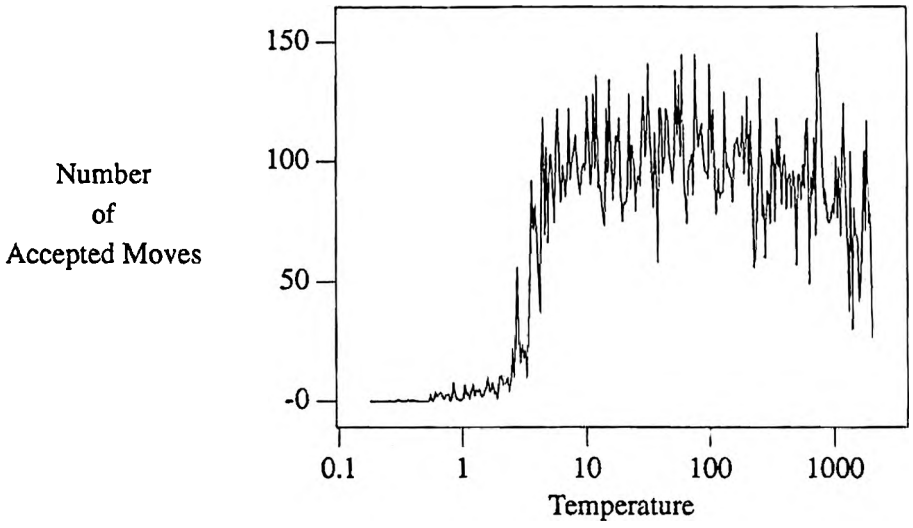


Figure 19. Total Number of Accepted Hill Climbing Moves (Stream length: 125)
($\cdots$: for optimistic moves, — : for pessimistic moves)

The acceptance ratio of the error-present algorithm is smaller than that of the sequential simulated annealing algorithm, because a pessimistic move occurs more frequently than a optimistic move and a pessimistic move decreases the acceptance ratio. In the critical (middle) temperature region, a decreasing number of hill climbing moves occurs. Since only hill climbing moves affect the acceptance ratio, the acceptance ratios of the error-present algorithm and the sequential simulated annealing algorithm are nearly the same. In the low temperature region, most accepted moves have negative cost change. However, with incomplete information, some moves are accepted, which would not have been accepted in the sequential simulated anncaling algorithm. So the acceptance ratio of the error-present algorithm is increased slightly (Figure 21).

The third phenomenon in the presence of cost error is the reduced fluctuation of the average change in cost as a function of temperature($< \Delta C >$ vs. T) at high and intermediate temperature regions [DaPf87]. Since pessimistic moves occur more frequently, the fluctuations in cost are reduced in the high temperature region. So the system is likely to be kept in the high local minimum. The average cost using the error-present algorithm is less than that of sequential simulated annealing algorithm because the hill climbing moves are rejected more frequently in the error-present algorithm (Figure 22).

Figures 17 through 21 are drawn from the stock cutting of 16 irregular patterns in 4 processors. The Markov Chain length is 500, and temperature decrement ratio is 0.98.

There are shortcomings in the traditional cost error measurement scheme (Definition 5-1). Since there is no way to calculate the actual cost without global information, the traditional error measurement scheme is to calculate the cost error after a global update as a difference between the actual cost change(ΔC_a) and the estimated cost change(ΔC_e) using Definition 5-1. However, this method has inherent problems.

This method counts only the accepted moves, i.e. if the candidate move is a pessimistic hill climbing move, $\Delta C_e > \Delta C_a > 0$, and this move is rejected, this move may be

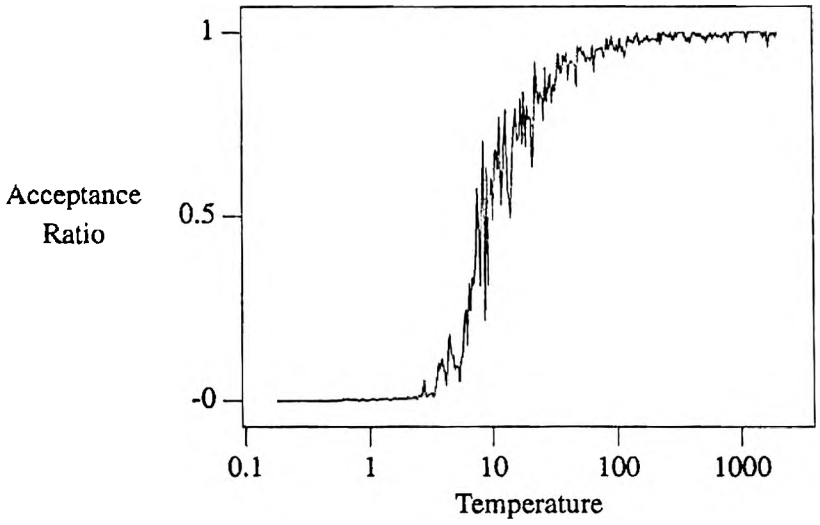


Figure 20. Acceptance Ratio of Hill Climbing Moves (Stream length: 125)
(··· : for optimistic moves, — : for pessimistic moves)

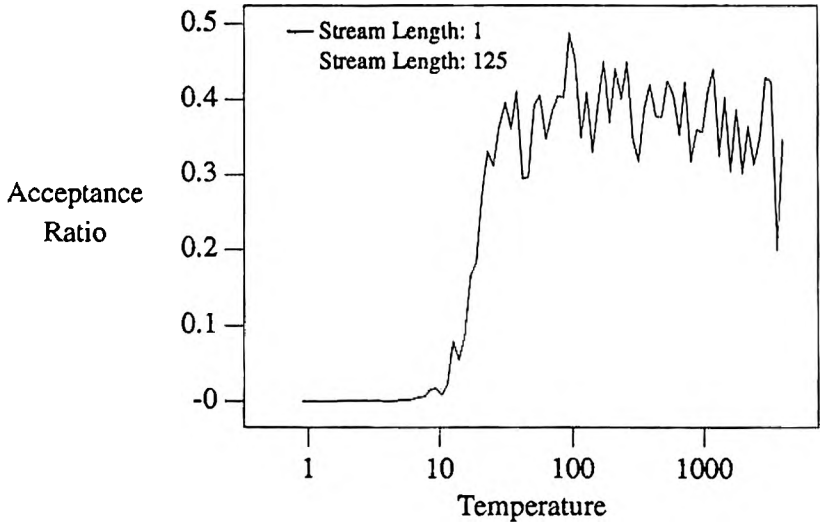


Figure 21. Acceptance Ratio of All Moves

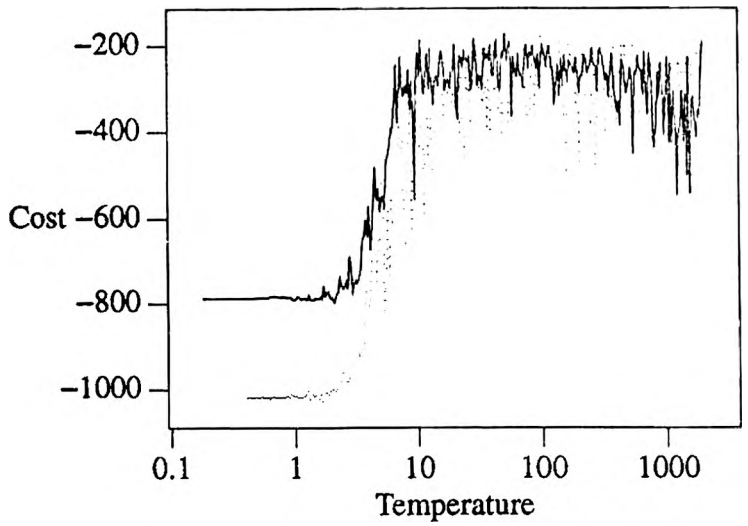


Figure 22. Flutuations of Cost

(· · · : sequential annealing, — : stream length is 125)

accepted in sequential simulated annealing because $e^{-\frac{\Delta C_e}{T}} < e^{-\frac{\Delta C_a}{T}}$. This kind of cost error cannot be included with this method. In other words, this method cannot calculate the cost error of rejected moves.

The second problem is that when both the actual cost change(ΔC_a) and the estimated cost change(ΔC_e) are negative, regardless of the cost error, the candidate move is accepted. However, the difference in cost, $\Delta C_a - \Delta C_e$, is added to the total amount of cost error, even though the acceptance of the move is correct, i.e. there is no error in the move decision.

Finally, the optimistic error ($\Delta C_a > \Delta C_e > 0$) and the pessimistic error ($\Delta C_e > \Delta C_a > 0$) are compensated during a stream length. Only the rough average error

can be conjectured. So this traditional error measurement scheme can be used only qualitatively. It indicates us at which temperature large cost errors occur.

These three problems are corrected by a new cost error measurement scheme, with some assumptions (see Section VI).

VI. A NEW ERROR TOLERANCE METHOD

In this section, a new cost error measurement scheme is presented. Using the measured amount of cost error, an optimal stream length is derived based on the hill climbing nature of simulated annealing. Bounds on the cost error are proved analytically to be a function of global update frequency, or stream length s. Erroneous move decisions due to the cost error (ΔE) will be proved to be exponentially distributed with respect to fixed temperature ($T > 0$). With this known distribution, the probability of an erroneous move decision and the amount of cost error due to the erroneous move decision can be determined in s parallel moves without global updating, or in stream length s.

Figure 22 shows one possible interpretation that as the stream length increases, the hill climbing power decreases since the fluctuations in cost reduce in the error-present annealing process. The decreased hill climbing power can be compensated for by an increased additional Markov chain length. That is, the additional move generations provide a greater chance of a hill climbing move. Since the cost error increases as the stream length increases, the optimal stream length and the additional Markov chain length are proportional to keep the convergence as in sequential (error-free) annealing process because as the stream length increases the cost error increases too. When the stream length is fixed, the generated cost error must be tolerated by changing the additional Markov chain length dynamically.

Meanwhile, when the additional Markov chain length is fixed, the tolerable amount of the cost error, bounds of the cost error, is fixed, so the stream length is varied according to the bounds of the cost error. With the increment of the Markov chain length, the annealing process converges to the good results with a reasonable speedups. Since the additional Markov chain length is fixed in the experiment, the amount of cost error must be controlled by increasing or decreasing the stream length. By adjusting the global

update frequency, the convergence property is maintained with the same degree as in the sequential annealing process.

Here the distribution of the move acceptance and the erroneous move decision are defined.

Theorem 6-1: The acceptance move decision is exponentially distributed with respect to the parameter $T > 0$.

$$Prob[\text{Move accepted with cost change } [0, \Delta C] \,]$$

$$= 1 - e^{-\frac{\Delta C}{T}}$$

Proof: Define the continuous random variable X to be a function which associates a positive real number, the hill climbing cost change (ΔC) with each possible outcome of an accepted move decision. The probability of move acceptance is $e^{-\frac{\Delta C}{T}}$ when the cost change is $(\Delta C, \infty)$. So the cumulative distribution of move acceptance is $1 - e^{-\frac{\Delta C}{T}}$ when the cost change is $[0, \Delta C]$, which is the exponential cumulative distribution function.

$$Prob[X \leq \Delta C]$$

$$= 1 - Prob[X > \Delta C]$$

$$= 1 - Prob[\text{Move accepted with cost change } \Delta C]$$

$$= 1 - e^{-\frac{\Delta C}{T}}$$

So the continuous random variable X has an exponential distribution with respect to the parameter $T > 0$. □

Since the estimated and actual cost changes are different, erroneous moves can result. Consider two possible cost changes, ΔC_1 and ΔC_2 where $\Delta C_1 > \Delta C_2$ where it is not known which is the actual and which is the estimated cost change. If a move is accepted with a smaller cost change, ΔC_1, while the move is rejected with the larger cost change, ΔC_2, then an erroneous move of error, $\Delta E = \Delta C_2 - \Delta C_1$, has occurred.

Theorem 6-2: The erroneous move decision is exponentially distributed with respect to the parameter $T > 0$, given that the candidate move is accepted with smaller cost change, ΔC_1, between the actual and the estimated cost changes.

Prob[The erroneous move decision with cost error $[0, \Delta E]$]

$$= Prob[\text{Move rejected with cost change } \Delta C_2 \mid \text{Move accepted with cost change } \Delta C_1]$$

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

Proof: Define a continuous random variable $\gamma_{\Delta C_1}$ to be a function which associates a positive real number, the cost error of the hill climbing move (ΔE), with each possible outcome of the erroneous move decision. Consider two cost change values ΔC_1 and ΔC_2 with $\Delta C_2 \geq \Delta C_1$. Then $\Delta E \equiv \Delta C_2 - \Delta C_1$. The erroneous move decision is the event that the candidate move with the smaller cost change, ΔC_1, is accepted, while the candidate move with the larger cost change, ΔC_2, is rejected. The random variable $\gamma_{\Delta C_1}$ represents the excess life of the move acceptance, i.e. $\gamma_{\Delta C_1} = S_{N(\Delta C_1)+1} - C_1$, where $N(\Delta C)$ is the number of acceptances with cost change $[0, \Delta C]$, and S_n is the sum of the cost change when the move is accepted n times. S is the sum of the random variable X in Theorem 6-1. So, $\gamma_{\Delta C_1}$ represents how long the acceptance move decision is maintained given that the candidate move with the smaller cost change, ΔC_1, is accepted. In other words, $S_{N(\Delta C_1)+1}$ is the cost change of the move rejection given that the candidate move with cost

change ΔC_1 is accepted.

$$Prob[\gamma_{\Delta C_1} \le \Delta E] = Prob[S_{N(\Delta C_1)+1} - \Delta C_1 \le \Delta E]$$

$$= Prob[S_{N(\Delta C_1)+1} \le \Delta E + \Delta C_1]$$

(from $N(C) \ge n \Leftrightarrow S_n \le C$.)

$$= Prob[N(\Delta E + \Delta C_1) \ge N(\Delta C_1) + 1]$$

$$= Prob[N(\Delta E + \Delta C_1) - N(\Delta C_1) \ge 1]$$

$$= 1 - Prob[N(\Delta E + \Delta C_1) - N(\Delta C_1) \le 0]$$

$$= 1 - Prob[N(\Delta E + \Delta C_1) - N(\Delta C_1) = 0]$$

(Since the number of rejections is non-negative.)

$$= 1 - Prob[N(\Delta E) = 0]$$

(From the memoryless property of exponential distribution. (Theorem 6-1))

$$= 1 - e^{-\frac{\Delta E}{T}}.$$ □

In Section VI.A, the move decisions are classified according to the actual cost change (ΔC_a) and the estimated cost change (ΔC_e) and the probability of the erroneous move decision is calculated from Theorem 6-2 by a case-by-case analysis. In Section VI.B, the amount of cost error is measured probabilistically regardless of whether the move is accepted or rejected. This cost error measurement method is unlike the traditional cost error measurement scheme (Definition 5-1). This method includes the cost error due to rejected moves. In Section VI.C, since cost error can be tolerated by hill climbing moves, the measured amount of cost error is used to derive the optimal stream length.

A. CASE-BY-CASE STUDY OF ERROR MODEL

There are 4 possible cost change cases (*Case 6-1* through *Case 6-4*) each with 4 possible sub-cases, that is:

1) A move is accepted based on an estimated cost change and also based on an actual cost change.

2) A move is accepted based on an estimated cost change, however will be rejected based on an actual cost change.

3) A move is rejected based on an estimated cost change, yet will be accepted based on an actual cost change.

4) A move is rejected based on an estimated cost change and also based on an actual cost change.

In sub-cases 1) and 4), the move decision is correct regardless of the cost error used. However, in sub-cases 2) and 3), an erroneous move decision occurs due to the cost error.

Since the actual cost change cannot be calculated at run time, the estimated cost change is used in an acceptance decision using the Metropolis criteria (equation 2-2).

Case 6-1: $\Delta C_e \geq \Delta C_a > 0$ (Pessimistic move)

The first case is that the actual cost change (ΔC_a) and the estimated cost change (ΔC_e) for one move are positive and the estimated cost change is greater than or equal to the actual cost change.

Define one move error $\Delta E_1 = \Delta C_e - \Delta C_a$, where $\Delta E_1 \geq 0$

1) The move is accepted with the estimated cost change ΔC_e, where the probability of a move acceptance is $\exp(-\frac{\Delta C_e}{T})$.

1-1) This move will be accepted with the actual cost change ΔC_a as well because the probability of a move acceptance with the actual cost change is greater than or equal to the probability of a move acceptance with the estimated cost change, i.e. $\exp\left(-\frac{\Delta C_e}{T}\right) \leq \exp\left(-\frac{\Delta C_a}{T}\right)$. So the move decision is correct regardless of the cost error.

2) The move is rejected with the estimated cost change ΔC_e, where the probability of a move rejection is $1 - \exp(-\frac{\Delta C_e}{T})$.

2-1) This move can be accepted with the actual cost change ΔC_a with a probability P_1, where

$$P_1 = Prob[\text{Move accepted with } \Delta C_a \cap \text{Move rejected with } \Delta C_e]$$

$$= Prob[\text{Move rejected with } \Delta C_e | \text{Move accepted with } \Delta C_a] \cdot Prob[\text{Move accepted with } \Delta C_a]$$

$$= Prob[\text{The erroneous move decision with cost error } \Delta E_1] \cdot Prob[\text{Move accepted with } \Delta C_e - \Delta E_1]$$

$$= \left(1 - e^{-\frac{\Delta E_1}{T}}\right) \cdot e^{-\frac{\Delta C_e - \Delta E_1}{T}} \qquad \text{from Theorem 6-2}$$

$$= e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E_1}{T}} - 1\right)$$

2-2) When this move is rejected with the actual cost change ΔC_a, there is no erroneous move decision.

Case 6-2: $\Delta C_a \geq \Delta C_e > 0$ (Optimistic move)

The second case is that the actual cost change and the estimated cost change for

one move are positive, however the actual cost change is greater than or equal to the estimated cost change.

We define one move error $\Delta E_2 = \Delta C_a - \Delta C_e$, where $\Delta E_2 \geq 0$

1) The move is accepted with the estimated cost change ΔC_e , where the probability of a move acceptance is $\exp(-\frac{\Delta C_e}{T})$.

1-1) If this move is accepted with the actual cost change ΔC_a, then there is no error.

1-2) This move can be rejected with the actual cost change ΔC_a with probability P_2, where

$P_2 = Prob$[move accepted with $\Delta C_e \cap$ move rejected with ΔC_a]

$= Prob$[Move rejected with ΔC_a | Move accepted with ΔC_e] ·

$Prob$[Move accepted with ΔC_e]

$= Prob$[The erroneous move decision with cost error ΔE_2] ·

$Prob$[Move accepted with ΔC_e]

$= e^{-\frac{\Delta C_e}{T}} \cdot \left(1 - e^{-\frac{\Delta E_2}{T}}\right)$ from Theorem 6-2

2) The move is rejected with the estimated cost change ΔC_e .

2-1) This move will be rejected with the actual cost change ΔC_a as well because $\Delta C_a \geq \Delta C_e$. So there is no erroneous move decision.

Case 6-3: $(\Delta C_e > 0 \cap \Delta C_a \leq 0) \cup (\Delta C_a > 0 \cap \Delta C_e \leq 0)$

In *Case 6-3*, the cost error is greater than the absolute value of the estimated cost change, so the signs of the estimated and actual cost changes are different. Computing these two probabilities is somewhat complex, thus any error control scheme will be

complex. These two cases happen rarely since the cost error is much smaller than the cost change in real experiments. Thus, the occurrence of these events is ignored.

Case 6-4: $\Delta C_e \leq 0 \cup \Delta C_a \leq 0$

In *Case 6-4*, a move will be always accepted and there is no cost error, since the decision of move is correct, i.e. move is accepted also with the actual cost change ΔC_a.

The summary of the above cases are in Table III, where ΔE is the amount of the cost error.

	Pessimistic Move($\Delta C_e \geq \Delta C_a$)		Optimistic move($\Delta C_a \geq \Delta C_a$)	
Move	acc w/ ΔC_e	rej w/ ΔC_e	acc w/ ΔC_e	rej w/ ΔC_e
acc w/ ΔC_a	0	$e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1\right)$	0	-
rej w/ ΔC_a	-	0	$e^{-\frac{\Delta C_e}{T}} \cdot \left(1 - e^{-\frac{\Delta E}{T}}\right)$	0

Table III. Probability of the Cost Error in a Hill Climbing Move

B. NEW COST ERROR MEASUREMENT SCHEME

In the new cost error measurement scheme, the total amount of cost error is calculated throughout a given stream length, s. Unlike previous methods (Section V.B) which ignore cost errors from rejected moves and find the optimal stream length heuristically,

this new method calculates the cost error analytically based on the results in the previous section.

Lemma 6-1: The actual cost change (ΔC_a) is represented as the sum of the estimated cost change (ΔC_e) and the cost error throughout an iteration i.

$$\Delta C_a = \Delta C_e \pm \Delta E$$
$$= \Delta C_e \pm i \cdot \alpha \cdot | < E > |$$

Proof: ΔE is the total amount of cost error throughout a fixed stream length, s. The absolute value of the average error ($| < E > |$) is calculated when one move is accepted. For example, one move is accepted in Processor P_1 and the Processor P_2 does not know that. Then the absolute value of the average cost error $| < E > |$ can be calculated when the Processor P_2 tries the move generation. The cost error ΔE throughout an iteration i, can be represented as an average error ($| < E > |$) times the total number of accepted moves throughout an iteration i. The total number of accepted moves is the acceptance ratio (α) times the iteration i. So $\Delta E = i \cdot \alpha \cdot | < E > |$. □

Since a cost error occurs only with a positive cost change in this analysis, to calculate the cost error, it is necessary to compute a probability for the conditions of *Case 6-1* and *Case 6-2*.

Theorem 6-3: $Prob[\Delta C_e \geq \Delta C_a > 0] + Prob[\Delta C_a \geq \Delta C_e > 0] = Prob[\Delta C_e > 0]$

Proof:

$$Prob[\Delta C_e \geq \Delta C_a > 0]$$
$$= Prob[\Delta C_e \geq \Delta C_a \mid \Delta C_e > 0, \Delta C_a > 0] \cdot Prob[\Delta C_e > 0, \Delta C_a > 0]$$
$$= Prob[\Delta C_e \geq \Delta C_a \mid \Delta C_e > 0, \Delta C_a > 0] \cdot Prob[\Delta C_a > 0 \mid \Delta C_e > 0]$$
$$\cdot Prob[\Delta C_e > 0]$$

$$= Prob[\Delta C_e \geq \Delta C_a \mid \Delta C_e > 0] \cdot Prob[\Delta C_e > 0].$$

Since $Prob[\Delta C_a > 0 \mid \Delta C_e > 0] = 1$, from the assumption that the cost error is not greater than the absolute value of the estimated cost change, i.e. the estimated cost change and the actual cost change have the same signs.

Similarly,

$$Prob[\Delta C_a \geq \Delta C_e > 0] = Prob[\Delta C_a \geq \Delta C_e \mid \Delta C_e > 0] \cdot Prob[\Delta C_e > 0]$$

So

$$Prob[\Delta C_e \geq \Delta C_a > 0] + Prob[\Delta C_a \geq \Delta C_e > 0]$$

$$= Prob[\Delta C_e \geq \Delta C_a \mid \Delta C_e > 0] \cdot Prob[\Delta C_e > 0] +$$

$$Prob[\Delta C_a \geq \Delta C_e \mid \Delta C_e > 0] \cdot Prob[\Delta C_e > 0]$$

$$= Prob[\Delta C_e > 0] \qquad \square$$

The next task is to estimate the $Prob[\Delta C_e > 0]$. Since $Prob[\Delta C_e > 0]$ is a function of state configuration, i.e. in a maximum cost, $Prob[\Delta C_e > 0]$ is zero in a move generation, while in a local minimum cost, $Prob[\Delta C_e > 0]$ is one in a move generation.

Lemma 6-2: The probability of positive estimated cost change is

$$Prob[\Delta C_e > 0] = \frac{\sum_{i \neq j} I_{\{\Delta C_e(i,j) > 0\}}(i)}{N(i)} \quad , \qquad \text{for any state } i$$

Proof: State j is any neighbor of state i. $\Delta C_e(i, j)$ is the estimated cost change of a move from state i to state j. $N(i)$ is the number of neighbor states from state i. The proof is obvious by using the specified indicator, I.□

Since $N(i)$ and $\Delta C_e(i, j)$ are not known in advance, it is difficult to estimate the $Prob[\Delta C_e > 0]$. However, during the running of the algorithm, the estimated cost change

can be calculated. So only when the estimated cost change is greater than zero, the total probability of cost error (P_T) is counted.

When the probability of optimistic and pessimistic errors is defined, the estimated cost change (ΔC_e) and cost error (ΔE) are fixed, i.e. the optimistic and pessimistic cost errors are the same (Figure 18) and the actual cost change is calculated from the predefined estimated cost change and cost error (Lemma 6-1).

Definition 6-1: The probability of optimistic cost error

$$P_{opt} = Prob[\Delta C_a \geq \Delta C_e > 0] \cdot \left| e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{-\frac{\Delta E}{T}} - 1 \right) \right|$$

Definition 6-2: The probability of pessimistic cost error

$$P_{pes} = Prob[\Delta C_e \geq \Delta C_a > 0] \cdot \left| e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right) \right|$$

In Section V.C, it is shown that the total pessimistic cost error are greater than the total optimistic cost error. This can be explained by the next theorem with some assumptions.

Theorem 6-4: The probability of a pessimistic cost error is greater than that of an optimistic cost error with the following four assumptions.

1. The cost error (ΔE) is less than the estimated cost change in the hill climbing move.
2. The amount of the pessimistic and optimistic cost error are same, $\Delta E_1 = \Delta E_2$ from *Case 6-1* and *6-2*.
3. Only measure the estimated cost change is measured and the actual cost change can be expected, or calculated, by the estimated cost change using Lemma 6-1.

4. The probabilities of the pessimistic move and the optimistic move are same in the hill climbing move generation, i.e. $Prob[\Delta C_a \geq \Delta C_e > 0] = Prob[\Delta C_e \geq \Delta C_a > 0]$.

Proof: With assumption 4, and Definition 6-1, 6-2, it is obvious that the probability of a pessimistic cost error is greater than that of an optimistic cost error. So a large pessimistic cost error is more likely to be accepted than an optimistic cost error. □

The probability of optimistic cost error (P_{opt}) is always in [0, 1] from Definition 6-1. The probability of pessimistic error (P_{pes}) must be checked to be in [0, 1]. Pessimistic errors only happen in *Case 6-1*.

So, $\Delta C_e \geq \Delta C_a > 0$

$\Rightarrow \Delta C_e \geq \Delta C_a , \quad \Delta C_a > 0$

$\Rightarrow \Delta C_e \geq \Delta C_e - \Delta E , \quad \Delta C_e - \Delta E > 0$

$\Rightarrow \Delta E \geq 0 , \quad \Delta E < \Delta C_e$

$\Rightarrow \Delta C_e > \Delta E \geq 0$

$\Rightarrow 1 \geq P_{pes} \geq 0$ from Definition 6-2

So the range of the probability of pessimistic error is well defined.

Theorem 6-5: Since a cost error occurs only in a positive cost change, the total probability of cost error, P_T, is given by

$$P_T = Prob[\Delta C_e > 0] \cdot e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right)$$

Proof:

$P_T = P_{pes} + P_{opt}$

$= Prob[C_e \geq \Delta C_a > 0] \cdot \left| e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right) \right| +$

$$Prob[\Delta C_a \geq \Delta C_e > 0] \cdot \left| e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{-\frac{\Delta E}{T}} - 1 \right) \right|$$

$$\leq (Prob[\Delta C_e \geq \Delta C_a > 0] + Prob[\Delta C_a \geq \Delta C_e > 0]) \cdot e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right)$$

$$= Prob[\Delta C_e > 0] \cdot e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right) \qquad \square$$

Now the cost error can be determined using the probability of cost error (P_T).

Theorem 6-6: The amount of cost error in the hill climbing move ($\Delta C_e > 0$) is

$$E = \Delta C_e \cdot e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right)$$

Proof: Since $Prob[\Delta C_e > 0] = 1$, the probability of cost error in the hill climbing move is given by

$$P_T = e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right) \qquad \text{From Theorem 6-5}$$

So,

$$E = \Delta C_e \cdot P_T$$

$$= \Delta C_e \cdot e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right), \qquad \text{given that } \Delta C_e > 0 \qquad \square$$

[Dura89] say that some problems or algorithms [JaRu87] are more resistant to the cost error than the others [RoKl90, RoBl86]. This robustness to the cost error can be explained by Theorem 6-7.

Theorem 6-7: The total amount of the cost error (E) depends on the portion of the cost error (ΔE or $i \cdot \alpha \cdot | < E > |$) in the estimated cost (ΔC_e).

Proof: The cost function is made up of the cost-error-dependent terms which result in the cost error in calculating the cost function and the cost-error-independent term which do not result in the cost error. For example, in the composite stock cutting problem, the cluster term and the overlap penalty term are the cost-error-independent terms, while the affinity relation term is the cost-error-dependent term (Section VII.C). From Theorem 6-6, when the cost error, ΔE, has only a small portion of the estimated cost error, ΔC_e, i.e. $\Delta E \ll \Delta C_e$, the total probability of the cost error goes to 0.

$$E \propto e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1 \right)$$

$$= e^{-\frac{\Delta C_e - \Delta E}{T}} - e^{-\frac{\Delta C_e}{T}}$$

Since $\Delta C_e > \Delta E > 0$ in the hill climbing move, the total amount of cost error (E) is always positive. So robustness to the cost error depends on the portion of the cost error (ΔE) in the estimated cost (ΔC_e). □

It was shown that the traditional cost error measurement scheme (Definition 5-1) has three shortcomings (Section V.C). These three shortcomings of the traditional method are corrected with the assumption in Theorem 6-4. The shortcomings are corrected as followings. First, the new cost error measurement method includes the cost error of the rejected moves (Definition 6-2 and Theorem 6-5). Second, this method does not include the cost error of the negative cost change moves, because the move decision is always correct regardless of the cost error used. Finally, there is no compensated cost error between the pessimistic and optimistic cost errors because this method adds the the probabilities of the pessimistic and optimistic cost error (Theorem 6-5).

C. MAXIMUM BOUND OF TOLERABLE ERROR

In this section, the optimal stream length is derived for the measured amount of cost error. Since a cost error is tolerated by hill climbing moves, a maximum bound on the cost error can be defined using a maximum bound on the hill climbing move.

Let $d(s)$ be the maximum amount (or depth) of cost which can be hill-climbed at a given temperature T and stream length s. Then

$$e^{-\frac{d(s)}{T}} \geq \frac{1}{s} \quad \Rightarrow \quad d(s) \leq T \ln s \tag{6-1}$$

This means that there is a possibility to choose $d(s)$ hill climbing move in s moves [Whit84]. The maximum hill climbing depth is a function of temperature and log of the stream length.

The error-present simulated annealing has a small hill climbing power than sequential simulated annealing, so the error-present algorithm is likely to be kept in a local minimum due to cost error (Figure 22). Hill climbing power is the degree of accepting the hill climbing move. In order to get out of the local minimum and converge to the optimal result, the error-present algorithm must have the same hill climbing power as the sequential simulated annealing algorithm. Since the decreased hill climbing power is due to the cost error, the following theorem is derived.

Theorem 6-8: The hill climbing depth of the error-present algorithm (d_e) is less than that of the sequential algorithm (d_a) by at most the amount of error (E).

$$d_a \leq d_e + E$$

where d_a is the hill climbing depth of sequential simulated annealing for one hill climbing move, d_e is the hill climbing depth of the error-present algorithm for one hill climbing move, and E is the hill climbing error derived from Theorem 6-6.

Proof: A loss of hill climbing power is introduced only by pessimistic errors ($\Delta C_e > \Delta C_a > 0$). Hill climbing power, probabilistically, is $d_a = \Delta C_a \cdot e^{-\frac{\Delta C_a}{T}}$ in the sequential annealing process and $d_e = \Delta C_e \cdot e^{-\frac{\Delta C_e}{T}}$ in the error-present annealing process. Using E (from Theorem 6-6) and pessimistic condition ($\Delta E = \Delta C_e - \Delta C_a$ where $\Delta C_e > \Delta C_a > 0$), we have

$$d_a - (d_e + E) = \Delta C_a \cdot e^{-\frac{\Delta C_a}{T}} - \left(\Delta C_e \cdot e^{-\frac{\Delta C_e}{T}} + \Delta C_e \cdot e^{-\frac{\Delta C_e}{T}} \cdot \left(e^{\frac{\Delta E}{T}} - 1\right)\right)$$

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

$$= \Delta C_a \cdot e^{-\frac{\Delta C_a}{T}} - \Delta C_e \cdot e^{-\frac{\Delta C_a}{T}} \qquad \text{From Lemma 6-1}$$

$$= e^{-\frac{\Delta C_a}{T}} \cdot (\Delta C_a - \Delta C_e) \leq 0 \qquad \text{From pessimistic condition}$$

So, $d_a \leq d_e + E$. □

Next, an extra stream length (u) is required for the decreased amount of hill climbing depth, $E(s)$, throughout the stream length s.

Definition 6-4: The extra move (u) to tolerate the cost error $E(s)$ is given by

$$e^{-\frac{E(s)}{T}} \geq \frac{1}{u}, \text{ so } u \geq e^{\frac{E(s)}{T}} \qquad \text{from Equation 6-1}$$

For a given temperature T, at least u moves have a hill climbing power $E(s)$; and with stream length s, there is a hill climbing power $d_e(s)$ in an error-present algorithm.

Lemma 6-3:

$$e^{-\frac{d_e(s)}{T}} \geq \frac{1}{s}$$

Proof: The proof is obvious from equation (6-1). □

Now the stream length s_a can be calculated for the error-tolerable algorithm having a regular hill climbing depth $d_a(s)$. That is, in order to increase the hill climbing depth to match that of sequential simulated annealing, the stream length s_a is needed.

Theorem 6-9: When the total amount of cost error ($E(s)$) occurs during stream length s, $s_a = s \cdot u$ stream length is needed to tolerate the cost error.

Proof:

$$e^{-\frac{d_a}{T}} \geq e^{-\frac{d_e+E}{T}} \qquad \text{from Theorem 6-8}$$

$$e^{-\frac{d_a(s)}{T}} \geq e^{-\frac{d_e(s)+E(s)}{T}} = e^{-\frac{d_e(s)}{T}} \cdot e^{-\frac{E(s)}{T}} \qquad \text{from ergodicity theory}$$

$$\geq \frac{1}{s} \cdot \frac{1}{u} \qquad \text{from Definition 6-4 and Lemma 6-3}$$

So $s_a = s \cdot u$ stream length is needed for the error present algorithm to have the same hill climbing depth as the sequential annealing process has in the stream length s. □

The next task is how to define the extra stream length factor u for speedups, considering the time for a global update. From Definition 6-4,

$$E(s) \leq T \cdot \ln u, \qquad \text{since } e^{-\frac{E(s)}{T}} \geq \frac{1}{u} \tag{6-2}$$

In order to decrease the extra stream length factor u for speedups, the maximum tolerable cost error $E(s)$ must be decreased as well. However, the extra stream length factor u and the maximum tolerable cost error $E(s)$ are inversely proportional.

PROCEDURE ERROR-TOLERANT SIMULATED ANNEALING

begin

INITIALIZE;

k:= 0;

repeat

calculate $| < E > |$ in T_k /* Average cost error in one accepted move */

$E(s) = 0$ /* $E(s)$ is the total amount of cost error in a given stream length */

repeat

PERTURB(config. $i \rightarrow$ config. $j, \Delta C_{ij}$);

if $\Delta c_{ij} \leq 0$ then accept

else

$$E(s) = E(s) + \Delta c_{ij} \cdot e^{-\frac{\Delta c_{ij}}{T_k}} \cdot \left(e^{\frac{i \cdot \alpha \cdot | < E > |}{T_k}} - 1 \right)$$

/* i is the ith iteration in the stream length */

/* α is the acceptance ratio */

if $\exp(-\Delta c_{ij}/T_k) >$ random [0,1) then accept;

if accept then

UPDATE(configuration j);

until **equilibrium is approached sufficiently closely**;

$T_{k+1} := f(T_k)$;

k:= k+1;

if ($E(s) \leq T_k \cdot \log u$) /* $u = 1.1$ */

increase stream length

else

decrease stream length

until **stop criterion == true (system is 'frozen')**;

end.

Figure 23. The Error-Tolerant Simulated Annealing

For example, if a 10% increase of Markov chain length is allowed, i.e. $u = 1.1$, then the maximum error bound $E(s)$ can be calculated using equation (6-2). If the measured amount of cost error in a given stream length s is greater than the maximum bound cost error $E(s)$, the stream length will be decreased. If the measured amount of cost error in a given stream length s is less than the maximum bound cost error $E(s)$, the stream length will be increased. When the stream length is changed, the Markov chain length s is kept fixed, i.e. Markov chain length M = stream length (s) $\times$ # of global updates in a given temperature. The pseudocode for the error-tolerant scheme is in Figure 23.

VII. IMPLEMENTATION OF PARALLEL SPACE-DECOMPOSITION

This section discusses the space-decomposition simulated annealing algorithm in distributed memory multiprocessors (multicomputers) such as a hypercube. The target problem is the composite stock cutting problem.

The composite (oriented 2D bin packing) stock cutting problem is discussed in Section I.A.2. The shapes are not constrained to be convex polygons or even regular shapes. The one side of the stock sheet, i.e. the length of the x-direction, which is called the *nest width*, is fixed. The objective is to minimize the other side (y-direction) given that all pattern information is known.

The input is the vertex coordinates of the patterns and the nest width. The output is the locations of all the patterns which minimize the y-length of the stock sheet bounding box. The stock sheet bounding box is the minimum square which surrounds all the patterns.

Section VII.A explains how an affinity relation is calculated. The affinity relation is denoted by a_{ij} for a pair of patterns i and j. It is used in the first term of the cost function in Section I.A.2. In Section VII.B, three types of moves, such as displacement, exchange, and rotation moves, are discussed. Section VII.C defines the cost function in detail and the cooling schedule is discussed in Section VII.D. In Section VII.E, the efficient method for finding overlap is presented. Finally, Section VII.F explains the parallel spatial decomposition implementation in detail.

A. PREPROCESS OF SIMULATED ANNEALING

In the preprocess of simulated annealing, a bitmap is generated for each pattern at all allowed rotations and the affinity relation is calculated for every pair of patterns. Since the patterns are irregular, they can have edges in arbitrary directions. Formulating

algebraic constraints that detect overlap between two patterns is very tedious. Therefore, a bitmap representation is used for the patterns. In generation of bitmap, first the bounding box is determined for the pattern, and the outside, boundary, and inside elements of bitmap are different each other for easy calculation of affinity relation and overlap detection.

Definition 7-1: The affinity relation, a_{ij}, is made up of edgewise adjacency term and bounding box term for every pair of patterns i and j.

$$a_{ij} = \alpha f_a + \beta f_d$$

where f_a is the weighted edgewise adjacency and f_d is the function of the density of bounding box for patterns i and j. α and β are nonnegative weights.

The boundary bitmap elements with distance n represents that the minimum distance of boundary bitmap elements of two patterns.

$$\text{Distance} = \min_{\substack{\forall l \in B_i \\ \forall m \in B_j}} D_{lm} \quad , \qquad \text{for all patterns } i \text{ and } j$$

where B_i is the set of boundary bitmap elements of a pattern i and D_{lm} is the distance between the bitmap element l of a pattern i and the bitmap element m of a pattern j

So the distance of the overlap boundary bitmap elements is 0.

Definition 7-2: Edgewise adjacency is determined by comparing the boundary bitmap elements of a pair of pattern i and j given that any two boundary bitmap elements are overlapped.

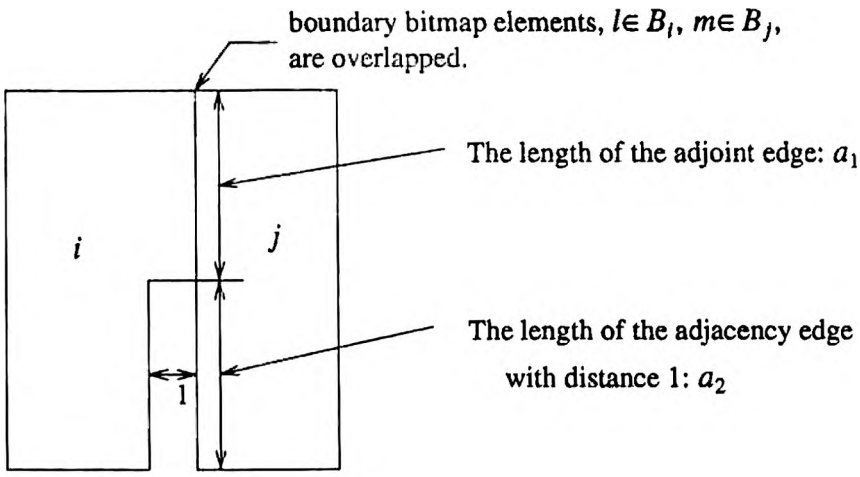
$f_{a_{lm}}$ = sum of overlap boundary bitmap elements +

$\frac{1}{2} \times$ sum of boundary bitmap elements with distance 1

where $f_{a_{lm}}$ represents the edgewise adjacency when the boundary bitmap element l of a pattern i and the boundary bitmap element m of the other pattern j are overlapped (Figure 24). Since two boundary bitmap elements are fixed by the overlap, the locations of two patterns are fixed.

The affinity relation, a_{ij}, represents the tendency of pattern i and to attract pattern j. Since the patterns are not regular, two patterns are unlikely to fit together perfectly. So, an adjacency edge with distance 1 is weighted half that of an adjacency edge with distance 0. This will help two patterns to get together with a small distance. For example, in Figure 24, the edgewise adjacency counts the number of the overlap boundary bitmap elements which is a_1. It also counts the number of boundary bitmap elements with distance 1 which is a_2. Then the total edgewise adjacency will be $a_1 + \frac{a_2}{2}$. This edgewise adjacency calculation is implemented easily by traversing the boundary bitmap elements of one pattern and counting the overlap and distance 1 boundary bitmap elements given that any two boundary bitmap elements are overlapped. The edgewise adjacency relation for every edge between two patterns is calculated. A maximum edgewise adjacency and (x, y) coordinate of the two patterns are selected.

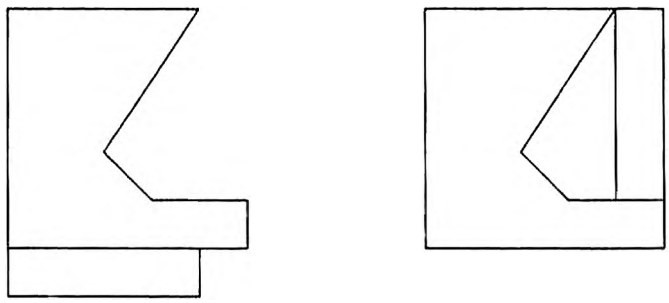
The minimal bounding box surrounding two patterns cannot be guaranteed by only the edgewise adjacency. In other words, maximizing edgewise adjacency is not the same as minmizing the bounding box (Figure 25). So the density of the bounding box, ρ, is incorporated to the affinity relation. When all patterns are packed, the density of the stock sheet is important. So when two patterns which form the minimal bounding box fit together, the density of the stock sheet comprising all patterns is likely to be large. The affinity relation is made up of two terms, edgewise adjacency and the density of the bounding box surrounding two patterns.



$$a_{i,j} = \left(a_1 + \frac{a_2}{2}\right) \times (\alpha + \beta\rho)$$

where α, β are weight constants and ρ is the density of the bounding box.

Figure 24. Calculation of Affinity Relation



Maximize edgewise adjacency Minimal bounding box

Figure 25. Edgewise Adjacency and Minimal Bounding Box

B. MOVE SET

Three types of moves are allowed in order to change the configuration of the given patterns.

Φ_1: Displace a pattern to a new location.

Φ_2: Exchange two patterns.

Φ_3: Change the rotation of a pattern.

Moves are selected randomly to guarantee the mobility of the pattern.

1. Description of Displace Move : Φ_1. In the operation Φ_1, an arbitrary pattern is picked randomly and a new location is selected randomly within the range-limiter window. At the beginning of the annealing process, the window size is to be large enough to contain a space of all patterns. Then it shrinks slowly as the temperature decreases. When the acceptance rate is less than 44%, the window size decreases slowly. Large distance moves usually imply large values of the cost change, ΔC. At low temperatures, only moves which result in small positive cost change have a reasonable chance of being accepted. Hence, at low temperatures, the large distance moves are almost invariably rejected. In order to generate moves which have a reasonable probability of acceptance, these large distance moves are prohibited by the use of a range-limiter window.

The *Rotation Move* (Φ_3) is incorporated into the *Displace Move* (Φ_1). In the operation Φ_1, all allowed rotations are tried. Since the patterns are irregular, this strategy increases the acceptance ratio significantly.

2. Description of Exchange Move : Φ_2. Two patterns are randomly selected and an interchange is attempted. Φ_2 does not apply the range-limiter window in selecting the two patterns. Also, for simplicity, the *Rotation Move* is not incorporated into this move. The *Exchange Move* (Φ_2) operation changes the cost larger than the *Displace Move* (Φ_1)

operation. So the exchange move is more likely to be rejected as the temperature goes down. The ratio of selecting moves Φ_1 vs. Φ_2 is 1:1 in the high temperature region. However, as the temperature decreases, the ratio of Φ_2 decreases also. In the final low temperature region, Φ_1 vs. Φ_2 is 20:1.

3. Description of Rotate Move : Φ_3. A pattern is selected at random to have its orientation changed. The amount of the rotation angle is randomly chosen among the allowed rotations. The coordinate rotation from $[x_i, y_i]$ to $[x_j, y_j]$ is done by

$$[x_j, y_j] = [x_i, y_i]\begin{bmatrix} \cos\theta_{ij} & \sin\theta_{ij} \\ -\sin\theta_{ij} & \cos\theta_{ij} \end{bmatrix}$$

where θ_{ij} is the difference of angle between rotation i and j.

Since the rotate move (Φ_3) is incorporated to the displace move (Φ_1), when the desired distance of the displace move is 0, we rotate a pattern for all possible rotation angles until the move is accepted.

C. COST METRIC: C

The cost function is made up of the affinity relation ($a_{i,j}$) between patterns, the distance from the origin (d_{i_0}) of a particular pattern, and the overlap penalty ($o_{i,j}$) between patterns.

$$C = -\alpha \sum_{\forall i} \sum_{\substack{\forall j \\ i \neq j}} \frac{a_{i,j}}{d_{i,j}} + \beta \sum_{\forall i} d_{i_0} + \gamma \sum_{\forall i} \sum_{\substack{\forall j \\ i \neq j}} o_{i,j}$$

where $d_{i,j}$ is the distance between pattern i and j, and α, β, and γ are positive real numbers that indicate the contribution of each of the components in the cost function.

1. The Affinity Relation Term : $-\sum_{\forall i}\sum_{\substack{\forall j \\ i \neq j}} \frac{a_{i,j}}{\dot{a}_{i,j}}$. The first term is minimized when two patterns having a high affinity relation are getting closer together. Since the affinity relation represents the degree of edgewise adjacency and the density of the bounding box of two patterns, the higher the affinity relation is, the greater the possibilty of saving scrap area. This term also influences packing density. As two patterns draw closer, the rotations with high affinity relations are likely to occur. However, the contribution of the affinity relation to the cost is smaller as the distance between two patterns increases. In other words, the affinity relation of the nearby patterns are given more weight than that of the patterns far away.

2. The Cluster Term : $\sum_{\forall i} d_{i_0}$. When the cost function minimizes the second term, all patterns cluster around the origin line (x-axis) and minimize the length of the y-direction. While the affinity relation term affects the match of two patterns, the second term makes all the patterns come together into a small bounding box.

3. The Overlap Penalty Term : $\sum_{\forall i}\sum_{\substack{\forall j \\ i \neq j}} o_{i,j}$. The third term is a penalty function for the overlap of two patterns. Experimentally it has been shown that good results are achieved by allowing overlap in the high temperature region. Since a bitmap is used to represent patterns, it is easy to count the number of overlap bitmap elements between two patterns.

D. COOLING SCHEDULE

In implementing the simulated annealing algorithm, the following parameters must be specified.

1. Initial value of the temperature: T_0
2. Final value of the temperature: T_f

3. Length of the Markov chain at a certain temperature T_k: L_k

4. Decrement strategy of the temperature: $T_{k+1} = f(T_k)$

A choice for these parameters is referred to as a cooling schedule. The cooling schedule must balance the quality of the final result with the computation time required.

1. Initial Value of the Temperature : T_0. The initial value of the temperature, T_0, is set so that virtually all moves (transitions) are accepted. Then all possible configurations can be considered from the initial state. In this experiment, the initial temperature is set such that the cost is constant for some fixed Markov chain length, i.e. for several consecutive temperatures.

2. Final Value of the Temperature : T_f. Since an important characteristic of simulated annealing is the hill-climbing move, the final temperature is defined such that a hill-climbing move does not occur any more below the final temperature, T_f. In this experiment, the stopping criterion is implemented by recording the value of the cost function at the end of each temperature in the annealing process. The final temperature is satisfied when the value of the cost function is not changed for some fixed Markov chain length.

3. Length of Markov Chain : L_k. L_k is the length of the Markov chain at a certain temperature T_k. The chain length must be long enough so that the annealing process reaches a quasi-equilibrium state after L_k tries at temperature T_k. The simple criterion for the fixed L_k is that for each temperature T_k, a minimum amount of transitions should be accepted. However, as T_k approaches 0, the transitions are accepted with decreasing probability, so L_k is bounded by some constant. Usually, L_k is some integer multiple of the maximum number of neighborhood states. In this experiment, L_k is set to be some integer multiple of the problem size, and the limit of transition is $4 \times L_k$.

4. Decrement Strategy of the Temperature : $T_{K+1} = f(T_k)$. A small temperature decrement is chosen in order to avoid the necessity of a long Markov chain length for re-establishing a quasi-equilibrium state at each new temperature. In this experiment, a fixed decrement ratio, α, is used.

$$T_{k+1} = \alpha \cdot T_k \qquad \text{where } k = 0, 1, 2, 3, \cdots$$

where is $0.95 \leq \alpha \leq 0.99$

This strategy decreases the temperature proportional to the logarithm of the temperature.

E. DATA STRUCTURE

Since simulated annealing is cpu-intensive, an efficient data structure is needed for move generation and evaluation of the cost function. The ability to propose and evaluate moves efficiently hinges on a good representation for the basic objects in the problem. Since a bitmap representation is used for the patterns, move generation moves all bitmap elements of a pattern to find the overlap with the other pattern. However, to enhance efficiency, another data structure is used which saves only the boundary bitmap elements of the pattern. When a pattern is moved, the relative location to the other pattern is calculated. When each bounding box of two patterns is overlapped, the boundary bitmap elements of one pattern are compared with a whole bitmap elements of the other pattern and vice versa.

In evaluating the cost function, most of the effort is spent on calculating the affinity relation cost. The preprocessor calculates the affinity relation for all allowed rotations of each pair of patterns. This will save much time at the expense of more storage space.

F. SPATIAL DECOMPOSITION METHOD

The problem must be decomposed among the processors to minimize the erroneous calculation of the cost function. Among the three terms of the cost function, only the

```
PROCEDURE FIND_OVERLAP
begin
  Pick a pattern i and a position (x, y) randomly;
  for all patterns j≠i do
    if bounding boxes of pattern i and j are overlapped
      compute the number of overlap between boundary bitmap
        of a pattern i and a whole bitmap of a pattern j;
      compute the number of overlap between boundary bitmap
        of a pattern j and a whole bitmap of a pattern i;
end;
```

Figure 26. The Pseudo Code for Finding Overlap

affinity relation term needs a global state, so it is desirable to cluster near patterns into one processor.

The stock sheet is nearly equally divided in the x-direction. Each processor governs a space and handles the pattern whose reference coordinate belongs to its own space. The reference coordinate of the pattern is the smallest (x, y) of the bounding box surrounding the pattern. The maximum range-limiter window size is restricted to the neighboring processor space. This restricts the mobility of patterns. However, in large sized problem, the mobility of a pattern is not impaired. Actually, when each processor handles at least 4 patterns, an optimal result is obtained with reasonable speedups. With this restriction of the maximum range-limiter window size, the number of synchronizing processors is at most two. This simplifies the code and facilitates the asynchronous run of all the processors.

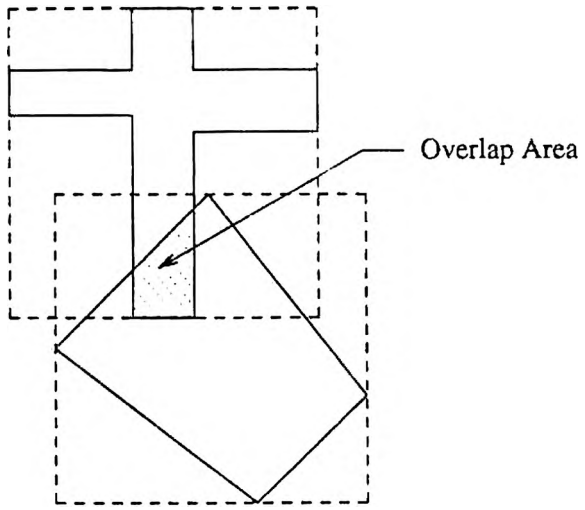


Figure 27. Finding Overlap

1. Move Operation. Since the maximum range-limiter window size is restricted to the neighboring processor space, move displacement occurs between at most two neighboring processors. Since the overlap penalty cost is high, every processor must have the exact bitmap information if the pattern lies in its own space. That is, if the pattern lies within the boundary of two processors, then each processor has exact bitmaps which pertain to its own space.

a) Intra-Processor Displacement. When the entire bitmap, i.e. all elements of the bitmap, of a picked pattern lies within the space before and after the displacement move, a local move displacement can be made because the neighboring processors are not affected in bitmap manipulation.

b) Inter-Processor Displacement. If the bitmap of a picked pattern lies across the boundary before or after the displacement, the two neighboring processors involved

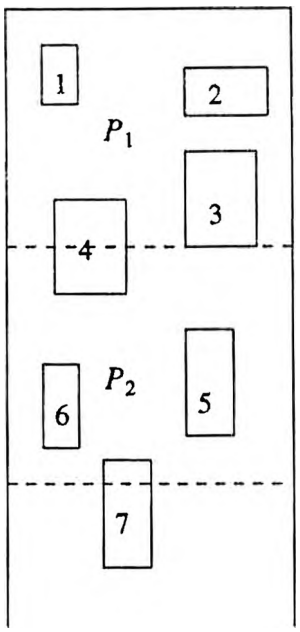


Figure 28. Spatial Decomposition

cooperate in the move to calculate the exact number of overlapping bitmap elements. This is an inter-process displacement.

When the move is an inter-processor displacement, the master processor picks a pattern and the neighboring processor involved in the bitmap manipulation becomes the slave processor in the inter-process displacement. The master processor sends the information of the selected pattern and the candidate displacement location to the neighboring processor. If the slave processor is the master processor and the master processor is the slave processor simultaneously, deadlock happens. In other words, if the two neighboring processors do inter-processor displacement move operation simultaneously, each processor will be waiting response from each other forever. To prevent deadlock, when a slave processor is asking another cooperation to the neighboring (master) processor, the arrived displacement request is rejected. If not, the slave processor accepts the request

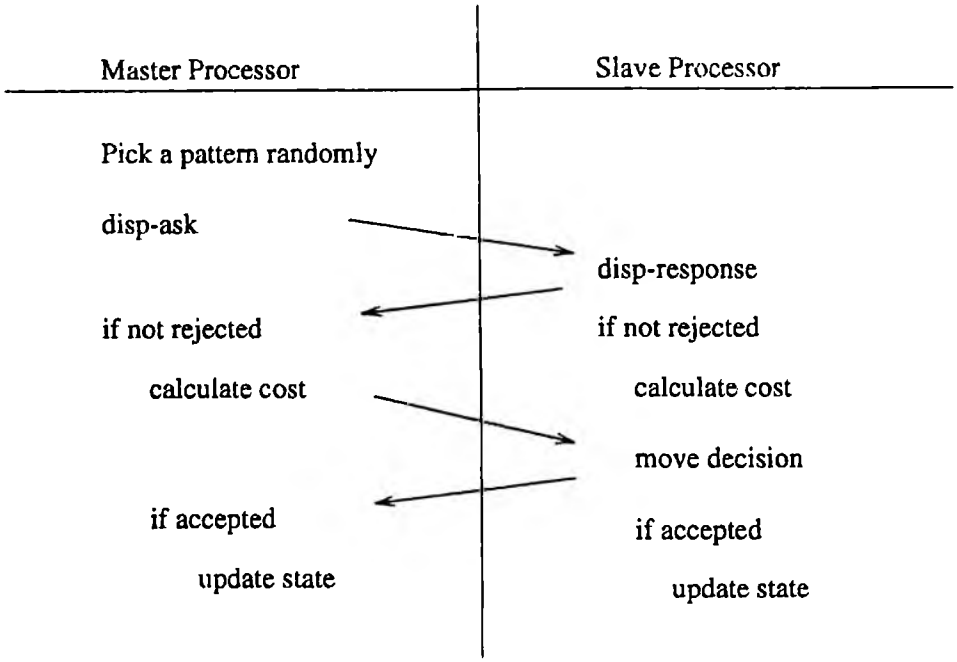


Figure 29. Inter-Processor Displacement

message from the master processor. The master processor calculates the cluster term of the cost function which calculates the distance from the origin line.

Each processor calculates its own affinity relation part of the cost function and the overlap penalty. The upper processor calculates the affinity relation on the its own and upper located patterns, while the lower processor calculates the affinity relation on its own and lower located patterns. Then the slave processor sums the costs from the two processors and makes the move decision. The slave processor sends a decision to the master processor. If the move is accepted, each processor changes its information. Since all allowed rotations are considered prior to an accepted move, the acceptance ratio increases. This scheme gives good results in the experiments.

c) Intra-Processor Exchange. The intra-process exchange operation is to interchange two patterns whose bitmaps are inside the same processor space. When the processor selects two patterns randomly, first the processor determines if the entire bitmaps of the two patterns are inside the space before and after the exchange operation. If both bitmaps are inside the space, the processor exchanges the reference coordinates of the two patterns locally.

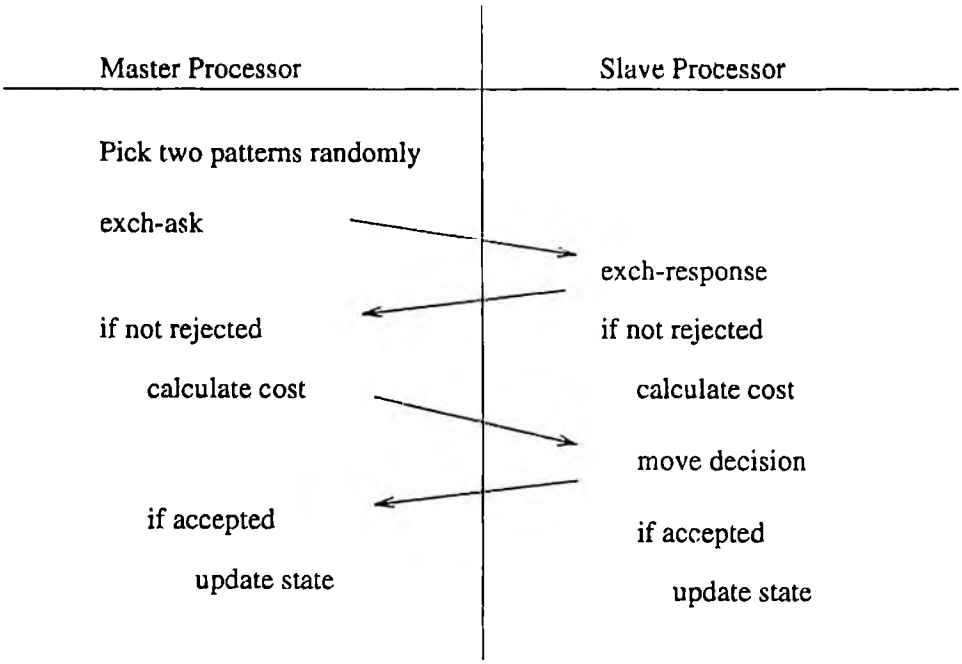


Figure 30. Inter-Processor Exchange of Type I

d) Inter-Processor Exchange. There are two kinds of inter-processor exchanges. The first case is when the master processor selects two patterns randomly from its own space; but, the selected two bitmaps lie across the boundary before or after the exchange operations. The two neighboring processors must cooperate to calculate the exact

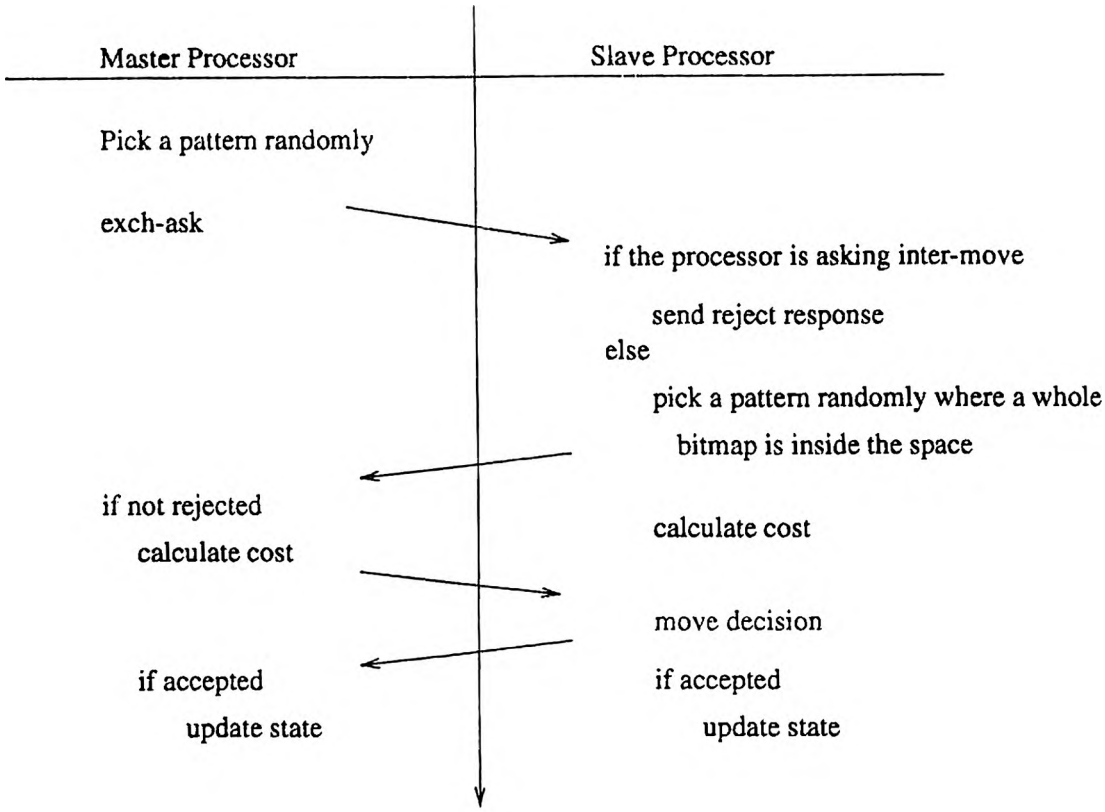


Figure 31. Inter-Processor Exchange of Type II

number of overlapped bitmap elements and the affinity relation. The master processor sends the selected pattern information to the neighboring slave processor. When the slave processor is requesting another inter-move, this exchange-ask is rejected. If not, the two processor cooperate in calculating the overlap penalty and the affinity relation costs. This is type I inter-processor exchange.

The second case is when two processors each pick a pattern randomly from its own space and exchange the two patterns with each other. This is type II inter-processor exchange. Since the cooperating processors must be at most two, type II inter-processor exchange selects patterns accordingly.

2. Fast Evaluation of the Move Decision. In the simulated annealing algorithm, the move decision, $e^{-\frac{\Delta C}{T}}$, is applied many times for a state change. Exponential evaluation requires 107 μsec cpu time on VAX 11/780 running VMS. Substantial reductions in the computation time for the evaluation of the exponential function can be achieved by using a table look-up technique [NgRa86]. Of course, the Metropolis criterion, $e^{-\frac{\Delta C}{T}}$, need only be evaluated when both ΔC and T are positive quantities since the new state is always accepted when $\Delta C \leq 0$. For simplicity, set $x \equiv \frac{\Delta C}{T}$ where x is always positive. Actually, if x exceeds 88, e^{-x} is essentially 0, and if x is less than 10^{-7}, e^{-x} is approximated as $(1 - x)$. So the range of interest x for evaluating e^{-x} is $10^{-7} < x < 88$. In terms of powers of 2, the range of interest is $2^{-23} < x < (2^7 - 40)$. So a range of 30 bits are of interest. If x is multiplied by 2^{23}, then the floating point number can be directly converted and stored as a 32-bit integer such that the least 30 significant bits are of interest.

We can divide 30 bits to 3 sets of 10 bits.

set 1 is e^{-x} for $2^{-3} < x \leq 2^7$

set 2 is e^{-x} for $2^{-13} < x \leq 2^{-3}$

set 3 is e^{-x} for $2^{-23} < x \leq 2^{-13}$

The first set is stored in **table 1** and uses the value of $\frac{x}{2^{-3}}$ as the table index. The second set is stored in **table 2** and uses the value of $\frac{x}{2^{-13}}$ as the table index. Finally, the third set is stored in **table 3** and uses the value of $\frac{x}{2^{-23}}$ as the table index. At the start of simulated annealing, these three sets of 1024 evaluations are performed using the exact exponential function and these values are stored in the table.

The evaluation then proceeds as follows: (1) The most significant group of 10 bits of x (bits 29 through 20) are shifted right by 20 bits and the ten bits are masked out. This

quantity determines *index 1*, the index into **table 1**. (2) The next most significant group of 10 bits of x (bits 19 through 10) are shifted right by 10 bits and the ten bits are masked out. This quantity determines *index 2*, the index into **table 2**. (3) The least significant group of 10 bits of x (bits 9 through 0) are masked out. This quantity determines *index 3*, the index into **table 3**. The value of e^{-x} is given by:

$$e^{-x} = (\textbf{table 1}[\textit{index 1}])\ (\textbf{table 2}[\textit{index 2}])\ (\textbf{table 3}[\textit{index 3}])$$

This technique requires only 3 table look-ups, 3 floating point multiplies and 2 shifting operations. The exact value of e^{-x} can differ from the table look-up value by at most $2^{-23} = 1.192 \times 10^{-7}$. However, since the most time is spent in the move generation and the cost calculation in the experiment of the composite stock cutting problem, this fast evaluation of the move decision does not lessen the running time significantly.

3. Global Update. During a predefined stream length, each processor generates moves asynchronously independent of the other processors. When a processor finishes the predefined stream length of move generations, it sends a termination signal to the host processor. When the host processor gets the termination signals from all processors, it sends a global update signal to all working processors. Then all working processors cooperate on the global update of information by communicating its own information to all other processors using a tree-reduction method and determine the new stream length according to the amount of cost error which has occurred during the previous stream length. In the global update, a maximum of 2 move-requests from the neighbor processors may be pending in the message buffer. So during the global update, any pending move-requests must be checked and removed in the first 2 tree-reduction operations.

4. Lazy Update. Since the global update takes much time, a lazy update is used which updates configuration information with out-of-date information. Because the cost error is caused only in calculating the affinity relation part and as the patterns are closer,

```
Procedure Evaluation of e^-x
begin
  if x ≤ 0
    e^-x ≥ 1;
  else if x > 88
    e^-x = 0;
  else if x < 10^-7
    e^-x = 1 - x;
  else
    i = x * 2^23;              /* i is 32-bit integer */
    e^-x = table1[i>>20] * table2[i>>10 & 0x3ff] * table3[i & 0x3ff];
end;
```

Figure 32. Fast Evaluation of Move Decision

the cost error can be larger, this lazy update scheme works well, i.e. it reduces the cost error fairly.

When a processor completes the given substream length for the lazy update, it sends information to the adjacent processor. The substream length for the lazy update is less than or equal to half of the stream length for the global update. This substream length for the lazy update can be defined by a user or dynamically by calculating the maximum bound of the cost error.

For example, processor P_i completes the substream length for the lazy update. Processor P_i sends processor P_{i-1} information of patterns which are owned by the current processor, P_i, and down processors, P_{i+1} through P_n where n is the total number of

processors. Processor P_i continues move generations without halting. Then processor P_{i-1} probes to determine if the lazy update messages came or not. If a lazy update message arrived, processor P_{i-1} updates the information, i.e. the lazy update is asynchronous. This lazy update can be implemented downward, that is processor P_i sends lazy update information to processor P_{i+1}.

5. Load Balancing. The patterns move around all the stock sheet across the boundary of the processor's space. Some processors may happen to have too small or large a number of patterns. These load unbalances can be corrected dynamically during the specified stream length. However, in this research, for simplicity, the patterns are redistributed after the predefined stream length or at the end of the temperature for simplicity.

6. Two High Specific Heat Regions.

A quantity corresponding to the thermodynamic specific heat is defined by taking the derivative with respect to temperature of the average value of the cost observed at a given temperature from equation (3-8), i.e.

$$S(T) = \frac{\partial < C(T) >}{\partial T} = \frac{\sigma^2}{T^2}$$

A large value of $S(T)$ indicates a change in state of the order of a system. This can be used in the optimization context to indicate that freezing has begun and hence that very slow cooling is required. Just as a maximum in the specific heat of a fluid indicates the onset of freezing or the formation of clusters, here specific heat maxima are found at two temperatures, each indicating a different type of ordering in the problem. In this research, high specific heat in the high temperature region corresponds to the aggregation of clusters of patterns, i.e. it represents the rapid cost change in the cluster term in the cost function. Lower specific heat in the lower temperature region corresponds to the proper rotation of patterns, i.e. it represents the rapid cost change in the affinity relation

term. Therefore patterns cluster near the origin line first and then proper rotations occur to further reduce the cost further.

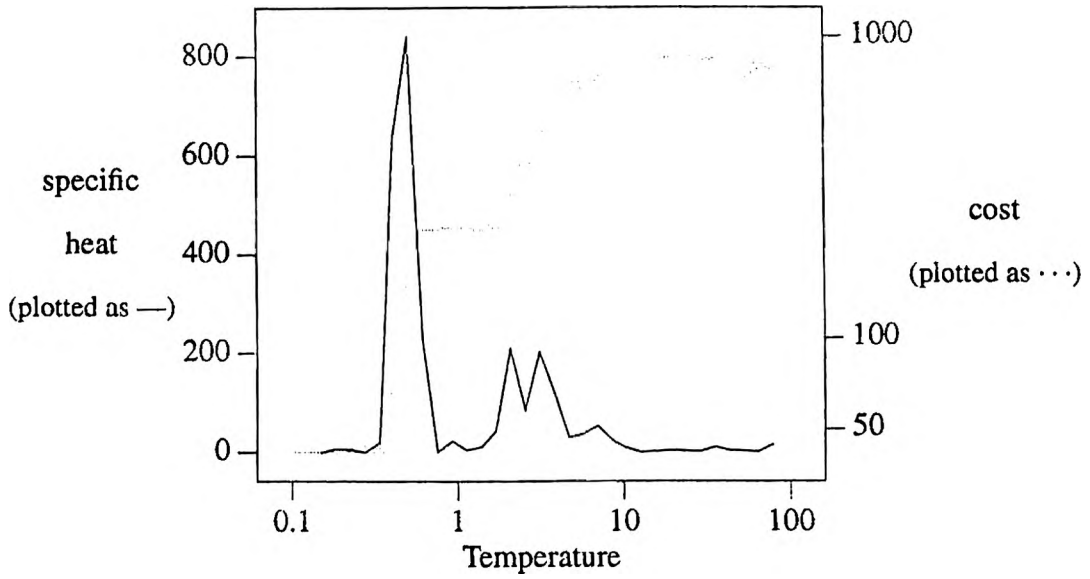


Figure 33. The Specific Heat and Cost

7. More Weight on the Larger Area Pattern. When large area patterns are near the origin line, the packing density may be high, i.e. the bounding box surrounding all patterns is small. This is because the large area pattern may have more scrap area in its bounding box, so when the large area pattern is at the end of stock sheet line, there may be more scraps. However, the small area patterns can be easily combined with other patterns, resulting in a high packing density bounding box. So more weight is given to the second term (cluster term) of the cost function for a large area pattern. This can be compared to give a large inertia moment to the origin line on the large area pattern. This strategy makes the large patterns cluster near the origin line, while the small area patterns fill out the scrap area resulted from clustering of the large area patterns.

The large area pattern has another problem of mobility. There are more probabilities of move rejection for the large area pattern due to the overlap penalty. Large area patterns are more likely to be stuck in their own places, while the small area patterns are moving toward the origin line more swiftly. So large area patterns congregate around the end of the stock sheet. To overcome this mobility bias, the pattern is selected for the move generation in proportion to the ratio of the area of the pattern. This increases the mobility of the large area patterns, while decreasing the mobility of the small area patterns.

VIII. EXPERIMENTAL RESULTS

The new adaptive error-tolerance method (Section VI), which will be referred to as the *adaptive method*, was implemented on a 16-node Intel iPSC/2. The target problem was the composite stock cutting problem (Section I), which was decomposed specially along space of the stock sheet (Section VII).

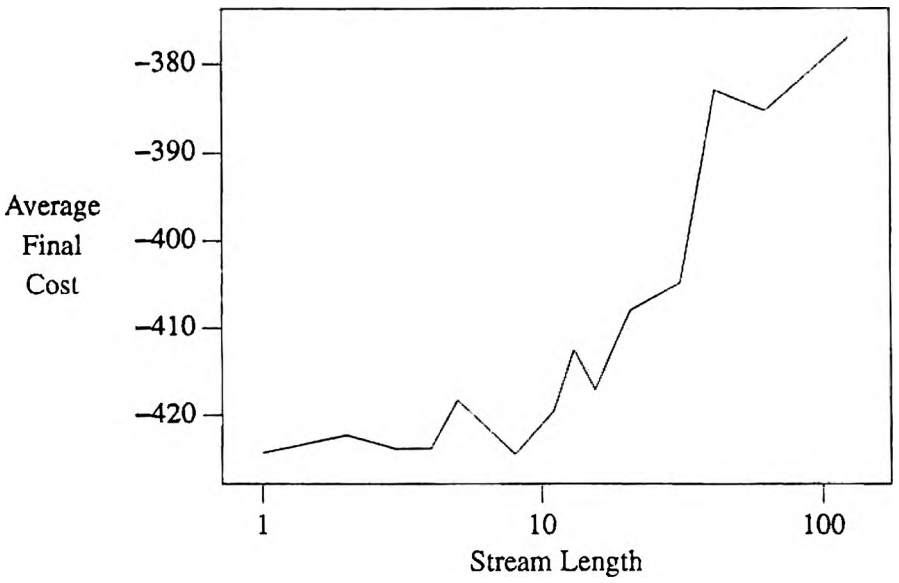


Figure 34. Final Cost vs. Stream Length

The parallel space-decomposition simulated annealing algorithm was implemented in 4 nodes. A total of 16 irregular patterns were used. The Markov chain length was 500. To track the behavior of the cost error, the weight of the affinity relation term was set much greater than that of the cluster term, since the cost error occurs only in the affinity relation term of the cost function. The fixed stream length method, which will be referred as the *static method*, was implemented twelve times on each stream length. The stream length was varied to note its effect on the cost error.

Figure 34 shows that the average final cost starts to increase above stream length 10. Using 4 patterns per node and an optimal stream length of 10, the herein algorithm is much more robust with respect to the cost error than the floor planning algorithms of [JaDa88, Dura89], where the optimal stream length was set equal to the number of patterns per node. This can be explained partly by Theorem 6-7. The portion of the cost error (ΔE) in the estimated cost (ΔC_e) of the composite stock cutting problem may be smaller than that of floor planning problems. This can be due to the large range limiter in the composite stock cutting problem, so the information of the moved pattern is propagated to the other processors. From Theorem 6-7, this reduces the total amount of the cost error, so the stream length can be increased keeping the convergence to the optimal results.

Comparing the stream length at each temperature (Figure 35) with the annealing curve (Figure 36), the stream length reduces to 2 in the critical region where specific heat is very high. However, the stream length increases to 125 far from the critical region, i.e. the global update is done only once at the end of each temperature. The stream length varies dynamically according to the annealing curve. This means the cost error has little affect on the annealing process away from the critical region, but affects it greatly in the critical region. This corresponds to the fact that the annealing process proceeds rapidly away from the critical region, but much more slowly in the critical region.

In Table IV, Adp means the adaptive method, and Static-10 means that the stream length was fixed at 10. Since the average final cost starts to increase above the stream length 10, the stream length of 10 was selected for the static method. The average final costs was almost the same. However, the standard deviation of the adaptive method was smaller than that of the static method, as expected. The average stream length of the adaptive method is larger than that of the static method. Since the number of global update was inversely proportional to the stream length, the average number of global

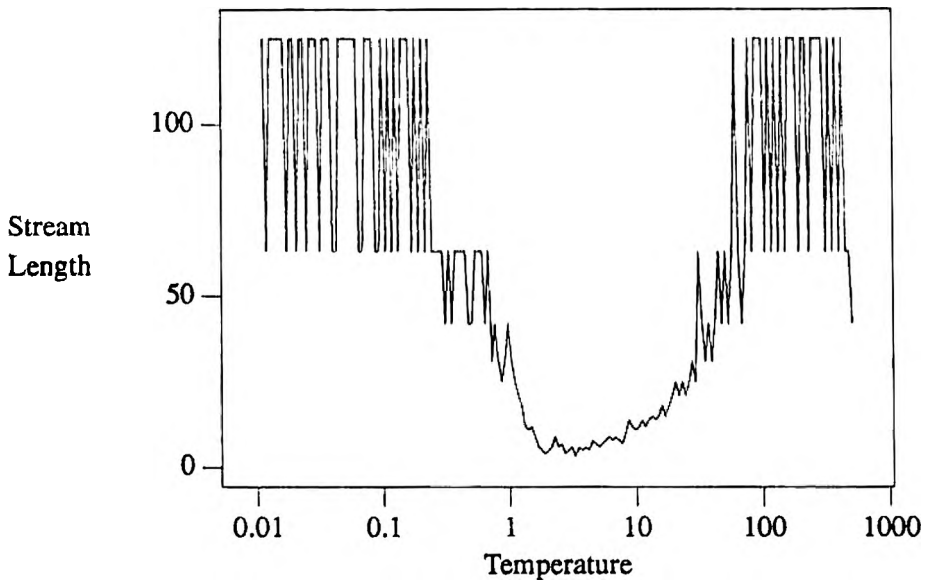


Figure 35. Stream Length vs. Temperature

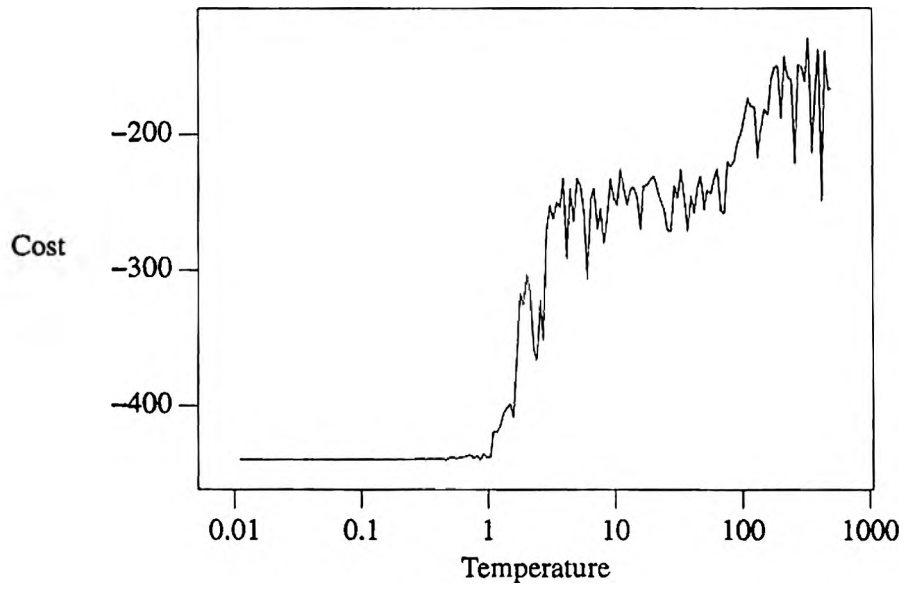


Figure 36. Annealing Curve

updates was reduced 6.3 times in the adaptive method, compared to the static method. From the above data (Figure 35, 36, and Table IV), the adaptive method adapts the stream length dynamically, with comparable final results.

	Mean	Std. Dev.	Worst	Best
Adp	-424.5	8.29	-414.4	-436.2
Static-10	-424.3	11.70	-402.9	-436.2

Table IV. Final Cost of Adaptive and Static Methods

As a second experiment, 16 different sets of patterns were implemented to observe the results of the adaptive method. The number of patterns varied from 128 to 160. Regular (rectangular) patterns were used for simple implementation, requiring no bitmap operation. A cooling schedule was set almost uniformly, such that the initial temperature was about 200,000, the temperature decrement ratio was 0.98 to 0.99, and the Markov chain length was 5,000 to 20,000. In this experiment, the packing density is considered. In other words, the weight of the cluster term in the cost function is balanced with that of the affinity relation term.

In Table V, Adp means the adaptive method and Static-5 represents the static method, where the stream length was fixed at 5. Table V indicates the speedups of both the adaptive method and the static method comparing with the sequential annealing process. The mean of speedup of the adaptive method was greater than that of the static

method over the entire processor range. However, the standard deviation of the adaptive method was always greater than that of the static method. Since the experiment was done on different sets of patterns, the run time may have varied according to the problem, in order to maintain the convergence in the adaptive method.

Figure 37 plots Table V. Figure 37 indicates that as the number of processors increased, the adaptive method was much better than the static method in speedups. As the number of processors increases, the global update time increased as well. Thus, the efficiency of the parallel implementation was reduced as the number of processors increased. Since the adaptive method reduced the global update frequency, the adaptive method achieved better speedups than the static method for a large number of processors.

Table VI compares the final cost of the adaptive method with that of the static method. The stream length of the static method varied from 5 to 100 where the convergence was assumed to be maintained. The mean final cost of the adaptive method was smaller than that of the static method for the entire processor range. However, using 16 nodes, the final cost of the parallel implementation was greater than that of the sequential annealing process. This may result from restricted mobility in move generation. Let the cost deviation of the parallel implementation be defined as:

$$\text{Cost Deviation} = \frac{\text{Cost of Parallel - Cost of Sequential}}{\text{Cost of Sequential}}$$

The cost deviation of the parallel implementation using 16 nodes was less than 1% for the adaptive method and 1.2% for the static method. The standard deviation of the adaptive method was smaller than that of the static method for all node ranges. This corresponds to the previous experiments (Table IV).

node size		Mean	Std. Dev.	Max.	Min.
2	Adp	1.47	0.16	1.67	1.15
	Static-5	1.18	0.09	1.28	1.02
4	Adp	3.55	0.50	3.95	2.11
	Static-5	2.76	0.29	3.16	2.05
8	Adp	6.98	0.94	7.64	4.13
	Static-5	4.89	0.47	5.46	3.97
16	Adp	11.94	1.34	13.04	8.10
	Static-5	7.63	0.84	9.45	6.35

Table V. Speedups of Adaptive and Static Methods

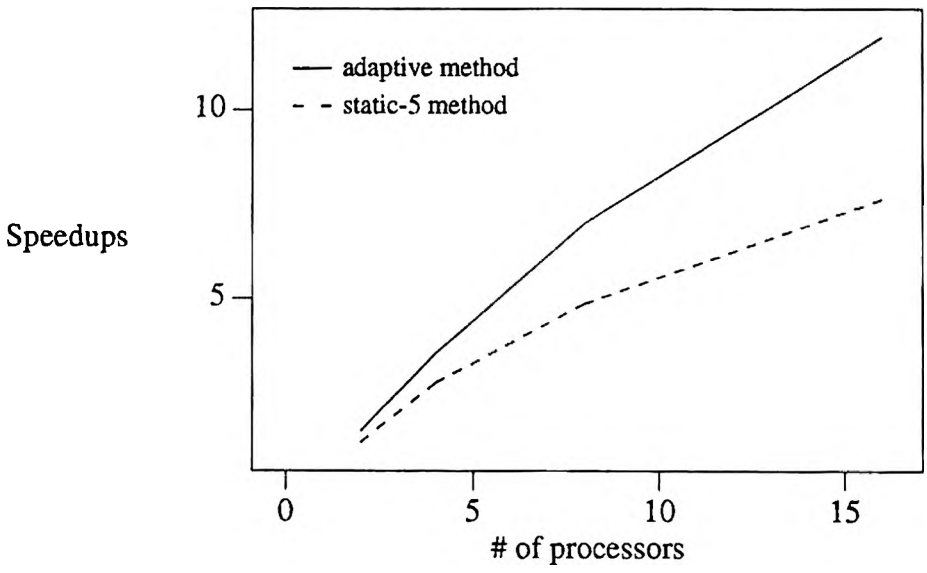


Figure 37. Speedups of Adaptive and Static Methods

node size		Mean	Std. Dev.
Seq.		583848	91797.9
2	Adp	587628	83453.7
	Static	588608	83138.0
4	Adp	578001	77181.2
	Static	575734	79882.2
8	Adp	574773	76009.6
	Static	575499	75534.0
16	Adp	589618	75242.4
	Static	591113	76073.4

Table VI. Final Cost of Adaptive and Static Methods.

determine the cost error behavior using a different set of the stream lengths. This experiment was similar to the the first experiment (Table IV). However, the weight of the cluster term was balanced with that of the affinity relation term in order to consider the packing density. The experiment was done 3 times using 128 regular patterns and 16 nodes. The initial temperature was set around 20,000; the decrement ratio was 0.98 to 0.985; and the Markov chain length was 10,000.

Stream Length	Time(Avg)	Cost(avg)	Performance(avg)
625	6.96378e+06	797019	5.55323e+12
208	7.0591e0+06	791015	5.58799e+12
125	7.31633e+06	790672	5.78597e+12
89	7.38138e+06	784053	5.78754e+12
69	7.48652e+06	794623	5.94925e+12
57	7.18042e+06	791005	5.67386e+12
45	7.52592e+06	787917	5.92617e+12
35	7.83123e+06	786564	6.15932e+12
26	8.44653e+06	784167	6.62457e+12
21	8.56733e+06	787604	6.75292e+12
16	9.43514e+06	788615	7.44400e+12
10	1.09276e+07	788676	8.61872e+12
5	1.74531e+07	783721	1.36727e+13
Avg	8.73649e+06	788896	6.88740e+12
Adaptive	7.20755e+06	786792	5.66971e+12
Sequential	9.34506e+07	753421	NA

Table VII. Performance of Static and Adaptive Methods

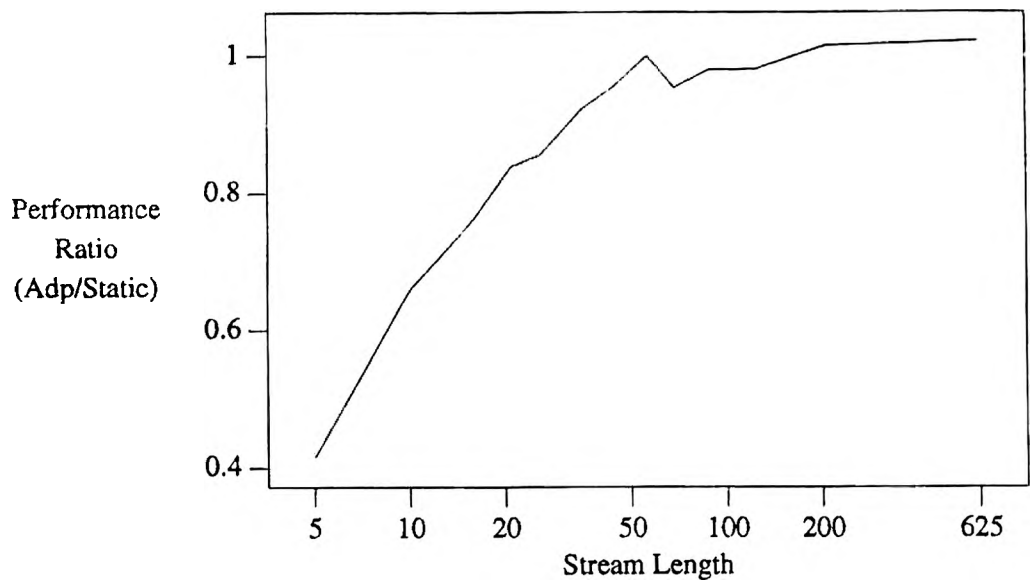


Figure 38. Performance of Adaptive vs. Static Methods

In Table VII, the *performance* is defined as the product of run time and cost.

$$\text{performance} = \text{run time} \times \text{final cost}$$

The performance was used as a parameter of a kind of goodness test for the trade-off between run time and cost. The cost deviation of the cost of the sequential process was 4.4% for the adaptive method and 4.7% for the static method.

In Figure 38, the *performance ratio* is defined as the performance of the static method over that of the adaptive method, $\frac{\text{Performance of Adaptive}}{\text{Performance of Static}}$. Figure 38 indicates that the static method can get a fairly good performance around the fixed stream length 100. In other words, considering the trade-off between the run time and the cost, 100 is a desirable stream length. When the stream length was larger than 200, the run time reduces with a relatively small increase of the cost. So the performance defined here

cannot represent the trade-off properly in large stream length regions. The performance of the adaptive method was lower than that of the static method in the proper regions. The average performance of the static method to that of the adaptive method was 0.82.

From the experimental results, the adaptive method is well suited for relaxing the frequency of the global updates, i.e. for increasing the stream length while maintaining the quality of the final results comparatively. Aside from the improved speedups, the adaptive method has an advantage over the static method in that, in the latter, much implementation is needed to determine the optimal stream length.

IX. CONCLUSION AND FUTURE RESEARCH

Simulated annealing is applied successfully to a broad range of combinatorial optimization problems, such as the traveling salesman problem, graph theory, VLSI design, and composite stock cutting. Since simulated annealing is a stochastic process, the real disadvantage is the massive computing time required to converge to a near optimal solution.

Many researchers have tried to fine tune the cooling schedule to reduce the computing time. This dissertation surveyed the theory and the sequential algorithms in Section II and III. However, the sequential annealing process is limited, being unable to reduce the massive computing time considerably due to the large amount of state changes required.

One promising approach for speeding up the simulated annealing is parallelization. From Section IV, distributed memory multicomputers show the most promise in achieving large parallel speedups. However, in a distributed memory architecture such as a hypercube, there is no globally available, centrally located system state. Updating the entire global state S thus involves explicit message traffic and is a critical bottleneck. To mitigate this bottleneck, it becomes necessary to amortize the cost of these state updates over as many parallel move evaluations as possible by using an approximate cost calculation. Thus, error in maintenance of the cost function $C(S)$ is inevitable and bounds must be placed on this error in order to assure convergence to the correct result.

Section V analyzed the behavior of the cost error. The analysis showed that a pessimistic move occurs more frequently than an optimistic move. Since pessimistic moves are more likely to be rejected, the hill climbing power decreases (due to the cost error) compared to the sequential annealing process. This reduced hill climbing power keeps the annealing process in the high local minima.

Section VI presented a new cost error measurement scheme which improved on the traditional methods, with some limitations. This method measures the cost error analytically based on the exponential distribution of the erroneous move decision. Section VI also proposed a new adaptive cost error-tolerance method in terms of stream length, based on the hill climbing power. This reduced hill climbing power is recovered by an additional small number of moves, i.e. by increasing the Markov chain length at a fixed temperature. The adaptive method derives the bounds on the cost error as a function of global update frequency, or stream length s.

Section VII discussed the details of parallel implementation on distributed multicomputers to solve the composite stock cutting problem. Although the simulated annealing process is conceptually straightforward, design of a successful annealing algorithm involves considerable engineering judgment. The sample results are included in the Appendix.

The adaptive cost error-tolerance method was implemented with a static stream length method in Section VIII. The experimental results showed that the adaptive method was well suited for relaxing the frequency of the global update, maintaining comparable quality of the results. The adaptive method has an advantage over the static stream length method in that, in the latter, to determine the optimal stream length, the static method has to run many experiments. However, the adaptive method varies the stream length dynamically by choosing a large stream length in high and low temperature regions, and a small stream length in the critical temperature region.

For future research, since the asynchronous spatial decomposition method achieves greater speedups in the distributed memory multiprocessor, it may be desirable to implement this method on different problems, such as the VLSI placement problem and the traveling salesman problem, using the adaptive cost error-tolerance method. Since the

adaptive method is independent of the problem, this will shed greater light on the utility of the adaptive method.

In the spatial decomposition algorithm, every processor has to hold sufficient space to ensure the mobility of patterns in move generation. If the space for the processor is too small, the pattern cannot move around freely because the pattern can move at most to the space of the neighbor processor. This is why the number of processor is limited two in inter-move operation for simplicity. So, the number of processors is limited in a small-sized problem, even though a small size of problem consumes considerable computing time. Since the division algorithm in Section IV.C.3, as a modified Systolic algorithm, gives linear speedups in the high temperature region, it is desirable to combine the division algorithm with a spatial decomposition algorithm in the high temperature region, and to combine a SSS algorithm with a spatial decomposition algorithm in the low temperature region. By doing this, the speedups increase considerably with large number of processors. In the high temperature region, a set of processors form a cluster, which cooperate to implement the spatial decomposition algorithm. Each cluster copies the current state, then completes a stream of moves at the same temperature. That is, a division algorithm of a spatial decomposition algorithm is employed. In the low temperature region, the division algorithm is replaced by the SSS algorithm.

The second approach to the parallel implementation is using the pyramid architecture (Figure 39). In a simulated annealing process, the ranger limiter scheme is used to increase the acceptance rate. The range limiter does not affect the irreducibility of the Markov chain, so it does not affect the convergence property. The range limiter simply prunes away some of those transitions whose probability becomes too small at low temperatures. The pyramid architecture is fitted to the range limiter. In the high temperature region, the size of the range limiter is large, i.e. a large cost change move is allowed. So one processor is used which covers all space. As the temperature goes down, the size of

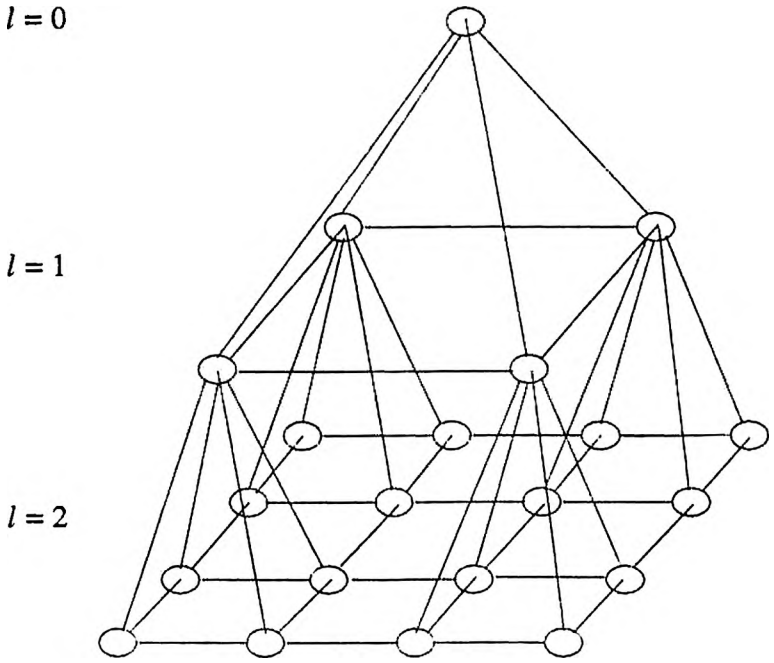


Figure 39. Pyramid Architecture

the range limiter decreases also, i.e. only a small cost change move is allowed. So more processors can be used without hindering mobility in move generations. This method also decreases the cost error in the high temperature region, preserving the convergence property of the sequential simulated annealing algorithm with a very high utilization of processors.

The third approach for speedups in the sequential and parallel implementation is using the multigrid method (Figure 40). In the stock-cutting and non-slicing placement problems, bitmaps are used to represent modules or patterns. The required time for move generation is excessive because of bitmap manipulation. In the high temperature region, the overlap cost error can be tolerated easily. So a high level grid can be used for the pattern, where the bitmap of every pattern is shrunk proportially, reducing the time for the

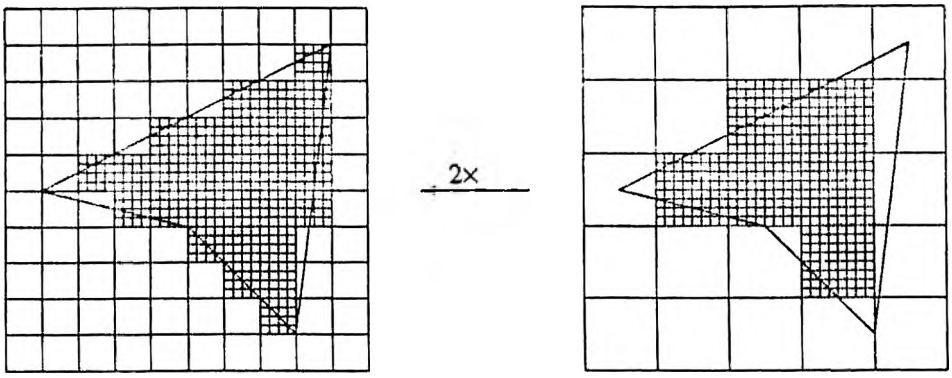
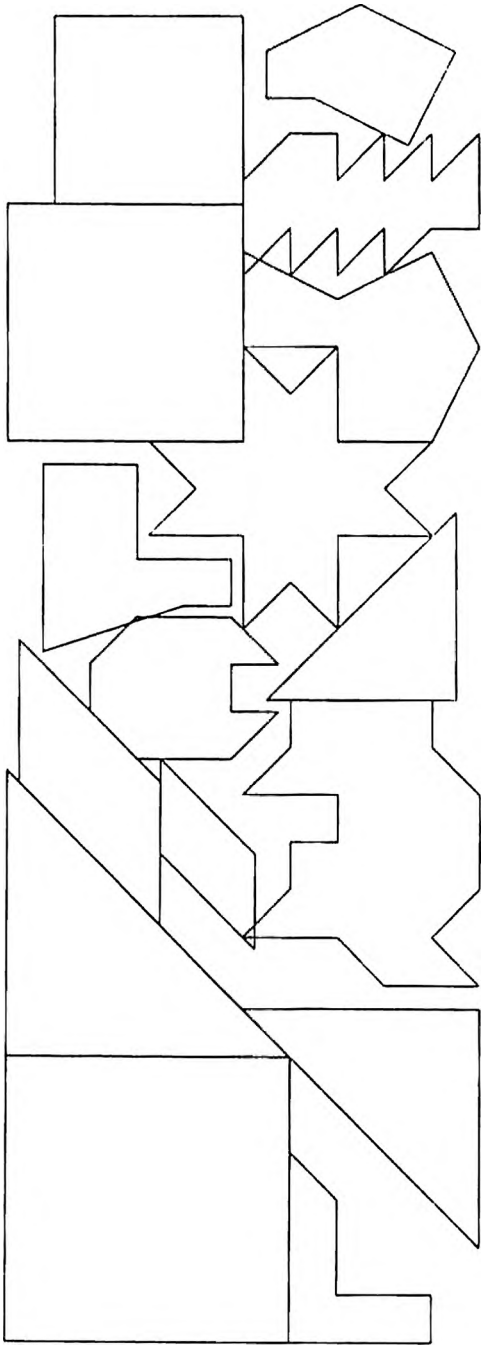


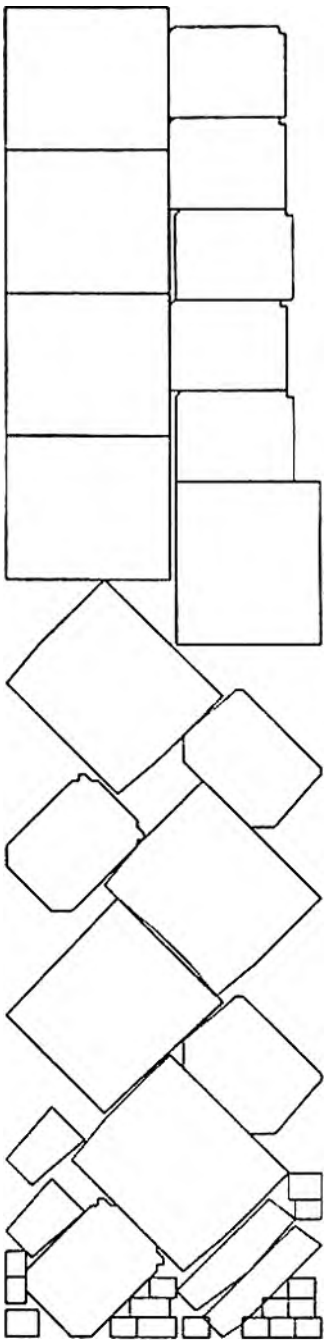
Figure 40. Multigrid Method

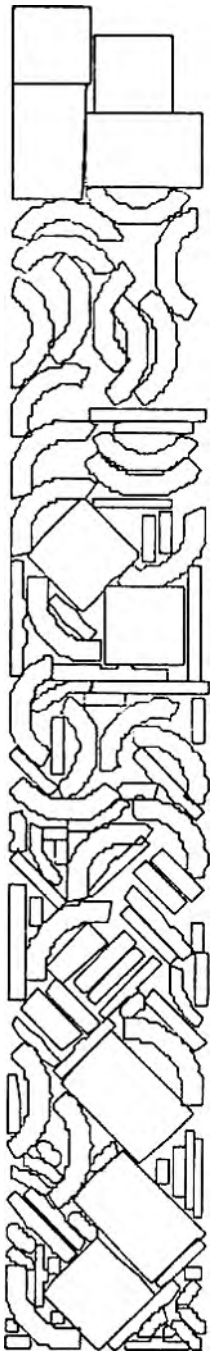
move. The overlap cost error is caused by an inaccurate bitmap of the module. As the temperature goes down, the cost error becomes prohibitive. So a more fine grid is used for the patterns, which decreases the overlap cost error. Consequently in the final temperature, the overlap cost error is removed. Finally, the bitmap represents all patterns accurately. The key issue is when to change the grid size.

APPENDIX

Sample Placement of Stock Cutting







BIBLIOGRAPHY

[AaBo86a] Aarts, E.H.L., de Bont, F.M.J., Habers, E.H.A., and van Laarhoven, P.J.M., "A Parallel Statistical Cooling Algorithm," *Proceedings of the Symposium on the Theoretical Aspects of Computer Science,* vol. 210, January 1986, pp 87-97.

[AaBo86b] Aarts, E.H.L., de Bont, F.M.J., Habers, J.H.A and van Laarhoven, P.J.M., "Parallel Implementations of the Statistical Cooling Algorithm," *Integration,* vol. 4, 1986, pp 209-238.

[AaLa85] Aarts, E.H.L. and van Laarhoven, P.J.M., "Statistical Cooling: A General Approach to Combinatorial Optimization Problems," *Philips J. of Research,* vol. 4, 1985, pp 193-226.

[AlCa89] Allwright, J., and Carpenter, D.B., "A Distributed Implementation of Simulated Annealing for The Traveling Salesman Problem," *Parallel Computing,* vol. 10(3), May 1989, pp 335-338.

[BaJo90] Banerjee, P., Jones, M.H., and Sargent, J.S., "Parallel Simulated Annealing Algorithm for Cell Placement on Hypercube Multiprocessors," *IEEE Trans. on Parallel and Distributed System,* VOL. 1, NO. 1, January 1990.

[Beas85a] J.E. Beasley, "Algorithms for Unconstrained Two-dimensional Guillotine Cutting," *J. Opl. Res. Soc.* 36 (4), pp 297-306, 1985

[Beas85b] J.E. Beasley, "An Exact Two-dimensional Non-guillotine Cutting Tree Search Procedure," *Operations Research,* 33 (1), 49-64, 1985

[BrBa88] Brouwer, R. and Banerjee, P., "A Parallel Annealing Algorithm for Channel Routing on Hypercube Multiprocessor," *Proc. Int. Conference on Computer Design,* 1988, pp 4-7.

[CaRo87] Casotto, A., Romeo, F. and Sangiovanni-Vincentelli, A., "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *IEEE Transactions on Computer-Aided Design,* September 1987, pp 838-847.

[Cern85] Cerny, V., "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *J. Opt. Theory Appl.,* vol. 45, 1985, pp 41-51.

[ChEd88] Chamberlain, R.D., Edelman, M.N., Franklin, M.A., and Witte, E.E., "Simulated Annealing on a Multiprocessor," *Proc. the International Conference on Computer Design,* 1988, pp 540-544.

[ChWh77] N. Christofides and C. Whitlock, "An Algorithm for Two-dimensional Cutting Problems," *Operations Research,* 25, pp 30-44, 1977.

[DaKi87] Darema, F., Kirkpatrick, S. and Norton, A.V., "Parallel Algorithms for Chip Placement by Simulated Annealing," *IBM Journal of Research and Development,* vol. 31(3), May 1987, pp 391-402.

[DaPf87] F. Darema and G.F. Pfister, "Multipurpose Parallelism for VLSI CAD on the RP3," *IEEE Design & Test of Computers,* pp 19-27, Oct. 1987.

[Dura89] M.D. Durand, "Parallel Simulated Annealing: Accuracy versus speed in placement," *IEEE Design and Test,* June, 1989, pp 8-34.

[GaJo79] Garey, M.R., and Johnson, D.S., "Computers and Intractability: A Guide to the Theory of NP-Completeness," *W.H. Freeman and Co., San Francisco,* 1979.

[GeMa72] A.M. Geoffrion and R.E. Marsten, "Integer Programming algorithms: A Framework and State-of-the-art survey," *Management Science,* 18 (19), pp 465-491, 1972.

[GiGo61] Gilmore, P.C., and Gomery, R.E., "A Linear Programming Approach to the Cutting-Stock Problem," *Operations Research,* 9, 1961, pp 849-859.

[GiGo63] Gilmore, P.C., and Gomery, R.E., "A Linear Programming Approach to the Cutting-Stock Problem," *Operations Research,* 11, 1963, pp 863-888.

[GiGo65] Gilmore, P.C., and Gomery, R.E., "A Multistage Cutting-Stock Problems of Two and More Dimensions," *Operations Research,* 13, 1965, pp 94-120.

[Glov90] Fred Glover, "TABU SEARCH: A TUTORIAL," Technical Reprot, System Science Center for Applied Artificial Intelligence, University of Colorado, Boulder, 1990.

[GrDa89] Greening, D.R. and Darema, F., "Rectangular Spatial Decomposition Methods for Parallel Simulated Annealing," *Proc. Int. Conference on Supercomputing,* Crete, Greece, June 1989, pp 295-302.

[Gree89] Greening, D.R., "A Taxonomy of Parallel Simulated Annealing Techniques," *Computer Science Technical Report, University of California, Los Angeles,* CSD-890050, August 1989.

[GrHa85] Gross, D. and Harris, C.M., "Fundamentals of Queueing Theory," *Wiley Series in Probability and Mathematics Statistics, John Wiley & Sons, Inc.,* 1985.

[Grov86] Grover, L.K., "A New Simulated Annealing Algorithm for Standard Cell Placement," *Proc. the International Conference on Computer-Aided Design,* November 1986.

[GrSu84] Greene, J.W. and Supowit, K.J., "Simulated Annealing Without Rejected Moves," *Proc. IEEE Int. Conference on Computer Design,* Port Chester, November 1984, pp 658-663.

[Haes80] R.W. Haessler, "A Note on Computational Modifications to the Gilmore-Gomory Cutting Stock algorithm," *Operations Research,* 28 (4), pp 1001-1005, 1980.

[Haje85] Hajek, B., "Tutorial survey of Theory and Applications of Simulated Annealing," *Proc. 24th Conf. on Decision and Control,* Ft. Lauderdale, December 1985, pp 755-760.

[Hinx80] A.I. Hinxman, "The Trim-loss and Assortment Problems: A Survey," *European Journal of Operational Research,* 5, pp 8-18, 1980.

[HuRo86] Huang, M.D., Romeo, F. and Sangiovanni-Vincentelli, A.L., "An Efficient General Cooling Schedule for Simulated Annealing," *Proc. IEEE Int. Conference on Computer-Aided Design,* Santa Clara, November 1986, pp 381-384.

[JaDa88] Jayaraman, R. and Darma, F., "Error Tolerance in Parallel Simulated Annealing Technique," *Proc. the International Conference on Computer Design,* 1988.

[JaRu87] Jayaraman, R. and Rutenbar, R.A., "Floorplanning by Annealing on a Hypercube Multiprocessor," *Proc. the International Conference on Computer-Aided Design,* 1987.

[JoAr87] Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C., "Optimization by Simulated Annealing: An Experimental Evaluation," *Part I and II, AT & T Bell Lab.,* 1987.

[KiGe83] Kirkpatrick, S., Gellatt Jr., C.D. and Vecci, M.P., "Optimization by Simulated Annealing," *Science,* vol. 220, 1983, pp 975-986.

[KiKi90] Kim, Y. and Kim, M., "A Stepwise-Overlapped Parallel Simulated Annealing Algorithm," *Integration,* vol. 10, 1990, pp 39-54.

[KrRu87] Kravitz, S.A. and Rutenbar, R.A., "Placement by simulated Annealing on a multiprocessor," *IEEE Trans. on Computer-Aided Design,* July 1987, pp 534-549.

[LaAa88] van Laarhoven, P.J.M., and Aarts, E.H.L., "Simulated Annealing: Theory and Applications," *D.Reidel Publishing Co.,* 1988.

[Lam88] Lam, J., "An Efficient Simulated Annealing Schedule," *Ph. D. Dissertation, Computer Science Dept., Yale Univ.* 1988.

[LiKe73] Lin, S. and B.W. Kernighan, "An Effective Heuristic for the Traveling-Salesman Problem," *Operation Research,* 21, 1973, pp 498-516.

[LMPD92] Lutfiyya H., McMillin B., Poshyanonda P., and Dagli C., "Composite Stock Cutting Through Simulated Annealing," *Mathl. Comput. Modeling,* Vol. 16, No. 1, 1992, pp 57-74. Pergamon Press, New York.

[LuMe86] Lundy, M. and Mees, A., "Convergence of an Annealing Algorithm," *Math. Prog.,* vol. 34, 1986, pp 111-124.

[MeRo53] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E., "Equation of State Calculations by Fast Computing Machines," *J. of Chem. Physics,* vol. 21, 1953, pp 1087-1092.

[Morr91] Mike Morrow, "Genetic Algorithms: A new class of searching algorithms," *Dr. Dobb's Journal,* April 1991, pp 26-32.

[NgRa86] A. Ng, P.Raghavan, and C. Thompson, "A Specification Language for Describing Rectilinear Steiner Tree Configurations," *Proc. 23rd Design Automation Conference,* 1986.

[PaSt82] Papadimitriou, C.H. and K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity," *Prentice-Hall, New York,* 1982.

[ReFu87] Reed, D.A. and Fujimoto, R.M., "Multicomputer Network: Message-Based Parallel Processing," *The MIT Press*, 1987.

[Reif65] Reif, F., "Fundamentals of Statistical and Thermal Physics," *McGraw-Hill, Inc., New York,* 1965.

[RoBl86] J.S. Rose, D.R. Blythe, W.M. Snelgrove, and Z.G. Vranesic, "Fast, High Quality VLSI Placement on an MIMD Multiprocessor," *Proc. International conference on Computer-Aided Design,* 1986, pp 42-45.

[RoDr90] Roussel-Ragot, P. and Dreyfus, G., "A Problem Independent Parallel Implementation of Simulated Annealing: Models and Experiments," *IEEE Trans. on Computer-Aided Design,* VOL. 9(8), August 1990.

[RoKl90] Jonathan Rose, Wolfgang Klebsch, and J. Wolf, "Temperature Measurement and Equilibrium Dynamics of Simulated Annealing Placement," *IEEE Trans. on Computer-Aided Design,* VOL. 9. NO. 3. March 1990

[Sark88] B.R. Sarker, "An Optimum Solution for One-dimensional Slitting Problems: A Dynamic Programming Approach," *J. Opl. Res. Soc.,* 39 (8), pp 749-755, 1988.

[Sech87] Sechen, C.M., "Placement and Global Routing of Integrated Circuits Using Simulated Annealing," *Ph. D. Dissertation, University of California, Berkeley,* 1987.

[Sech88] Sechen, C.M., "Chip-Planning, Placement and Global Routing of Macro/Custom Cell Integrated Circuit Using Simulated Annealing," *25th ACM/IEEE Design Automation Conference,* 1988.

[Sene81] Seneta, E., "Non-negative Matrices and Markov Chains," *Springer Verlag, New York,* 2nd ed., 1981.

[ShMa91] Shahookar, K. and Mazumder, P., "VLSI Cell Placement Techniques," *Acm Computing Surveys,* vol. 23(2), June 1991.

[Wayn91] Peter Wayner, "Genetic Algorithms," *BYTE,* January 1991, pp 361-368.

[Whit84] White, S.R., "Concepts of Scales in Simulated Annealing," *Proc. IEEE Int. Conference on Computer Design,* Port Chester, November 1984, pp 646-651.