Combinatorial Optimization in VLSI Physical Design

by

Peter Anthony Walsh

B.Sc. (Honours), National University of Ireland, 1982
M.Sc. (Honours), National University of Ireland, 1985

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor in Philosophy
in the Department of Computer Science

We accept this dissertation as conforming
to the required standard

Dr. D.M. Miller, Supervisor (Department of Computer Science)

Dr. L.A. Ellis, Departmental Member (Department of Computer Science)

Dr. M. Serra, Departmental Member (Department of Computer Science)

Dr. F. M-Guibaiy, Outside Member (Department of Electrical and Computer Engineering)

Professor J.T. Mowchenko, External Examiner (Department of Electrical Engineering, University of Alberta.)

Supervisor: Dr. D.M. Miller

Abstract

Simulated Annealing is a general purpose combinatorial optimization technique which has been applied to many problems in VLSI design. In essence, simulated annealing is Monte Carlo iterative improvement with the ability to conditionally accept uphill moves. The notion of a cooling schedule is common to all simulated annealing implementations. A cooling schedule can be thought of as simulated annealing's control mechanism.

Experiential work has been done on estimating the cost of an optimal solution to some combinatorial optimization problem instances. Such an estimate can be used to determine termination criteria for general purpose optimization techniques such as iterative improvement or simulated annealing. We have extended this idea and designed a complete simulated annealing general cooling schedule based on the cost of an optimal solution to a problem instance. We call the resultant schedule an *extended goal-directed* general cooling schedule.

One of the major problems with simulated annealing is its long computation times. This problem can be addressed by first using a fast heuristic to find a good initial configuration and then applying simulated annealing. This approach is called *Simulated Sintering*.

To exploit the potential of simulated sintering one needs an appropriate general cooling schedule. The extended goal-directed cooling schedule is equally applicable to simulated annealing and simulated sintering.

To date, no one cooling schedule has proven suitable for all optimization problem instances. In our view, no such cooling schedule exists. Consequently, we have attempted to identify the type of problem best suited to optimization by simulated annealing and simulated sintering using the extended goal directed schedule.

We have applied the extended goal-directed schedule to standard cell placement and floorplanning problems using both simulated annealing and simulated sintering. Within this context, we have compared the performance of the extended goal directed schedule to other published schedules. Our results indicate that in terms of layout quality, the extended goal-directed schedule performs as well or better than the other schedules.

In this dissertation, we have developed a new general cooling schedule. Our evaluation of the extended goal-directed schedule suggests that it is a useful research contribution in the area of simulated annealing algorithms.

Examiners:

Dr. D.M. Miller, Supervisor (Department of Computer Science)

Dr. J.A. Ellis, Departmental Member (Department of Computer Science)

Dr. M. Serra, Departmental Member (Department of Computer Science)

Dr. F. El-Guibaly, Outside Member (Department of Electrical and Computer Engineering)

Professor J.T. Mowchenko, External Examiner (Department of Electrical Engineering, University of Alberta.)

Contents

List of Figures

List of Tables

Notation

C	Cost function.
$\hat{C}$	Average cost.
ΔC	Change in cost $C(s_i) - C(s_{i-1})$.
$\widehat{\Delta C}$	Average ΔC.
ΔC_{max}	Greatest ΔC for a particular temperature value.
E_0	Initial target configuration cost.
E_j	Target configuration cost for temperature index j.
E_q	Optimal configuration cost.
$\mathcal{F}$	Simulated annealing configuration acceptance function.
$\mathcal{G}$	Temperature decrement function.
G	Neighbourhood mapping.
N	Number of configurations in a neighbourhood.
$mean$	Mean.
Q	Problem size.
s_{max}	Configuration with greatest cost accepted at a particular temperature value.
s_{min}	Configuration with smallest cost accepted at a particular temperature value.
$s = x_0, x_1, .., x_n$	Configuration s of parameters $x_0, x_1, ..., x_n$.

S	Configuration space.
$s_0, s_1, .., s_i, ..s_p \ (0 \leq i \leq p)$	Sequence of accepted configurations.
s_0	Initial configuration.
std	Standard deviation.
T	Temperature.
$T_0, T_1, ..T_j, .., T_q \ (0 \leq j \leq q)$	Sequence of temperature values.
T_0	Initial temperature.
T_q	Final temperature.
$\mathcal{T}$	Temperature for extended goal-directed schedule.
x_{MIN}	Terminal configuration with best cost.
x_{MAX}	Terminal configuration with worst cost.

Acknowledgments

I wish to thank my supervisor, Dr. D.M. Miller, for his help and guidance during the course of this work.

I would also like to thank my committee members, Dr. J.A. Ellis, Dr. M. Serra, Dr. F. El-Guibaly and Dr. J.T. Mowchenko.

I acknowledge the financial support from a University of Victoria Fellowship and I wish to thank the Department of Computer Science for the use of its facilities.

Dedication

To my parents Marie and Jerry

To my wife Barbara and our son, Michael John Patrick

Chapter 1

Introduction

1.1 Motivation

Many problems in engineering and the sciences involve choosing a 'best' solution from among a large number of possible solutions. This type of problem is known as *optimization*. In the past, optimization techniques have been developed in an *ad hoc* fashion yielding a multitude of single purpose procedures. Some general purpose techniques also exist. In this dissertation we study one of the more recent general purpose optimization techniques called Simulated Annealing (and a derivative of simulated annealing called Simulated Sintering).

1.1.1 Simulated Annealing

All existing simulated annealing implementations strive to find a 'best' solution to an optimization problem. One solution is judged better than another by some metric known as solution cost. A best solution is found by searching a portion of the total

number of solutions to the optimization problem. The existing simulated annealing procedures terminate once certain termination criteria are satisfied. A solution of least cost encountered during the search is deemed the 'best' solution. Typically, these termination criteria are time related and are chosen without regard to the cost of an optimal solution to the optimization problem. The primary motivation for this dissertation is to determine how the cost (exact, estimated, or lower-bound) of an optimal solution can be used to control simulated annealing's configuration search.

Research on simulated annealing algorithms can be divided into three areas namely, applications, acceleration and foundations. The area of *applications* is concerned with applying the simulated annealing technique to problems. Research in *acceleration* is concerned with implementation issues such as efficient data structures and efficient cost function evaluation. Research on the *foundations* of simulated annealing is concerned with determining which problems are amenable to optimization by simulated annealing; the development of new cooling schedules and determining which cooling schedule best suites a particular problem instance.

This dissertation is concerned with research on the foundations of simulated annealing. In particular, we develop a new cooling schedule called the extended goal-directed schedule which is based on the cost (exact, estimated or lower-bound) of an optimal solution to a problem instance. We give guidelines on when to use the extended goal-directed schedule based on problem-instance statistics.

We apply simulated annealing using the extended goal-directed schedule to five instances of an idealized problem from VLSI physical design. This is done to evaluate the schedule's performance. In order to put this work in context, what follows is a brief discussion of the digital system design process of which physical design is the last phase.

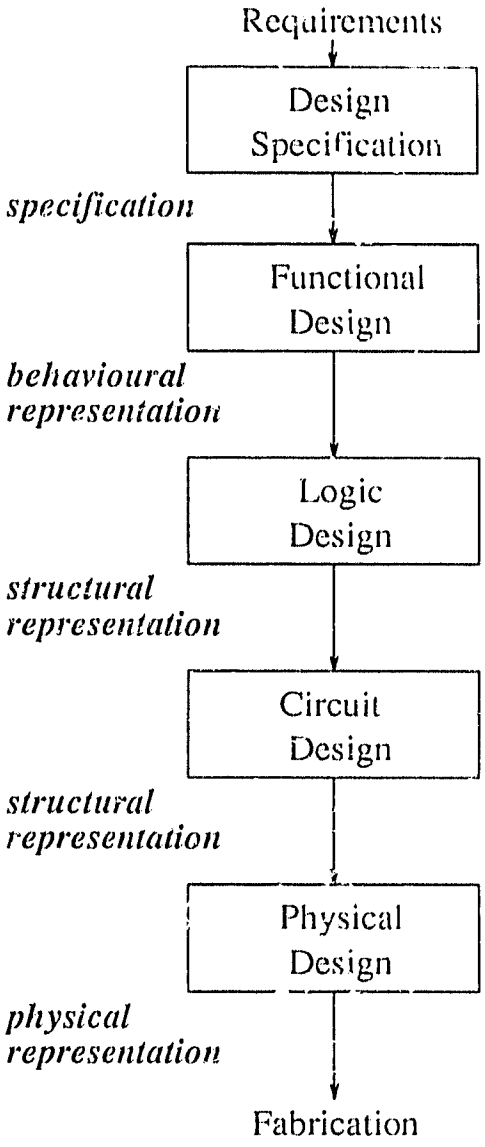


Figure 1.1: Phases of Digital System Design.

1.1.2 The Digital System Design Process

The digital system design process can be divided into five phases (Fig. 1.1). In phase one, a *design specification* is produced based on the requirements of the system. The design specification phase is usually a time-consuming manual step.

Functional design is concerned with system behaviour. For a given system input, a *behavioural representation* enables one to determine the system's output. For example, a truth table is one type of behavioural representation.

The *logic design* phase is concerned with the logic structures that implement the functional design. The *circuit design* phase is concerned with the the electrical characteristics of basic circuit elements. Both these phases contribute to a *structural representation* of the system which describes components of the system and their interaction.

In the *physical design* phase, the structural and behavioural representations from the previous phases are transformed to a *physical representation*. A physical representation defines geometric shapes used in the fabrication of the digital system.

The secondary motivation for this dissertation is to gain experience in the physical design of integrated circuits. Much work has been done on automating the physical design phase. The importance of this problem continues to grow due to the increasing complexity of integrated circuits. Consequently, work on physical design automation is still, and will continue to be, an active and challenging research area.

1.1.3 Goal Oriented Optimization in VLSI

Optimization in VLSI can be goal oriented in the sense that a circuit must be optimized to fit into a certain area or perform to some specification. This type of scenario

is well suited to the notion of using the estimated cost or lower bound cost of an optimal solution to control simulated annealing's search for a best solution. This is the case since the cost of an acceptable solution can be substituted for the cost of an optimal solution. In this way, simulated annealing's search for a best solution can be tailored for a specific problem instance.

1.2 Dissertation Organization

In Chapter 2, we introduce a framework for discussing optimization. This framework includes definitions, an optimization benchmark and a discussion of one of the simplest optimization techniques namely, iterative improvement.

Optimization by simulated annealing is introduced in Chapter 3 along with several cooling schedules (or control mechanisms).

In Chapter 4 we introduce the goal-directed cooling schedule and the extended goal-directed schedule. Both are based on the cost (exact, estimated or lower bound) of an optimal solution to a problem instance. The performance of the extended goal-directed schedule is evaluated on six instances of an idealized placement problem.

Chapter 5 describes simulated sintering which is another general purpose optimization technique similar to simulated annealing. We show how the extended goal directed schedule can be used as a cooling schedule for simulated sintering.

In Chapter 6, we describe two VLSI physical design styles namely, standard cell and macro/custom. We report the results obtained by applying simulated annealing (and simulated sintering) using the extended goal-directed to eight standard-cell placement problems. We also demonstrate how simulated annealing (and simulated sintering) using the extended goal-directed schedule can be applied to macro/custom

placement.

In Chapter 7, we give guidelines on when to use the extended goal-directed schedule based on problem-instance statistics. We close Chapter 7 with a dissertation summary, including research contributions, and suggestions for possible future work.

Chapter 2

Optimization

The purpose of this chapter is to provide background material for this dissertation. We begin by defining combinatorial optimization. A framework for discussing general purpose combinatorial optimization is then introduced. This framework consists of definitions and notation, and an idealized placement problem. The purpose of the idealized placement problem is to provide a benchmark for comparing the performance of general purpose combinatorial optimization procedures.

Iterative improvement is one of the simplest general purpose combinatorial optimization techniques. The latter part of this chapter contains a description of iterative improvement and an evaluation of its performance on our idealized placement problem. We include this now because later in the dissertation, we describe and evaluate other more elaborate general purpose combinatorial optimization techniques relative to the description and evaluation of iterative improvement.

2.1 Combinatorial Optimization

A *problem instance* can be described by a title, a specification and a question. For example, we can describe an instance of two-level Boolean minimization [8] as follows:

Title :	Two-level Boolean Minimization.
Specification :	$f(a,b) = \overline{a}b+ab$
	(Canonical sum of products expression.)
Question :	Find a sum of products representation for $f(a,b)$
	where the number of product terms is minimized.

The above problem instance has four *feasible solutions* namely, $\overline{a}b+ab$, $\overline{a}b+b$, $ab+b$ and b. All feasible solutions to a problem instance solve the given problem.

A real number is associated with each feasible solution. This number is the *cost* of the feasible solution with respect to some optimization criteria. A function which maps feasible solutions to costs is called an *objective function.* If we define the objective function in our example problem to be the number of product terms in a sum of products expression then $\overline{a}b+ab$, $\overline{a}b+b$ and $ab+b$ each have a cost of 2 and, b has a cost of 1.

An *optimal solution* to a problem instance is a feasible solution with minimum cost. With reference to our example problem, the feasible solution b is an optimal solution since it has the least cost from among the four feasible solutions. A *near-optimal solution* to a problem instance is a feasible solution with cost close (in some sense) to the cost of an optimal solution.

An optimization problem instance is completely characterized by a set of feasible solutions and their costs. An optimization problem instance is solved by selecting an optimal solution or in some cases, a near optimal solution from the set of feasible so-

lutions. If an objective function is a function of discrete variables then, the associated optimization problem is called *combinatorial optimization* (CO).

2.2 Definitions and Notation

Let s be a *configuration* of parameters which represents a feasible solution to an optimization problem instance.[1] Let the set of all feasible solutions to a problem instance be represented by the set of configurations S ($s \in S$). The set S is called a *configuration space*. Let C be an objective cost function. (Note, definitions 1 through 6 come from Papadimitriou and Steiglitz [22]. Definitions 7, 8 and 9 are provided by the author.)

Definition 1 *An instance of a CO problem is a pair (S, C) where S is a configuration space and C is a mapping*

$$C : S \rightarrow \Re$$

Definition 2 *A CO problem is a set of CO problem instances.*

Definition 3 *The aim of optimization is to find an optimal configuration $s_x \in S$ where*

$$C(s_x) \leq C(s_y) \quad \forall\, s_y \in S$$

The term global minimum configuration is a synonym for optimal configuration.

Definition 4 *A neighbourhood is a mapping*

$$G : S \rightarrow 2^S \text{ (the power set of } S)$$

defined for each CO problem instance.

[1] From this point on, the terms feasible solution and configuration are used interchangeably

Definition 5 *The configuration s_y is a neighbour of the configuration s_x ($s_x, s_y \in S$) if*

$$s_y \in G(s_x)$$

Definition 6 *The configuration $s_x \in S$ is a local minimum if*

$$C(s_x) \leq C(s_y) \quad \forall\, s_y \in G(s_x)$$

Definition 7 *A down-hill path of length n is a sequence $(s_1, s_2, .., s_n)$ where*

(a) $s_x \in S$ $(1 \leq x \leq n)$

(b) $C(s_n) < C(s_{n-1}), C(s_{n-1}) < C(s_{n-2}), .., C(s_2) < C(s_1)$

(c) $s_n \in G(s_{n-1}), s_{n-1} \in G(s_{n-2}), .., s_2 \in G(s_1)$

Definition 8 *A region is a pair (s_x, R) where $s_x \in S$, $R \subseteq S$ and $\forall\, s_y \in R$, both s_x and s_y exist on the same down-hill path and $C(s_x) < C(s_y)$.*

Definition 9 *The configuration $s_x \in S$ is a regional local minimum in the region (s_x, R).*

Throughout this dissertation, the notions of *hills* and *valleys* are used to describe CO problems. Figure 2.1 shows a model cost versus configuration curve. In reality, a plot of cost versus configuration for a CO problem instance would be a multi-dimensional contour. For discussion purposes, we model this contour as a two-dimensional curve. In keeping with this model, we use descriptive language which is continuous in nature. This is done with full knowledge that cost versus configuration contours are discrete.

With regard to Figure 2.1, adjacent configurations on the horizontal axis are neighbours and the curve itself is viewed as a terrain of hills and valleys. Each valley is considered a region. A regional local minimum is located at the bottom of each valley.

A procedure that solves a CO problem is called a CO *algorithm*. Within the analogy of hills and valleys, an optimization algorithm corresponds to placing a *ball* somewhere on a cost versus configuration curve and giving it the ability to roll. Algorithm termination corresponds to stopping the ball. The configuration at which the ball starts to roll is called the *initial configuration*. The configuration with least cost, encountered while the ball is rolling, is called the *terminal configuration*.

2.3 Heuristic Optimization

An optimization algorithm is *exact* if it yields an optimal configuration to every instance of an optimization problem. One obvious exact algorithm for CO is to examine each configuration in the configuration space of the problem instance and select a configuration which minimizes the objective function.

Many CO problems are NP-hard [33]. All known exact algorithms for NP hard optimization problems require a computational effort whose worst case time complexity is at least exponential in the size of the problem instance. For large instances of NP-hard optimization problems, the use of any known exact optimization algorithm would be impractical since it would require prohibitively large execution times.

As an alternative to exact algorithms, *heuristic* or approximate algorithms may be used to solve CO problems. Heuristic algorithms yield optimal or near optimal configurations to NP-hard optimization problems and typically require a computa

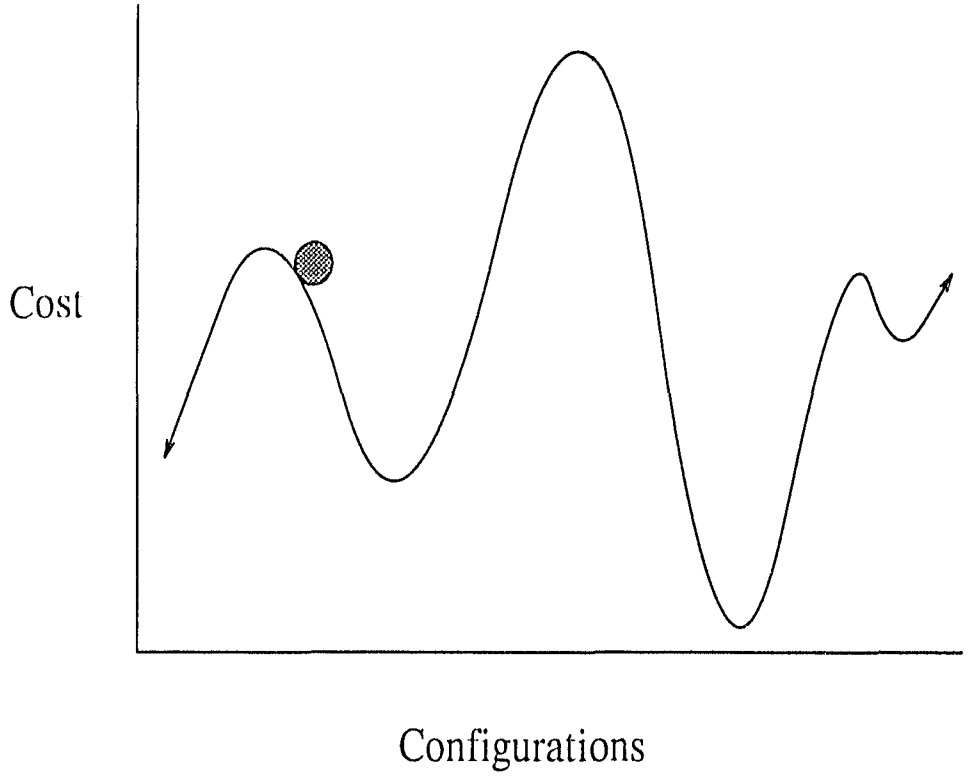


Figure 2.1: Cost *vs.* Configuration: Hills and Valleys.

```
procedure general purpose combinatorial optimization(in s_0; out s_p);

    var s_0,s_p,s,X:configuration;

        (* s_0  =  initial configuration          *)
        (* s_p  =  terminal configuration         *)
        (* X    =  current configuration          *)
        (* s    =  selected neighbour configuration *)

    begin
    X := s_0;
    repeat
        select s; (* s ∈ G(X) *)
        if accept(C'(s)) then
            X := s
        else
            NOP
    until termination
    s_p:=X
    end;
```

Figure 2.2: General Purpose Combinatorial Optimization.

tional effort whose worst-case time complexity is bounded polynomialy in the size of the problem instance. Low order polynomial time algorithms can be practical.

2.4 General Purpose Optimization

Optimization algorithms which are problem independent are called *general purpose*. General purpose optimization algorithms presuppose the existence of a current configuration, a neighbour selection mechanism and an objective function. We limit our attention to algorithms which proceed by selecting a neighbour of the current config

uration and *accepting* or *rejecting* it based on its cost. In the event that the selected neighbour configuration is accepted, then it is made the current configuration. If the selected neighbour configuration is rejected then it is discarded. The above procedure is repeated until some termination criteria are satisfied. A pseudocode definition of general purpose combinatorial optimization is shown in Figure 2.2.

2.5 Monte Carlo Combinatorial Optimization

The term *Monte Carlo Method* is used to describe any algorithm that employs random numbers [15]. Combinatorial optimization using a Monte Carlo method is called *Monte Carlo combinatorial optimization*. Monte Carlo combinatorial optimization techniques come in many different flavours [20]. In this dissertation, we examine four Monte Carlo combinatorial optimization techniques namely, *Monte Carlo Iterative Improvement* (Chapter 2), *Simulated Annealing*, *Simulated Sintering* (Chapters 3,4 and 5) and *Simulated Evolution* (Chapter 7).

2.6 Iterative Improvement

Iterative improvement is a general purpose optimization technique. Starting from an initial configuration $s_0 \in S$, iterative improvement transforms s_0 to a terminal configuration $s_p \in S$. This is achieved by constructing a sequence of configurations $s_0, s_1, ..., s_i, ... s_p$. The configuration s_i must come from s_{i-1}'s neighbourhood $(1 \leq i \leq p)$. In addition, $C(s_i) < C(s_{i-1})$, $(1 \leq i \leq p)$.

```
procedure iterative improvement(in s_0; out s_p);

    var s_0,s_p,s,X:configuration;

    begin
    X := s_0;
    repeat
        select s; (* s ∈ G(X) *)
        if C(s) < C(X) then
            X := s
    until C(X) ≤ C(s_x)   ∀ s_x ∈ G(X);
    s_p:=X
    end;
```

Figure 2.3: Iterative Improvement.

The acceptance of s_i as the successor to s_{i-1} is said to be a *down-hill move* since the cost of s_i must be less than the cost of s_{i-1} $(1 \leq i \leq p)$.[2] Iterative improvement permits only down-hill moves and terminates when no further down hill moves are possible, *i.e.*, when $C(s_p) \leq C(s_k)\ \forall\ s_k \in G(s_p)$. A pseudocode definition of iterative improvement is shown in Figure 2.3.

To date, it has not been possible to give a useful upper bound on iterative improvement's worst case time complexity [33]. Nevertheless, iterative improvement has been shown to be effective for solving many different combinatorial optimization problems [11, 16]. In the absence of any meaningful theoretical analysis, we turn our attention to an empirical study.

[2]Note that $C(s_i)$ must be strictly less than $C(s_{i-1})$, equality does not constitute a *down-hill* move.

2.7 An Empirical Evaluation

A suitable problem is required for evaluating optimization procedures. We choose square array placement [29] as a benchmark since it is representative of placement in VLSI physical design.

A single application of an optimization procedure to a problem instance is called a *trial*. We evaluate an optimization procedure by performing multiple trials on a given problem instance.

Each trial starts with a different initial configuration. Let us say that we perform 100 trials. This would result in 100 potentially different terminal configurations.

We record terminal-configuration cost statistics to measure a procedure's ability to produce, **on average**, quality terminal configurations for a given problem instance. We also record the average number of configurations examined over all trials for a given problem instance. This statistic is a measure of how long a procedure takes, **on average**, to locate a terminal configuration. Knowledge of these statistics is important for the purpose of comparing and contrasting different procedures. For example, consider the situation where we compare two optimization procedures and find that the average terminal configuration cost for procedure 1 is less that the average terminal configuration cost for procedure 2. We use this as an indication that procedure 1 is more likely to produce a terminal configuration with overall least cost. This is not to say that for a given number of trials, procedure 1 will *always* produce a terminal configuration with overall least cost.

Square array placement (SAP) is an idealized form of floorplanning. SAP is a geometric problem concerned with placing unit-square objects in the plane. These unit-square objects are called *cells*.

The placement plane is partitioned into an m×m array of cell locations. Each cell

location can accommodate a single cell. The total number of cells to be placed in the array is m^2. Cells are labeled 1 to m^2.

Let M be the set of cell labels for the SAP problem. A *two-terminal net-list* is a subset W of the Cartesian product of M with itself. An element of W is called a *net*. A net represents a connection between two cells. In terms of VLSI physical design, a net is realized by a wire in an integrated circuit.

A SAP problem *configuration* is an assignment of cells to cell locations. The configuration s_y is a *neighbour* of s_x ($s_x, s_y \in S$) if s_x can be transformed to s_y by swapping the positions of two cells in s_x. Consequently, each configuration has $\binom{m^2}{2} = N$ neighbours and $|S| = (m^2)!$.

For a given configuration, each cell has a location which we describe by the array coordinates (x, y) $(1 \leq x, y \leq m)$. Each net is given a cost which corresponds to the sum of the horizontal distance and vertical distance between the net's cells. For example, if a net connects cell 1 located at (3,5) to cell 2 located at (6,1) then the vertical distance between cells 1 and 2 is 3 and the horizontal distance between cells 1 and 2 is 4. The sum of the horizontal distance and the vertical distance is 7 and is called the Manhattan distance.

The Manhattan distance for each net is calculated and summed to give the total Manhattan distance. The total cost of a configuration is defined as five times the total Manhattan distance [29]. The factor five is of Siarry's invention. It has been carried over into our work simply because we want to use the same cost function as Siarry used in his trials.

The objective of the SAP problem is to determine an assignment of cells to cell locations such that the total Manhattan distance between connected cells is minimized.

Two different net-lists are considered. The first net-list comes from Siarry [29]

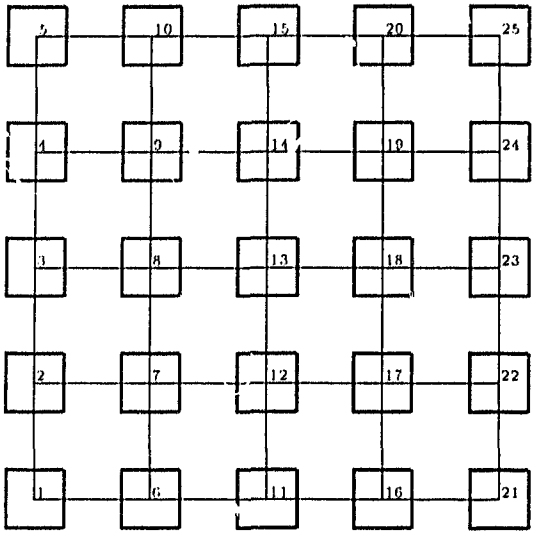


Figure 2.4: 5×5 SAP: Optimal Configuration. (Siarry's net-list).

and is constructed such that in an optimal placement, each net is connected to its nearest horizontal and vertical neighbours only. An optimal solution has cost:

$$5 * 2(m^2 - m)$$

Figure 2.4 shows an optimal 5×5 placement (nearest neighbours connected). For comparison purposes, Figure 2.5 shows a non-optimal placement. Each configuration has 300 neighbours and $|S| = 25! \simeq 10^{25}$. There are 8 unique optimal configurations. Each optimal configuration has a cost of 200.

The second net-list applies only to 5×5 SAP and comes from Sechen [27]. Figure 2.6 shows an optimal placement. There are 16 unique optimal configurations. Each optimal configuration has a cost of 190.

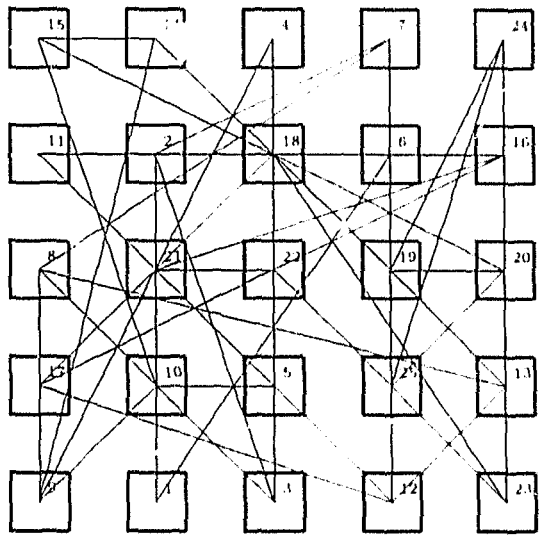


Figure 2.5: 5×5 SAP: Non Optimal Configuration (Siarry's net list).

Neighbour Label	1	2	3	...	m^2-1	m^2	m^2+1	...	N
Cell Pair	1,2	1,3	1,4	...	$1,m^2$	2,3	2,4	...	m^2-1,m^2

Table 2.1: Neighbour Labeling

2.7.1 SAP Neighbour Selection

The essence of any neighbour selection mechanism is the order in which neighbour configurations are examined. For ease of reference, let Table 2.1 define a neighbour labeling for the m×m SAP problem, *e.g.*, the configuration found by swapping the positions of cell 1 and cell 2 is called neighbour 1. Neighbour selection corresponds to selecting a neighbour from the current configuration's neighbourhood. Many *different* neighbour orderings are possible.

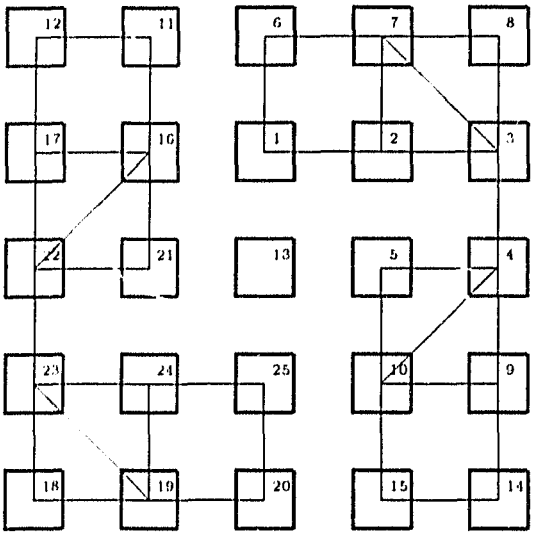


Figure 2.6: 5AP: Optimal Configuration (Sechen's net-list).

We use the following procedure. Selection from the neighbourhood is performed in a pseudo-random order. A pseudo-random number in the range $[1, N]$ is generated. If the number is say 5, then neighbour 5 is selected. Once a neighbour has been selected, it is not considered again for selection until such time as all other neighbours have been selected. Neighbour selection continues by generating additional pseudo-random numbers. Note that for iterative improvement, the order in which neighbours are examined is usually fixed. Iterative improvement is sometimes referred to as *Monte Carlo* iterative improvement when neighbourhood examination is performed in a pseudo random fashion.

We evaluated iterative improvement on the 5×5 SAP problem. The results are shown in Table 2.2. For both net-lists, 100 trials were performed. Each trial had a different starting configuration which was chosen at random. In the cost and config-

Net-list	Cost (Opt.=200)				Configurations		Optimal	Time (s)
	mean	*std*	x_{MIN}	x_{MAX}	*mean*	*std*	%	*mean*
Siarry	343	34	220	426	896	211	0	0.36
Sechen	261	20	215	325	765	172	0	0.32

Table 2.2: 5×5 SAP, Iterative Improvement, (100 Trails.)

urations columns, *mean* and *std* stand for mean and standard deviation respectively. The best and worst terminal configuration costs are given by x_{MIN} and x_{MAX} and, the optimal column contains the percentage of trials which terminated in an optimal configuration.[3]

From Table 2.2, it can be seen that iterative improvement failed to find a single optimal configuration in any trial (Siarry's and Sechen's net lists).

For Siarry's net-list, an average of 896 configurations were examined yielding an average terminal configuration cost of 343 (71.5% off the optimal). For Sechen's net list, an average of 765 configurations were examined yielding an average terminal configuration cost of 261 (37.3% off the optimal).

All of iterative improvement's drawbacks stem from the fact that only down hill moves are allowed. Terminal configurations are always local minima which are rarely optimal. Terminal configurations are also dependent on the starting configuration. In general, there are no guidelines available to select a good starting configuration.

Some of the above problems can be addressed by extending iterative improvement. In the following section, three extensions are considered.

[3]All timings were performed on a SUN SPARC 2

2.8 Iterative Improvement Extensions

A single application of iterative improvement finds a local minimum in the region of the initial configuration. The repeated application of iterative improvement to different initial configurations yields a number of local minima [38]. We call this approach *multiple improvement*. In multiple improvement, the local minimum with the least cost is selected as the terminal configuration. For example, if the 100 trials reported in Table 2.2 are considered a single application of multiple improvement then 89,644 configurations were examined yielding a terminal configuration cost of 220 (Siarry's net-list).

The question arises as to how many initial configurations should be examined during multiple improvement. The probability of finding a global minimum tends to 1 as the number of initial configurations examined in multiple improvement approaches ∞ [8]. Such an observation is of no practical use in deciding how many initial configurations would have to be considered to yield a near optimal terminal configuration. There is no known relationship between the number of initial configurations considered by multiple improvement and the quality of its terminal solution other than the limiting property stated above.

Neighbourhood enlargement increases the number of configurations in a neighbourhood. In the case of the SAP problem, neighbourhood enlargement can be achieved by, for example, swapping the positions of four modules instead of two. For the 5×5 SAP problem, the resulting neighbourhood size would increase from $\begin{pmatrix} 25 \\ 2 \end{pmatrix} = 300$ to $\begin{pmatrix} 25 \\ 4 \end{pmatrix} = 12,650$. We evaluated iterative improvement with an enlarged neighbourhood (12,650 neighbours) on the 5×5 SAP problem (Siarry's and Sechen's net-lists).

Net-list	Cost (Opt.=190)				Configurations		Optimal	Time (s)
	mean	*std*	x_{MIN}	x_{MAX}	*mean*	*std*	%	*mean*
Siarry	295	34	200	350	33,539	10,567	1	15.75
Sechen	233	18	200	285	28,904	6,907	0	13.2

Table 2.3: 5×5 SAP: Iterative Improvement (Enlarged Neighbourhood). (100 Trials.)

The results are shown in Table 2.3.

From Table 2.3, it can be seen that iterative improvement with an enlarged neighbourhood found an optimal configuration once for Siarry's net list and failed to find an optimal configuration for Sechen's net-list. Relative to the results obtained using iterative improvement, the average terminal cost for both net-lists has been reduced by 14% and 11% respectively. This increase in the quality of the average terminal configuration comes at a cost, *i.e.*, a substantial increase in the average number of configurations examined. For Siarry's net-list, an average of 33,539 configurations were examined yielding an average terminal configuration cost of 295 (17.5% off the optimal). For Sechen's net-list, an average of 28,904 configurations were examined yielding an average terminal configuration cost of 233 (22.6% off the optimal).

The detrimental effect of iterative improvement's strictly down hill policy is lessened by neighbourhood enlargement. This is the case since increasing the size of a neighbourhood increases the chance of finding a down-hill move. However, neighbourhood enlargement also results in an increase in the number of configurations that must be examined. The question arises as to how large the neighbourhood should be so that terminal configuration costs are near optimal and the number of configurations that must be examined remains small (relative to the size of the configuration space). In general, there are no guidelines available to determine an ideal neighbourhood size.

2.9 Summary

In this chapter, a framework for discussing optimization has been introduced. Within this framework, iterative improvement has been examined.

Iterative improvement has one main weaknesses *i.e.*, its inability to free itself from local minima and continue towards a global minimum. This is the case since only down-hill moves are allowed. To overcome this problem, some form of controlled *up-hill* movement (*i.e.*, the acceptance of cost increasing moves) must be incorporated into iterative improvement. Multiple improvement is one approach. Another approach is termed *hill climbing* and was first introduced by Kirkpatrick [12] in the form of Simulated Annealing. Simulated Annealing is the subject of Chapter 3.

Chapter 3

Simulated Annealing

In Chapter 2, we saw how iterative improvement is usually unable to free itself from local minima and continue to a global minimum. One way to address this problem is to allow controlled up-hill movement. Simulated annealing is a Monte Carlo combinatorial optimization technique that allows controlled up hill movement. We begin this chapter with some comments on the origins of simulated annealing. This is followed by the presentation of an abstract model for simulated annealing with the intent of giving the reader an appreciation for simulated annealing and how it works. The latter sections of this chapter discuss simulated annealing implementation and performance issues.

3.1 Background

Annealing is a physical process used to improve the properties of a material by heating and then cooling. Physical annealing is used in the production of crystalline solids [27]. The substance to be annealed is initially raised to a high temperature (above

melting point). Subsequently, the temperature is lowered slowly until a crystal is formed.

In 1953, Metropolis *et al.* [18], developed a procedure which simulates physical annealing. In 1983, Kirkpatrick *et al.* [12] noticed that an adaptation of Metropolis' procedure could be used to solve combinatorial optimization problems. They called their procedure *Simulated Annealing* (SA).

Much of the terminology used in SA has been carried over from physical annealing. While noting their origins, all carried over terms are defined only in the context of combinatorial optimization. The latter sections of this chapter discuss SA implementation and performance issues.

3.2 Simulated Annealing Model

In the realm of combinatorial optimization, SA is Monte Carlo iterative improvement with the ability to conditionally accept up-hill moves. The conditional acceptance of up-hill moves allows SA to escape from some local minima which in turn, increases the chance of finding a global minimum.

The probability of accepting an up-hill move during SA is a function of the magnitude of the cost increase, a control parameter T and a random number. T is referred to as *temperature* which is a term carried over from physical annealing. Small up-hill moves have a higher probability of acceptance than larger ones and the probability of accepting any up-hill move diminishes as T decreases. As in the case of iterative improvement, SA accepts all down-hill moves. Unlike iterative improvement, SA also accepts all moves where the change in cost is zero *i.e.*, $\Delta C = 0$.[1]

[1] With reference to SA, we consider all such moves to be down-hill moves.

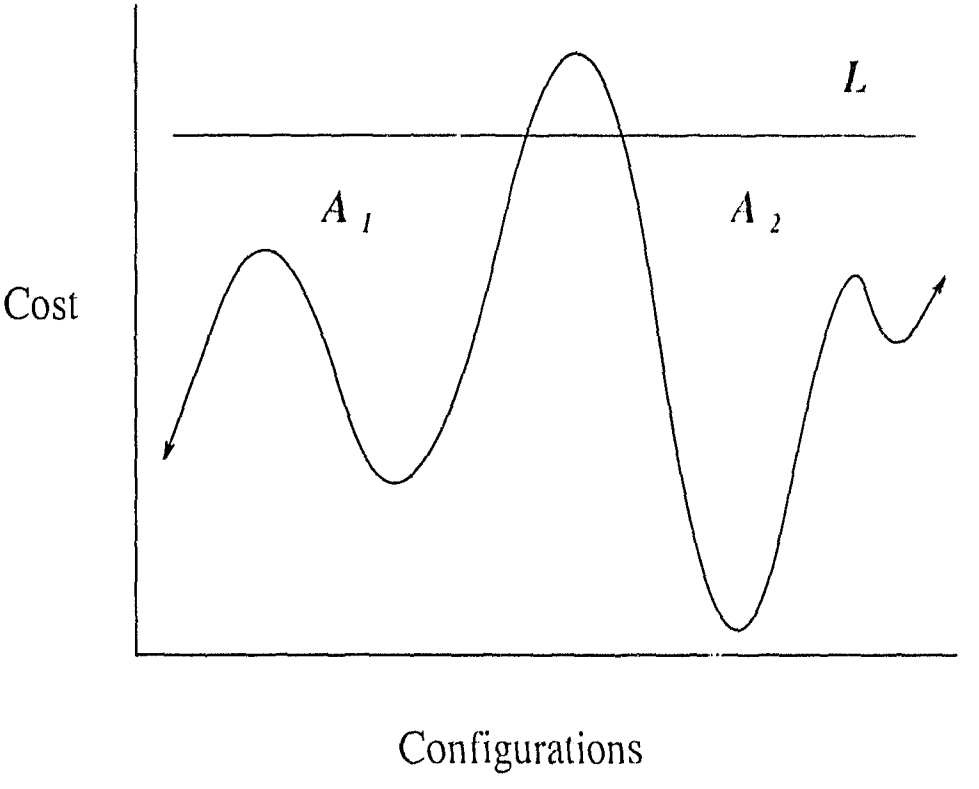


Figure 3.1: Cost *vs.* Configuration Curve.

The SA procedure can be divided into three stages:

(1) Initially, T is large enough so that large up-hill moves have a probability of acceptance close to one. During this stage, SA behaves as a random walk through the configuration space. No useful work is being performed during stage 1. As we will explain later, a sufficiently high initial value for T is critical to the performance of SA. An initial value of T which is neither too high nor too low is difficult to determine. Consequently, a 'better safe than sorry' approach is taken. T is given an initial value which is higher than the lowest value required to accept large up hill moves with a probability close to one. This results in the random walk behaviour mentioned above.

(2) As T decreases, the configuration space is partitioned. Within the analogy of hills and valleys, the value of T defines the position of a horizontal line L (Fig. 3.1). Recall that a *region* is informally described as a valley in our model cost versus configuration curve (see formal definition on page 10). We define an *area* within this curve as one or more contiguous regions. As T decreases, L is lowered and two areas A_1 and A_2 are formed. Areas A_1 and A_2 are formed when the acceptance of a sequence of configurations starting in A_1 and finishing in A_2 becomes unlikely due to the value of T.

(3) As T decreases further, the probability of accepting any up-hill move approaches zero. At the beginning of this stage, small up-hill moves may be accepted. Towards the end of this stage, SA behaves as iterative improvement, *i.e.*, only down hill moves are accepted.

The essence of SA can now be stated as follows: Stage 1 allows for the examination of the complete configuration space. Stage 2 divides the configuration space into areas.

An area is then selected at random. As T is decreased further, the selected area is in turn divided into areas. This process is repeated into stage 3. No further area division is performed when the last selected area contains a single region. Stage 3 ends when the last area selected is searched for its local minimum.

The control of T is critical to the performance of SA both in terms of the quality of the terminal configuration and the time taken to find it. If T is not large enough during stage 1, then areas of the configuration space which could contain optimal or near-optimal configurations become inaccessible. On the other hand, if T is too large, then time is wasted performing a fruitless random walk through the configuration space.

During stage 2, T must decrease slowly to increase the chance that the last area selected in stage 3 contains an optimal or near optimal configuration. The term *rapid quenching* is a carryover from physical annealing and is used to describe the situation where T decreases too quickly. Rapid quenching during SA usually results in terminal configurations which are neither optimal nor near optimal. If T is not small enough during stage 3, then it is possible to accept cost increasing moves which result in an escape from an area which contains an optimal or near optimal configuration.

In the following sections, a more detailed description of SA is provided and the procedure is evaluated on the SAP problem as described in Chapter 2.

3.3 Simulated Annealing Procedure

SA is a general purpose combinatorial optimization technique which has been applied to many problems in VLSI design [38, 35]. Starting from an initial configuration $s_0 \in S$, SA constructs a sequence of configurations $s_0, s_1, .., s_i, .., s_r$ where $s_i \in G(s_{i-1})$ (1 <

$i \leq r$). The acceptance of a configuration s_i within the sequence $s_0, ..., s_i, ..., s_r$ is determined by the function:

$$\mathcal{F}(s_{i-1}, s_i, T_j) = \begin{cases} \text{accept, if } C(s_i) \leq C(s_{i-1}) & 1 \leq i \leq r \\ f(s_{i-1}, s_i, T_j), \text{ otherwise} & 0 \leq j \leq q \end{cases}$$

$$f(s_{i-1}, s_i, T_j) = \begin{cases} \text{accept, if } u \leq e^{-\frac{C(s_i)-C(s_{i-1})}{T_j}} \\ \text{reject, otherwise} \end{cases}$$

where u is a pseudo-randomly generated number in the interval [0,1). The acceptance function is an adaptation of the Metropolis Criteria [18] used in the simulation of physical annealing. T_j is referred to as the j^{th} *temperature value* where T_0 and T_q denote the initial and final temperatures, respectively, and j is called the *temperature index*.

The value of temperature is decreased once certain criteria are satisfied. Once there criteria have been satisfied, *equilibrium* is said to have been detected. As in the case of temperature, the term *equilibrium* is carried over from physical annealing.

A function $\mathcal{G}$ is employed to determine temperature decrements. $\mathcal{G}$ is a monotonically decreasing function of T so $T_j = \mathcal{G}(T_{j-1})$ and $T_j < T_{j-1}$ $(1 \leq j \leq q)$. The overall objective of SA is to *converge* to an optimal or near optimal configuration. A pseudocode definition of SA is shown in Figure 3.2

3.4 General Cooling Schedules

The notion of a *cooling schedule* is common to all SA implementations. The value of T_0, the function $\mathcal{G}$, the ability to detect equilibrium and criteria for convergence together make up a cooling schedule. An annealing schedule is said to be general if equilibrium, convergence, T_0 and $\mathcal{G}$ are defined in a problem independent manner.

```
construct initial configuration

initialize temperature

repeat
        repeat
                select a neighbour configuration

                accept or reject new configuration

        until equilibrium

        decrease temperature

until converged
```

Figure 3.2: Simulating Annealing Pseudocode.

Most general cooling schedules are constructed such that (a) T_j $(1 \leq j \leq q)$ is found by multiplying T_{j-1} by a constant and (b) equilibrium exists after the examination of a fixed number of configurations. The schedules proposed by Wong [37], Siarry [29] and Sechen [27] are typical of this approach. In the following paragraphs, we describe a general cooling schedule that is representative of the above style of cooling schedule. It is a hybrid of Wong's, Siarry's and Sechen's schedules. For ease of reference, we call this representative schedule a *classical* schedule.

3.4.1 The Classical General Cooling Schedule

An initial sampling of the problem's configuration space is performed in order to estimate $\widehat{\Delta C}$, where $\widehat{\Delta C}$ is the average $|\Delta C| = |C(s_i) - C(s_{i-1})|, \quad s_i \in G(s_{i-1})$. We examine a sequence of 2,000 configurations. $\widehat{\Delta C}$ is found by calculating the average absolute cost difference between successive pairs of configurations within the sequence. In theory, a more accurate estimate of $\widehat{\Delta C}$ could be found by examining a larger number of configurations. In our experience, a more accurate estimate of $\widehat{\Delta C}$ is not necessary since its effect on the classical schedule is not significant.

In the initial stages of SA, ΔC can be approximated by $\widehat{\Delta C}$ and the probability P of accepting an up-hill move should be close to one. Consequently, an expression for T_0 can be derived as follows:

$$P = e^{-\frac{\widehat{\Delta C}}{T_0}} \qquad (\widehat{\Delta C} > 0\ ,\ T_0 > 0)$$
$$\Rightarrow \quad T_0 = -\frac{\widehat{\Delta C}}{\ln(P)}$$

The expression for T_0 used in the classical schedule comes from Wong [37] and is a weighted product of the above expression:

$$T_0 = -\frac{\widehat{\Delta C}\lambda_T}{\ln(P)}$$

Wong indicates that the λ_T factor in the above expression is used to reduce the probability of accepting configurations which tend to maximize the given cost function. Wong does not give values for λ_T and P other than to say that $\lambda_T \sim 1$ and $P \sim 1$. We will return to the issue of assigning values to λ_T and P once the remainder of the schedule has been described. In the following paragraph, two further parameters r and w are introduced. Like λ_T and P, the assignment of values to r and w is not considered until the classical schedule has been completely described.

Temperature reduction in the classical schedule is determined by:

$$T_j = \mathcal{G}(T_{j-1}) = rT_{j-1}$$

for some constant r.

Equilibrium is said to exist at temperature value T_j if the total number of configurations examined at this temperature value exceeds wN. Convergence occurs when no up-hill moves are accepted at a particular temperature value.

The assignment of values to λ_T, P, r and w is called *schedule parameter assignment*. The first stage of schedule parameter assignment is to assign values to λ_T and P in the expression for T_0. Figure 3.3 shows the relationship between λ_T, P and $-\lambda_T/ln(P)$. For practical reasons, we limit the range of P to $0 < P < 0.99$.

Typically, r is assigned a value in the range 0.85 to 0.99 and w is assigned a discrete value in the range 2 to 5.

We evaluated the classical schedule on the 5×5 SAP problem using a range of values for $-\lambda_T/ln(P)$. We completed parameter assignment by using Wong's values for r and w, *viz.*, 0.85 and 2 respectively. The results are shown in Table 3.1.

For each value of $-\lambda_T/ln(P)$ shown in Table 3.1, 100 trials were performed. Each trial had a different starting configuration which was chosen at random.

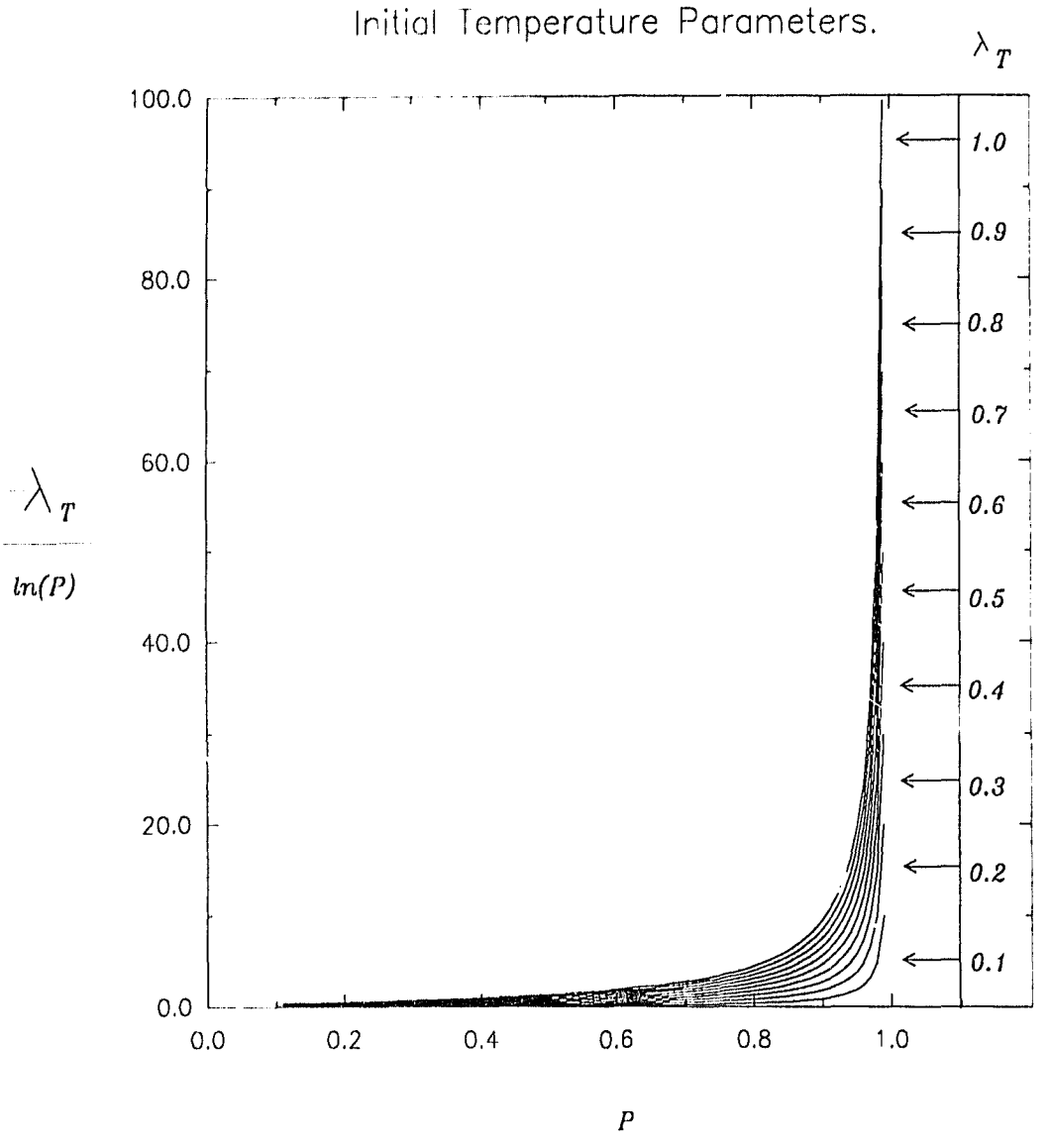


Figure 3.3: Relationship between P and λ_T $(0 < P \leq 0.99)$.

$-\lambda_T/ln(P)$	λ_T	P	Cost (Opt. = 200)			Configurations		Optimal	Time
			mean	*std*	x_{MAX}	*mean*	*std*	%	(s)
99.49	1.00	0.99	243	38	320	25,230	1,049	39	11.0
59.69	0.6	0.99	251	33	310	23,610	1,129	21	10.6
19.89	0.2	0.99	245	37	320	19,325	1,200	33	9.3
0.56	0.7	0.29	240	36	320	6,066	1,085	12	1.9
0.48	0.6	0.29	246	36	320	5,562	1,220	31	1.7
0.40	0.5	0.29	245	36	310	4,926	1,083	33	1.5
0.32	0.4	0.29	246	36	330	4,104	1,180	33	1.2
0.24	0.3	0.29	257	36	320	3,360	918	21	3.9
0.16	0.2	0.29	278	36	355	2,385	831	8	3.6
0.08	0.1	0.29	312	33	365	960	390	3	3.0
$r = 0.85$						w =			

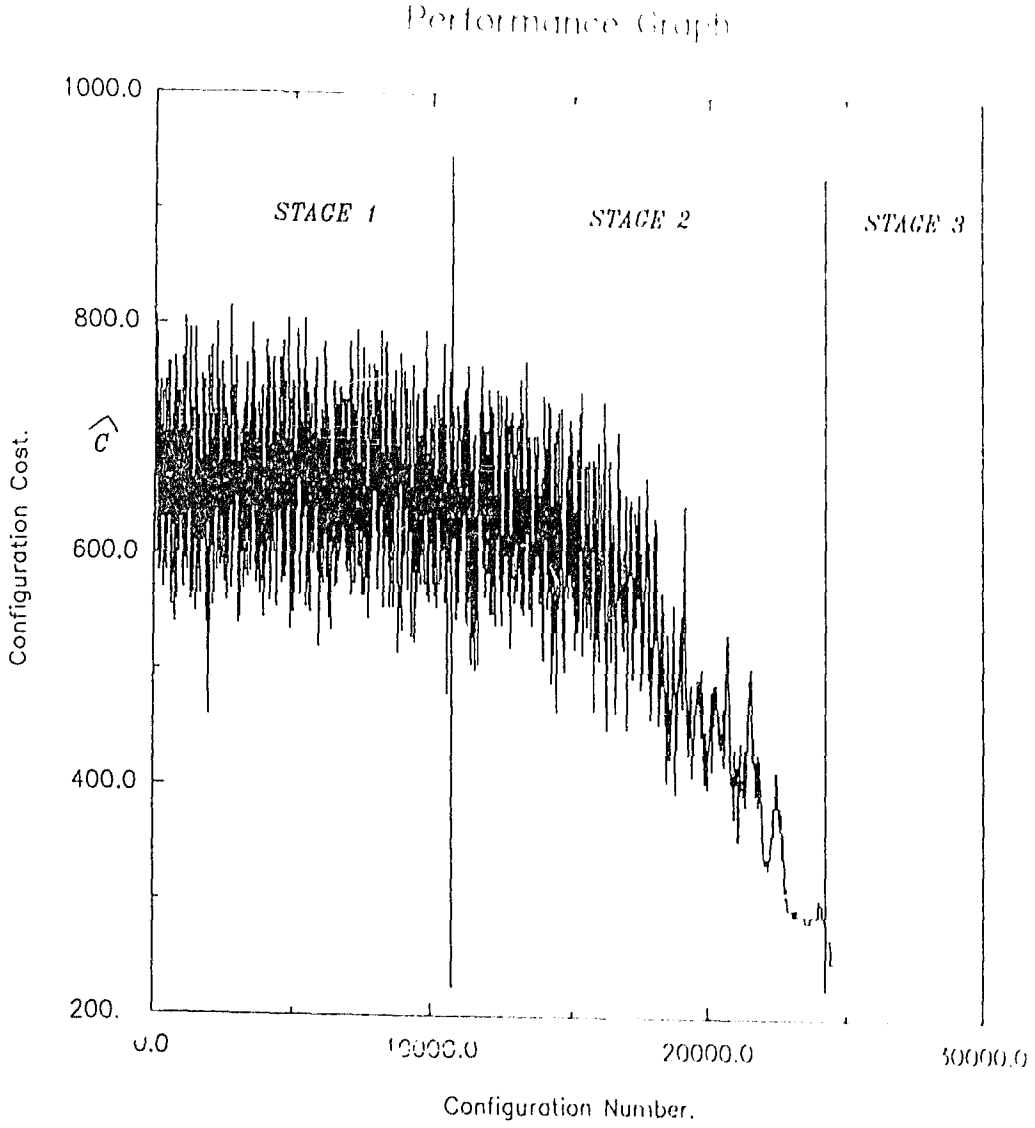
Table 3.1: 5×5 SAP: Siarry's Net-list – Classical Schedule

From Table 3.1, it can be seen that the average terminal configuration cost is approximately constant for values of $-\lambda_T/ln(P)$ greater than or equal to 0.32. For values less than or equal to 0.24, the average terminal configuration cost increases as the value of $-\lambda_T/ln(P)$ decreases. This is consistent with the properties of the SA model presented earlier in that a low initial temperature will have a detrimental effect on the quality of the terminal configuration and a high initial temperature will result in an increase in the number of configurations examined without an increase in the quality of the terminal configuration.

Consider a graph of accepted configuration cost versus the number of configurations examined by the SA procedure. We call such a graph a *performance graph*. Figure 3.4 shows a performance graph based on an instance of the 5×5 SAP problem (Siarry's net list) using the classical schedule with $-\lambda_T/ln(P) = 99.49$. A similar graph with $-\lambda_T/ln(P) = 0.16$ is shown in Figure 3.5. In both graphs, the stages of the annealing procedure are shown. It should be noted that the boundary between stages is fuzzy and not well defined as depicted in Figures 3.4 and 3.5.

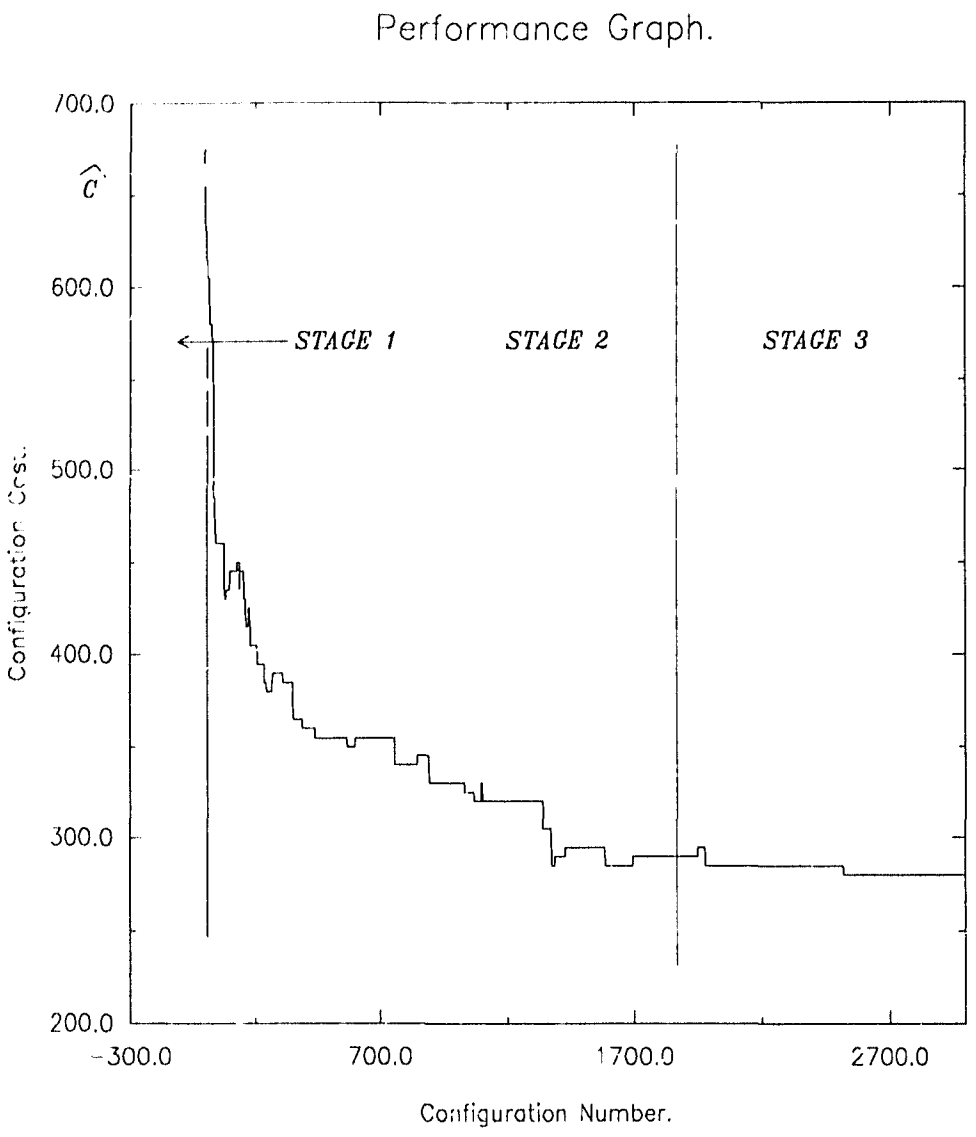
As already indicated, the excessively high initial temperature results in an long stage 1 (Fig. 3.4). By contrast, an insufficiently high initial temperature causes the procedure to resemble iterative improvement (Fig. 3.5).

Returning to Table 3.1, we can see that terminal configuration costs were optimal 39% of the time on average ($-\lambda_T/ln(P) = 99.49$). Perhaps a better rate can be achieved by increasing r or by increasing w? To this end, we again evaluated the classical schedule on the 5×5 SAP problem using a range of values for $-\lambda_T/ln(P)$. On this occasion, we assigned r and w the values 0.99 and 5 respectively. The results are shown in Table 3.2. It can be seen that the average terminal configuration cost is approximately constant for values of $-\lambda_T/ln(P)$ greater than or equal to 0.40. For values less than or equal to 0.32, the average terminal configuration cost increases as



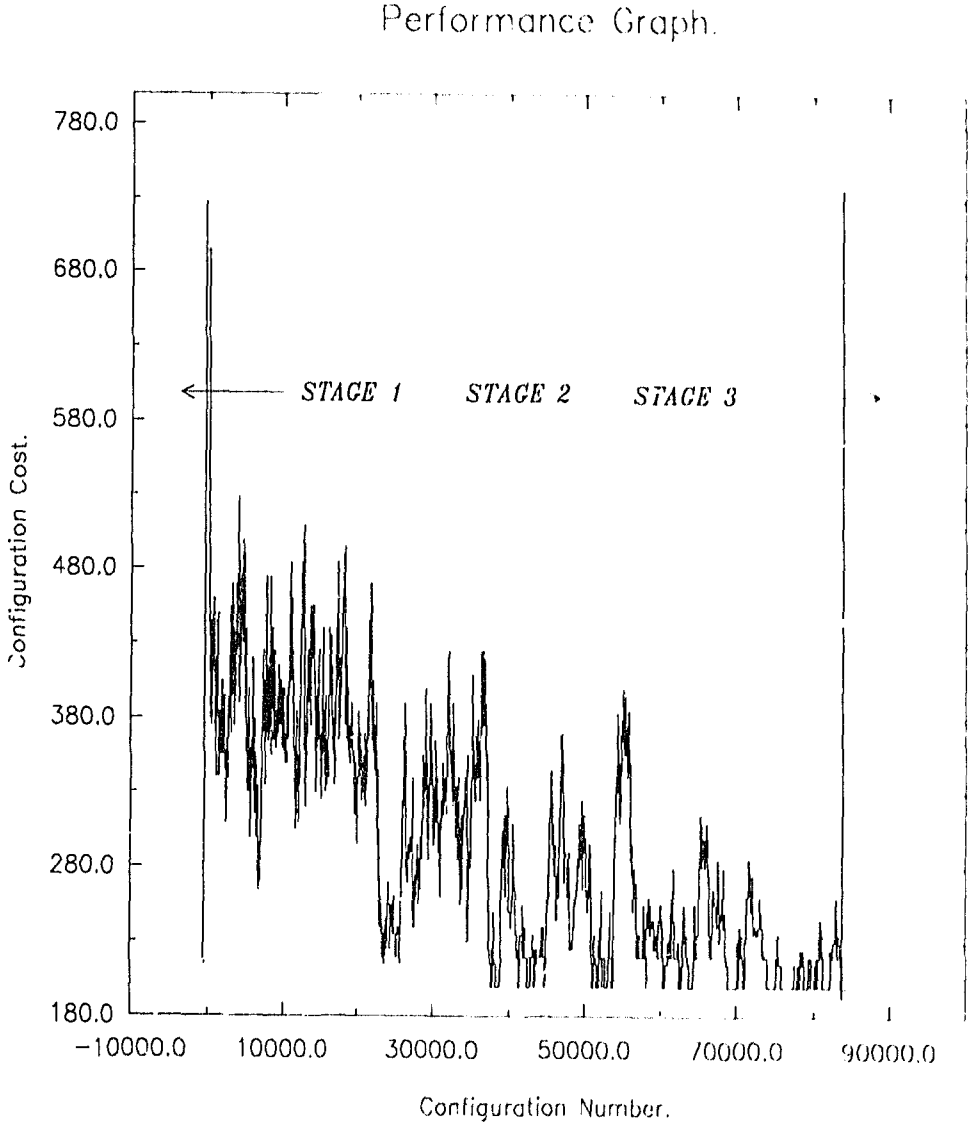
$$-\lambda_T / ln(P) = 99.49 \quad r = 0.85 \quad w = 2$$

Figure 3.4: Performance Graph (1).



$-\lambda_T/ln(P) = 0.16 \quad r = 0.85 \quad w = 2$

Figure 3.5: Performance Graph (2).



$$-\lambda_T/ln(P) = 0.40 \quad r = 0.99 \quad w = 5$$

Figure 3.6: Performance Graph (3).

$-\lambda_T/ln(P)$	λ_T	P	Cost (Opt.=190)			Configurations		Optimal	Time
			mean	*std*	x_{MAX}	*mean*	*std*	%	(s)
0.99	1.00	0.99	20.	4.5	245	221,145	23,020	99	81.4
0.48	0.6	0.29	200	0	200	111,705	11,808	100	44.0
0.40	0.5	0.29	200	0	200	84,825	12,977	100	34.6
0.32	0.4	0.29	202	11	260	56,250	25,833	96	23.8
0.24	0.3	0.29	211	23	265	36,255	36,354	82	16.3
0.16	0.2	0.29	245	31	280	33,840	25,643	31	15.0
0.08	0.1	0.29	275	36	340	3,780	2,359	12	4.0
$r = 0.99$						$w = 5$			

Table 3.2: 5×5 SAP: Siarry's Net-list - Classical Schedule.

the value of $-\lambda_T/ln(P)$ decreases.

From Table 3.2, it can be seen that the procedure almost always converged in an optimal configuration for values of $-\lambda_T/ln(P)$ greater than or equal to 0.4. Recall that iterative improvement never terminated in an optimal configuration when it was evaluated using the 5×5 SAP problem (Siarry's net-list).

Figure 3.6 shows a performance graph based on an instance of the 5×5 SAP problem (Siarry's net-list) using the classical schedule with $-\lambda_T/ln(P) = 0.40$, $r = 0.99$ and $w = 5$. It can be seen from Figure 3.6 that the procedure found an optimal configuration well before convergence. In other words, assigning r and w the values 0.99 and 5 respectively, was an over kill resulting in the examination of an excessive number of configurations. The corresponding increase in the execution time was not problematic for the 5×5 SAP problem. However, for large values on N and a high initial temperature, the increase in the number of configurations examined could

result in prohibitively large execution times.

In conclusion, the effectiveness of SA using the classical schedule is limited by the need to perform schedule parameter assignment. Tables 3.1 and 3.2 show that for the 5×5 SAP problem (Siarry's and Sechen's net lists), a poor schedule parameter assignment can cause the performance of the procedure to fluctuate wildly both in terms of the quality of the terminal solution and i.. the time taken to find it. For a given problem, a good schedule parameter assignment can only be found experimentally. This in itself is not easy. One need look no further than Tables 3.1 and 3.2. After 17,000 trials (500,000 configurations) we are still unable to say what values or $-\lambda_T/ln(P)$, r and w best suit the 5×5 SAP problem (Siarry's net list).

3.4.2 A Statistical General Cooling Schedule

The size of each temperature decrement and the number of configurations examined at each temperature value are predetermined in the classical schedule. In contrast to schedules like the classical schedule, Huang's schedule [10] exerts *dynamic control over* the size of each temperature decrement and the number of configurations examined at a particular temperature value. In the following section, Huang's general cooling schedule is described and evaluated on the 5×5 SAP problem.

The initial value for T is given as $T_0 = k\sigma$ where σ is the standard deviation of the cost distribution and k is a constant. An initial sampling of the problem's configuration space is performed. The costs associated with all configurations encountered during the traversal are used to estimate σ. Huang does not specify how many configurations should be examined. In our implementation of Huang's schedule, we examine 2,000 configurations, the same number as we examined in the classical schedule. The value of k is computed such that a configuration s_i is accepted with

probability P when $\Delta C = C(s_i) - C(s_{i-1}) \leq 3\sigma$ and $T = T_0$. An expression for k can be derived as follows:

$$P = e^{-\frac{\Delta C}{T}} \Rightarrow P = e^{-\frac{3\sigma}{k\sigma}}$$
$$\Rightarrow k = -\frac{3}{\ln(P)}$$

During the initial stages of SA ($T = T_0$), a reasonable acceptance rate for cost increasing configurations is 80%. Therefore, a typical value for k is 20.

Huang defines the curve of average cost $< C >$ versus the logarithm of T as the *annealing curve*.[2] The slope of the annealing curve is:

$$\frac{d(\ln(T))}{d<C>} = \frac{d(\ln(T))}{dT}\frac{dT}{d<C>} = \frac{dT}{T\,d<C>}$$
$$\Rightarrow \frac{d<C>}{d(\ln(T))} = \frac{T\,d<C>}{dT}$$

From Reif [24], it is known that:

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

The slope of the annealing curve can be approximated by:

$$\frac{\Delta<C>}{\Delta\ln(T)} \approx \frac{\Delta<C>}{\ln(\mathcal{G}(T)) - \ln(T)} = \frac{\sigma^2}{T}$$

Consequently,

$$e^{\frac{-T\Delta<C>}{\sigma^2}} = \frac{\mathcal{G}(T)}{T}$$
$$\Rightarrow \mathcal{G}(T) = Te^{\frac{-T\Delta<C>}{\sigma^2}}$$

Huang requires that $\Delta < C >$ be less than σ. Consequently, let $\Delta < C >= \lambda\sigma$ where $\lambda \leq 1$ and

$$\mathcal{G}(T) = Te^{\frac{-\lambda T}{\sigma}}$$

[2]Note that $< C >$ is a running average which is updated with the cost of each new configuration selected.

A lower bound of 0.5 is placed on the $\mathcal{G}(T) \mid T$ ratio. Such a bound prevents drastic reductions in the value of the control parameter due to a flat annealing curve when T is approximately equal to T_0. Typically, λ is assigned the value 0.7.

Let S_j be the set of configurations accepted when $T = T_j$. Let S_{ζ_j} be comprised of elements of S_j whose costs are in the range $(< C > -\zeta, < C > +\zeta)$. Equilibrium is established when the ratio of S_{ζ_j} to S_j reaches a stable value γ. Such a ratio depends on the nature of the cost distribution and the sampling range ζ. At high temperatures, the cost distribution is assumed to be a normal distribution. Consequently, [10]

$$\gamma = \mathrm{erf}(\frac{\zeta}{\sigma})$$

where erf() is the error function. Huang assigns ζ the value $\frac{\sigma}{2}$. Therefore, γ = erf(0.5) = 0.38.

Equilibrium is detected by a counting process. Two counters A_1 and A_2 are maintained. Counters A_1 and A_2 are initially set to zero. A_2 is incremented whenever a configuration is accepted. A_1 is incremented whenever a configuration is accepted whose cost is within the range $(< C > -\zeta, < C > +\zeta)$. A_1 and A_2 are assigned target counts. Equilibrium is said to be established if A_1 reaches its target count before A_2 reaches its target count. In the event that A_2 reaches its target count before A_1 then both counters are reset to zero and the counting is continued. Huang sets the target count for A_1 and A_2 to $\gamma(3Q)$ and $\mid 1 - \gamma \mid (3Q)$ respectively where Q is the size of the problem instance.

The above equilibrium detection mechanism is used after Q moves have been accepted. By default, equilibrium is said to exist after $4*N$ configurations have been examined at a particular temperature value if the above equilibrium conditions have not already been satisfied.

The ability to accept up-hill moves is no longer required when all configuration

Net-list	Cost (Opt.=200)			Configurations		Optimal	Time
	mean	*std*	x_{MAX}	*mean*	*std*	%	(s)
Siarry	223.25	32.7	280	24,864	6,155	66	7.8

Table 3.3: 5×5 SAP: Siarry's Net-list - Huang's Schedule. (100 Trials)

accepted at a given temperature value have similar cost. Consequently, SA can terminate or converge whenever the similar-cost condition is detected. To this end, consider the situation where equilibrium has been established at temperature $T = T_j$. Let the sequence of configurations accepted when $T = T_j$ be $s_a, s_{a+1}, ..., s_b, ..., s_c$ $(a \leq b \leq c)$ and let $S_j = \{s_b : a \leq b \leq c\}$. For all configurations in S_j, let $s_{max} \in S_j$ maximize the objective function. For all configurations in S_j, let $s_{min} \in S_j$ minimize the objective function. Finally, let $\Delta C'_{max}$ be the absolute maximum change in cost incurred during the construction of the sequence $s_a, ..., s_b, ...s_c$. The similar-cost condition is considered satisfied when $C'(s_{max}) - C'(s_{min}) = \Delta C'_{max}$.

We evaluated Huang's schedule on the 5×5 SAP problem (Siarry's net-list). The results are shown in Table 3.3. The procedure found an optimal configuration 66% of the time with an average terminal configuration cost of 219 (11.5% off the optimal). Its performance corresponds roughly to the performance of the classical schedule with $\lambda_I/ln(P) = 0.24$, r=0.99 and $w = 2$.

The calculation of initial temperature in Huang's schedule corresponds to a $\lambda_I/ln(P)$ value of 1.2. Based on the experience gained evaluating the classical schedule, it would appear that Huang's initial temperature calculation is high. However, due to the dynamic control exerted over temperature and equilibrium detection, the effect (in terms of the excess number of configurations examined) of a high initial temperature is reduced.

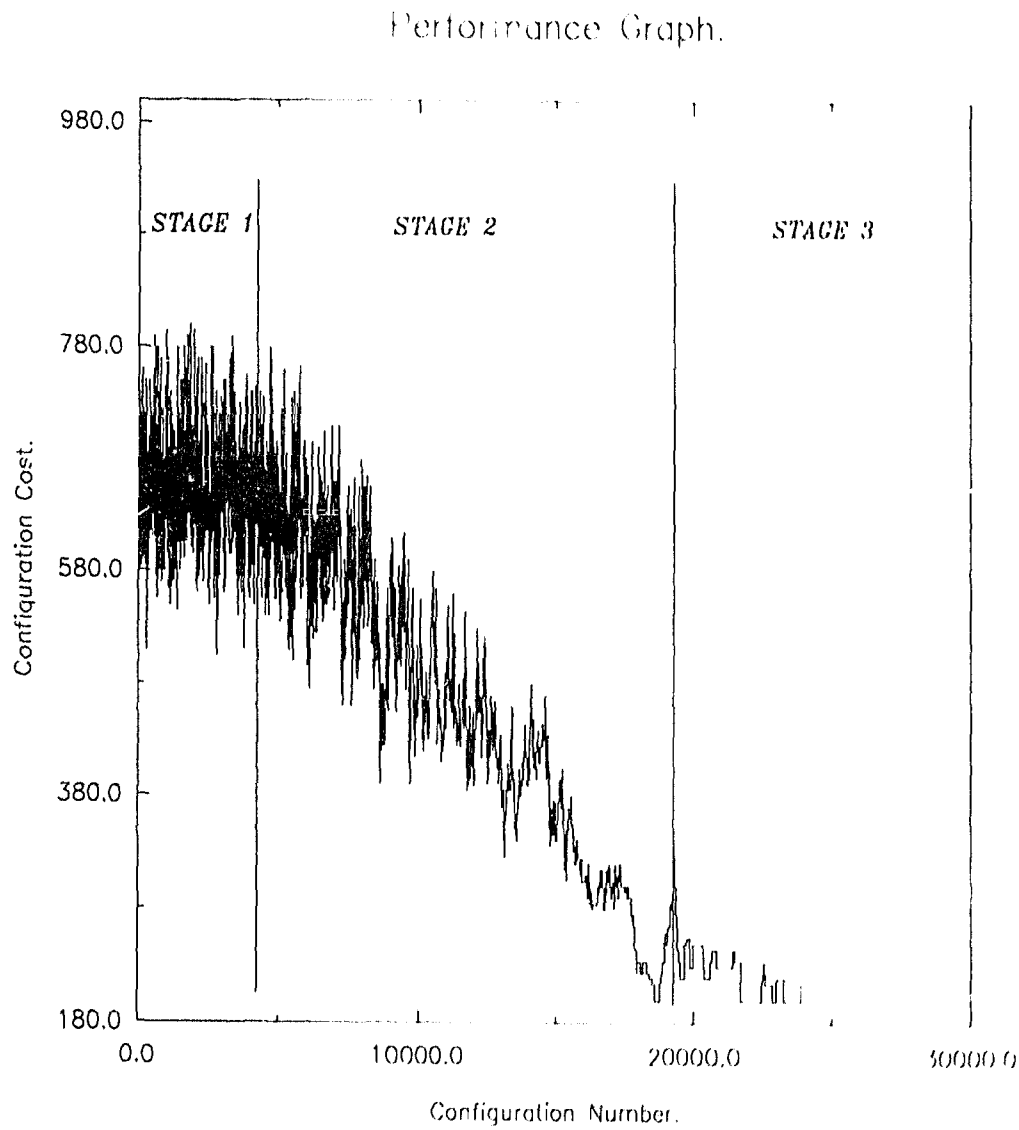


Figure 3.7: 5×5 SAP: Siarry's Net list — Huang's Schedule.

Net-list	Cost (Opt.=190)			Configurations		Optimal	Time
	mean	*std*	x_{MAX}	*mean*	*std*	%	(s)
Sechen	200.3	6.5	225	31,008	4,884	38	8.2

Table 3.4: 5×5 SAP: Sechen's Net-list - Huang's Schedule. (100 Trials.)

Figure 3.7 shows a performance graph based on an instance of the 5×5 SAP problem (Siarry's net-list) using Huang's schedule. It can be seen that stage 1 is longer than necessary and the procedure again reached an optimal configuration long before convergence. The procedure converged prematurely in 34% of the trials yielding non optimal terminal configurations.

We evaluated Huang's schedule on the 5×5 SAP problem (Sechen's net-list). The results are shown in Table 3.4. Huang's schedule found an optimal configuration 38% of the time with an average terminal configuration cost of 200.3 (5% off the optimal).

3.5 Summary

In the past, the salient features of SA have been described relative to their counterparts in physical annealing. We took a different approach and introduced an abstract model for SA. This model is an extension of the model proposed by Grove [9]. Within the context of this model, the reader was given an appreciation for SA and how it works.

The notion of a cooling schedule is common to all SA implementations. In this chapter, we described a general cooling schedules called the classical schedule. One of the main drawbacks with this schedule is the need to perform parameter assignment.

We have demonstrated that for the 5×5 SAP problem using Siarry's and Sechen's net-lists, a good parameter assignment is difficult to achieve.

Huang's general cooling schedule was also described in this chapter. The problem of parameter assignment does not occur in Huang's schedule. With regard to average terminal configuration quality, Huang's schedule performed better than the classical schedule on the 5×5 SAP problem (Siarry's and Sechen's net lists). However, on several occasions, SA using Huang's schedule terminated in a configuration which was neither optimal nor near-optimal (Tables 3.3 and 3.4). This problem could be caused by inappropriate decreases in temperature or by poor equilibrium or convergence criteria. Most likely, it is a combination of all three. All we can say for sure is, on the occasions that the procedure terminated in a non-optimal configuration, the value of temperature was not high enough to enable the SA procedure to continue to an optimal configuration.

All existing general cooling schedules for SA are similar in the sense that they only allow temperature decreases. To our knowledge, none of the existing schedules provide a mechanism for increasing temperature should the SA procedure become trapped at a configuration that is neither optimal nor near-optimal. In the following chapter, we propose a general cooling schedule which is based on the cost (exact, estimated or lower-bound) of an optimal solution to a problem instance. We exert dynamic control over temperature as does Huang's schedule. The key difference between the two approaches is that we allow temperature to increase should the procedure become trapped at a non-optimal configuration. By providing a mechanism for temperature increases, we increase the chance that the SA procedure can proceed to an optimal configuration.

Chapter 4

Goal-Directed Annealing

Recently, Sastry *et al.* proposed a general procedure to estimate the cost of an optimal solution to some combinatorial optimization problem instances [26]. Sastry suggested that such an estimate could be used to determine termination criteria for general purpose optimization techniques such as iterative improvement or simulated annealing. We have taken this suggestion further and designed a complete simulated annealing general cooling schedule based on the cost (exact, estimated or lower-bound) of an optimal solution to a problem instance.

In this chapter, we introduce a goal-directed general cooling schedule. The goal-directed cooling schedule uses the cost (exact, estimated or lower-bound) of an optimal solution to a problem instance to determine the probability of accepting an up-hill move [36]. We evaluate the goal-directed schedule on three instances of the SAP problem. This evaluation provides evidence that the goal-directed schedule is an effective SA control mechanism for some problem instances. However, the evaluation also uncovers several problems with the goal-directed approach. We examine these problems and solve them through extensions to the schedule. We call the resulting

schedule an *extended* goal-directed cooling schedule.

The extended goal-directed schedule is a variation on the goal directed theme which permits temperature to increase as well as decrease. In the latter part of this chapter, the extended goal-directed schedule is evaluated on five instances of the SAP problem. We show that for each instance considered, there is an improvement in the quality of the average terminal configuration achieved by the extended goal directed schedule over Huang's schedule and, this improvement in quality is statistically significant.

4.1 A Goal-Directed Approach

Let K_j be the probability of accepting a configuration with a cost increase of ΔC. The relationship between K_j and T_j is

$$K_j = e^{-\frac{\Delta C}{T_j}}$$

An initial random search of the problem's configuration space is performed to estimate $\widehat{\Delta C}$. We examine a sequence of 2,000 configurations and average the absolute value of the cost difference between successive pairs in the sequence. In theory, a more accurate estimate of $\widehat{\Delta C}$ could be found by examining a larger number of configurations. In our experience, a more accurate estimate of $\widehat{\Delta C}$ is not necessary since its effect on the goal-directed schedule is not significant.

Our approach is to let K_j decay geometrically with time. K_j follows the relation

$$K_j = K_{j-1}\, d = K_0\, d^j$$

for some constant d called the K *multiplier* $(1 < j < q)$. An expression for G is derived as follows:

$$K_j = e^{-\frac{\Delta C}{T_j}} = K_0\, d^j = K_{j-1}\, d$$

$$\Rightarrow T_j = \frac{-\widehat{\Delta C}}{\ln(K_{j-1}\, d)}$$

$$\Rightarrow T_j = \frac{-\widehat{\Delta C}}{\ln\left(e^{\frac{-\widehat{\Delta C}}{T_{j-1}}}\, d\right)}$$

$$\Rightarrow T_j = \mathcal{G}(T_{j-1}) = \frac{-\widehat{\Delta C}}{\frac{-\widehat{\Delta C}}{T_{j-1}} + \ln d}$$

Notice that in the goal-directed schedule, T_j's rate of decrease is approximately geometric ($T_j = c_1 T_{j-1} + c_2$, for some constants c_1 and c_2). Several schedules from the literature [27, 29, 37] allow temperature to decrease geometrically. We justify our decision to let K_j decrease geometrically based on the precedent set by existing cooling schedules.

A high acceptance rate for cost-increasing configurations is desirable during the initial stages of SA ($T = T_0$). We set K_0 to 0.99. This yields a temperature value that almost certainly will accept a cost-increasing move of $\widehat{\Delta C}$. In fact, the probability of accepting a cost-increasing move of $\widehat{\Delta C} \times 50$ when K_0=0.99 is 0.60. In our opinion, there is a strong case to suggest that setting K_0 to 0.99 yields a temperature value that results in a random walk through the configuration space.

A low acceptance rate (almost nil) for cost-increasing moves is desirable during the final stages of SA ($T = T_q$). We set K_q to 0.01. This yields a temperature value that almost certainly will not accept a cost increasing move of $\widehat{\Delta C}$. In fact, the probability of accepting a cost-increasing move of $\frac{\widehat{\Delta C}}{10}$ when K_q=0.01 is 0.63. In our opinion, there is a strong case to suggest that setting K_q to 0.01 yields a temperature value that results in the rejection of most cost-increasing moves.

The K multiplier, T_0 and T_q are found as follows: (given values for K_0 and K_q)

$$d = e^{\frac{\ln\left(\frac{K_q}{K_0}\right)}{q}}$$

$$T_0 = \frac{-\Delta\bar{C}}{\ln K_0}$$

$$T_q = \frac{-\Delta\bar{C}}{\ln K_0 \, d^q}$$

For each temperature value T_j, a *target cost* E_j is defined. From all the configurations accepted so far, let B be the configuration with least cost. Equilibrium is established at temperature value T_j if $C(B)$ is less then or equal to E_j. For notational convenience, we call a configuration with cost less than or equal to E_j a j equilibrium configuration.

The value of E_j $(1 \leq j \leq q)$ is determined such that the temperature value T_j is capable of locating a j-equilibrium configuration. By experimentation, we found the best E_j to T_j match is achieved by letting E_j decay in a geometric series. Even so, there is no guarantee that an equilibrium configuration will be encountered at a particular temperature value. We address this problem later in this chapter. For the moment, let convergence occur when equilibrium is reached at temperature value T_q.

An E multiplier (denoted g) is defined in an analogous manner to the K multiplier above. In particular,

$$E_j = E_{j-1} \, g = E_0 \, g^j$$

and

$$g = e^{\frac{\ln\left(\frac{E_q}{E_0}\right)}{q}}$$

E_0 and E_q correspond to the costs of an easily accessible configuration and an optimal configuration respectively. We set E_0 to $\bar{C}$ which is the average of the costs of the configurations encountered in the initial random search of the configuration space. In theory, a more accurate estimate of $\bar{C}$ could be found by examining a larger number

of configurations. In our experience, a more accurate estimate of $\hat{C}'$ is not necessary since its effect on the goal-directed schedule is not significant. E_q may be derived from knowledge of the problem or estimated. The impact a poor estimate has on goal-directed annealing is discussed in Chapter 7.

We determine q, the number of temperature indices, such that the last decrease in target cost equals $\widehat{\Delta C}'$ *i.e.*, $E_q + \widehat{\Delta C}' = E_{q-1}$. We experimented with different final decreases in the target cost for the SAP problem and found $\widehat{\Delta C}'$ to be the most appropriate.

The derivation of q is as follows:

$$E_{j-1}\, g = E_j \quad (1 \leq j \leq q)$$

$$\Rightarrow E_{q-1}\, g = E_q$$

$$\Rightarrow g = \frac{E_q}{E_q + \widehat{\Delta C}'} \qquad \text{since} \qquad E_q + \widehat{\Delta C}' = E_{q-1}$$

Now,

$$E_0\, g^q = E_q$$

$$\Rightarrow q = \frac{\ln\left(\frac{E_q}{E_0}\right)}{\ln g}$$

$$\Rightarrow q = \frac{\ln\left(\frac{E_q}{E_0}\right)}{\ln\left(\frac{E_q}{E_q + \widehat{\Delta C}'}\right)}$$

4.1.1 Iterative Improvement within Simulated Annealing

Consider the situation where the SA procedure accepts a configuration with cost slightly greater than $C'(B)$. We conjecture that a likely location for a configuration with cost less than $C'(B)$ is in the neighbourhood of the current configuration.

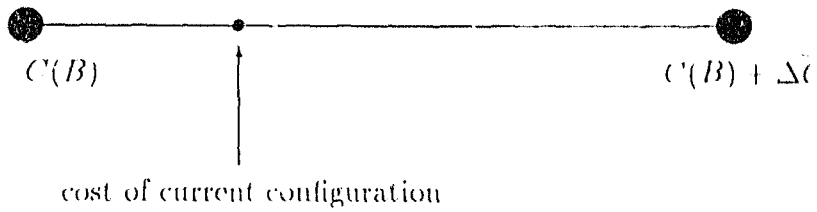


Figure 4.1: Iterative Improvement within SA

Existing schedules fail to exploit this locality property since they are all capable of immediately accepting a sequence of up-hill moves which would distance the procedure from the current configuration's neighbourhood. Our approach is to invoke iterative improvement whenever a configuration with cost slightly greater than $C'(B)$ is accepted. This ensures that the neighbourhood of a configuration with cost slightly greater than $C'(B)$ is searched for a configuration with cost less than $C'(B)$.

It is not practical to invoke iterative improvement for every accepted configuration because of the increase in the number of configurations that would have to be examined. We use a probabilistic technique to determine when iterative improvement is to be invoked.

A uniformly distributed pseudo-random number in the range $[C'(B), C'(B) + \Delta C']$ is generated. If the cost of the current configuration is less than the generated number, then iterative improvement is invoked. By reference to Figure 4.1, it can be seen that the closer the cost of the current configuration is to $C'(B)$, the greater the chance that the generated pseudo-random number will be greater than the cost of the current configuration. Simply stated, the probability of invoking iterative improvement increases as the cost of the current configuration nears $C'(B)$.

An invocation of iterative improvement is terminated when no cost decreasing

moves are possible or when the appropriate j-equilibrium configuration is located.

Appendix A shows pseudocode for SA using the goal-directed schedule.

4.1.2 An Empirical Evaluation

In this section, we compare the goal-directed schedule to schedules proposed by Siarry [29] and Sechen [27] using the SAP problem (Siarry's and Sechen's net-lists) as a benchmark. The schedules proposed by Siarry and Sechen are instances of the classical schedule described in Chapter 3. Their performance is considered here for completeness since the two net-lists used thus far for the 5×5 SAP problem were first used to benchmark the general cooling schedules proposed by Siarry and Sechen. The results are shown in Tables 4.1 and 4.2. We include the results obtained from Huang's schedule for comparison purposes.

The goal-directed schedule found an optimal configuration in all trials. Siarry's schedule found an optimal configuration 38% of the time with an average terminal configuration cost of 211 (Siarry's net-list). Sechen's schedule found an optimal configuration 72% of the time with an average terminal configuration cost of 192 (Sechen's net-list).

The low standard deviation of the number of configurations examined by Siarry's and Sechen's schedules (Tables 4.1 and 4.2) results from their static nature. On the other hand, the high standard deviation of the number of configurations examined by the goal directed schedule (Tables 4.1 and 4.2) results from its dynamic nature and adaptive ability.

The iterative improvement component of the goal-directed schedule had no impact on the results for the 5×5 SAP problem with Siarry's net-list. For Siarry's net-list,

Schedule	Cost (Opt.=200)			Configurations		Optimal	CPU Time (s)
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean*
SA-Si	241	35	310	12,228	1,004	38	4.52
SA-H	223.25	32.7	280	24,864	6,115	66	7.8
SA-G	200	0	200	13,714	11,538	100	5.6

Key	Method
SA-Si	SA using Siarry's schedule
SA-H	SA using Huang's schedule
SA-G	SA using the goal schedule

Table 4.1: 5×5 SAP Performance Summary: Siarry's net-list. (100 Trials.)

Schedule	Cost (Opt.=190)			Configurations		Optimal	CPU Time (s)
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean*
SA-Se	192	3	200	348,000	0[1]	72	90.1
SA-H	200.3	6.5	225	31,008	4,884	38	8.2
SA-G	190	0	190	292,468	309,623	100	65

Key	Method
SA-Se	SA using Sechen's schedule
SA-H	SA using Huang's schedule
SA-G	SA using the goal schedule

Table 4.2: SAP Performance Summary: Sechen's net-list.(100 Trials.)

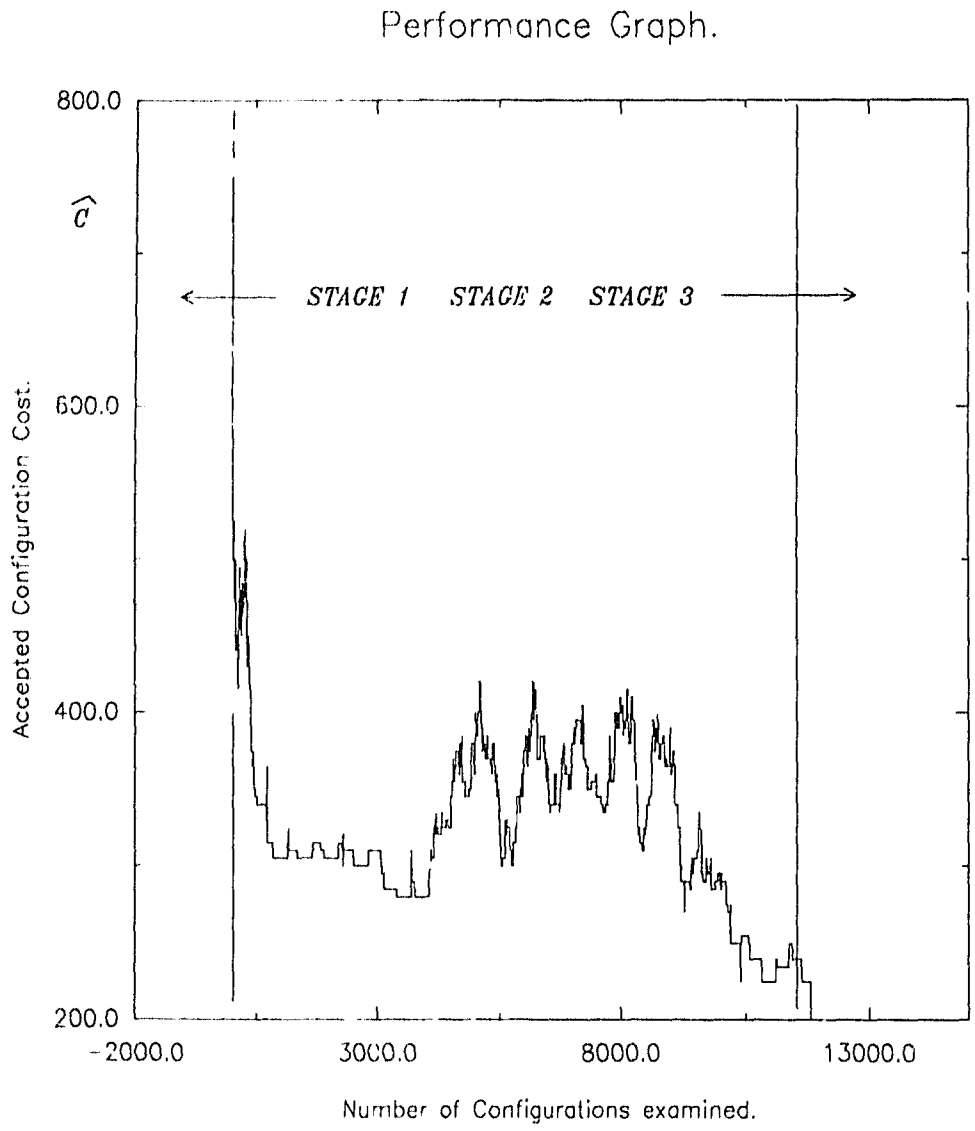


Figure 4.2: 5×5 SAP: Siarry's Net-list - Goal Schedule.

the integration of iterative improvement into the goal directed schedule resulted in a 50% decrease in the average number of configurations examined.

Figure 4.2 shows a performance graph based on an instance of the 5×5 SAP problem (Siarry's net-list) using the goal-directed schedule. It can be seen in this case that stages 1 and 3 are essentially eliminated. If the initial temperature is too high during stage 1, the goal-directed schedule quickly reacts to reduce it. No time is wasted during stage 3 since the procedure terminates after it discovers an optimal configuration.

The horizontal segments in the graph (Fig. 4.2) correspond to invocations of iterative improvement which failed to improve the current best cost. While the number of configurations examined was between 1,500 to 3,500, the procedure sought a downhill path to an optimal configuration through invocations of iterative improvement. When no such path presented itself, the goal-directed schedule was flexible enough to resume hill climbing. By the time the number of configurations examined reached the 10,000 mark, the procedure again started to invoke iterative improvement. On this occasion, the aggressive nature of iterative improvement paid off since an optimal configuration was discovered shortly afterwards.

Consider a graph of the log (base 10) of temperature versus temperature index. We call such a graph a *temperature graph*. Figure 4.3 shows two temperature graphs based on an instance of the 5×5 SAP problem (Siarry's net list) using the goal directed schedule and Huang's schedule.

It can be seen from Figure 4.3 that the goal-directed schedule's initial temperature is higher than Huang's initial temperature (in fact, it is *about twice as high*). We again note that a high initial temperature does not adversely impact the performance

[1]Sechen's schedule employs equilibrium and convergence criteria that result in the examination of a fixed number of configurations derived from the problem size.

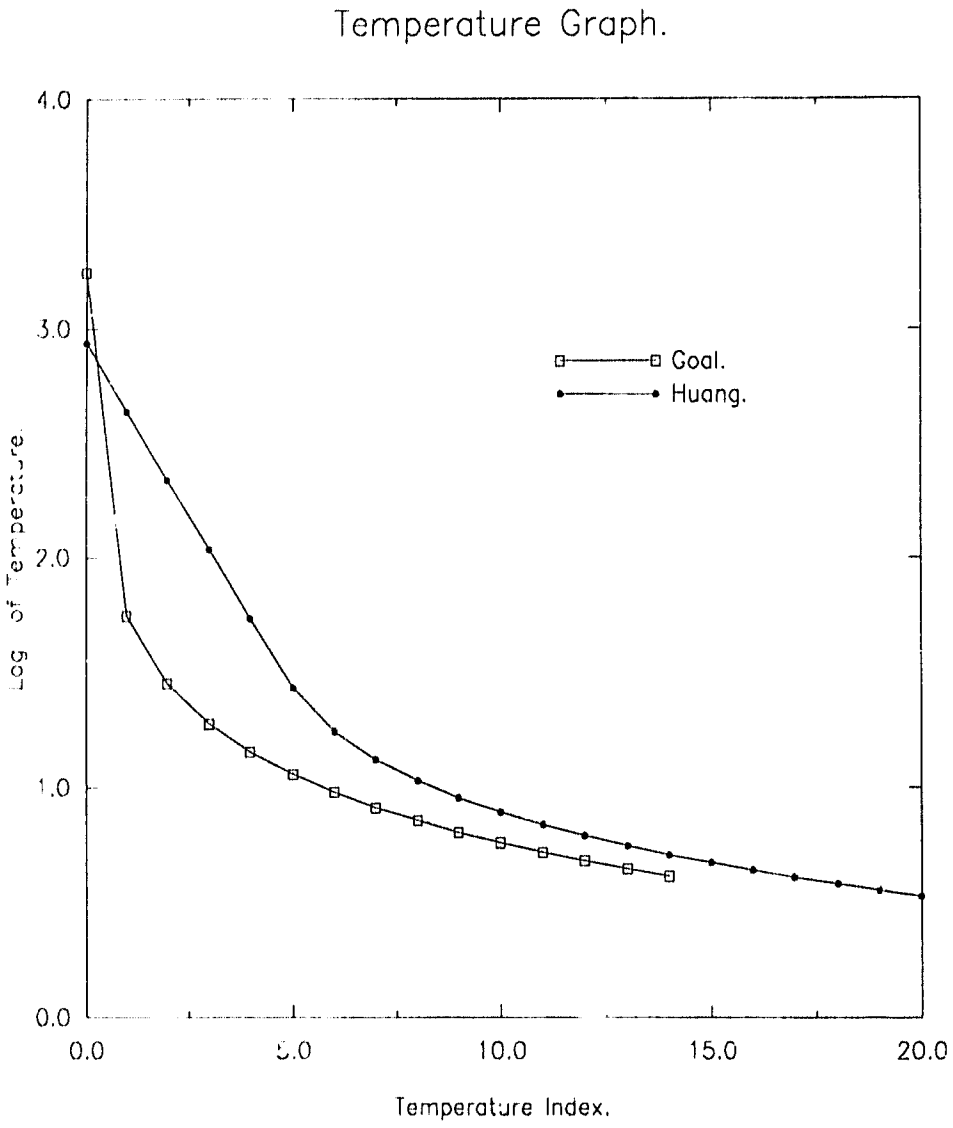


Figure 4.3: Log of Temperature *vs.* Temperature Index.

of the goal-directed schedule. By contrast, an excessively high initial temperature would impact the performance of schedules which examine a constant number of configurations per temperature value. We detailed an example of this in Chapter 2 using the classical schedule.

Consider a graph of the cost of the best configuration seen so far versus temperature index. We call such a graph a *cost graph*. Figure 4.4 shows two cost graphs based on an instance of the 5×5 SAP problem (Siarry's net list) using the goal directed schedule and Huang's schedule. The third plot in Figure 4.4 shows the goal target cost E_j for each temperature index j.

It can be seen from Figure 4.4 that the cost of the best configuration seen so far found by the goal-directed schedule closely follows its target costs. We point this out to support our contention that the goal-directed schedule does not suffer from rapid quenching *i.e.*, the situation where temperature is reduced too quickly resulting in terminal configurations which are neither optimal nor near optimal.

Stage 1 of the SA procedure using Huang's schedule occurs in the first 4 temperature indices (Fig. 4.3). In the goal-directed schedule, the corresponding temperatures are lower with the exception of the initial temperature.

For Siarry's net-list, Figure 4.5 shows a highly structured sub optimal configuration with cost 260 (30% off the optimal). For comparison purposes, an optimal configuration is shown in Figure 4.6. For Huang's schedule, 25% of its terminal configurations which were non-optimal were topologically similar to the configuration in Figure 4.5. This occurred because the temperature was allowed to decrease below a threshold value resulting in the rejection of the up-hill moves necessary to transform a configuration similar to the one in Figure 4.5 to an optimal configuration. The goal schedule is less susceptible to this problem since the temperature is only reduced

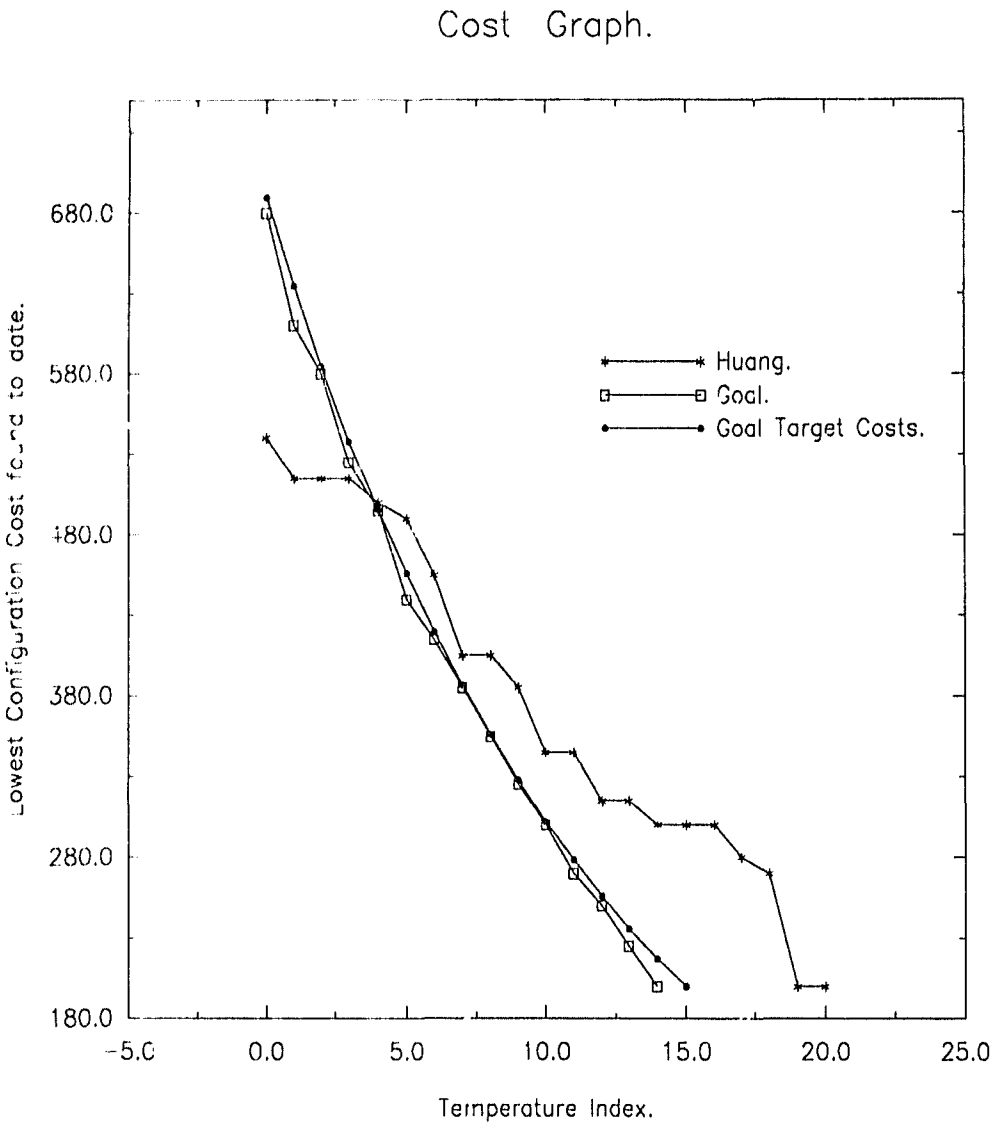


Figure 4.4: Configuration Cost *vs.* Temperature Index.

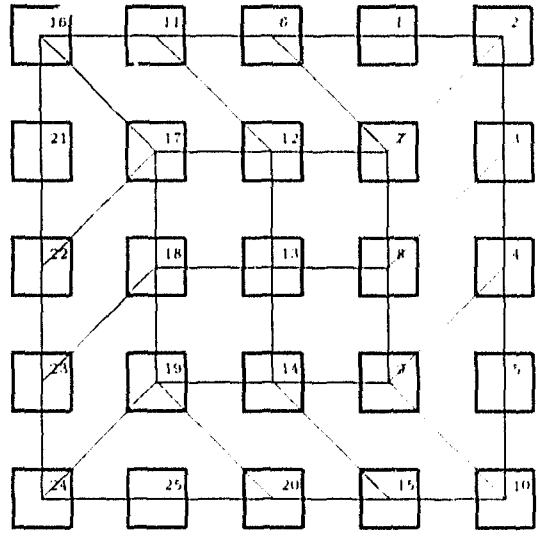


Figure 4.5: SAP: Sub-optimal Configuration - Siarry's Net-list (Cost = 260).

when an equilibrium configuration is encountered.[2]

With Sechen's net-list, it is not difficult to identify a near optimal configuration. But, it is difficult to transform near optimal configurations to optimal configurations. Figure 4.7 shows a near optimal configuration with cost 200 for the 5 × 5 SAP problem (Sechen's net-list) and Figure 4.8 shows an optimal configuration with cost 190 for the 5×5 SAP problem (Sechen's net-list). Recall that there are 16 optimal configurations to the SAP problem (Sechen's net-list). All 16 optimal configurations have the same topology as the configuration shown in Figure 4.8. To transform the configuration shown in Figure 4.7 to any of the 16 optimal configurations would require several up-hill moves resulting in the repositioning of modules 4, 5, 9, 10, 13, 14 and 15.

For Huang's schedule, 60% of its terminal configurations which were non optimal

[2]Assuming that the value of T_j is appropriate for finding a j-equilibrium configuration.

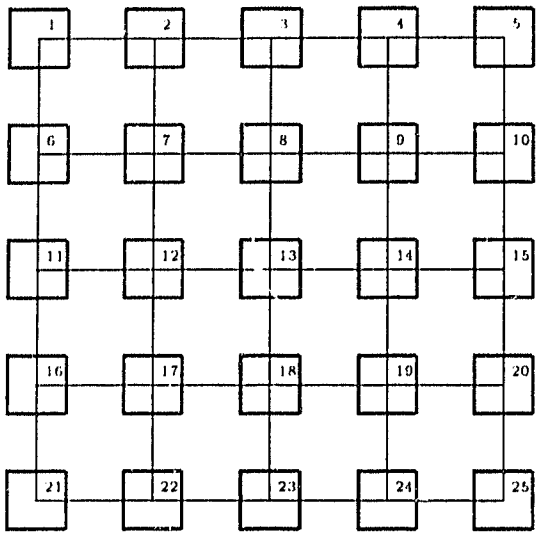


Figure 4.6: SAP: Optimal Configuration - Siarry's Net-list (Cost = 200).

were very similar in structure to the configuration in Figure 4.7. As in the case of Siarry's net-list, temperature was allowed to decrease below a threshold value resulting in the rejection of the up-hill moves necessary to transform a configuration similar to the one in Figure 4.7 to an optimal configuration.

Figure 4.9 shows a performance graph based on an instance of the 5×5 SAP problem (Sechen's net-list) using the goal-directed schedule. It can be seen from Figure 4.9 than near optimal configurations were easily accessible. Failed invocations of iterative improvement are also visible (horizontal line segments).

In a recent article, Kling and Banerjee [14] draw on their experience in cell placement to quantify the size of a placement problem which is complex enough to discriminate between CO techniques, but at the same time, is simple enough to permit formal analysis. They postulate that a placement problem with 100 modules satisfies

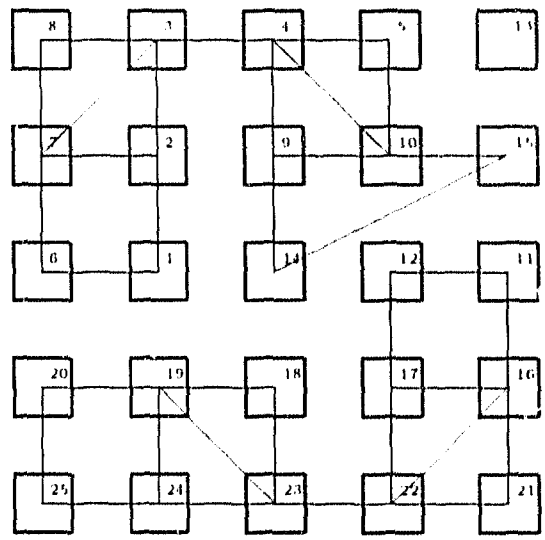


Figure 4.7: SAP: Near Optimal Configuration Sechen's Net list (Cost 200).

the above criteria. With this in mind, we now move from the 5 × 5 SAP problem to the 10×10 SAP problem.

Consider the 10×10 SAP problem using Siarry's net-list and cost function. In an optimal configuration, all modules are connected only to their nearest horizontal and vertical neighbours the cost of an optimal configuration is 900. We evaluated the goal directed schedule using the 10×10 SAP problem (Siarry's net-list). A typical number of temperature indices required by the goal-directed schedule for this problem was 40. None of the trials found a 16-equilibrium configuration *i.e.*, a configuration with cost less than E_{16}. In essence, the temperature value T_{16} was inappropriate for locating a 16-equilibrium configuration.

We address this problem in the following section. Some *modifications* to the goal-directed schedule are proposed. We call the resulting schedule the *extended*

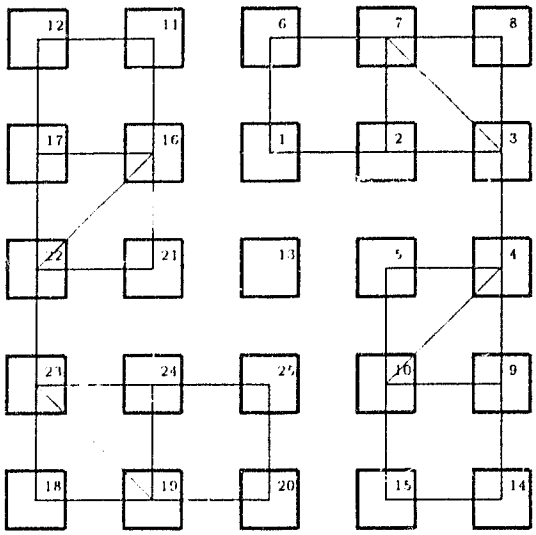


Figure 4.8: SAP: Optimal Configuration - Sechen's Net-list (Cost = 190).

goal-directed general cooling schedule or the e-goal schedule.

It should be noted that the extended goal-directed schedule supersedes the goal-directed schedule. We are not suggesting that the goal-directed schedule is more suited to a certain class of problem that the extended goal-directed schedule. Rather, we have described the goal-directed schedule in order to show how the extended goal-directed schedule evolved.

4.2 The Extended Goal-Directed Approach

For the goal directed schedule, equilibrium is established at temperature T_j if $C(B)$ is less than or equal to the target cost E_j. We assume that T_j is an appropriate temperature value for locating a j-equilibrium configuration. However, there is no guarantee

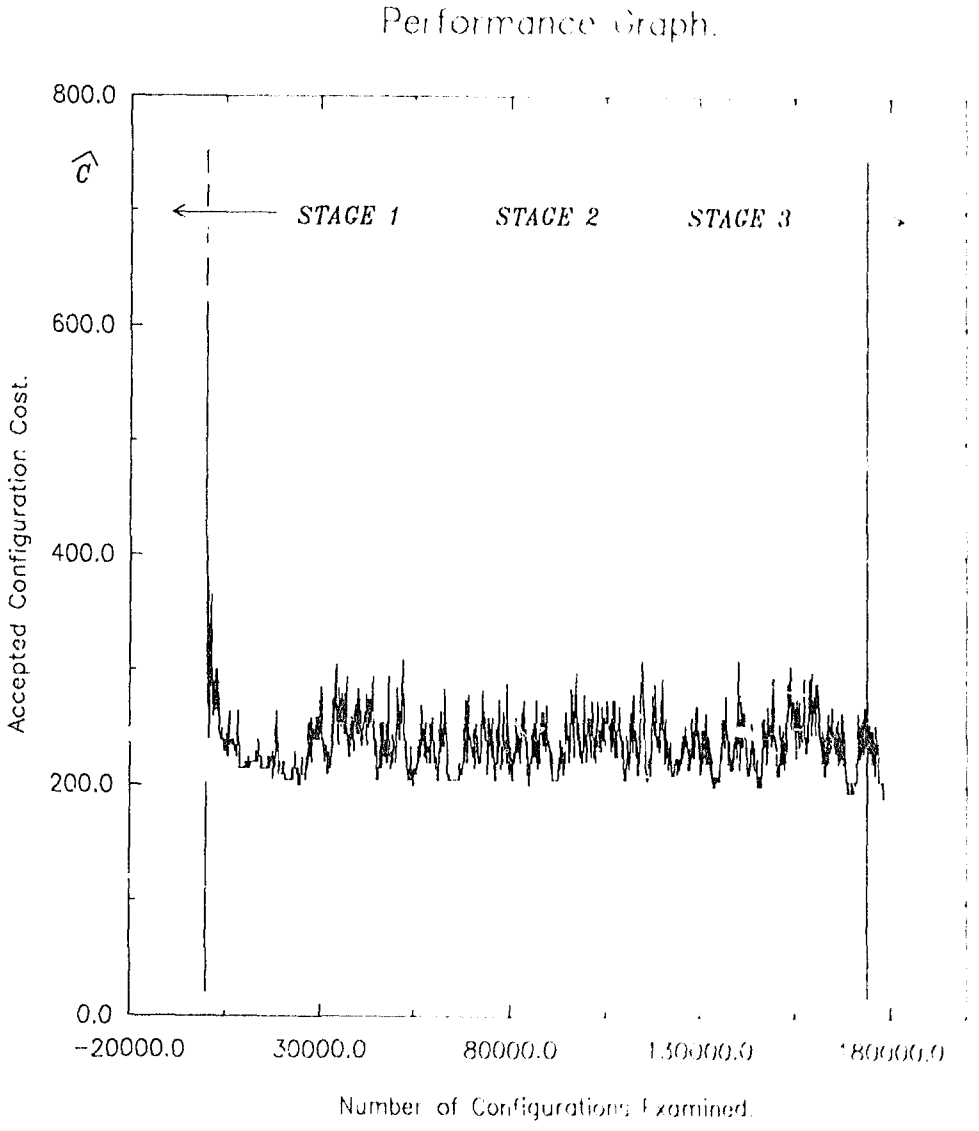


Figure 4.9: SAP: Sussman's Net list - Goal Schedule.

that an equilibrium configuration will be encountered at a given temperature. In our experience, failure to detect an equilibrium configuration occurred primarily because the value of temperature was too high. We address this problem by considering a range of temperature values for each temperature index. This new approach is based on the assumption that T_j is at least high enough to locate a j-equilibrium configuration.

In the following paragraphs, we relate temperature in the goal directed schedule to temperature in the extended goal-directed schedule. To this end, we expand our notation such that temperature in the extended goal-directed schedule is denoted by $\mathcal{T}$ and temperature in goal-directed schedule is denoted by T.

An example of an extended goal-directed temperature graph is shown in Figure 4.10. Transition from one temperature index to the next is represented by a solid circle. For a given temperature index, $\mathcal{T}_j$ is never set higher that T_j. Subsequently, $\mathcal{T}_j$ may be set to T_{j+1}, T_{j+2} etc. This results in the wave pattern seen in Figure 4.10. Note that the temperature increases and decreases with time. We are not aware of any other general cooling schedule which controls temperature in this manner. For existing general cooling schedules, temperature increases are forbidden. We now describe the temperature control mechanism for the extended goal-directed schedule.

$\mathcal{T}_j$ is initialized to T_j. The search for a j-equilibrium configuration is then initiated. We assume that the temperature is too high if a j-equilibrium configuration is not located after the examination of $500Q$ configurations.[3] To correct this, the temperature is decreased. For example, if $\mathcal{T}_j = T_j$ then $\mathcal{T}_j$ is set to T_{j+1}. If the temperature is still too high, $\mathcal{T}_j$ is subsequently set to T_{j+2}, T_{j+3} etc. This process is continued until a j-equilibrium configuration is found.

[3]The constant 500 was arrived at through experimentation with the SAP problem.

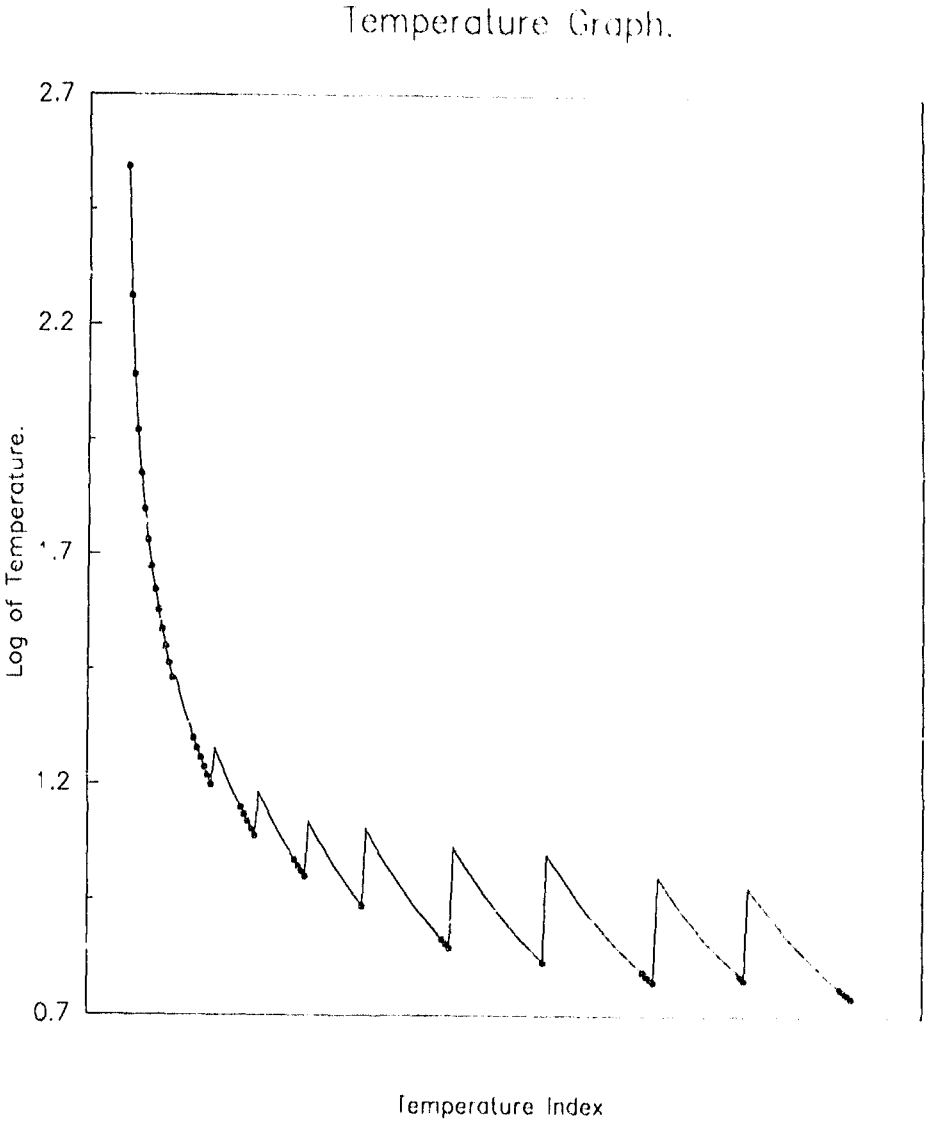


Figure 4.10: An Example e-goal Temperature Graph.

At this point, the SA procedure moves to the next temperature index where the objective is to find a $(j+1)$-equilibrium configuration *i.e.* a configuration with cost less than or equal to E_{j+1}.

For discussion purposes, assume $\mathcal{T}_j = T_{j+2}$ when equilibrium was established at temperature index j. It is reasonable to assume that a temperature lower than T_{j+2} would be suitable for locating a $(j+1)$-equilibrium configuration. Consequently, $\mathcal{T}_{j+1}$ is set to T_{j+3} and the search for a $(j+1)$-equilibrium configuration is initiated.

If a $(j+1)$-equilibrium configuration is not found after the examination of $1000Q$ configurations then we assume that the temperature is too low.[4] To correct this, we reset $\mathcal{T}_{j+1}$ to T_{j+1} and continue the search for a $(j+1)$-equilibrium configuration as described above.

With reference to Figure 4.10, the trough of each wave contains a sequence of solid circles. With the exception of the first circle in each sequence, the circles correspond to the occasions where a j-equilibrium configuration was found without resorting to increasing the temperature.

Recall that in the goal-directed schedule, temperature decreased geometrically. The same is true within each temperature index in the extended goal-directed schedule. Between each wave crest and its corresponding trough in Figure 4.10, we can see the effect of geometric temperature decrements.

We now turn our attention to SA's convergence criteria. Several strategies have been proposed, *viz.*,

- Converge when the number of accepted configurations, expressed as a percentage of the total number of configurations generated at a particular temperature, falls below a predetermined value [37].

[4]The constant 1000 was arrived at through experimentation with the SAP problem.

- Converge when the cost of the current best configuration remains unchanged for a number of consecutive temperature values [27].
- Converge when the 'similar cost' condition is satisfied, *i.e.*, when $C(s_{max}) - C(s_{min}) = \Delta C_{max}$ (see Huang's schedule in Chapter 3) [10].

We have found through experimentation with the SAP problem that the third convergence strategy has the best performance. Figure 4.11 shows a portion of a performance curve for the temperature value prior to convergence using Huang's strategy. We show this to emphasize that s_{max} and s_{min} must be neighbours. We feel that this neighbour constraint in the convergence strategy is unnecessary and undesirable.

Ideally, the SA procedure should converge when an optimal configuration has been found or when the temperature has decreased to a point that prohibits meaningful hill climbing. We define *meaningful* hill-climbing as hill-climbing capable of freeing the SA procedure from the current local minimum. The first part of this ideal is captured by the extended goal-directed schedule since convergence occurs when equilibrium is reached at temperature index q. With regard to the second part of the ideal, we have found through experimentation with the SAP problem that meaningful hill climbing is absent when $C(s_{max}) - C(s_{min}) \leq \widehat{\Delta C}$ for a particular temperature value. Our SA procedure converges when this situation persists for Q consecutive temperature values where Q is the size of the problem instance (for the SAP problem Q is the number of cells).

4.2.1 Iterative Improvement within Simulated Annealing

The above equilibrium detection mechanism is based on the number of configurations examined at a temperature value. Unfortunately, this mechanism is adversely

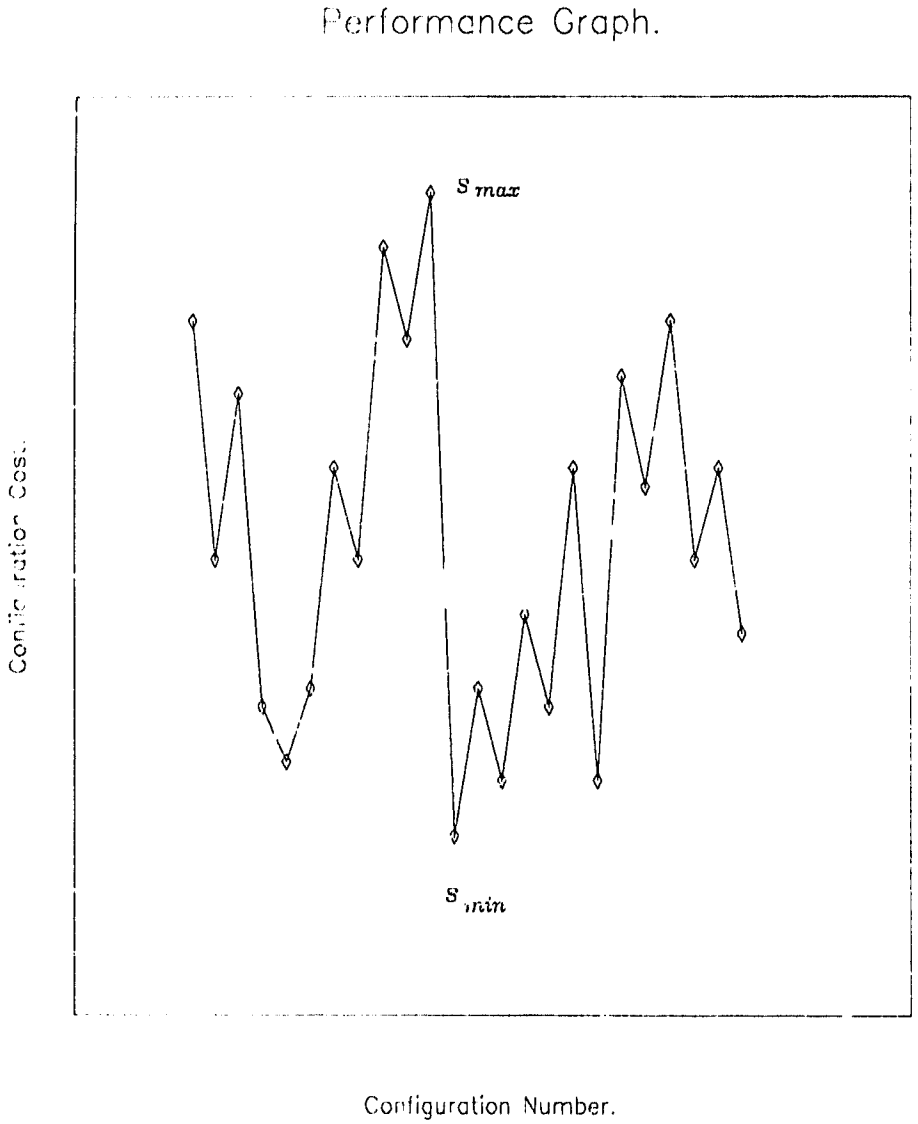


Figure 4.11: Convergence Strategy - Huang's Schedule.

affected by the probabilistic scheme for invoking iterative improvement used in the goal-directed schedule. Consequently, for the extended goal directed schedule, an alternative scheme is used *i.e.*, iterative improvement is invoked whenever a best so far configuration is encountered. With this new mechanism, the number of iterative improvement invocations is relatively small. This is the case since the number of occasions a best-so-far configuration is encountered is usually small relative to the total number of configurations examined. We can say with reasonable confidence that with this new mechanism for invoking iterative improvement, the negative impact on the SA procedure with regard to the increase in the number of configurations examined is negligible. However, it is likely that there is some positive impact with regard to locating equilibrium configurations. This in turn, may speed the procedure along.

Appendix B shows pseudocode for SA using the extended goal directed schedule.

4.2.2 An Empirical Evaluation

In this section, we evaluate the extended goal directed schedule on the 5×5 SAP problem using Siarry's and Sechen's net lists. The results are shown in Tables 4.3 and 4.4. We include the results obtained from Huang's schedule and the goal directed schedule for comparison purposes.

The extensions to the goal-directed schedule had little effect on the results for the 5×5 SAP problem (Siarry's net list). For Sechen's net list, the average total number of configurations examined was reduced by slightly less that one half. The extended goal-directed schedule yielded terminal configurations which were non optimal three times in 100 trials resulting in an average terminal configuration cost of 190.2 (0.1% off the optimal).

Schedule (100 Trials)	Cost (Opt.=200)			Configurations		Optimal	CPU Time (s)
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean*
SA-H	223.25	32.7	280	24,864	6,155	66	7.8
SA-G	200	0	200	13,714	11,538	100	5.6
SA-E	200	0	200	10,129	9,268	100	4.3

Key	Method
SA-H	SA using Huang's schedule
SA-G	SA using the goal schedule
SA-E	SS using the e-goal schedule

Table 4.3: 5×5 SAP Performance Summary: Siarry's net-list.

Schedule (100 Trials)	Cost (Opt.=190)			Configurations		Optimal	CPU Time (s)
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean*
SA-H	200.3	6.5	225	31,008	4,884	38	8.2
SA-G	190	0	190	292,468	309,623	100	65
SA-E	190.2	1.2	200	147,110	92,083	97	37.5

Table 4.4: SAP Performance Summary: Sechen's net-list.

Schedule	Cost (Opt.=900)			Configurations *10^6		Optimal	CPU Time (s)
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean*
SA-II	1,059	137	1,280	1.62	0.5	36.6	561
SA-E	920	58	1,180	9.1	5.3	80	3,311

Table 4.5: 10×10 SAP Performance Summary: Siarry's net list. (30 Trials.)

For the 10×10 SAP problem we use three different net lists. The first is Siarry's net-list *i.e.*, in an optimal configuration, all modules are connected only to their nearest horizontal and vertical neighbours. The cost function used is the same as for the 5×5 SAP problem yielding an optimal configuration cost of 900.

The second net-list is a variation on Siarry's net list where individual nets are weighted *i.e.*, the cost of a net is given as the product of weight and Manhattan distance. Weights are chosen at random in the range 1 to 8. In an optimal configuration, all modules are connected to only their nearest horizontal and vertical neighbours. The cost of an optimal configuration is the sum of the net weights. For our benchmark, this cost is 818. We call the resulting net list a *random weight* net list.

The third net-list describes a 3-regular, 3-connected bipartite graph called Horton's graph [5]. Horton's graph is shown in Figure 4.12. From the point of view of placement, each node in the graph is a cell and each edge is a net. Configuration cost is calculated as the sum of the Manhattan distance between connected cells. The cost of an optimal configuration is unknown. Consequently, we use the lower bound cost of 145, *i.e.*, the situation where all connected cells are placed adjacent to each other.

We evaluated Huang's schedule, and the extended goal directed schedule using the 10×10 SAP problem (Siarry's, Horton's and the random net lists). The results are shown in Tables 4.5, 4.6 and 4.7.

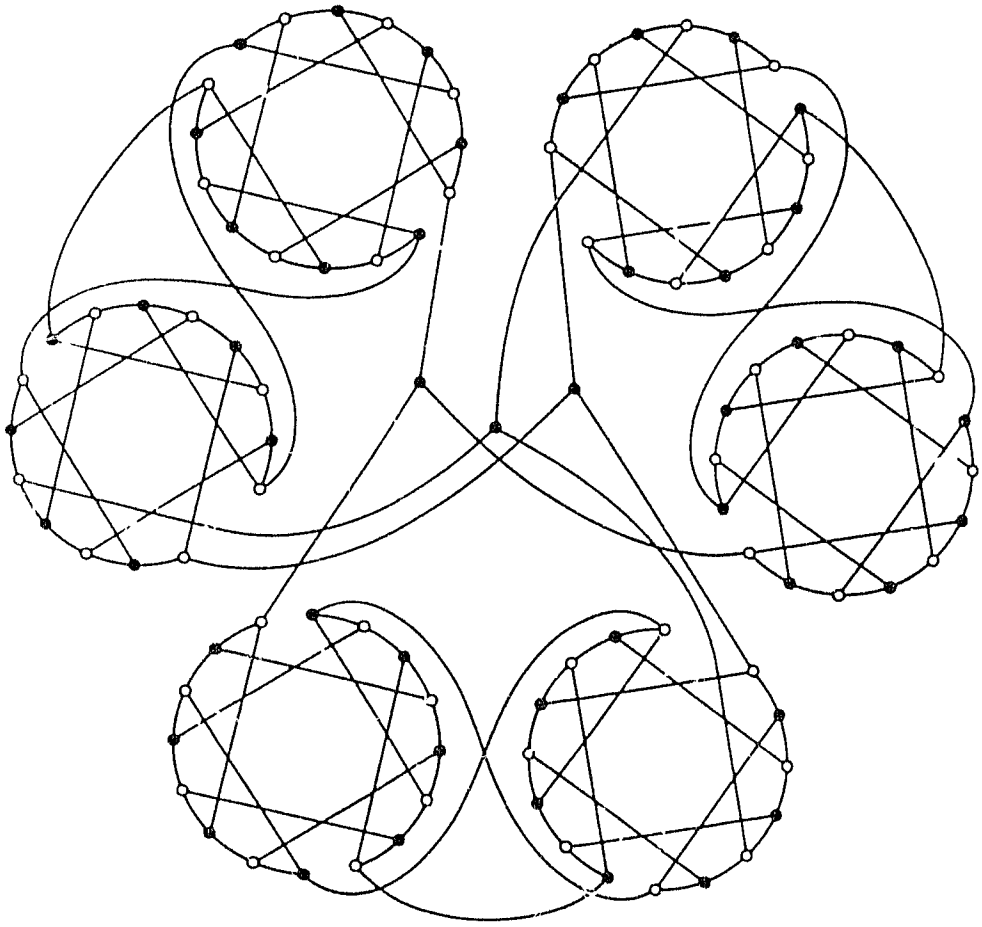


Figure 4.12: Horton's Graph.

Schedule	Cost (Opt.=818)			Configurations *10^6		Optimal	CPU Time (s)
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean*
SA-H	982	80	1,097	2.7	1.2	13.3	1,049
SA-E	884	76	1,048	20	6.0	36.6	7,735

Table 4.6: 10×10 SAP Performance Summary: Random Weight net list. (30 Trials.)

Schedule	Cost (Opt.=145)				Configurations *10^6		CPU Time (s)
	mean	*std*	x_{MAX}	x_{MIN}	*mean*	*std*	*mean*
SA-H	225.83	3.88	233	215	2.06	0.5	684
SA-E	218.76	2.33	223	214	29.6	3.1	9,990

Table 4.7: 10×10 SAP Performance Summary: Horton's net list. (30 Trials.)

For the 10×10 SAP problem (Siarry's net-list), the extended goal directed schedule found an optimal configuration 80% of the time with an average cost of 920 (2.2% off the optimal). Huang's schedule found an optimal configuration 36% of the time with an average cost of 1,059 (17.6% off the optimal).

For the 10×10 SAP problem (the random net-list), the extended goal directed schedule found an optimal configuration 36.6% of the time with an average cost of 884 (8.0% off the optimal). Huang's schedule found an optimal configuration 13.3% of the time with an average cost of 982 (20.0% off the optimal).

For the 10×10 SAP problem (Horton's net-list), the extended goal directed schedule had an average terminal configuration cost of 218.76. Huang's schedule had an average terminal configuration cost of 225.83.

As in the case of the 5×5 SAP problem, the improved quality of the terminal

configuration found by the extended goal-directed schedule comes at a cost. The extended goal directed schedule examined on average, between 6 and 10 times as many configurations as did Huang's schedule on the 10×10 SAP problem (Siarry's, Borton's and the random weight net-lists).

4.3 Statistical Evaluation

We now turn our attention to comparing the statistical difference between the performance of the extended goal directed schedule and the performance of Huang's schedule. Both schedules were evaluated on five instances of the SAP problem. For each problem instance, both schedules yielded a random sample of terminal configuration costs. Our objective is to demonstrate statistically that the samples come from different populations and therefore, one schedule is statistically superior to the other.

We postulate two populations where

- the first population has mean μ_1 and variance ${\sigma_1}^2$
- the second population has mean μ_2 and variance ${\sigma_2}^2$
- n_1 and n_2 independent samples are drawn from the first and second populations respectively
- $\overline{y}_1$ and ${std_1}^2$ are estimates of the first population's mean and variance calculated from the n_1 data points
- $\overline{y}_2$ and ${std_2}^2$ are estimates of the second population's mean and variance calculated from the n_2 data points.

The point estimator of the difference between the two population means is $(\overline{y}_1 - \overline{y}_2)$. A distribution of point estimates results if we repeat the sampling process (including the calculation of the point estimate) several times. From Mendenhall [17], we know that the probability distribution of the estimator, $(\overline{y}_1 - \overline{y}_2)$ is approximately normal for n_1 and n_2 both greater than or equal to 30. The mean and standard deviation of this distribution are [17]

$$E(\overline{y}_1 - \overline{y}_2) = (\mu_1 - \mu_2) \qquad \sigma_{(\overline{y}_1 - \overline{y}_2)} = \sqrt{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}}$$

The bound on the error of the point estimate is

$$2\sqrt{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}}$$

We can use ${std_1}^2$ and ${std_2}^2$ as estimates for ${\sigma_1}^2$ and ${\sigma_2}^2$ in the above formulae. This approximation is reasonable when n_1 and n_2 are greater than or equal to 30 [17]. We can conclude that our initial postulate of two populations is true if

$$(\overline{y}_1 - \overline{y}_2) > 2\sqrt{\frac{{std_1}^2}{n_1} + \frac{{std_2}^2}{n_2}}$$

Table 4.8 shows the appropriate statistics for comparing the statistical difference between the performance of the extended goal directed schedule and the performance of Huang's schedule on the 5×5 and 10×10 SAP problems. With reference to Table 4.8, we conclude that the performance of the extended goal directed schedule on the 5×5 and 10×10 SAP problems is statistically superior to that of Huang's schedule since $\overline{y}_1$ was always greater than $\overline{y}_2$ and $(\overline{y}_1 - \overline{y}_2)$ was always greater than $2\sqrt{\frac{{std_1}^2}{n_1} + \frac{{std_2}^2}{n_2}}$.

4.4 Summary

In this chapter we described a goal directed general cooling schedule and the extended goal-directed general cooling schedule for SA.

Problem (SAP)	Net-list	n_1	n_2	SA H $\overline{y}_1$	std_1	SA F $\overline{y}_2$	std_2	$(\overline{y}_1 - \overline{y}_2)$	$2s_{(\overline{y}_1 - \overline{y}_2)}$
5×5	Siarry	100	100	223.25	32.7	200	0	23.25	6.54
5×5	Sechen	100	100	200.4	6.5	190.2	1.2	10.1	1.32
10×10	Siarry	30	30	1,059	137	920	58	139	54.32
10×10	Random	30	30	982	80	884	76	98	10.29
10×10	Horton	30	30	225.83	3.88	218.76	2.33	7.05	1.5

Table 4.8: Schedule Statistics.

We compared the performance of the extended goal directed schedules with that of Huang's schedule on five instances of the SAP problem. The extended goal directed schedule consistently outperformed Huang's schedule in terms of the quality of the terminal configuration. We demonstrated that the improvement in the quality of the average terminal configuration achieved by the extended goal directed schedule over Huang's schedule is statistically significant.

The extended goal-directed schedule has two shortcomings, *viz.*, the cost of an optimal configuration to a problem instance has to be derived or estimated and, a large number of configurations have to be examined before an optimal or near optimal configuration can be found.

In Chapter 5, we describe a procedure which reduces the number of configurations that need to be examined by the extended goal-directed schedule.

Chapter 5

Simulated Sintering

One of the major problems with SA is its long computation times. This problem can be addressed by using a fast heuristic to find a good initial configuration and then applying simulated annealing. Such an approach is called *Simulated Sintering* [9].

In this chapter, we describe simulated sintering. To exploit the potential of simulated sintering one needs an appropriate general cooling schedule. We show how the extended goal-directed schedule can be used as an effective general cooling schedule for simulated sintering.

5.1 Simulated Sintering Procedure

Simulated Sintering (SS) is a general purpose CO technique which has evolved from SA. Like SA, SS starts from an initial configuration and constructs a sequence of configurations. SA and SS differ in the manner in which the initial configuration is chosen.

The initial configuration for SA is chosen randomly from a problem's configuration

space. For SS, a fast heuristic is used to find a good initial configuration. SS consists of two parts *viz.*, the search for the initial configuration and the subsequent hill climbing under the control of a cooling schedule.[1]

The rationale for using SS over SA is as follows. If the time taken to find a good initial configuration using a heuristic is less than the time taken to find a similar configuration using SA then it is likely that the computational effort required by SS will be less than the computational effort required by SA.

SS can be viewed in terms of the three stage SA model described in Chapter 3. Within that context, SS consists only of stage 3 and part of stage 2. Stage 1 is eliminated.

5.2 General Cooling Schedule

The success of SS hinges on the identification of an appropriate initial temperature. If the initial temperature is too high, then SS degenerates to SA and no reduction in computation time is achieved. If the initial temperature is too low, then rapid quenching occurs *i.e.*, the situation where temperature is reduced too quickly resulting in terminal configurations which are neither optimal nor near optimal.

Let s_0^* be the starting configuration for SS.[2] An appropriate initial temperature for SS would be the temperature at which SA could locate an equilibrium configuration similar to s_0^*.

What is required is the ability to relate configuration costs to temperature values. Grover [9] was the first to discuss the problem of finding an appropriate initial tem

[1] Note, we do not exclude the possibility that the first part of SS contains a hill climbing element.

[2] For notational convenience, we use s_0^* to represent SS's initial configuration and s_0 to represent SA's initial configuration.

perature for SS. His approach was to use knowledge of the problem gained through experience to estimate the initial temperature. For general purpose CO, this approach is unsatisfactory since it assumes knowledge of the optimization problem.

Rose [25] also addresses the problem of relating configuration costs to temperature values. His approach is based on measuring the probability distribution of $\widehat{\Delta C}$. By his own admission, this probability distribution is difficult to measure exactly and initial temperatures are consistently overestimated.

The extended goal-directed schedule is equally applicable to both SA and SS. For SS, we proceed in exactly the same way as we did for SA with the exception of the initial temperature calculation. The initial temperature for SS is set to T_j $(1 \leq j \leq q)$ where E_j is the largest target cost less than or equal to $C(s_0^*)$.

Appendix C shows pseudocode for SS using the extended goal-directed schedule.

5.3 An Empirical Evaluation

We use two different heuristics for SS *viz.*, multiple improvement (consisting of 10 applications of iterative improvement) and SA using Huang's schedule. We evaluated SS on the SAP problem using Siarry's net-list, Sechen's net-list, the random-weight net list and Horton's net list. The results are shown in Tables 5.1, 5.2, 5.3, 5.4 and 5.5. We have included the results obtained by SA using Huang's schedule and the extended goal directed schedule for comparison purposes.

As already indicated, we use two different heuristics for SS. Both of these heuristics are general purpose CO procedures. A general purpose heuristic is not a requirement for SS. In fact, it is more likely that a suitable heuristic for SS is problem dependent and constructive in nature.

The breakdown of the average number of configurations examined by the two parts of SS is shown in Tables 5.6 and 5.7. The column labeled 'percentage reduction' in Tables 5.6 and 5.7 shows the decrease in the average number of configurations examined by SS using the extended goal-directed schedule as a percentage of the average number of configurations examined by SA using the extended goal directed schedule. Our intent is to reduce the computation time required by the hill climbing part of SS relative to the corresponding computation time required by SA. Consequently, the average number of configurations examined by SS's heuristic is not included in the above percentage calculation.

In terms of the average terminal configuration cost, Tables 5.8 and 5.9 show the appropriate statistics for comparing the statistical difference between the performance of SA and SS using the extended goal directed schedule on the 5 × 5 and 10 × 10 SAP problems.

From Tables 5.8 and 5.9, we deduce that in terms of the average terminal configuration cost, the performance difference between SA and SS using the extended goal directed schedule on SAP problem (Starry's net list, Sechen's net list, the random weight net list and Horton's net-list) is not statistically significant. This is an important result since Grover postulated that the use of SS over SA would inevitably result is some decrease in terminal configuration quality.

For the 5×5 SAP problem, neither multiple improvement nor SA using Huang's schedule satisfy the criteria for a fast heuristic discussed earlier. They are not considered appropriate heuristics for the 5×5 SAP problem since they examined a large number of configurations relative to the total number of configurations examined by SS. For the 10×10 SAP problem, SA using Huang's schedule proved to be a good heuristic for SS using the extended goal directed schedule.

5.4 Summary

In this chapter, we have described SS and demonstrated how the extended goal-directed schedule can be used as an effective general cooling schedule for SS. We compared the performance of SA and SS using the extended goal-directed schedule on five instances of the SAP problem. We demonstrated that in terms of the average terminal configuration cost, the performance difference between SA and SS using the extended goal-directed schedule on SAP problem (Siarry's net-list, Sechen's net-list, the random weight net-list and Horton's net-list) is not statistically significant.

We achieved our goal of reducing the computation times required by SA using the extended goal-directed schedule for some problems. However, these times are still substantial when compared to the computation times of existing general cooling schedules for SA. The increased computation time may be justifiable if there is a corresponding increase in the quality of the terminal configuration. In the following chapter we detail one example where this is not the case *i.e.*, no increase in terminal configuration quality is achieved. This begs the question, can we predict when it is likely that use of the extended goal directed schedule over Huang's schedule will result in superior quality terminal configurations? We address this question in Chapter 7.

In the following chapter, we show how SS and the extended goal-directed schedule can be applied to two problems from VLSI physical design.

Schedule	Cost (Opt. = 200)			Configurations		Optimal	CPU Time
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean* (*s*)
SA-H	223.25	32.7	280	24,864	6,155	66	7.8
SA-E	200	0	200	10,129	9,268	100	4.3
SS-M	200.6	6	260	24,058	59,559	99	14.7
SS-H	200	0	200	29,936	14,633	100	9.4

Key	Method
SA-H	SA using Huang's schedule
SA-E	SA using the e-goal schedule
SS-M	SS using the e-goal schedule and multiple improvement as heuristic
SS-H	SS using the e-goal schedule and SA using Huang's schedule as heuristic

Table 5.1: 5×5 SAP Performance Summary: Siarry's net list. (100 Trials.)

Schedule	Cost (Opt. = 190)			Configurations		Optimal	CPU Time
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean* (*s*)
SA-H	200.3	6.5	225	31,008	4,884	38	8.2
SA-E	196.2	1.2	200	147,110	92,083	97	37.5
SS-M	190.3	1.7	200	148,866	105,174	96	41
SS-H	190.2	1.21	200	126,433	95,759	97	31.8

Table 5.2: SAP Performance Summary: Sechen's net list. (100 Trials.)

Schedule	Cost (Opt. 900)			Configurations *10^6		Optimal	CPU Time
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean* (s)
SA H	1,059	137	1,280	1.62	0.5	36.6	561
SA E	920	58	1,180	9.4	5.3	80	3,311
SS M	930	65	1,175	9.0	5.4	66.6	3,934
SS H	931	60	1,100	6.4	6.0	70	2,292

Table 5.3: 10×10 SAP Performance Summary: Siarry's net-list. (30 Trials.)

Schedule	Cost (Opt. = 818)			Configurations *10^6		Optimal	CPU Time
	mean	*std*	x_{MAX}	*mean*	*std*	%	*mean* (s)
SA H	982	80	1,097	2.7	1.2	13.3	1,049
SA E	884	76	1,048	20.0	6.0	36.6	7,735
SS M	896	86	1,075	16.4	6.0	40	6,896
SS H	891	79.1	1,038	13.1	6.6	36.6	5,029

Table 5.4: 10×19 SAP Performance Summary: Random Weight net-list. (30 Trials.)

Schedule	Cost (Opt. = 145)				Configurations *10^6		CPU Time
	mean	*std*	x_{MAX}	x_{MIN}	*mean*	*std*	*mean* (s)
SA H	225.83	3.38	233	215	2.06	0.5	684
SA E	218.76	2.33	223	214	29.6	3.1	9,990
SS-M	219.76	2.86	225	214	28.1	3.6	9,805
SS-H	217.7	2.16	222	215	19.9	4.3	6,675

Table 5.5: 10×10 SAP Performance Summary: Horton's net-list. (30 Trials.)

Problem (SAP)	Net-list	SA-E	SS-M		Percentage Reduction
			heuristic	e goal	
5×5	Siarry	10,129	8,861	15,197	50
5×5	Sechen	147,110	7,820	111,646	1.1
10×10	Siarry	$9.4 * 10^6$	$0.20 * 10^6$	$8.8 * 10^6$	6.3
10×10	Random	$20.0 * 10^6$	$0.30 * 10^6$	$16.1 * 10^6$	19.5
10×10	Horton	$29.6 * 10^6$	$0.21 * 10^6$	$27.89 * 10^6$	5.9

Table 5.6: Mean Number of Configurations Examined.

Problem (SAP)	Net-list	SA-E	SS-H		Percentage Reduction
			heuristic	e goal	
5×5	Siarry	10,129	24,865	5,071	49.9
5×5	Sechen	147,110	31,010	95,423	35
10×10	Siarry	$9.4 * 10^6$	$1.64 * 10^6$	$4.8 * 10^6$	48.9
10×10	Random	$20.0 * 10^6$	$2.8 * 10^6$	$10.3 * 10^6$	48.5
10×10	Horton	$19.9 * 10^6$	$2.07 * 10^6$	$17.8 * 10^6$	10.6

Table 5.7: Mean Number of Configurations Examined.

Problem (SAP)	Net list	n_1	n_2	SS-M $\overline{y}_1$	std_1	SA-E $\overline{y}_2$	std_2	$(\overline{y}_1 - \overline{y}_2)$	$2\sigma_{(\overline{y}_1-\overline{y}_2)}$
5×5	Siarry	100	100	200.6	6	200	0	0.6	1.2
5×5	Sechen	100	100	190.3	1.7	190.2	1.2	0.1	0.41
10×10	Siarry	30	30	930	65	920	58	10	31.8
10×10	Random	30	30	896	86	884	75	12	41.9
10×10	Horton	30	30	219.76	2.89	218.76	2.33	1.0	1.35

Table 5.8: Schedule Statistics: SS-M and SA-E.

Problem (SAP)	Net list	n_1	n_2	SS-H $\overline{y}_1$	std_1	SA-E $\overline{y}_2$	std_2	$(\overline{y}_1 - \overline{y}_2)$	$2\sigma_{(\overline{y}_1-\overline{y}_2)}$
5×5	Siarry	100	100	200	0	200	0	0	0
5×5	Sechen	100	100	190.2	1.21	190.2	1.2	0	0.34
10×10	Siarry	30	30	931	60	920	58	11	30.47
10×10	Random	30	30	891	79.1	884	76	7	40
10×10	Horton	30	30	217.7	2.16	218.76	2.33	1.06	1.66

Table 5.9: Schedule Statistics: SS-H and SA-E.

Chapter 6

Integrated Circuit Physical Design

Physical design is concerned with transforming a structural representation of an integrated circuit into a physical representation which contains information used in circuit fabrication. In this chapter, we discuss two physical design styles namely, standard-cell and macro/custom. Both are concerned with the placement of rectangular modules in the plane. We report the results obtained by applying simulated annealing (and simulated sintering) using the extended goal directed to eight standard-cell placement problems. We also demonstrate how simulated annealing (and simulated sintering) using the extended goal directed schedule can be applied to macro/custom placement.

6.1 The Big Picture

An integrated circuit (IC) design can be expressed in terms of behavioral, structural and physical properties [23].

A *behavioral representation* describes an IC's function.

A *structural representation* describes the components of an IC and component interconnections. Structural representations say nothing about the functionality of the IC other than what can be inferred from the behavior of specific components. We refer to IC components as *cells* and their interconnections are specified by *nets*. A structural representation consisting of cells and nets is called a *net-list*.

A *physical representation* contains information used in the fabrication of an IC. The lowest level of physical representation contains photo-mask information. A more abstract physical representation can be maintained provided it can easily be translated to the photo mask format. An example of such a representation would be one that specifies the physical location of cells, the physical realization of cells and the physical realization of nets.

The problem of determining the physical location of cells is called *placement* [23]. The problem of determining the physical realization of nets is called *routing* [23].

As indicated in Chapter 1, the secondary motivation for this dissertation is to gain experience in the physical design of integrated circuits. To this end, we examine two different forms of placement namely, standard-cell placement and macro/custom placement. What follows in this chapter is not an evaluation of the extended goal-directed schedule. Rather, it is an illustration of the extended goal-directed schedule applied to the problems of standard-cell placement and macro/custom placement.

6.2 Standard Cell Physical Design

The phases of standard-cell physical design are shown in Figure 6.1. The atomic components of a standard-cell physical design are *standard cells*. Standard-cells are pre designed and implement well defined logic functions. Usually, standard cells are of

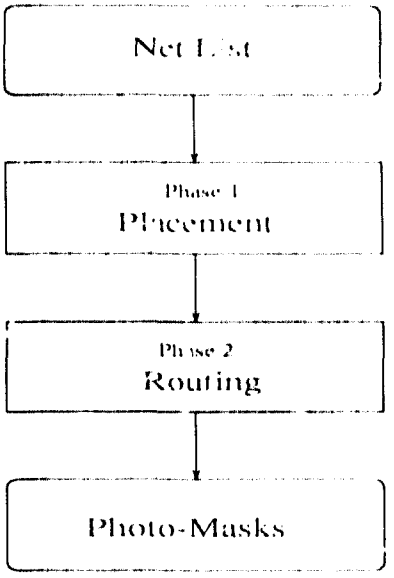


Figure 6.1: Phases of Standard Cell Physical Design

uniform height with varying width. Standard cells have uniform heights to facilitate placing them in horizontal rows.

Figure 6.2 shows a particular skeleton standard cell layout consisting of four rows of standard cells. Input and output *terminals* are located on the periphery of each standard cell. Logically equivalent terminals are connected by wires which run between, over and around standard cell rows. A wire is the physical realization of a net.

Wires consist of horizontal and vertical segments. Wire segments are placed in tracks which run parallel and at right angles to standard cell rows. The space between two standard-cell rows is called a *channel*.

Terminals are of two types namely, logic terminals and power and ground terminals. Wires which carry logic input and output signals are connected to logic

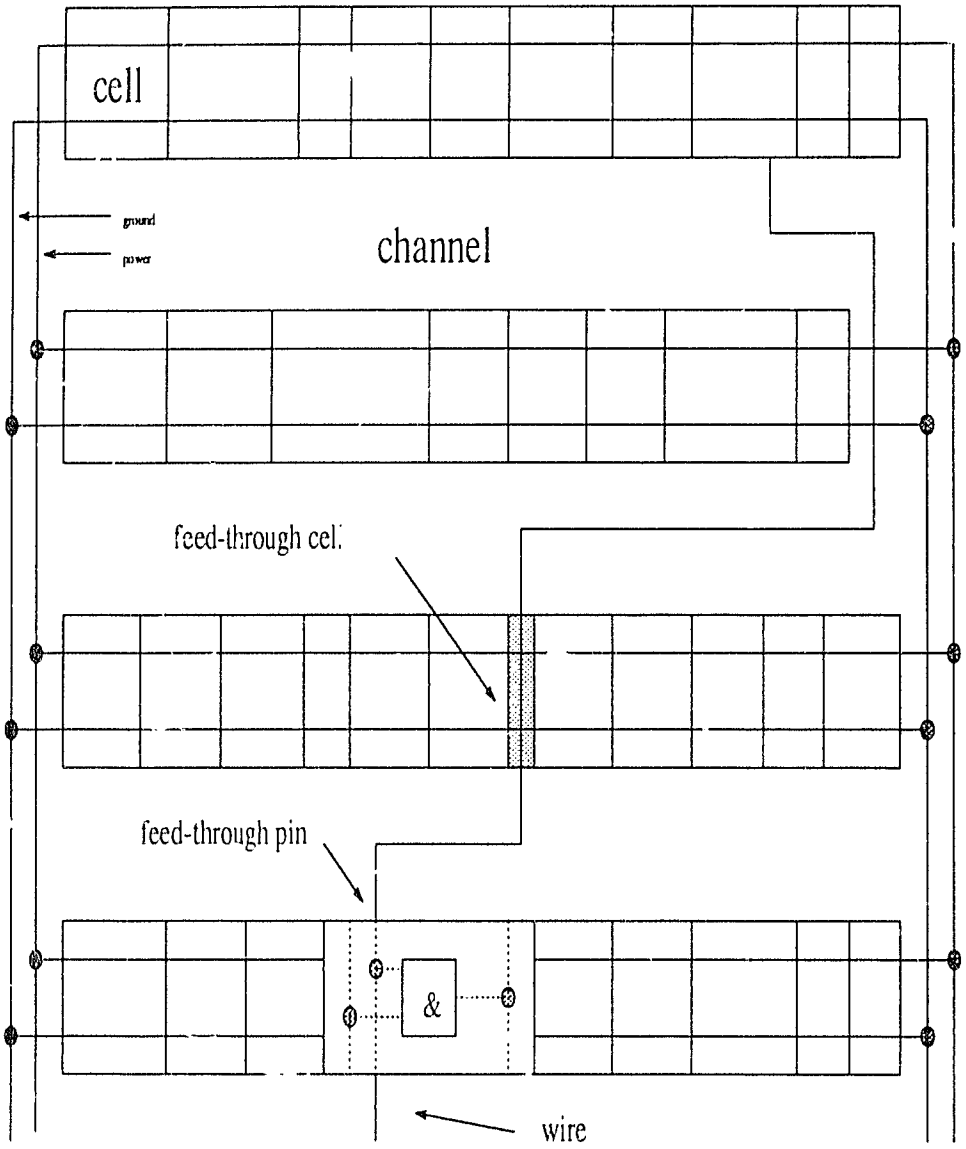


Figure 6.2: Skeleton Standard-Cell Layout.

terminals. Wires which carry power and ground signals are connected to power and ground terminals.

Logic terminals are located on the top of each standard cell. Typically the same logic terminals are also available on the bottom of the standard cell. Power and ground terminals are located on each standard cell such that when standard cells are abutted horizontally, power and ground rails are formed.

Vertical wire segments may be used to make connections that span two or more standard-cell rows. These segments come in the following four forms:

1. A wire segment which runs around the end of a row of standard cells.

2. A feed-through standard cell: A standard cell consisting solely of a single vertical wire segment.

3. An independent feed-through pin: A vertical wire segment which runs over a standard cell. The wire segment is not connected to any components within the standard cell.

4. A logic feed-through pin: A vertical wire segment which runs over a standard cell. The wire segment is connected to a logic terminal on the top of the standard cell and to a logically equivalent terminal on the bottom of the standard cell.

With reference to Figure 6.2, the single wire shown spans row 2 (counting rows from the top) using form 1 above. Form 2 is used to span row 3. Row four is spanned using a logic feed-through pin. For demonstration purposes, we have expanded the fourth standard cell (counting standard cells from the left) of the fourth row. The expanded standard cell is a two-input AND gate and all three of its logic terminals are available on the top and bottom of the standard cell.

Routing is used to perform the interconnection of terminals given a placement of standard cells in rows. This is achieved by assigning wires to tracks. Such an assignment must satisfy technology dependent design rules.

The number of tracks required to achieve a complete interconnection between standard cells is greatly influenced by the standard cell placement. In turn, this number impacts the total size of the design as tracks must be separated spatially as specified by the design rules.

The standard-cell placement problem involves arranging a collection of standard cells in horizontal rows such that the overall area (standard-cell area and routing area) is minimized. The standard-cell placement problem is reported in [28] to be NP hard.

6.2.1 Standard-Cell Placement Model

Placement and routing are two separate phases in standard-cell physical design. This is the case because the combined problem of placement and routing is considered too difficult a task to tackle in a single phase. It is not practical to route different placements in order to determine which placement is superior. This is due to the complexity of the routing problem. Consequently, some measure of the quality of a placement is required without resorting to routing.

An estimate of the total wire length can be used as a measure of placement quality. Figure 6.3 shows a wire connecting four standard cells. An estimate of the length of this wire is half the perimeter of the minimal bounding box which encloses the centres of the four standard cells. A common estimate of the total wire length is the sum of the estimated wire lengths for each wire in the design.

We model standard-cell placement as an extension of the square array placement

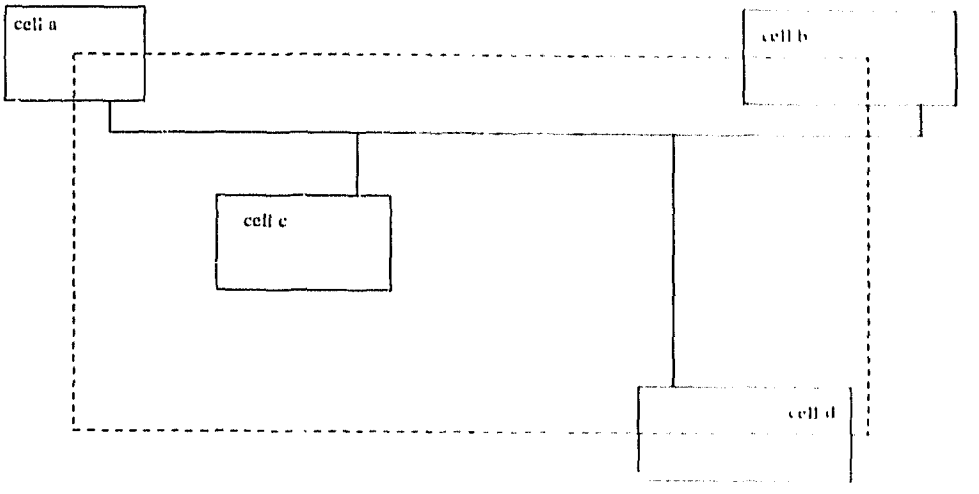


Figure 6.3: Wire Bounding Box.

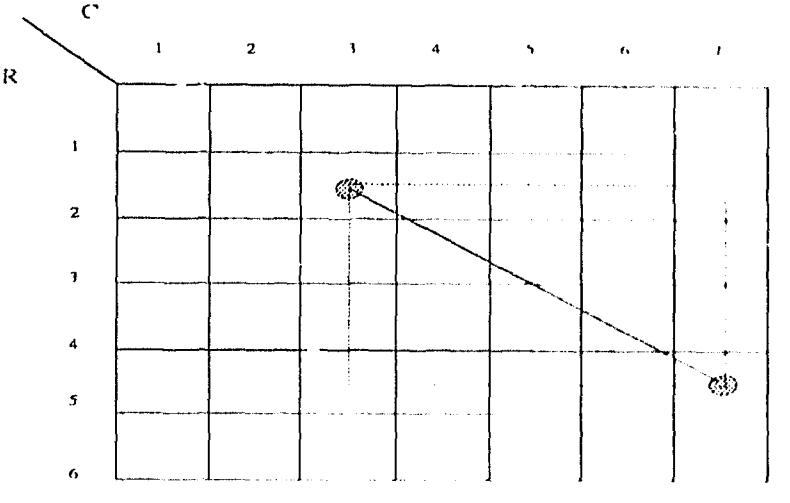


Figure 6.4: Placement Array.

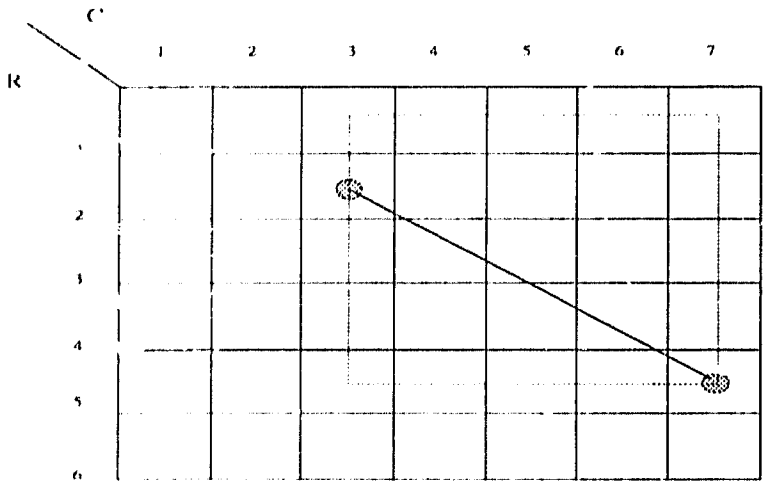


Figure 6.5: Placement Array.

(SAP) problem. We assume the existence of (a) an $R \times C$ array of standard cell locations (b) a collection of standard cells not exceeding $R \cdot C$ in number and (c) a net list. We also assume that logic inputs enter the design at right angles to the first cell row and logic outputs exit the design at right angles to the last cell row.[1]

A standard cell *configuration* is an assignment of standard cells to standard-cell locations. Each standard-cell location can accommodate a single standard cell. Two configurations are neighbours if one can be transformed to the other by swapping the positions of two standard cells. Each configuration has $\begin{pmatrix} R \cdot C \\ 2 \end{pmatrix}$ neighbours and the configuration space has $(R \cdot C)!$ configurations. A pseudo-random ordering of the integers 1 to N is generated.[2] Neighbour configurations within a neighbourhood are then selected in this order.

[1]Logic inputs and outputs that enter and exit the design from the left and right of the design can also be accommodated but are not considered.

[2]Note N is the number of configurations in a neighbourhood.

The cost of a configuration consists of two components. The first component is the total wire length as described previously. Wire length is expressed in terms of array units. For example, consider an array where $R = 6$ and $C = 7$. If a wire connects a standard cell located at (2,3) to a standard cell located at (5,7) then the length of this wire is estimated as half the perimeter of the minimal bounding box enclosing the two points which is $3 + 4 = 7$ (Fig. 6.4). If the wire is a primary input to the design then the bounding box is extended vertically to the top of the array (Fig. 6.5). This increases the estimated wire length from 7 to 8. The extra cost incurred accounts for routing the wire to the top of the design. If the wire is a primary output from the design then the bounding box is extended vertically to the bottom of the array. The extra cost incurred accounts for routing the wire to the bottom of the design.

The second component in the cost of a configuration is a penalty for creating standard-cell rows of unequal length. Each standard cell in the library is assigned a length-cost for the purpose of determining the length of a row. With reference to Figure 6.2, the second row is shorter that the other three rows resulting in some unused space called white-space. We account for white-space in the cost of a configuration by adding the following quantity to the overall cost:

$$w = R \cdot (L_{max} - L_{min})$$

where L_{max} and L_{min} are the lengths of the longest and shortest rows in the placement.

Typically, a large number of configurations are examined during SA. With regard to execution time, the cost of examining a configuration is usually dominated by the time taken to evaluate the objective cost function. Therefore, a cost function which can be evaluated incrementally is highly desirable since it usually requires considerably less time to evaluate.

It is possible to evalnate our cost function for standard-cell placement incremen

tally. To this end, the estimated length of each wire is maintained separately. When the positions of two standard cells are swapped, only the wires connected to these standard cells have their lengths re-estimated. For each of the affected wires, the change in estimated wire length is added to the estimate of the total wire length.

The length of each row is also maintained separately. If standard cells are rearranged in the same row then no adjustment to row lengths is required. If a swap affects two rows then for each of the affected rows, the difference in widths of the standard cell leaving the row and the standard cell entering the row is added to the row's length. L_{max} and L_{min} are then updated.

6.2.2 Optimization

Our intent is to apply SS using the extended goal-directed schedule to the problem of standard cell placement. Consequently, we require an estimate of the cost of an optimal configuration to an instance of the standard-cell placement problem and a procedure capable of producing a good initial configuration.

We had hoped to use a general purpose procedure such as Sastry's [26] statistical estimation technique for estimating the cost of an optimal configuration to a problem instance. Our attempts at implementing Sastry's technique did not result in a reliable estimation procedure. More will be said on this subject in Chapter 7. For the present, we use a problem-specific heuristic to estimate a lower bound on the cost of an optimal configuration to a standard-cell placement problem instance. This heuristic is based on the ideal that, in an optimal configuration, any two interconnected cells will either be placed adjacent to each other in the same row or immediately above or below each other on two different rows. We estimate the optimal length of a wire to be

$$l = (\text{the number of standard cells connected by the wire} + 1)\ div\ 2$$

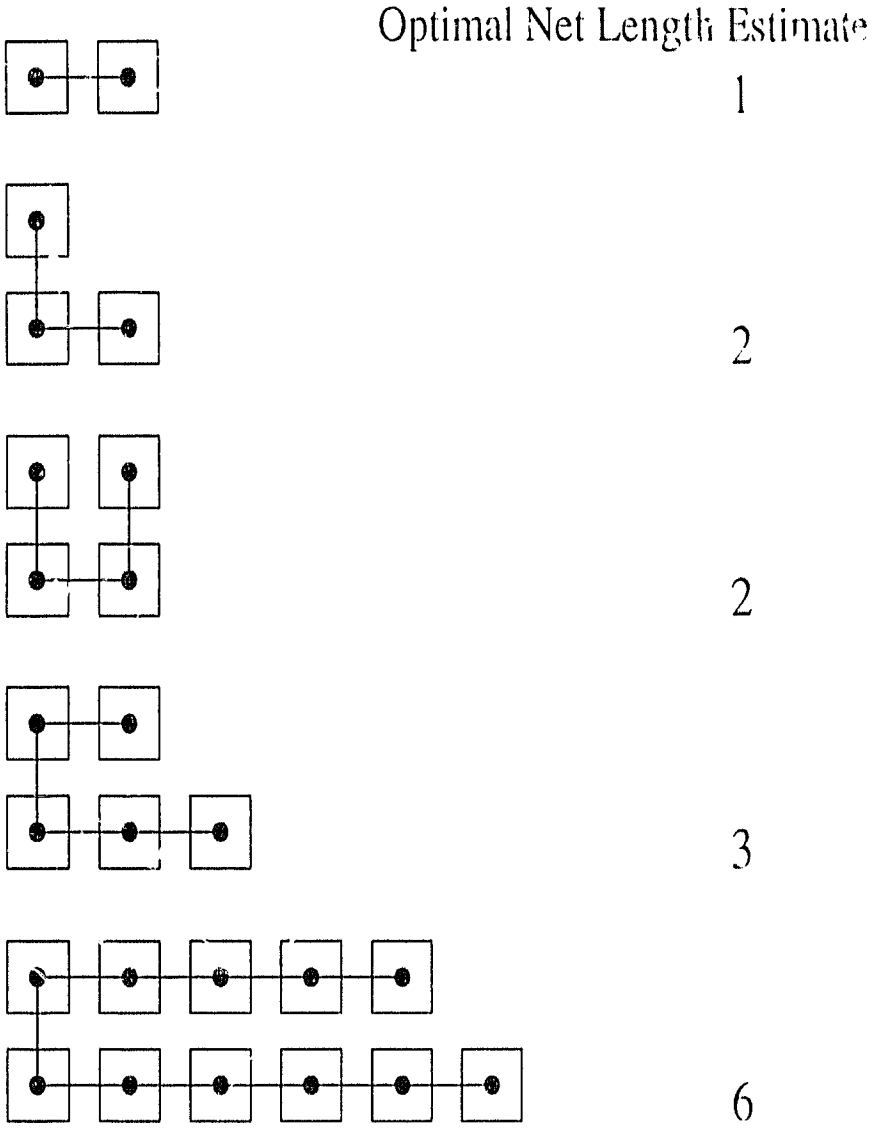


Figure 6.6: Optimal Net Length Estimation.

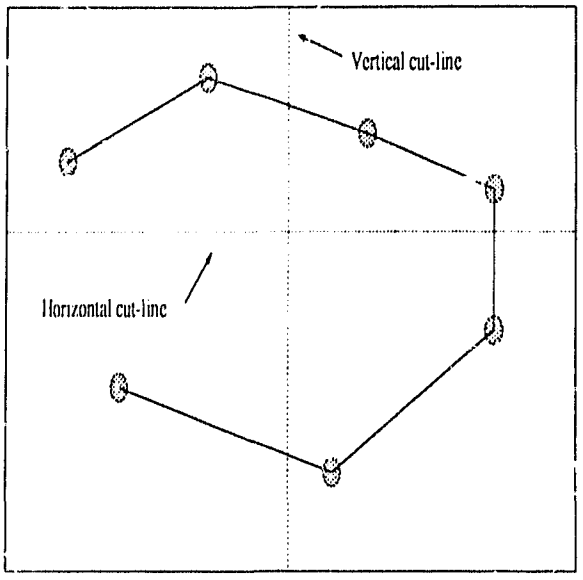


Figure 6.7: Quadrisection (Step 1).

Figure 6.6 shows pictorially how we envisage an optimal placement for nets with 1, 2, 3, 4, 5 and 11 cells respectively.

We also assume that an optimal configuration contains no white-space. Consequently, we use the sum of the estimated wire lengths as the cost of an optimal configuration.

6.2.3 Experimental Results

We have integrated our SA and SS procedures into the Open Architecture Silicon Implementation Software System (Oasis) [2]. Oasis is a set of integrated design tools including tools for logic synthesis and placement and routing. Oasis takes as input a function specification and produces a standard-cell design. The standard-cell placement procedure employed by Oasis is based on the quadrisection method [31].

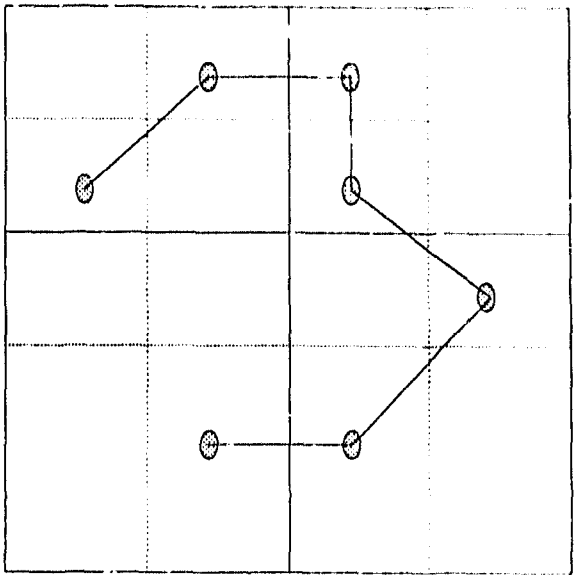


Figure 6.8: Quadrisection (Step 2).

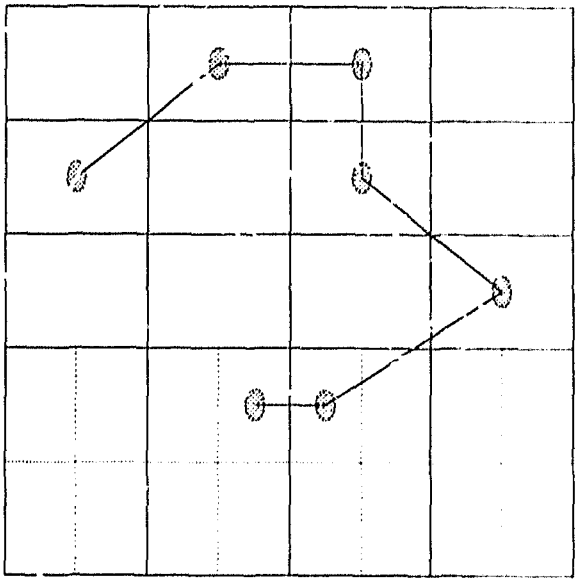


Figure 6.9: Quadrisection (Step 3).

Suppose that a design is to contain 5 rows of standard-cells. With quadrisection, the design is divided into four quadrants using horizontal and vertical cut-lines (Fig. 6.7). Cells are placed in quadrants in an attempt to minimize the number of wires that cross a cut line. Each quadrant is further divided into four until each quadrant can no longer be divided using a horizontal cut-line (Fig. 6.8 and Fig. 6.9). At this stage the standard cells have been assigned to rows, and the individual locations of standard cells within a row remains to be determined. This last step is performed using a min cut [31] approach.

We have compared Oasis's placement procedure to that of SA using Huang's schedule and SS using the extended goal-directed schedule. In this comparison, 8 example circuits were used as benchmarks. Both the SA and SS implementations used the extended SAP placement model as described above. Where SS was concerned, we used both SA using Huang's schedule and Oasis's placement procedure to provide good initial configurations. The example circuits used in the comparison were

- Circuit 1 [28] (circuit contains only 2-input gates).
- Circuit 2 Circuit 1 after inverter minimization [34].
- Circuit 3 Risc processor (circuit contains up to 8-input gates).
- Circuit 4 Realization of misex1 [1] using decomposition [19] (2-input gates).
- Circuit 5 Realization of divide-by-7 using decomposition, re-composition and inverter minimization [34] (up to 8-input gates).
- Circuit 6 C432 [6] (up to 4 input gates).
- Circuit 7 Realization of bw [1] (2-input gates).
- Circuit 8 Circuit 5 after an alternative decomposition (2-input gates).

Design	Cells	Placement Cost				CPU Time (minutes)			
		Oasis	SA-II	SS Q	SS II	Oasis	SA II	SS Q	SS II
1	16	44	35	33	33	< 1.0	0.05	1.06	1.11
2	15	39	33	32	32	< 1.0	0.05	1.20	1.18
3	86	461	285	288	285	< 1.0	11.7	130	81
4	89	386	238	232	232	< 1.0	19	120	83.7
5	112	603	341	334	333	< 1.0	25	285	204
6	171	1265	683	677	679	< 1.0	128	481	433
7	237	1361	637	631	637	< 1.0	571	2337	1175
8	256	14048	724	702	699	< 1.0	241	2152	1538

Key	Method
Oasis	Quadrisection
SA-II	SA using Huang's schedule
SS-Q	SS using the e-goal schedule and quadrisection as heuristic
SS-II	SS using the e-goal schedule and SA using Huang's schedule as heuristic

Table 6.1: Standard Cell Results (Placement Cost).

Each of the placement procedures was applied once to each of the 8 example circuits. Table 6.1 shows the terminal costs in terms of total estimated wire length plus the penalty for unequal rows. For brevity, we will refer to this sum as the *placement cost*. With regard to SA and SS, we used R and C values supplied by Oasis's placement phase.

From Table 6.1, it can be seen that SA using Huang's schedule and SS using the extended goal-directed schedule produced designs with shorter wire length compared to the designs produced using quadrisection. In our comparison, wire length has the meaning set out above. However, it should be noted that the quadrisection procedure

Design	Cells	Area			
		Oasis	SA-H	SS-Q	SS-H
1	16	36000	34000	35200	35200
2	15	37600	33672	31096	32016
3	86	288288	261048	257521	261048
4	89	270272	238896	240160	239528
5	112	388756	369360	361816	354240
6	171	669768	559896	547128	606144
7	237	801568	708032	710400	708032
8	256	849020	783840	769295	780432

Key	Method
Oasis	Quadrisection
SA-H	SA using Huang's schedule
SS-Q	SS using the e-goal schedule and quadrisection as heuristic
SS-H	SS using the e-goal schedule and SA using Huang's schedule as heuristic

Table 6.2: Standard-Cell Results (Area).

performs optimization using a different definition of wire length. Again with respect to wire length, SS performed marginally better than SA but at a cost of up to a factor of five in execution time.

We next proceeded to route these designs using Oasis's router. Our intent here was to determine how a reduction in wire length affected the total area of a design. The results are shown in Table 6.2.[3] Area is reported in Magic [21] square grid units. In our area calculation, we did not include the area required by power and ground

[3]We do not give precise CPU times for Oasis. For the purpose of our comparison, it suffices to say that Oasis requires CPU times of less than one minute.

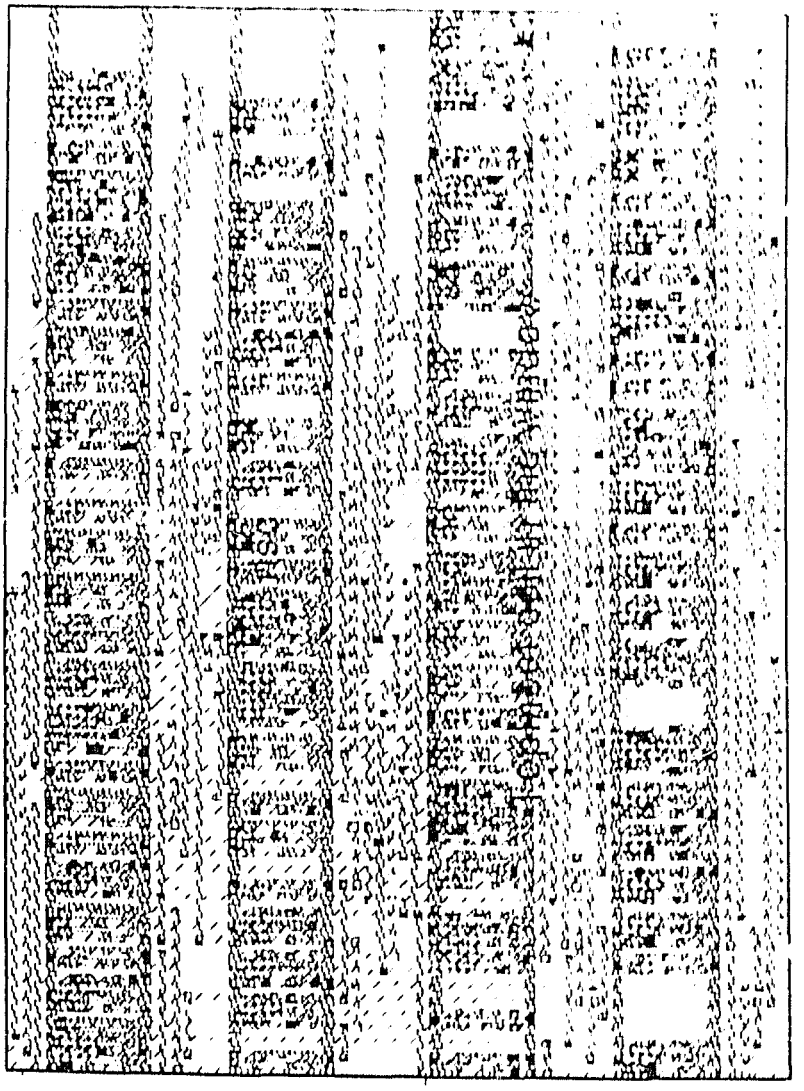


Figure 6.10: Circuit 3 – Physical Representation

rails *i.e.*, the rails at the periphery of the design as depicted in Figure 6.2. Figure 6.10 shows an example design realization. We used a CMOS 2.0 micron technology [2].

Figure 6.11 shows a graph of area versus placement cost for our 8 example circuits. The data show a linear tendency. We have drawn a regression line to improve readability. This linear tendency is an indication that our cost function is a good predictor of placement quality.

The designs produced by SA and SS are on average 10% smaller than the designs produced using quadrisection.

The designs produced by SS using the extended goal-directed schedule are marginally smaller (less than 2% on average) than the designs produced by SA using Huang's schedule. We note that in some cases a design with a smaller wire length required more area than a design with a larger wire length. This only occurred when the difference in wire length was small and is caused by a combination of the heuristic nature of the router and inaccuracies in the placement model.

As to whether an increase in execution time of up to a factor of five is acceptable for such a modest decrease in area, there is no definite answer. The cost of IC area in dollars is a step function. Consequently, a modest reduction in area may result in a large dollar savings. For example, a small decrease in area might mean that a design can be fabricated as a single IC rather than as a pair of ICs.

6.3 Macro/Custom Physical Design

The phases of macro/custom physical design are shown in Figure 6.12. The atomic components of macro/custom physical design are *macro cells* and *custom cells*. Macro

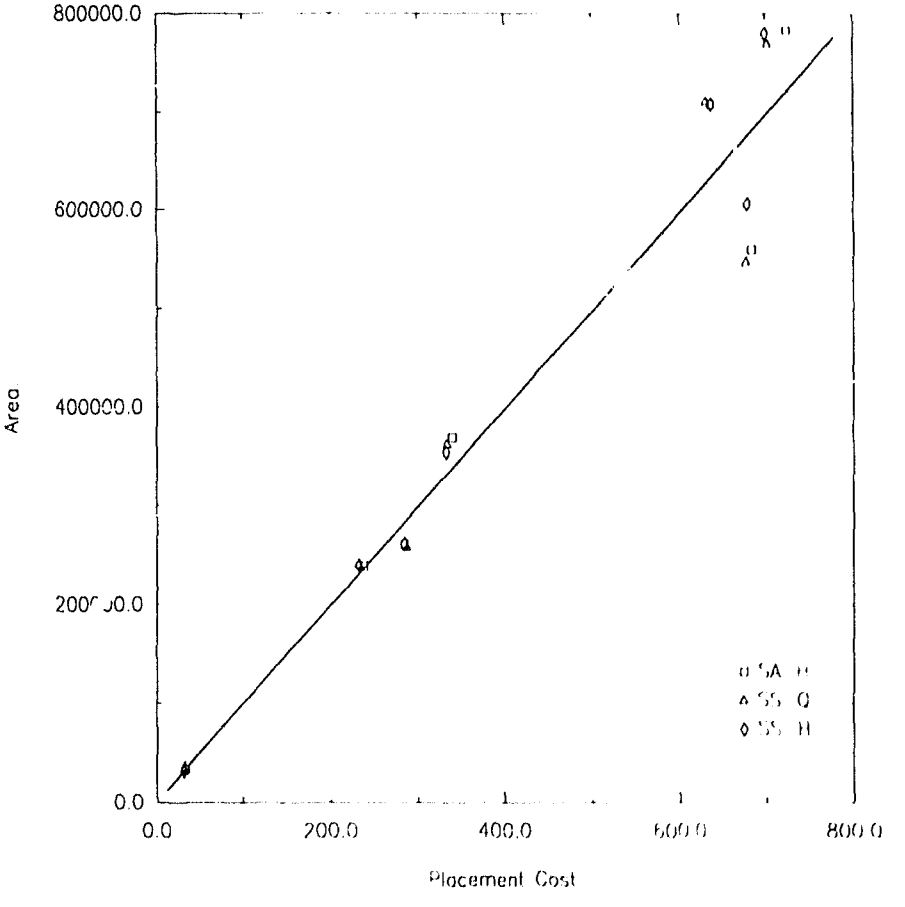


Figure 6.11: Placement Area versus Placement Cost.

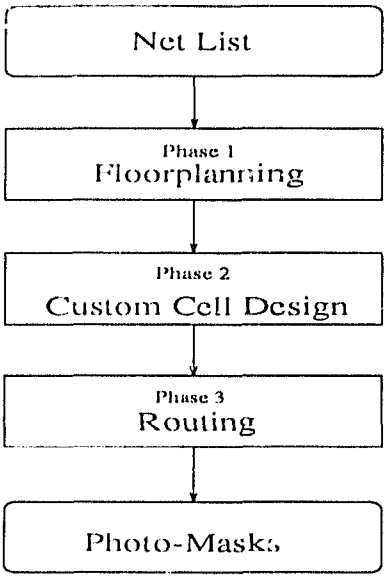


Figure 6.12: Phases of Macro/Custom Physical Design.

cells are pre-designed as in the case of standard cells. Macro cells and standard cells differ in that a macro cell may have several physical realizations. All realizations of a macro cell are logically equivalent. Macro cell realizations differ only in their dimensions and possibly the relative location of terminals.

Custom cells are designed specifically for a particular application. The detailed design of custom cells is performed after its physical location in the plane has been determined. Placement is performed using an estimate of custom-cell area and a range of acceptable height/width ratios called aspect ratios. For a given cell, we define aspect ratio to be the length of its shortest side divided by the length of its longest side, *i.e.*, aspect ratio is less than or equal to one.

The first phase of macro/custom physical design is concerned with the allocation of space for custom and macro cells. This process is called *floorplanning* and its

function is to determine the aspect ratio and position in the plane of each cell. For macro cells, the determination of aspect ratio corresponds to selecting a macro cell realization and its orientation.

After floorplanning, each custom cell is designed to conform to the previously determined aspect ratio and terminals are placed on the periphery of the cell. Subsequently, a router is employed to perform the interconnection of terminals.

Figure 6.13 shows a skeleton macro/custom layout. We restrict our attention to designs with rectangular shaped cells. Wire segments are again placed in tracks which may run on all four sides of a cell. Typically, wires can not be routed on top of custom or macro cells.

The problem of *floorplanning* involves arranging a collection of macro and custom cells in the plane such that the overall area (cell area and routing area) is minimized.

6.3.1 Slicing Floorplans and Slicing Trees

A floorplan for n cells consists of a bounding rectangle divided into n or more non overlapping rectangles. Cells and rectangles are labeled 1 to n and, rectangle i must be large enough to accommodate cell i ($1 \leq i \leq n$).

A *slicing* floorplan is arrived at by recursively bisecting a rectangle into two non overlapping rectangles. A binary slicing tree is a hierarchical representation of a slicing floorplan [37]. We restrict our attention to slicing floorplans and we only consider designs consisting of macro cells.

Figure 6.14 (a) shows a slicing floorplan which has been bisected into two non overlapping rectangles using a *horizontal cut-line*. Figure 6.14 (b) shows its slicing tree representation. Figure 6.14 (c) shows a slicing floorplan which has been bisected

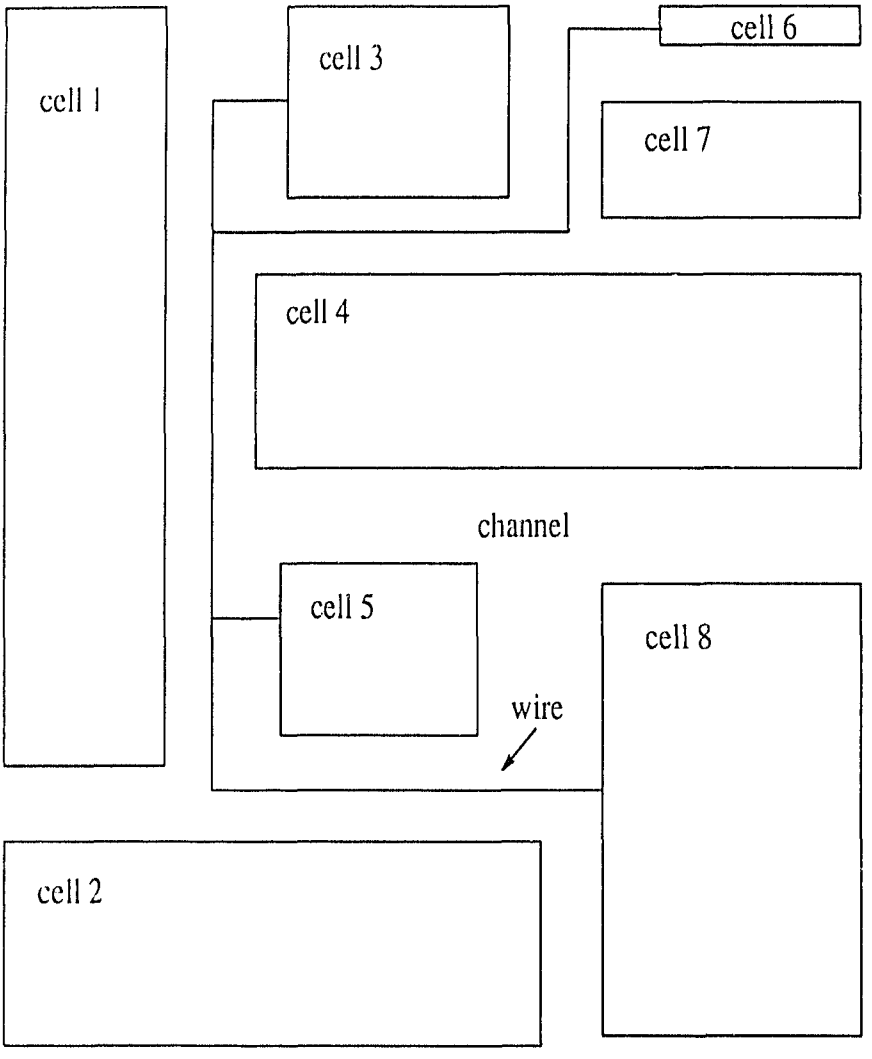


Figure 6.13: Skeleton Macro/Custom Layout.

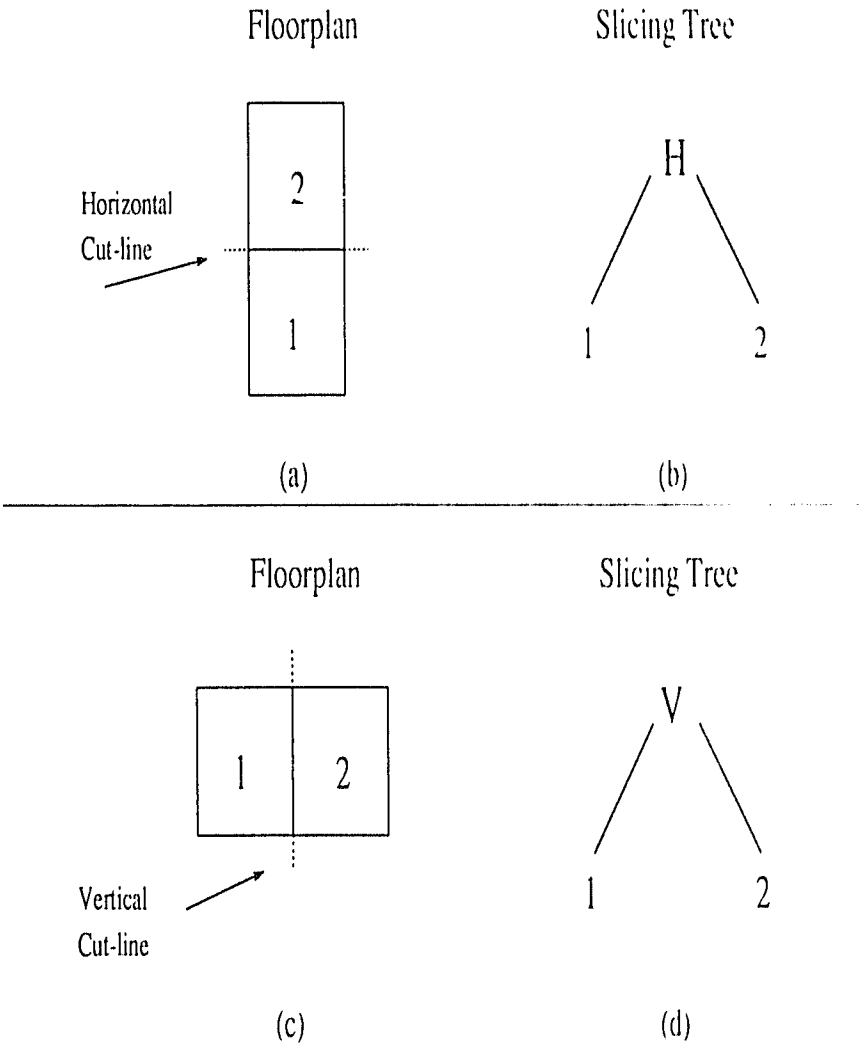


Figure 6.14: Floorplan and Slicing Tree Representation

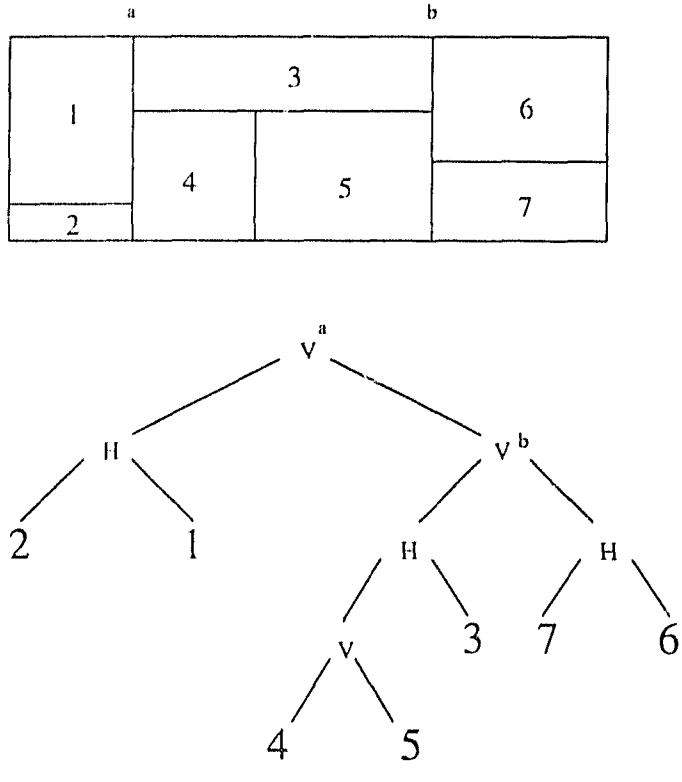


Figure 6.15: Floorplan and Slicing Tree Representation.

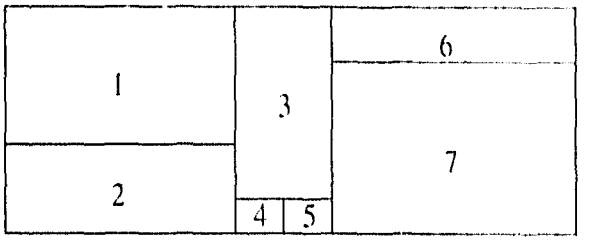


Figure 6.16: Alternative Slicing Floorplan.

into two non-overlapping rectangles using a *vertical cut-line*. Figure 6.11 (d) shows its slicing tree representation.

In general, each leaf node of a slicing tree represents a bounding rectangle and each non-leaf node represents a horizontal or vertical cut-line. Horizontal and vertical cut-lines in a slicing tree are represented by the symbols **H** and **V** respectively. The corresponding horizontal and vertical cut-lines in a slicing floorplan are shown as line segments.

Figure 6.15 shows a slicing floorplan and its slicing tree representation. Note that a slicing tree contains no dimensional information. In fact, a slicing tree may represent more than one slicing floorplan. For example, the slicing tree in Figure 6.15 is also a representation of the slicing floorplan in Figure 6.16.

In addition, a slicing floorplan may have more than one slicing tree representation. For example, the slicing trees in Figures 6.15 and 6.17 both represent the slicing floorplans of Figures 6.15 and 6.16.

Wong [37] defines an equivalence relation E on the set of all slicing floorplans with n rectangles. Let f and g be two slicing floorplans with n rectangles. fEg is true if f and g have the same slicing tree representation. E partitions the set of slicing floorplans with n rectangles into equivalence classes. Each equivalence class is called

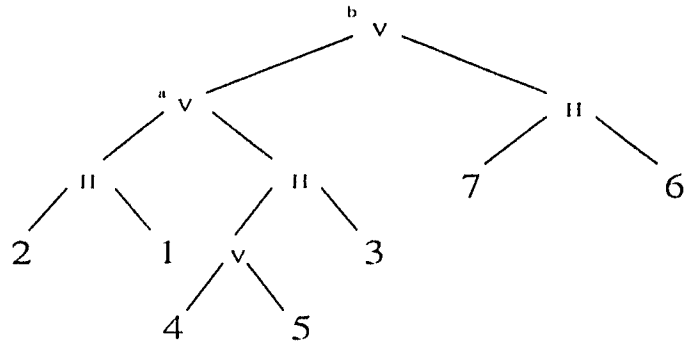


Figure 6.17: Alternative Slicing Tree Representation.

a *slicing structure*.

Our intent is to use a slicing tree with n leaf nodes to represent a slicing structures with n rectangles. As we have seen above, there is a many-to-1 relationship between slicing trees with n leaf nodes and a slicing structure with n rectangles. This is undesirable from an optimization point of view since any increase in the number of feasible solutions makes the search for an optimal solution more difficult.

We desire a 1-to-1 relationship between a slicing tree with n leaf nodes and a slicing structure with n rectangles. To this end, consider a slicing tree where the cut-line at a non leaf node is the same as the cut-line at one of its non-leaf children. Implicit in such a structure is a cut-line ordering. Consider the slicing tree of Figure 6.15. The vertical cut-line with superscript a precedes the vertical cut-line with superscript b because of their relative positions in the slicing tree. With regard to the slicing structure, this ordering imposes a left to right order in the corresponding vertical cut-lines. For clarity, we have labeled the vertical cut-lines in the slicing structure of Figure 6.15 with the superscripts of their corresponding cut-lines in the slicing tree of Figure 6.15.

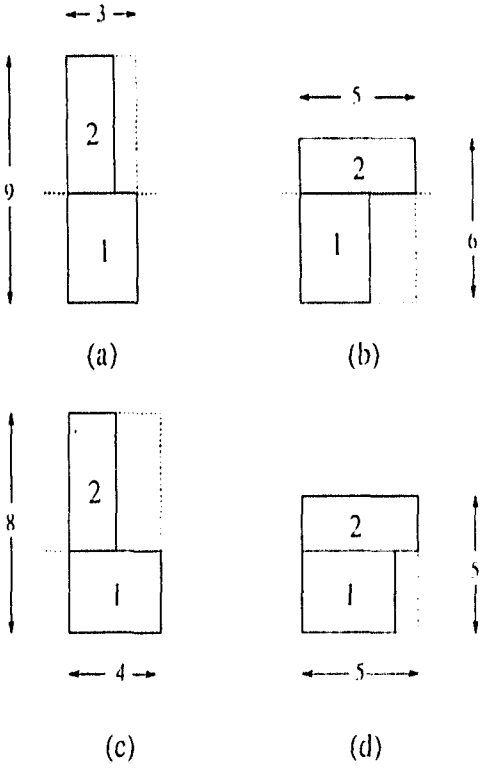


Figure 6.18: Floorplan with Cells Embedded.

A *skewed* slicing tree is a slicing tree in which the cut lines at each non leaf node and their respective non-leaf right children are different.

From Wong [37], we know there is a 1-to-1 correspondence between slicing structures and skewed slicing trees. Consequently, in our floorplanning procedures, we represent slicing floorplans as skewed binary slicing trees. Figure 6.17 shows the skewed slicing tree representation for the slicing floorplan in Figure 6.15.

Our next step is to add dimensional information to a skewed slicing tree. Consider the situation where we have two cells. Let cell 1 have dimensions (x_1, y_1) where x_1 denotes the width of the cell and y_1 denotes the height of the cell. Let cell 2 have

dimensions (x_2, y_2). We define two binary operators h and v such that the dimensions of the smallest rectangle enclosing cell 1 and cell 2 are: (MAX(x, y) returns the greater of x and y and SUM(x, y) returns the arithmetic sum of x and y.)

$$(x_1, x_2)h(y_1, y_2) = (X, Y) \; : \; X = \text{MAX}(x_1, x_2) \; ; \; Y = \text{SUM}(y_1, y_2)$$

$$(x_1, x_2)v(y_1, y_2) = (X, Y) \; : \; X = \text{SUM}(x_1, x_2) \; ; \; Y = \text{MAX}(y_1, y_2)$$

The operators h and v correspond to horizontal and vertical cut-lines in a slicing structure.

Consider again the floorplan and slicing tree of Figure 6.14 (a) and (b). Let cell 1 and cell 2 have dimensions (3,4) and (2,5) respectively. From the floorplan, we know that cell 2 is to be positioned on top of cell 1. The resulting dimensions of the bounding rectangle enclosing cell 1 and cell 2 are:

$$(X, Y) = (3,4)h(2,5) = (3,9)$$

Let cell 1 have two alternative dimensions namely, (3,4) and (4,3). Let cell 2 also have two alternative dimensions namely, (2,5) and (5,2). The alternative dimensions of the the bounding rectangle enclosing cell 1 and cell 2 are (see Fig. 6.18):

$$(X_1, Y_1) = (3,4)h(2,5) = (3,9)$$

$$(X_2, Y_2) = (3,4)h(5,2) = (5,6)$$

$$(X_3, Y_3) = (4,3)h(2,5) = (4,8)$$

$$(X_4, Y_4) = (4,3)h(5,2) = (5,5)$$

With reference to Figure 6.18, rectangles which are not labeled represent wasted space called white-space.

6.3.2 Slicing Tree Evaluation

A non-leaf node of a slicing tree is *evaluated* by determining the unique alternative dimensions of the rectangle enclosing the cells on its right branch and on its left branch. In the previous paragraphs, we have described how the alternative dimensions of the bounding rectangle are calculated. Slicing tree nodes are evaluated in postorder. The evaluation of the root node yields the alternative dimensions of the bounding rectangle for the slicing floorplan.

Figure 6.19 shows a slicing floorplan and its skewed slicing tree representation. Adjacent to each leaf node in the slicing tree, we show alternative cell dimensions. Adjacent to each non-leaf node, we show the alternative dimensions of the node's bounding rectangle as determined by node evaluation.

It should be noted that our method of node evaluation is simple yet instructive. This is the case since all possible alternative bounding rectangle dimensions are calculated at **each node**. A bounding rectangle with dimensions (x_1, y_1) is discarded if there exists an alternative rectangle with dimensions (x_2, y_2) and

$$(x_1 = x_2, \quad \text{and} \quad y_1 \geq y_2) \quad \text{or} \quad (y_1 = y_2, \quad \text{and} \quad x_1 > x_2)$$

Wong [37] describes a more efficient procedure based on *bounding curves*. The bounding curves approach was first introduced by Stockmeyer [30]. At each node, Stockmeyer considers only relevant bounding rectangles. In other words, the bounding rectangles that are discarded in our naive approach are not considered at all by Stockmeyer. The end result is the same for both our algorithm and Stockmeyer's algorithm. However, in terms of execution time, it would be worth converting to Stockmeyer's scheme for a production system.

Let the process of selecting one of the alternative dimensions of a floorplan's bounding rectangle be called *fixing the shape* of the root node. The next step is to

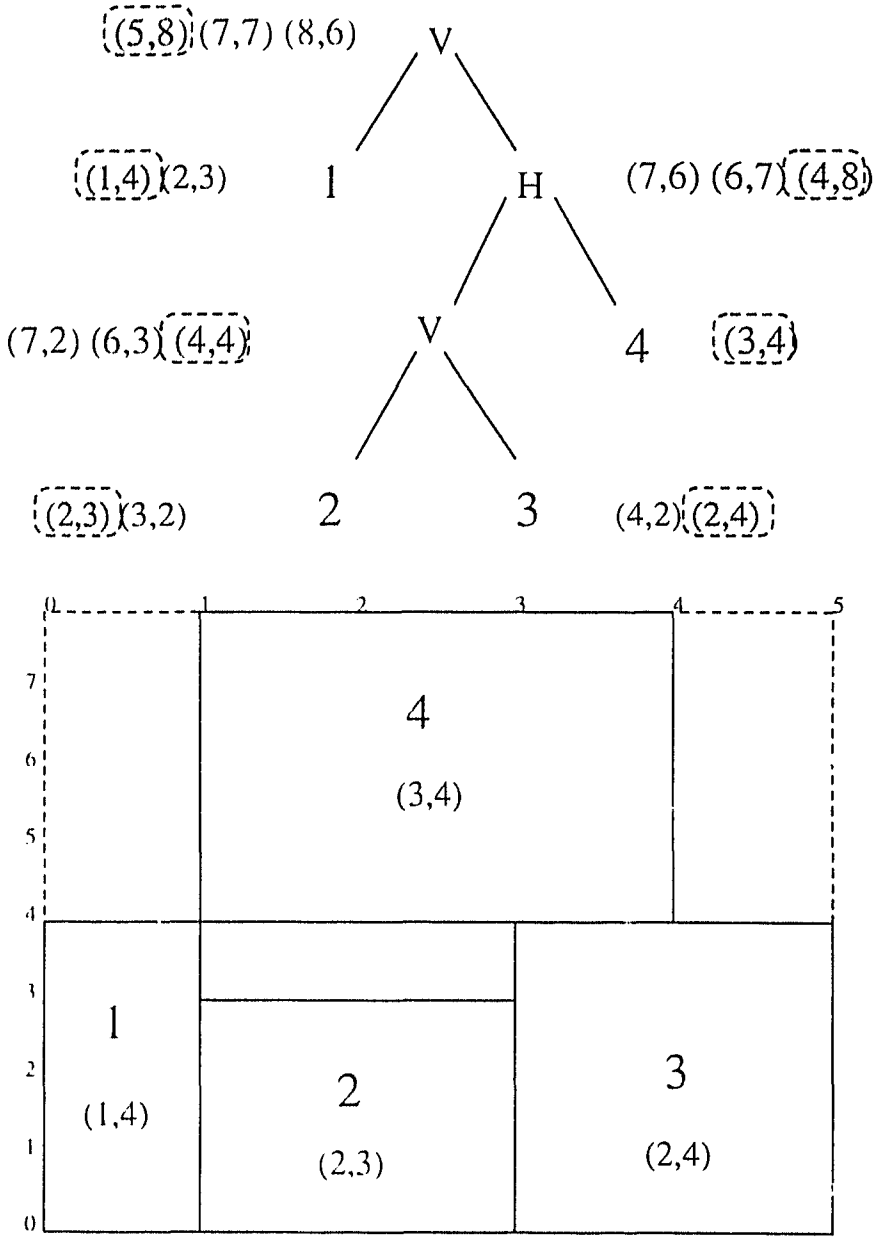


Figure 6.19: Slicing Tree Evaluation.

fix the shape of the root's left and right children subject to the constraints imposed by the root shape. Once the root's children have their shapes fixed, the process continues in preorder until the shape of each node is completely determined. During this traversal, the lower left corner coordinate of each shape is also determined. Given the shape of a node is fixed to (X, Y), the shape of its left and right children are fixed to (x_1, y_1) and (x_2, y_2) where:

Node is a vertical cut-line : $SUM(x_1, x_2) = X$; $MAX(y_1, y_2) = Y$

Node is a horizontal cut-line : $MAX(x_1, x_2) = X$; $SUM(y_1, y_2) = Y$

Let (M, N) be the lower left corner coordinates of the node with shape (X, Y). The lower-left-corner coordinates of the node's left and right children are (m_1, n_1) and (m_2, n_2) where:

Node is a horizontal cut-line : $m_1 = M$; $n_1 = N$; $m_2 = M$; $n_2 = SUM(N, y_1)$

Node is a vertical cut-line : $m_1 = M$; $n_1 = N$; $m_2 = SUM(M, x_1)$; $n_2 = N$

The slicing tree preorder traversal is initiated with the selected values for the shape of the root node and (0,0) for its lower left corner coordinates.

Consider again the skewed slicing tree in Figure 6.19. The shape of the root can be fixed in any one of three ways. Let the shape of the root be fixed to (5,8). With $(X = 5, Y = 8)$, the only acceptable values for $(x_1, y_1), (x_2, y_2)$ are (1,1) and (4,8) and their lower-left-corner coordinates are (0,0) and (1,0). In the skewed slicing tree shown in Figure 6.19, we have highlighted the fixed shape of each node. We also show the resulting slicing floorplan with cells embedded.

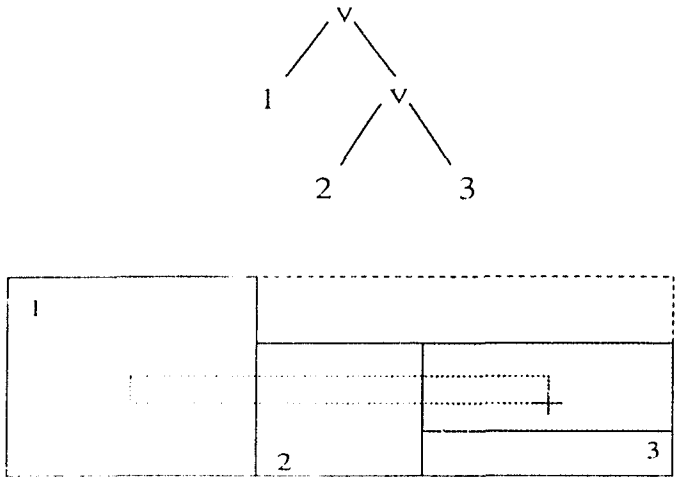


Figure 6.20: Wire Length Estimate.

6.3.3 Floorplan Model

A floorplanning *configuration* is a skewed slicing tree with dimensional information. Two configurations are neighbours if one can be transformed to the other by interchanging the branch-roots of two disjoint sub-trees or by changing the cut-line direction of a non-leaf node. By interchanging the branch-roots of two disjoint sub-trees we mean, take two branches that do not have a node in common and swap their position in the tree. For both types of transformations, the resulting tree must be a valid skewed slicing tree. A configuration has a maximum of $\begin{pmatrix} 2n-1 \\ 2 \end{pmatrix} + (n-1)$ possible neighbours. Some of these configurations are invalid neighbours since their associated slicing tree may not be a valid skewed slicing trees. Neighbour selection is performed in a pseudo-random order as in the case of the SAP problem (see Chapter 2).

We note that Wong represents a slicing floorplan as a *normalized Polish expression*. Neighbours are found by manipulating the polish expression rather than the

slicing tree itself. For a given configuration, Wong defines a completely different neighbourhood than the one described above. We do not claim that our neighbourhood is superior to Wong's neighbourhood. However, in our opinion, a neigbourhood defined on the slicing tree is more intuitive than a neighbourhood defined on the corresponding polish expression.

The cost of a configuration is the sum of two components. The first component is an estimate of total wire length. Wong [37] estimates the length of a wire as half the perimeter of the minimal bounding box which encloses the centers of the cells. Figure 6.20 shows a slicing floorplan with three cells and its skewed slicing tree representation. Cell 1 and cell 3 are connected by a wire. Cell 3 is free to move vertically into the space represented by the stipple pattern in Figure 6.20. Consequently, we estimate the length of a wire as half the perimeter of the minimal bounding box which encloses the centers of the bounding rectangles in which the embedded cells are free to move.

The second component of the cost of a configuration is a measure of the area occupied by the configuration's slicing floorplan. Wong [37] used the area of the slicing floorplan's bounding rectangle as his area metric. We take a slightly different approach and use the slicing floorplan's white-space area as our area metric. The rational for this is as follows.

Wong uses the sum of total area and weighted wire length as his configuration cost. No guidelines are given as to the magnitude of this weight. In our experience, superior floorplan quality results when the wire length component dominates the cost function. Consequently, we speculate that the weight should at least be greater than one. An appropriate weight is difficult to determine especially since estimated wire length is linear and area is quadratic.[4]

[4]Suppose that wire length and area are functions of n. Since they have different growth rates, a suitable weight for a particular value of n may not be suitable for larger values of n

By using white-space area rather than total area, we implicitly weight the estimated total wire length component of the cost of a configuration. We have found that our implicit weighting strategy causes the estimated total wire length component to dominate the cost function.

Let (X, Y) be the outer dimensions of a floorplan with n non-overlapping rectangles. From our perspective, a floorplan with outer dimensions (X, Y) can be transformed to a floorplan with outer dimensions (Y, X) by a simple rotation. Our calculation of white-space takes this into account.

Let Z be a given aspect ratio. We favour floorplans with aspect ratio between Z and 1. Floorplans with aspect ratio less than Z are penalized by increasing the value of white-space.

Let A be the sum of the area occupied by the floorplan's n cells. We calculate the value of white-space as follows:

$$ws = (X \cdot Y) - A \text{ if } X > Y \text{ and } \frac{Y}{X} \geq Z$$
$$ws = (X \cdot X \cdot Z) - A \text{ if } X > Y \text{ and } \frac{Y}{X} < Z$$
$$ws = (X \cdot Y) - A \text{ if } X \leq Y \text{ and } \frac{X}{Y} \geq Z$$
$$ws = (Y \cdot Y \cdot Z) - A \text{ if } X \leq Y \text{ and } \frac{X}{Y} < Z$$

For example, the floorplan in Figure 6.20 has three cells labeled 1, 2 and 3. Their dimensions are (9,9), (6,6) and (9,2) respectively. The total cell area is 135 and the outer dimensions of the floorplan are (23,9). Let the aspect ratio be 0.9. The calculation for ws is as follows:

$$ws = (23 \cdot 23 \cdot 0.9) - 135 = 341.1$$

The overall cost of a configuration is the sum of white-space and the total estimated wire-length.

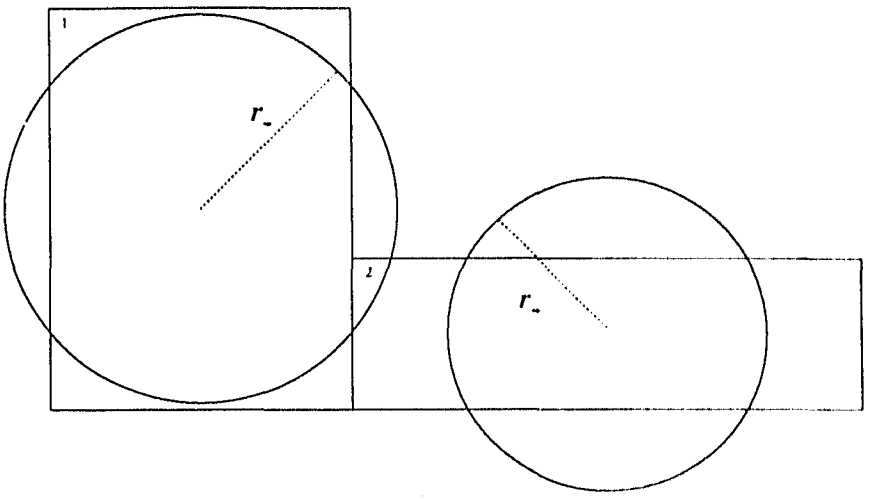


Figure 6.21: Wire Length Estimate.

Note that slicing tree evaluation can be performed incrementally. To this end, we maintain backward links in the slicing tree from each node to its parent. Given a configuration and one of its valid neighbours, the cost of the neighbour can be determined without resorting to a complete slicing tree evaluation. Recall that two configurations are neighbours if one can be transformed to the other by interchanging the branch-roots of two disjoint sub-trees or by changing the cut line direction of a non-leaf node. If a slicing tree node has its cut line direction changed, only the nodes encountered on the backward path from the changed node to the tree root need to be re-evaluated. If the branch-roots of two disjoint sub trees are swapped then only the nodes encountered on the backward paths from the parents of both branch roots to the tree root are re-evaluated.

6.3.4 Optimization

Our intent is to apply SA and SS using the extended goal-directed cooling schedule to the problem of floorplanning. For SS, we require an estimate of the cost of an optimal configuration to an instance of the floorplanning problem and a procedure capable of producing a good initial configuration. As in the case of standard-cell placement, we use a problem specific heuristic to estimate a lower-bound on the cost of an optimal configuration to a problem instance. This heuristic is based on the ideal that interconnected cells will be placed adjacent to each other and that the floorplan will have no white-space.

Figure 6.21 shows two cells. For each cell, we have drawn a circle of equal area such that both cell and circle share a common center. We estimate the length of the wire connecting these cells to be the sum of the radii of their associated circles. We do this so as not to bias our wire length estimation in favour of a particular cell orientation. The sum of the estimated wire lengths plus the sum of cell areas is used as the cost of an optimal configuration.

6.3.5 Experimental Results

The Microelectronics Center of North Carolina (MCNC) provides a set of floorplanning benchmark circuits. There circuits are not used extensively in the literature. Our benchmark need is one of illustration rather than evaluation. In particular, we required a problem where we knew the optimal configuration cost in advance. Consequently, we decided to construct our own benchmark problem.

We constructed an example floorplan by repeatedly bisecting a rectangle into 26 non overlapping rectangles. Into each non-overlapping rectangle we placed a cell such that the outer dimensions of the floorplan was 27 by 30 and the floorplan had

no white-space. We placed wires between arbitrary selected pairs of cells. We also weighted each wire. Wire weights are in the range 1 to 8 and were chosen at random. Figure 6.22 shows our example floorplan with cells embedded. Wires are represented by dotted lines. The distance between dots indicates a wire's weight. For example, the wire connecting cell 6 and cell 17 has weight 8 and the wire connecting cell 1 and cell 8 has weight 1.

The floorplan in Figure 6.22 has an area of $27 \cdot 30 = 810$ square units and a total wire length of 987.5 units. Its cost is 987.5. Note that our example floorplan can not be laid out in less than 810 square units. However, a smaller total wire length is possible at the cost of increasing the layout area.

We applied SA and SS using the extended goal-directed schedule on our 26 cell example floorplan using an aspect ratio of 0.25 (1:4). The results are shown in Table 6.3. Thirty trials were performed. Each trial started with a different initial configuration which was chosen at random.

In terms of average terminal configuration cost, both SA and SS using the extended goal-directed schedule performed better that Huang's schedule (Table 6.3).

In Chapter 4, we alluded to a problem with the convergence criteria used in Huang's schedule. This problem manifested itself twice in our thirty trials resulting in terminal configurations which were far from optimal. Figure 6.23 shows the less extreme of these two cases.

In Figures 6.24, 6.25 and 6.26, we show the terminal configurations with least cost for each of Huang's schedule and SA and SS using the extended goal directed schedule. Note that for simplicity, cells are positioned left and bottom justified within bounding rectangles. We envisage that the exact position of a cell within a bounding rectangle will be determined during the routing phase. Note also that each of the

Schedule	Cost				Configurations $*10^6$		CPU Time
	mean	*std*	x_{MAX}	x_{MIN}	*mean*	*std*	*mean* (sec.)
SA H	1165.8	591.2	4125.5	954	0.93	0.339	7857.2
SA E	1021.1	35.5	1080.5	939.5	1.22	0.24	11007.6
SS H	1006.2	40.0	1084	912	1.7	0.322	14612.1

Key	Method
SA H	SA using Huang's schedule
SA E	SA using the e goal schedule
SS H	SS using the e goal schedule and SA using Huang's schedule as heuristic

Table 6.3: Floorplanning Problem. (30 Trials.)

floorplans in Figures 6.3, 6.23, 6.24, 6.25 and 6.26 are drawn to the same scale.

Tables 6.4 and 6.5 show the appropriate statistics for comparing the statistical difference between the performance of Huang's schedule and the extended goal-directed schedule on our example floorplanning problem. From Tables 6.4 and 6.5, we deduce that in terms of the average terminal configuration cost, the performance difference between Huang's schedule and the extended goal-directed schedule on our 26 cell floorplanning problem is not statistically significant. This is **not** a negative result. In Chapter 7, we explain why the extended goal-directed schedule and Huang's schedule have a similar performance (terminal configuration quality) for this problem instance.

		SA-H		SA-E			
n_1	n_2	$\overline{y}_1$	std_1	$\overline{y}_2$	std_2	$(\overline{y}_1 - \overline{y}_2)$	$2\sigma_{(\overline{y}_1 - \overline{y}_2)}$
30	30	1021.1	35.5	1165.8	591.2	144.7	216.2

Key	Method
SA-H	SA using Huang's schedule
SA-E	SA using the e-goal schedule
SS-H	SS using the e-goal schedule and SA using Huang's schedule as heuristic

Table 6.4: Schedule Statistics: SA H and SA E.

		SA-E		SS-H			
n_1	n_2	$\overline{y}_1$	std_1	$\overline{y}_2$	std_2	$(\overline{y}_1 - \overline{y}_2)$	$2\sigma_{(\overline{y}_1 - \overline{y}_2)}$
30	30	1006.2	40.0	1021.1	35.5	14.9	19.5

Key	Method
SA-H	SA using Huang's schedule
SA-E	SA using the e-goal schedule
SS-H	SS using the e-goal schedule and SA using Huang's schedule as heuristic

Table 6.5: Schedule Statistics: SA-E and SS H.

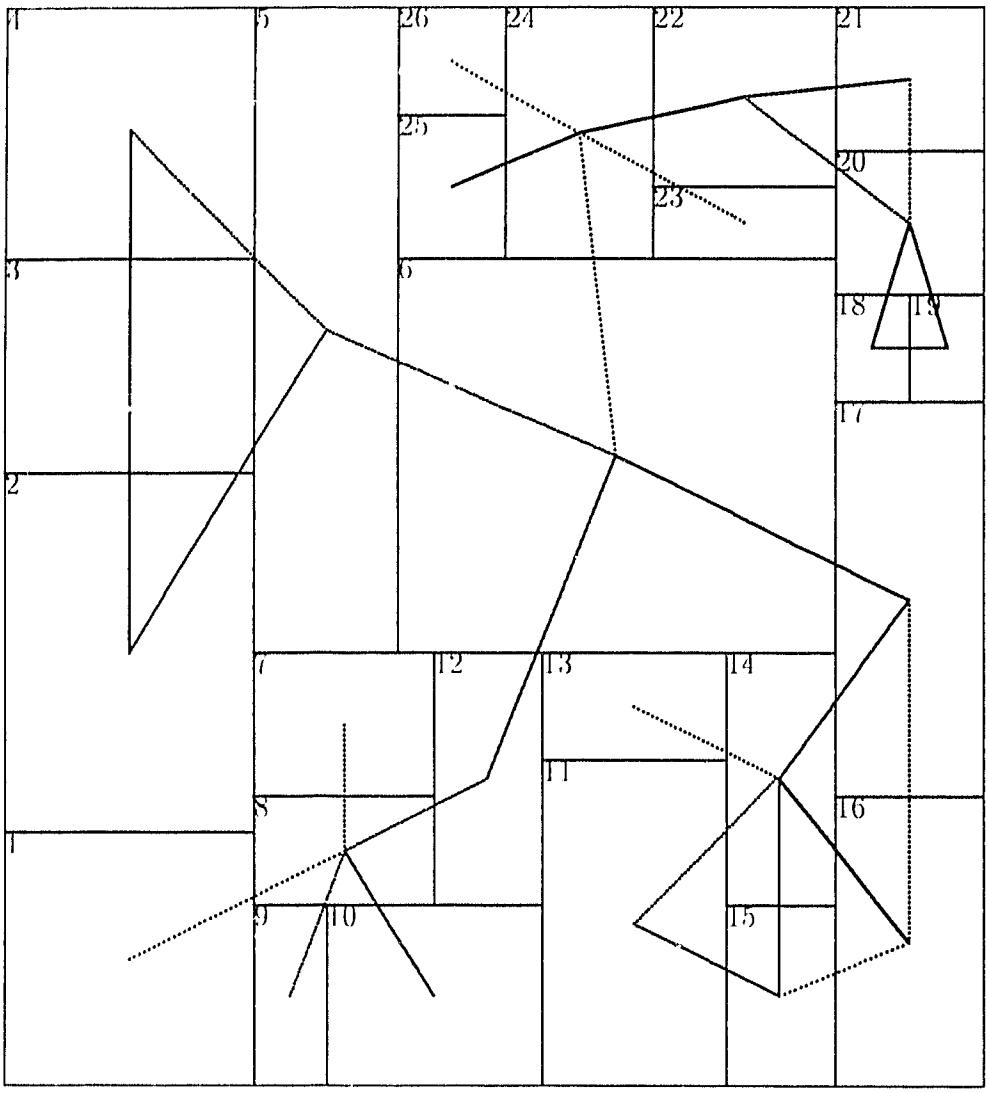


Figure 6.22: Example Floorplan: Cost=987.5 Area=810 Wire len.= 987.5

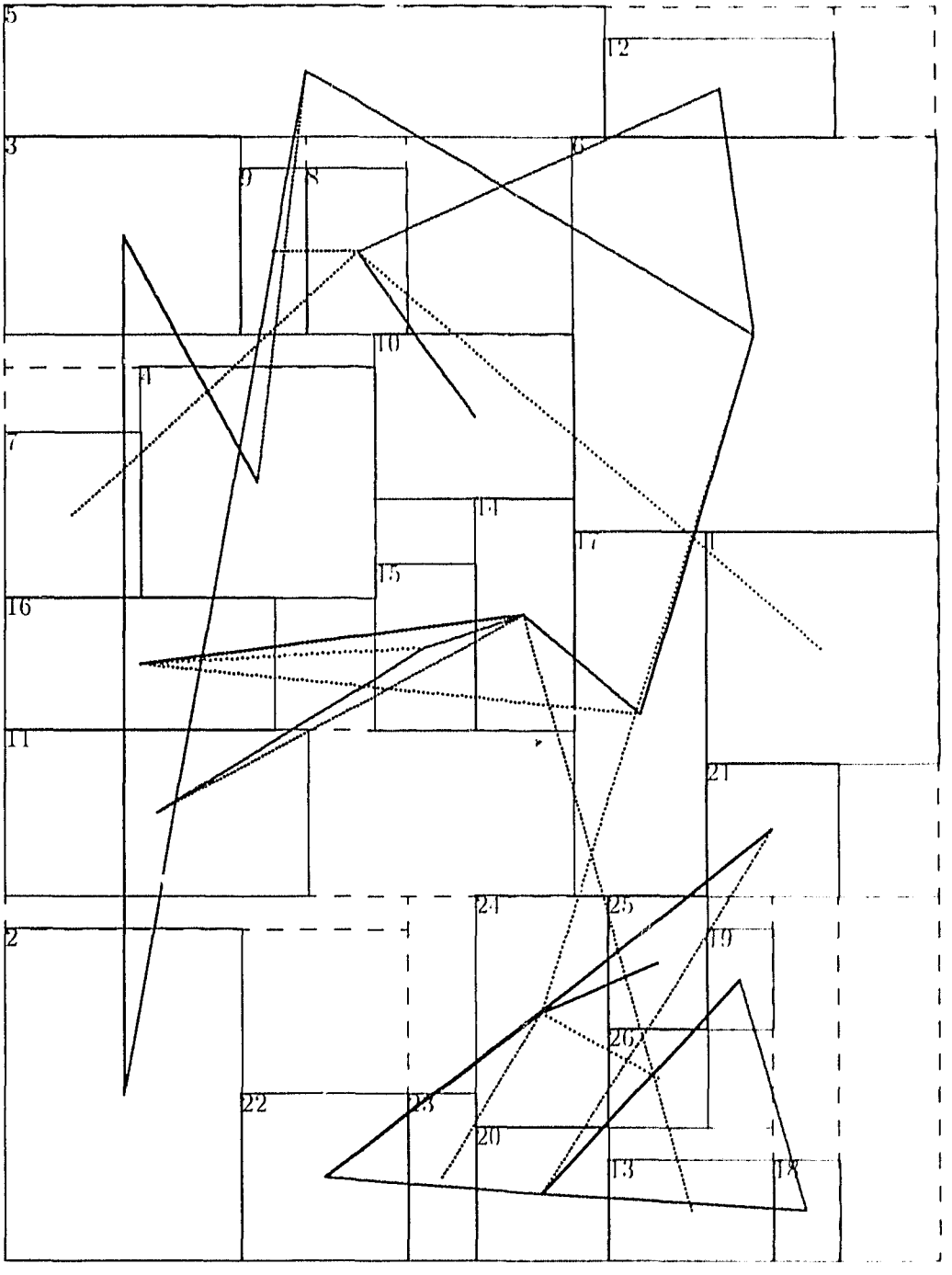


Figure 6.23: Premature Convergence: Cost=2057.5 Area=1064 Wire len.=1804.5

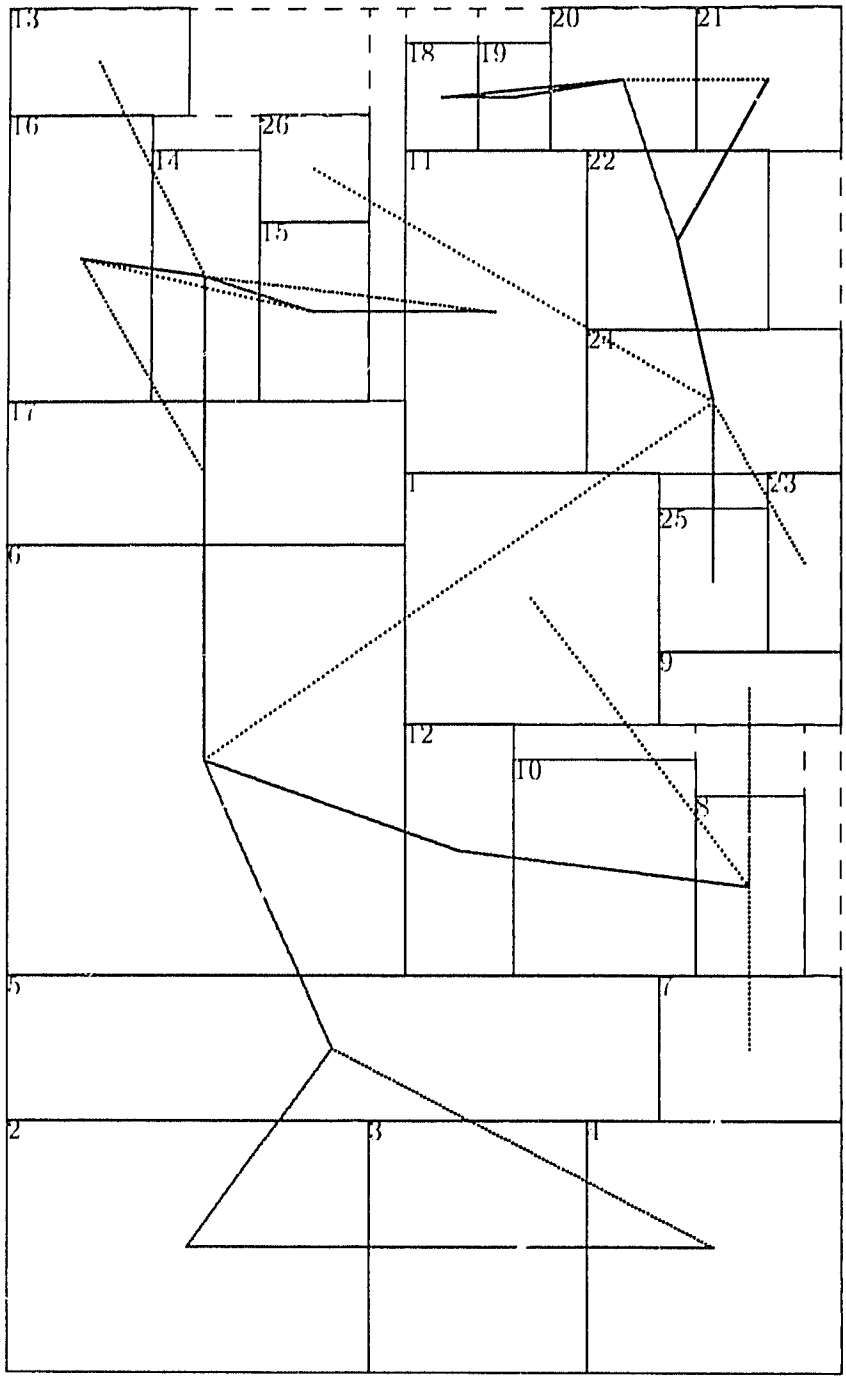


Figure 6.24: SA-II's Best: Cost=954 Area=874 Wire len.=890

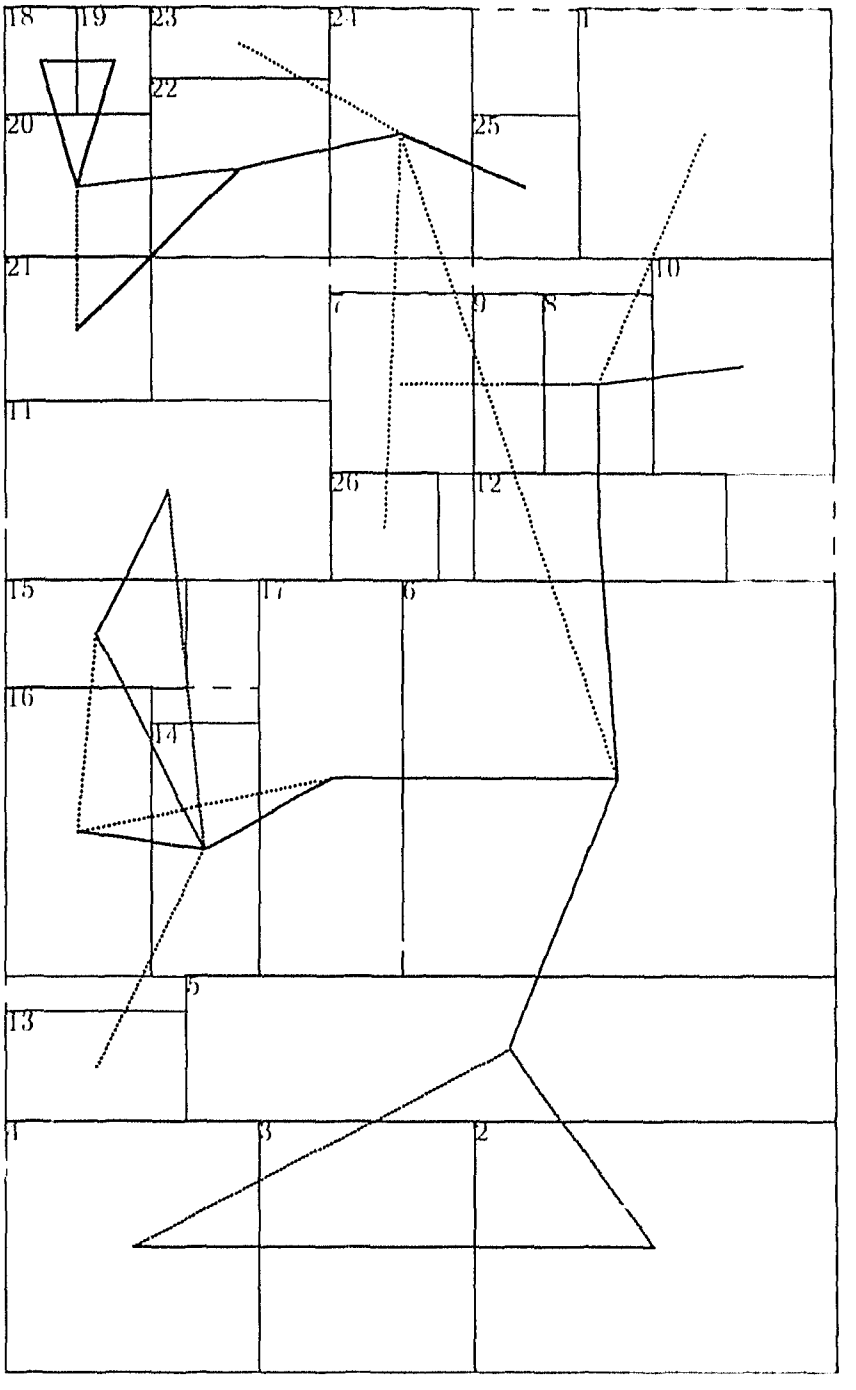


Figure 6.25: SA-E's Best: Cost=939.5 Area=874 Wire len. 866.5

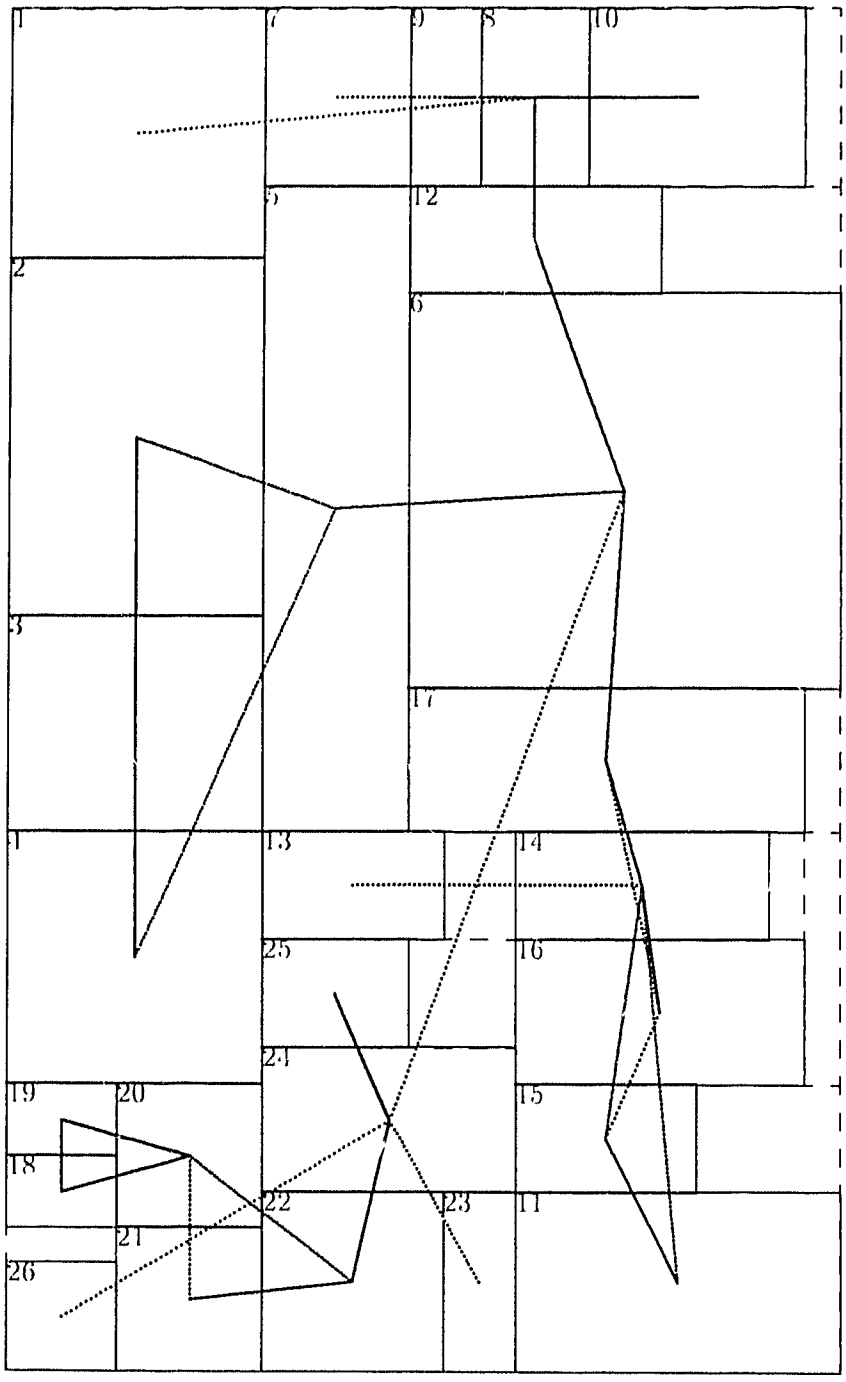


Figure 6.26: SS-II's Best: Cost=912 Area=874 Wire len. 846

6.4 Summary

In this chapter, we have discussed two physical design styles namely, standard cell and macro/custom. We have demonstrated placement procedures for both styles.

With regard to standard cell design, we integrated our standard cell placement procedure using the extended goal-directed schedule into the Oasis [2] logic synthesis system. We applied our procedure to 8 example circuits. The results show that in terms of layout area, our procedure performed better than the placement procedure used by Oasis.

With regard to floorplanning, we applied our procedure to an example floorplanning problem. The results show that in terms of layout area, Huang's schedule and the extended goal-directed schedule performed in a similar manner.

In the following chapter, we give guidelines on when to use the extended goal directed schedule based on problem instance statistics. We use the results from Chapters 4, 5 and 6 to justify these guidelines.

Chapter 7

Concluding Remarks

As stated in Chapter 1, research in the foundations of SA is concerned with determining which problems are amenable to optimization by SA and which cooling schedule best suites a particular problem. In this chapter, we give guidelines on when to use the extended goal-directed schedule based on experience and insight gained while working on this dissertation.

We close this chapter with a dissertation summary, including research contributions, an outline of possible future work and an epilogue.

7.1 Cooling Schedule Selection

Recall that q is the number of temperature values used in the extended goal-directed schedule. The value of q measures the difference between the cost of an easily accessible configuration and the cost (exact, estimated or lower-bound) of an optimal configuration relative to $\widehat{\Delta C}$. In Table 7.1, we show the q values for each of the

Chapter	Problem	Problem Size (Q)	q
3,4,5	5×5 SAP: Siarry's Net-list	25	13
3,4,5	5×5 SAP: Sechen's Net-list	25	11
4,5	10×10 SAP: Siarry's Net-list	100	13
4,5	10×10 SAP: Weighted Net-list	100	37
4,5	10×10 SAP: Horton's Net-list	100	38
6	Circuit 1	16	9
6	Circuit 2	16	8
6	Circuit 3	92	32
6	Circuit 4	100	27
6	Circuit 5	120	38
6	Circuit 6	190	48
6	Circuit 7	258	58
6	Circuit 8	291	63
6	Floorplan	26	6

Table 7.1: Example Problems and their q values.

problems considered in this dissertation.[1]

The extended goal-directed schedule is less susceptible to becoming trapped in local minima than schedules like the classical schedule or Huang's schedule. This is because temperature in the extended goal-directed schedule can increase when the procedure suspects that it is trapped in a local minimum. Temperature in the

[1] For the standard cell placement problems, Q in Figure 7.1 is the product of the number of rows and columns in the underlying placement array. Consequently, the value of Q is greater than or equal to the number of cells to be placed.

extended goal-directed schedule can increase on up to q occasions. The larger the value of q, the less likely it is that SA using the extended goal-directed schedule will become trapped in a local minimum relative to the chances that SA using either the classical schedule or Huang's schedule will become trapped in a local minimum.

The extended goal-directed schedule resembles Huang's schedule when the value of q is 1, *i.e.*, no temperature increases are allowed.

We surmise that in terms of the average terminal configuration cost and the least-cost terminal configuration cost, the extended goal-directed schedule will perform at least as well as Huang's schedule. We also surmise that the larger the value of q, the greater the chance that the extended goal-directed schedule will perform better than Huang's schedule. In both cases, it is likely that the extended goal-directed schedule will examine more configurations than SA using Huang's schedule.

Temperature in the extended goal-directed schedule can increase on up to q occasions. In terms of the number of configurations examined, a penalty is incurred each time the temperature is increased. The larger the value of q, the more potential there is for incurring multiple instances of this penalty. For SS, the initial temperature is set to T_j where the target cost E_j is the largest target cost less than or equal to $C(s_0^*)$. Therefore, temperature can increase on $(q - j)$ occasions. This results in a reduction in the potential of incurring multiple instances of the above penalty. Consequently, the larger the value of q, the more likely it is that SS using the extended goal-directed schedule will examine a smaller number of configurations than SA using the extended goal-directed schedule given a suitable heuristic for SS.

Based on our experience with the extended goal-directed schedule, we recommend the use of SS over SA when the value of q exceeds 20. This is because we can expect a significant decrease in the computation times of SS relative to the computation times of SA when the value of q is large. If the value of q is greater that 5, then

we recommend using the extended goal-directed schedule in the hope it will yield a quality terminal configuration and with the knowledge that its computational effort will most likely be greater than that of Huang's schedule. If the value of q is less that 5 then we recommend Huang's schedule be used in preference to the extended goal-directed schedule.

To support the above recommendations, we point to the benchmark results in Chapters 4, 5 and 6. In Chapter 4, we showed that for the 5×5 and 10×10 SAP problems, the extended goal-directed schedule consistently outperformed Huang's schedule in terms of the quality of the average terminal configuration. The 5×5 and 10×10 SAP problems using Siarry's net-list have typical q values of 13 and 13 respectively. For the 10×10 SAP problem using SA with the extended goal directed schedule, a typical number of temperature increases is 10.

In Chapter 6, we showed that for the floorplanning problem, the extended goal directed schedule had a similar performance to Huang's schedule in terms of the quality of the average terminal configuration. For the floorplanning problem, a typical value of q is 6.

From a users point of view, the quality of the least-cost terminal configuration is more important than the quality of the average terminal configuration. With regard to the 5×5 and 10×10 SAP problems, the extended goal directed schedule also outperformed Huang's schedule in terms of the quality of the least cost terminal configuration. This is evident from the somewhat higher percentage of terminal configurations which were optimal.

With regard to the standard-cell placement problems, we did not perform enough trials to say which schedule performed better on average. However, we can say that the extended goal-directed schedule performed at least as well as Huang's schedule in terms of terminal configuration quality. In fact, in all but one of our 8 exam

ple circuits, the extended goal-directed schedule yielded a superior quality terminal configuration.

In Chapter 5, we demonstrated the effectiveness of the extended goal-directed schedule for SS. For the 10×10 SAP problem using Siarry's net-list, a typical value for q is 43. When SA using Huang's schedule is used as the heuristic for SS, a typical initial temperature is T_{35}. Consequently, the potential for temperature increases is reduced resulting in a decrease in the number of configurations examined as compared to SA using the extended goal-directed schedule. In our experience, this reduction in potential for temperature increases is not sufficiently large to result in a decrease in the quality of the average terminal configuration.

Note that for the standard-cell circuits in Chapter 6, the value of q for all but circuits 1 and 2 is greater than 20. Consequently, we did not consider SA using the extended goal-directed schedule on any of these problems.

We emphasize that the recommendations contained in the proceeding paragraphs are based on insight and experience. No definitive statement can be made about the value of q. This is the case since q is a function of $\widehat{\Delta C}$ and the cost of an optimal configuration. We estimate $\widehat{\Delta C}$ and we most likely will also have to estimate the cost of an optimal configuration. Consequently, we can expect some variation in the value of q for multiple trials of a given problem instance.

7.2 Dissertation Summary

In Chapter 2, we introduced a framework for discussing optimization. Within that framework, iterative improvement was examined and evaluated.

SA was introduced in Chapter 3. In the past, the salient features of SA have been

described relative to their counterparts in physical annealing. We took a different approach and introduced an abstract model for SA. This model is an extension of the model proposed by Grover [9]. Within the context of this model, we gave the reader an appreciation for SA and how it works.

In Chapter 3, we also described a general cooling schedule which we referred to as the classical cooling schedule. We demonstrated the main drawback with this schedule, namely the need to perform parameter assignment. We also reviewed and evaluated a more elaborate cooling schedule called Huang's schedule.

In Chapter 4, we introduced the goal-directed cooling schedule and the extended goal-directed schedule. Both are based on the cost of an optimal solution to a problem instance. The latter schedule is the only schedule we are aware of that permits increases in the value of the control parameter, temperature. We compared the performance of the extended goal-directed schedules with Huang's schedule on the 5×5 and 10×10 SAP problems. With regard to the 5×5 and 10×10 SAP problems, we demonstrated that the improvement in quality of the average terminal configuration achieved by the extended goal-directed schedule over Huang's schedule is statistically significant.

One of the shortcomings of the extended goal-directed schedule is its long computation times. In Chapter 5, we addressed this problem by using SS. We showed how the extended goal-directed schedule can be used as a general cooling schedule for SS. In doing so, we achieved our goal of reducing the computation times required by SA using the extended goal-directed schedule. In terms of the 5×5 and 10×10 SAP problems, this reduction in computation times did not result in a decrease in the quality of the average terminal configuration.

In Chapter 6, we described two VLSI physical design styles namely, standard cell and macro/custom. We demonstrated placement procedures for both styles.

With regard to standard-cell design, we integrated our standard cell placement procedure using the extended goal-directed schedule into the Oasis [2] logic synthesis system. We evaluated our procedure on 8 example circuits. The results show that in terms of layout area, our procedure performs better than the placement procedure distributed with Oasis.

With regard to floorplanning, we compared the performance of Huang's schedule and the extended goal-directed schedule on an example floorplanning problem. The results show that in terms of average terminal configuration quality, Huang's schedule and the extended goal-directed performed in a similar manner. We stress that this is **not** a negative result. On the contrary, we view it as a positive result since it supports our contention that the performance of the extended goal-directed schedule is a function of q, the number of temperature values *i.e.*, problems with high q values are more amenable to optimization by SA using the extended goal-directed schedule than by Huang's schedule.

In this chapter, we discussed cooling schedule selection. We gave guidelines on when to use the extended goal-directed schedule based on q, the number of temperature values.

In the next section, we outline several areas of possible future work.

7.3 Future Work

7.3.1 Optimal-Configuration Cost Estimation

Recently, Sastry [26] demonstrated that the location parameter of the Weibull distribution can be used to estimate the cost of an optimal solution to some CO problems.

Sastry contends that the type 3 (Weibull) extreme value distribution provides an excellent model for the distribution of local minima in a CO problem. Consequently, the location parameter of the Weibull can be used to estimate the cost of an optimal solution.

Estimation of all three parameters of the Weibull distribution can be achieved using the method of moments or the method of maximum likelihood [4]. With regard to the Weibull distribution, the method of maximum likelihood requires the iterative solution of two non-linear equations. Sastry takes a simplified approach. Nonetheless, it requires the iterative solution of non-linear equations.

Our first objective was to gain experience in probabilistic estimation using the Weibull model. Consequently, we took the easier route of parameter estimation using the method of moments. In particular, we used the estimation formulae given by Ang, [4].

Sample mean, sample variance and sample skewness are required to estimate all three Weibull parameters using the method of moments. In our experience, there was a considerable fluctuation in sample skewness depending on the data sample. This led to an unacceptable variance in the location parameter estimate. We experimented with several sample gathering techniques. Unfortunately, we were unable to reduce the fluctuation in the sample skewness. Consequently, we abandoned Ang's method of moments estimation formulae and plan to investigate the method of maximum likelihood in future work.

For the standard-cell placement and floorplanning problems, we crudely estimated the cost of an optimal configuration. By design, these estimates were more than likely less than the actual cost of an optimal configuration. In future work, we intend to investigate the affect of a poor target cost estimate on the performance of the extended goal-directed schedule.

7.3.2 Initial Temperature Value for Simulated Sintering

In Chapter 5, we showed how to calculate the initial temperature for SS using the extended goal-directed schedule. There is every reason to suspect that the same initial temperature can be used for SS regardless of the schedule used. In future work, we intend to investigate how our calculation of the initial temperature can be used to turn cooling schedules for SA into cooling schedules for SS.

7.3.3 TimberWolf

TimberWolf [27] is one of the best known standard-cell placement procedures based on the SA paradigm. In future work, we intend to compare our standard-cell placement procedure to that of TimberWolf. We note that Suaris [31] compared quadrisection and TimberWolf on a number of example circuits. The results indicated that in terms of layout quality, quadrisection and TimberWolf had a similar performance. In terms of execution time, TimberWolf was at least an order of magnitude slower. Note that for our eight example circuits, our standard-cell placement procedure produced layouts which were on average 10% smaller than the layouts produced by quadrisection. This is not to say that our procedure will produce layouts 10% smaller than TimberWolf. Nonetheless, it is an indication that in terms of terminal configuration quality, our procedure can likely compete with TimberWolf. This issue can only be settled by an evaluation of both procedures on the same example circuits.

7.3.4 Multi-Goal Cost Functions for Simulated Annealing

Recently, Upton [32] introduced a SA cooling schedule with two independent control parameters. He contended that such an approach removed the problem of having to

weight wire length relative to area in floorplanning problems *i.e.* he used one control parameter for area and the other for wire length. We incorporated this approach into the goal-directed schedule and called it a two goal schedule [36]. In terms of terminal configuration quality, the two-goal schedule performed very well on idealized floorplanning problems. However, for the floorplanning problem described in Chapter 6, its performance in terms of terminal configuration quality was inferior (approximately 20%) to that of the extended goal directed schedule. Based on the idealized floorplanning results, we believe the two goal schedule has promise. Consequently, we plan to investigate multi-goal schedules for SA in future work.

7.3.5 Simulated Evolution

Simulated Evolution (SE) is another general-purpose Monte Carlo combinatorial optimization technique [13]. Given a current configuration, SE generates the next configuration by *mutating* the current configuration *i.e.*, some components of the current configuration are randomly rearranged. Non optimal configuration components are then *eliminated* from the mutated configuration yielding a partial solution. The eliminated components are added back into the partial solution using application specific heuristics and cost functions to determine their position and/or composition. This yields a complete solution or new configuration. The entire process is repeated until the initial configuration evolves into an optimal or near optimal configuration.

Ly *et al.* [3] have compared the performance of SA and SE using the scheduling problem in high-level synthesis. They cite two reasons why SE should outperform SA. The first reason concerns the probability of accepting a new configuration. The probability of accepting a new configuration in SA is a function of the overall configuration cost. In SE, eliminated configuration components are added back into the

partial solution using application specific heuristics and cost functions to determine their position and/or composition. SA may be forced to make gross tradeoffs because of its monolithic cost function whereas SE is free to apply specific heuristics and cost functions to configuration components. This argument is somewhat nullified by using a multi-goal cost function in SA. We propose investigating multi-goal cost functions with the intent of comparing and contrasting SA and SE.

7.3.6 Field-Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) have emerged as an attractive means of implementing logic circuits providing instant manufacturing turnaround and negligible prototype costs. They hold the promise of replacing much of the VLSI market now held by Mask-Programmed Gate Arrays [7].

The underlying structure of a Symmetrical-Array FPGA is a square array. The number of cells in a FPGA is typically in the low hundreds. Consequently, the problem of placement in Symmetrical-Array FPGAs maps to the SAP problem discussed in this dissertation both in terms of the number of cell involved and the underlying physical structure. Based on our results with the SAP problem, there is evidence to suggest that SS using the extended goal-directed schedule could yield quality FPGA placements. One further attribute that makes the extended goal-directed schedule suitable for FPGA placement is we can accurately estimate the total net-length for an optimized FPGA placement using procedures given by Sechen [27]. In future work, we propose an investigation of Symmetrical-Array FPGA placement using SS and the extended goal-directed schedule.

7.4 Epilogue

It has been said that SA is an optimization technique which should be included in everyone's bag of computing tricks. The technique itself is only as good as the cooling schedule which controls it.

To date, no one cooling schedule has proven suitable for all problem instances. In our view, no such cooling schedule exists.

In this dissertation, we have developed a new general cooling schedule called the extended goal-directed schedule. We have attempted to identify the type of problem best suited to optimization by SA and SS using the extended goal directed schedule. We believe that the extended goal-directed schedule is a useful extension to the so called bag of computing tricks.

Bibliography

[1] *ESM Benchmarks.* Microelectronics Center of North Carolina, 1987.

[2] *Open Architecture Silicon Implementation Software (Oasis).* Microelectronics Center of North Carolina, 1990.

[3] Ly T. A. and J. T. Mowchenko. Comparing simulated evolution and simulated annealing using the scheduling problem in high level synthesis. *Canadian Conference on VLSI*, pages 5.4.1 5.4.8, August 1991.

[4] A. H-S. Ang and W.H. Tang. *Probability Concepts in Engineering Planning and Design Vol. II-Decision,Risk, and Reliability.* John Wiley and Sons Inc., 1984.

[5] J. A. Bondy and U.S. R. Murty. *Graph Theory with Applications.* North-Holland., 1976.

[6] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. *Proc. IEEE Symposium on circuits and systems*, pages 663 698, June 1985.

[7] S. Brown, R. Francis, J. Rose, and Z. Vranesic. *Field Programmable Gate Arrays.* Kluwer Academic Publishere., 1992.

[8] David Green. *Modern Logic Design.* Addison-Wesley Publishing Company, 1986.

[9] L. K. Grover. Standard cell placement using simulated sintering. *Proc. 24th. Design Automation Conference*, pages 56–59, 1987.

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

[11] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 19(2):291–307, 1970.

[12] G. D. Kirkpatrick, C. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[13] R. Kling and P. Banerjee. Esp: A new standard cell placement package using simulated evolution. *ACM/IEEE Design Automation Conference*, pages 60–66, 1987.

[14] R. M. Kling and J. Tyszer. Empirical and theoretical studies of the simulated evolution method applied to standard cell placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(10):1303–1315, October 1991.

[15] D. E. Knuth. *The Art of Computer Programming Volume 2 Seminumerical Algorithms*. Addison Wesley., 1969.

[16] S. Lin and B. W. Kernighan. An efficient heuristic for the traveling salesman problem. *Operations Research*, pages 498–516, 1973.

[17] W. Mendenhall. *Introduction to Probability and Statistics: Fifth Edition*. Duxbury Press, North Scituate, Massachusetts, 1979.

[18] N. A. Metropolis, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chem. Phys.*, 21(6):1156–1159, 1953.

[19] D. M. Miller. *Decomposition in Many-Valued Logic Design.* Ph.D Thesis, Department of Computer Science, University of Manitoba, 1976.

[20] S. Nahar, S. Sahni, and E. Shragowitz. Experiments with simulated annealing. *22nd Design Automation Conference*, (22):748–752, 1985.

[21] J. K. Ousterhout. *Magic Manual.* EECS Department, University of California at Berkeley, 1986.

[22] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity.* Prentice Hall., 1982.

[23] B. Preas and M. Lorenzetti. *Physical Design Automation of VLSI Systems.* Benjamin/Cummings Publishing Company, Inc., Menlow Park, California, 1988.

[24] F. Reif. *Statistics and Thermal Physics.* McGraw Hill Inc., 1965.

[25] J. Rose, W. Klebsch, and J. Wolf. Temperature measurement and equilibrium dynamics of simulated annealing placements. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 9(3):253–259, March 1990.

[26] S. Sastry and J. Pi. Estimating the minimum of partitioning and floorplanning problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(2):273–282, February 1991.

[27] C. Sechen. *VLSI Placement and Global Routing using Simulated Annealing.* Kluwer Academic Publishers, 1988.

[28] K. Shahookar and P. Mazumder. Vlsi cell placement techniques. *ACM Computing Surveys*, 23(2):144–220, June 1991.

[29] P. Siarry, L. Bergonzi, and G. Dreyfus. Thermodynamic optimization of block placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 6(2):211–221, March 1987.

[30] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, (57):91–101, 1983.

[31] P. R. Suaris and G. Kedem. Quadrisection: A new approach to standard cell layout. *Intl. Conf. on Computer Aided Design*, pages 474–477, 1987.

[32] M. Upton, K. Samii, and S. Sugiyama. Integrated placement for mixed macro cell and standard cell designs. *27th. Design Automation Conference*, pages 32–35, 1990.

[33] P. van Laarhoven and E. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[34] P. A. Walsh. *Combinatorial Optimization in Logic Design*. Ph.D Dissertation Proposal, Department of Computer Science, University of Victoria, 1989.

[35] P. A. Walsh and D. M. Miller. Single row routing by simulated annealing. *Canadian Conference on VLSI*, pages 7.4.1–7.4.10, October 1990.

[36] P. A. Walsh and D. M. Miller. A goal directed cooling schedule for simulated annealing. *Canadian Conference on VLSI*, pages 6.4.1–6.4.8, August 1991.

[37] D. Wong and C. Liu. A new algorithm for floorplan design. *Proc. 23rd. Design Automation Conference*, pages 101–107, 1986.

[38] D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated Annealing for VLSI Design.* Kluwer Academic Publishers., Boston, 1988.

Appendix A

Goal for SA

```
procedure SAGoal(in s0; out sp);

  (* Simulated Annealing Goal-Directed Schedule *)

  const
    K0=0.99;
    Kq=0.01;

  var
    s0,sp,s,X, B:configuration;
    E0, Eq, ΔC', d, g, T, K, E:real;
    q:integer;

  (* s0 = initial configuration
    sp = terminal configuration
    X = current configuration
    s = selected neighbour configuration
    B = best configuration

    ΔC' = average ΔC'
    C' = average C'
    E = equilibrium configuration cost
    E0 = initial E
    Eq = final E (optimal configuration cost)
    g = the E multiplier

    K = probability of accepting a cost-increasing move of ΔC'
    K0 = initial K
    Kq = final K
    d = the K multiplier
    q = number of temperature decrements
    T = temperature *)

  begin
  X := s0;
  ΔC':=getAverageΔC';                C':=getAverageC';
  E0:=C';                            Eq:=getOptimalCost;
  q:=round(exp(ln(Eq/E0))/ln(Eq/(Eq+ΔC')));
  d:=exp(((ln(Kq/K0))/q);            g:=exp(((ln(Eq/E0))/q);

  K:=K0;
  E:=E0;
  B:=X;
```

```
while C'(B) > E_q do
    begin
    T:= (-ΔC') * ln(K);
    while C'(B) > E do
        begin
        select s; (* s ∈ G(X) *)
        if F(X,s,T) then
            begin
            X := s
            if C'(X) < C'(B) then
                B:=X;
            if (C'(X) - C'(B))/ΔC' ≤ random[0,1) then
                improve(X)
            end
        end;
    K:=K*d;
    E:=E*g;
    end;
s_p:=B
end;
```

Appendix B

E-Goal for SA

```
procedure SAEgoal(in s0,Q; out sp);

    (* Simulated Annealing Extended Goal-Directed Schedule *)

    const
        K0=0.99;
        Kq=0.01;

    var
        s0,sp,s,X, B:configuration;
        E0, Eq, ΔC', d, g, T, K, E:real;
        Sent, Cnt, q, smax, smin:integer;
        Inc:boolean;
```

(*
s_0 = initial configuration
s_p = terminal configuration
Q = problem instance size
X = current configuration
s = selected neighbour configuration
B = best configuration found so far
Cnt = number of configurations examined at a particular temperature value
s_{max} = worst cost over all configurations accepted at a particular temperature value
s_{min} = best cost over all configurations accepted at a particular temperature value
$Sent$ = number of occasions temperature is decreased without detecting equilibrium and $(s_{max} - s_{min}) < \widehat{\Delta C'}$
Inc = a true value indicates that temperature may increase

$\widehat{\Delta C'}$ = average $\Delta C'$
$\hat{C}'$ = average C'
E = target configuration cost
E_0 = initial E
E_q = final E (optimal configuration cost)
g = the E multiplier

K = probability of accepting a cost-increasing move of $\Delta C'$
K_0 = initial K
K_q = final K
d = the K multiplier
q = number of temperature decrements
T = temperature
*)

```
begin
X := s0;
Cnt:=0;
Sent:=0;
Inc:=false;
smax:=0;                                  smin:=MAXREAL;
ΔC':=getAverageΔC';                       C':=getAverageC';
E0:=C';                                   Eq:=getOptimalCost;
q:=round(exp((ln(Eq/E0))/ln(Eq/(Eq+ΔC'))));
d:=exp(((ln(Kq/K0))/q));                  g:=exp(((ln(Eq/E0))/q));

K:=K0;
E:=E0;
B:=X;

while C'(B) > Eq do
    begin
    T:=(-ΔC')*ln(K);
    while C'(B) > E do
        begin
        select s; (* s ∈ G(X) *)
        Cnt:=Cnt+1;
        if F(X,s,T) then
            begin
            X := s
            if C'(X) < C'(B) then
                begin
                B:=X;
                improve(X)
                end
            end;

        if (not Inc) and (Cnt > (500*Q)) then
            (* DECREASE TEMP. - equilibrium not detected*);
            begin
            if (smax - smin) < ΔC' then
                begin
                Sent:=Sent+1;
                if Sent=Q then
                    Eq:=C'(B)  (* QUIT *)
                end
            else
```

```
                    Sent:=0;
                K:=K*d;
                T:=(-ΔC)*ln(K);
                s_max:=0;
                s_min:=MAXREAL;
                Cnt:=0
                end;

            if (Inc) and (Cnt > (1000*Q)) then
                (* INCREASE TEMP - stabalize *);
                begin
                K:=K_0;
                E:=E_0;
                while E ≥ C(B) do
                    begin
                    K:=K*d;
                    E:=E*g
                    end;
                T:=(-ΔC)*ln(K);
                Inc:=false;
                s_max:=0;
                s_min:=MAXREAL;
                Cnt:=0
                Sent:=0
                end;

            end;

        (* DECREASE TEMP - equilibrium detected *);
        K:=K*d;
        E:=E*g;
        Inc:=true;
        s_max:=0;
        s_min:=MAXREAL;
        Cnt:=0;
        Sent:=0
        end;
    s_p:=B
    end;
```

Appendix C

E-Goal for SS

```
procedure SSEgoal(in s0,Q; out sp);

    (* Simulated Sintering Extended Goal-Directed Schedule *)

    const
        K0=0.99;
        Kq=0.01;

    var
        s0,sp,s,X, B:configuration;
        E0, Eq, ΔC', d, g, T, K, E:real;
        Scnt, Cnt, q, smax, smin:integer;
        Inc:boolean;
```

(*
s_0 = initial configuration
s_p = terminal configuration
Q = problem instance size
X = current configuration
s = selected neighbour configuration
B = best configuration found so far
Cnt = number of configurations examined at a particular temperature value
s_{max} = worst cost over all configurations accepted at a particular temperature value
s_{min} = best cost over all configurations accepted at a particular temperature value
$Scnt$ = number of occasions temperature is decreased without detecting equilibrium and $(s_{max} - s_{min}) < \widehat{\Delta C'}$
Inc = a true value indicates that temperature may increase

$\widehat{\Delta C'}$ = average $\Delta C'$
$\hat{C}$ = average C
E = target configuration cost
E_0 = initial E
E_q = final E (optimal configuration cost)
g = the E multiplier

K = probability of accepting a cost-increasing move of $\Delta C'$
K_0 = initial K
K_q = final K
d = the K multiplier
q = number of temperature decrements
T = temperature
*)

```
begin
X := s0;
Cnt:=0;
Sent:=0;
Inc:=false;
smax:=0;                                  smin:=MAXREAL;
ΔC':=getAverageΔC';                       C':=getAverageC';
E0:=C';                                   Eq:=getOptimalCost;
q:=round(exp((ln(Eq/E0))/ln(Eq/(Eq + ΔC'))));
d:=exp(((ln(Kq/K0))/q));                  g:=exp(((ln(Eq/E0))/q));

K:=K0;
E:=E0;
B:=X;

while E ≥ C'(B) then (* adjust parameters for sintering *)
    begin
    E := E * g;
    K := K * d
    end;

while C'(B) > Eq do
    begin
    T:= (-ΔC') * ln(K);
    while C'(B) > E do
        begin
        select s; (* s ∈ G(X) *)
        Cnt:=Cnt+1;
        if F(X,s,T) then
            begin
            X := s
            if C'(X) < C'(B) then
                begin
                B:=X;
                improve(X)
                end
            end;

        if (not Inc) and (Cnt > (500 * Q)) then
            (* DECREASE TEMP. - equilibrium not detected*);
            begin
            if (smax - smin) < ΔC' then
```

```
                begin
                Sent:=Sent+1;
                if Sent=Q then
                    E_q:=C(B)  (* QUIT *)
                end
            else
                Sent:=0;
            K:=K*d;
            T:=(-ΔC)*ln(K);
            s_max:=0;
            s_min:=MAXREAL;
            Cnt:=0
            end;

        if (Inc) and (Cnt>(1000*Q)) then
            (* INCREASE TEMP.    stabalize *);
            begin
            K:=K_0;
            E:=E_0;
            while E>C(B) do
                begin
                K:=K*d;
                E:=E*g
                end;
            T:=(-ΔC)*ln(K);
            Inc:=false;
            s_max:=0;
            s_min:=MAXREAL;
            Cnt:=0
            Sent:=0
            end;

        end;

    (* DECREASE TEMP.    equilibrium detected *);
    K:=K*d.
    E:=E*g;
    Inc:=true;
    s_max:=0;
    s_min:=MAXREAL;
    Cnt:=0;
    Sent:=0
```

```
        end;
    s_p := B
    end;
```