

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI[®]

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

A Parallel Tabu Search Algorithm for VLSI Standard Cell Placement

BY

Ahmad Abdul-Jabbar Al-Yamani

A Thesis Presented to the
FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In
Computer Engineering

May 1999

UMI Number: 1395608

UMI Microform 1395608

Copyright 1999, by UMI Company. All rights reserved.

**This microform edition is protected against unauthorized
copying under Title 17, United States Code.**

UMI

**300 North Zeeb Road
Ann Arbor, MI 48103**

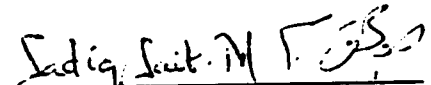
KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA
DEANSHIP OF GRADUATE STUDIES

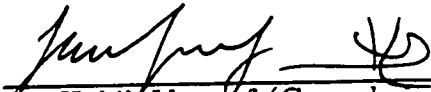
This thesis, written by

AL-YAMANI, AHMAD ABDUL-JABBAR

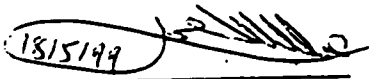
under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the Deanship of Graduate Studies, in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN COMPUTER ENGINEERING


Thesis Committee


Dr. Sadiq. M. Sait (Chairman)


Dr. Habib Youssef (Co - chairman)


Dr. Hassan Barada (Member)


Dr. Abdullah I. Almojel
Department Chairman


Dr. Abdullah M. Al - Shehri
Dean of Graduate Studies

18/5/99
Date



"هَذَا مِنْ فَضْلِ رَبِّي لِيَبْلُوَنِي أَأَشْكُرُ أَمْ أَكْفُرُ وَمَنْ
شَكَرَ فَإِنَّمَا يَشْكُرُ لِنَفْسِهِ وَمَنْ كَفَرَ فَإِنَّ رَبِّي غَنِيٌّ
كَرِيمٌ"

النمل - ٤٠ -

Dedicated

to

my beloved parents

and to the memory of my sister

Ameerah

Acknowledgments

All praise be to Allah, *Subhanahu-wa-ta-Aaala*, for his unlimited help and guidance. May Allah bestow peace on his prophet, Muhammad (pbuh), and his family. I acknowledge the support and facilities provided by King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

Thanks to my father and mother who were -after Allah- the source of all success in my life. I was carried through the most difficult moments in my life by their prayers, love, and support. My sisters and brothers were a great source of support in all achievements I accomplished.

I would like to express my extreme thanks to the one who did the difficult part of the work. The one who was living the difficult moments before the nice ones. The one whose support, appreciation and patience made it possible. All thanks are to my wife 'Alaa Olwi'. I also thank my father-in-law and my mother-in-law who were in touch throughout all the steps in my MS program.

I would like to express my profound gratitude and appreciation to my thesis committee Dr. Sadiq M. Sait, Dr. Habib Youssef and Dr. Hassan Barada, for their guidance and patience throughout this thesis. It was because of them that the work at any point of time, never got stressful.

I also wish to thank the Chairman of Computer Engineering Department, Dr. Abdullah Almojel and Dean of College of Computer Science and Engineering, Dr. Khalid AlTawil, for all the support they provided in order to achieve this work. Also, my thanks go to the faculty and staff members of Computer Engineering Department for their encouragement. I also thank Hussain Ali for helping in providing a lot of useful functions without which the comparison would not have been possible.

Contents

Acknowledgments	v
List of Tables	ix
List of Figures	x
Abstract (English)	xii
Abstract (Arabic)	xiii
1 Introduction	1
1.1 VLSI Placement Problem Definition	2
1.2 VLSI Design Styles	3
1.3 Multi-Objective Placement Problem	3
1.3.1 Interconnection Length	4
1.3.2 Area	6
1.3.3 Critical Path Delay	6
1.3.4 Overall Solution Quality Evaluation	7
1.4 Heuristics Applied to VLSI Placement	10
1.5 Organization of the Thesis	12
2 Tabu Search	13
2.1 Basic Tabu Search Algorithm	14
2.2 Tabu Search Parameters	16
2.2.1 Candidate List	16

2.2.2	Moves and Move Attributes	19
2.2.3	Evaluation Function	19
2.2.4	Tabu List	20
2.2.5	Aspiration Criteria	21
2.3	Tabu Search Classes	23
2.4	Summary	24
3	Literature Review	26
3.1	Heuristics Applied to VLSI Placement	26
3.2	Tabu Search in VLSI Placement	28
3.3	Tabu Search Parallelization	29
3.4	Summary	32
4	Parallel TS for VLSI Standard-Cell Placement	33
4.1	Proposed Tabu Search Algorithm	33
4.2	Parallelization of the Algorithm	35
4.3	Applying the Algorithm in a Heterogeneous Environment	38
4.4	Diversification of the Search Process	41
4.5	Summary	43
5	Experimental Results	45
5.1	Effect of Degree of Low-level Parallelization	47
5.2	Effect of Degree of High-level Parallelization	53
5.3	Accounting for Speed and Load Heterogeneity	57
5.4	Effect of Diversification	61
5.5	Interarrival of a q -Quality Solution	65
5.6	Fuzzy Cost Evaluation vs. Weighted Sum	67
5.7	Comparison with Previous Work	68
5.8	Summary	70
6	Conclusions	71

A	Parallel Virtual Machine	74
A.1	Introduction	74
A.2	Parallel Virtual Machine (PVM)	75
A.2.1	PVM Daemon (pvmd)	75
A.2.2	PVM Library (PVML)	75
A.2.3	Architectural Description	76

List of Tables

5.1	Characteristics of circuits and layouts used. (<i>LH</i> = layout heights and <i>Avg. RCH</i> = average routing channel height in microns).	46
5.2	Parameters of parallel experiments. (GI = Global Iterations and LI = Local Iterations).	47
5.3	Runtime of homogeneous and heterogeneous runs in seconds for seven circuits.	59
5.4	Best solution achieved by fuzzy evaluation run vs. weighted sum run.	68
5.5	Best solution achieved by parallel tabu search vs. classical simulated evolution.	69
5.6	Best solution achieved by parallel tabu search vs. simulated evolution with fuzzy evaluation.	69
5.7	Best solution achieved by parallel tabu search vs. simulated evolution with fuzzy allocation.	70

List of Figures

1.1	Levels of abstraction for VLSI circuits design.	2
1.2	Configuration of a VLSI standard cell layout.	4
1.3	Range of acceptable solutions for a tri-valued cost vector. . . .	8
1.4	Range of acceptable solutions for a the proposed cost vector. .	9
1.5	The membership function <i>within acceptable criterion i</i>	10
2.1	Algorithmic description of tabu search (TS).	14
2.2	Basic tabu search algorithm flowchart.	15
2.3	Aspiration plus strategy.	17
2.4	Elite candidate list strategy.	17
2.5	Sequential fan candidate list strategy.	18
4.1	Paradigm of tabu search parallel implementation.	36
4.2	Algorithmic description of master process of parallel TS. . . .	39
4.3	Algorithmic description of worker process of parallel TS. . . .	39
4.4	A scenario of the <i>master</i> , the <i>TSW</i> and the <i>CLW</i> processes. .	40
5.1	Effect of number of CLWs on the solution quality.	49
5.2	Runtime needed to achieve a solution of cost less than x for different numbers of CLWs.	50
5.3	Speedup achieved in reaching a solution of cost less than x for different numbers of CLWs.	51
5.4	Efficiency of using more CLWs in reaching a solution of cost less than x	52
5.5	Effect of number of TSWs on the solution quality.	54

5.6	Runtime needed to achieve a solution of cost less than x for different numbers of TSWs.	55
5.7	Speedup achieved in reaching a solution of cost less than x for different numbers of TSWs.	56
5.8	Efficiency of using more TSWs in reaching a solution of cost less than x	58
5.9	Heterogeneous vs. homogeneous runs.	60
5.10	Best cost vs. runtime for heterogeneous and homogeneous runs.	62
5.11	Number of solutions provided by machines of different speeds within various solution ranges.	63
5.12	Effect of diversification.	64
5.13	Local vs. global iterations.	66
5.14	CDF for getting a solution of quality $< q$	67

THESIS ABSTRACT

Name: AL-YAMANI, AHMAD ABDUL-JABBAR
Title: A PARALLEL TABU SEARCH ALGORITHM
FOR VLSI STANDARD CELL PLACEMENT
Major Field: COMPUTER ENGINEERING
Date of Degree: May 1999

VLSI standard cell placement is an NP-hard problem to which various heuristics have been applied. In this work, tabu search, which is an iterative heuristic, is used. The objective of the algorithm is to achieve the best possible placement solution in terms of interconnection length, overall area of the circuit, and critical path delay (circuit speed). The algorithm is parallelized on a network of stations using PVM. The proposed tabu search algorithm integrates two parallelization strategies. These are functional decomposition strategy and multi-search threads strategy. Furthermore, domain decomposition strategy is implemented probabilistically. The performance of each strategy is observed and analyzed. The goal of the parallelization is to speedup the search and to improve the solution quality. A diversification scheme is applied to make sure that different processes investigate different regions of the search space. The performance of this scheme is studied and analyzed. The algorithm is applied in a heterogeneous environment and a suitable strategy is adopted to account for that heterogeneity. The effect of accounting for heterogeneity is observed and analyzed. Experimental results are presented.

Index Terms: VLSI, Standard Cell Design, Placement, Tabu Search, Parallel Tabu Search, Functional Decomposition, Multi-Search Threads, Combinatorial Optimization, Diversification.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
Dhahran, Saudi Arabia

May 1999

خلاصة الرسالة

الاسم: أحمد بن عبد الجبار اليماني

عنوان الرسالة: خوارزم متوازي للبحث المحظور لمشكلة ترتيب الخلايا في الدوائر عالية الكثافة جداً

التخصص: هندسة الحاسب الآلي

تاريخ الشهادة: مايو ١٩٩٩

ترتيب الخلايا في الدوائر عالية الكثافة جداً مشكلة لا يمكن تمثيلها بكثيرات الحدود ويطبق عليها أكثر من خوارزم. في هذه الرسالة، يستخدم خوارزم البحث المحظور وهو خوارزم عشوائي تسلسلي. هدف الخوارزم المستخدم هو تحقيق أفضل الحلول الممكنة من حيث طول التوصيلات، المساحة الكلية للدائرة، وسرعة عمل الدائرة. ينفذ الخوارزم بالتوازي على شبكة من محطات العمل باستخدام (PVM). الخوارزم المقترح يكامل استراتيجيتين للتنفيذ المتوازي هما استراتيجية التحليل الوظيفي واستراتيجية عمليات البحث المتعددة. بالإضافة إلى ذلك، فإن استراتيجية تقسيم المجال تطبق احتمالياً. يتم تحليل و ملاحظة أداء كل من هذه الاستراتيجيات. هدف التنفيذ المتوازي هو تطوير سرعة البحث ونوعية الحل. تطبق طريقة لتطبيق تباعد البحث لزيادة كفاءة التنفيذ المتوازي، كما تتم دراسة وتحليل أداء هذه الطريقة. يطبق الخوارزم في بيئة من محطات عمل متغايرة. تتم ملاحظة وتحليل تأثير اعتبار التغير بين الأجهزة. وأخيراً، تتم مقارنة نتائج الخوارزم بنتائج عمل سابق من أجل التحقق من عمله.

مصطلحات البحث: الدوائر عالية الكثافة جداً، التصميم القياسي، ترتيب الخلايا، زمن التنفيذ، طول التوصيلات، البحث المحظور، البحث المحظور المتوازي، التحليل الوظيفي، عمليات البحث المتعددة، التكفيء التجميعي، التباعد.

درجة ماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

مايو ١٩٩٩

Chapter 1

Introduction

Complexity of Very Large Scale Integration (VLSI) circuits requires breaking the design process into various levels of abstraction. These levels are shown in Figure 1.1 [1]. Architectural design is the task of human experts who determine the behavior of the product under various circumstances and different inputs. This stage has a great effect on the cost and the performance of the final product. Logical design phase takes care of defining the data path and the control path of the circuit to be designed. The fabrication of the circuit, i.e., Printed Circuit Board (PCB) or VLSI, is also part of this design level.

Physical design refers to all steps that follow the logical design before the fabrication of the circuit. These steps include partitioning, floorplanning, placement and routing. The wire length of the circuit, the overall area, and the delay, are dramatically affected by how the circuit is laid out. In this work, placement problem, which is one of the VLSI physical design problems, is addressed and *Tabu Search* algorithm, which is an iterative heuristic, is applied to it.

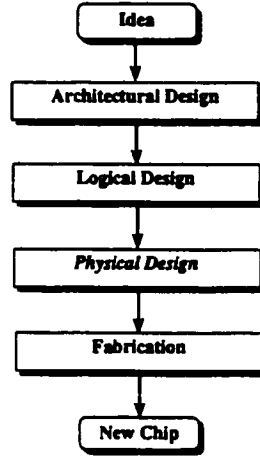


Figure 1.1: Levels of abstraction for VLSI circuits design.

1.1 VLSI Placement Problem Definition

In VLSI design, placement is the process of arranging components on the circuit surface [1]. These components are interconnected functional blocks that vary in size according to their functions.

The process of placement is directed towards satisfying some objectives. These objectives can be the overall area of the circuit [2, 3], the total wire length [4], the delay of the critical paths [5, 6, 7], the routability of the interconnections [8], or a combination of these.

The problem of cell placement can be defined as finding suitable locations for all cells in the two-dimensional layout surface of a VLSI circuit. A suitable location is one that optimizes some objectives like wire length, area, and/or delay. Cell placement problem can be formally defined as follows. Given a set of modules $M = \{m_1, m_2, \dots, m_n\}$, and a set of signals $S = \{s_1, s_2, \dots, s_k\}$, each module $m_i \in M$ is associated with a set of signals S_{m_i} , where $S_{m_i} \subseteq S$. Also each signal $s_i \in S$ is associated with a set of modules M_{s_i} , where $M_{s_i} = \{m_j | s_i \in S_{m_j}\}$. M_{s_i} is called a signal net. Placement problem is to assign each module $m_i \in M$ to a unique location such that a given cost function is optimized and a number of constraints are satisfied. Cost function

and constraints depend on the layout style and the objectives [9].

Even in its simple case, where components and slots are of equal sizes and one objective is to be satisfied, placement is an NP-hard¹ problem [8]. Since the number of cells can be in thousands, it is impractical to use brute force technique to solve it.

1.2 VLSI Design Styles

Various layout styles differ in the structural constraints imposed on the components and the layout surface. These styles include the full-custom layout where no constraints are applied, the automation becomes difficult and the layout is hand-crafted. Gate array methodology imposes the restriction that the surface is a two dimensional array of equal-size slots and the components have to be equally sized [1]. Macro cell methodology allows cells to vary in both dimensions to allow smaller area placement or better routing of interconnections [1, 11]. Standard cell methodology assumes the cells to be of same height and variable width to be arranged in rows. Channels between the rows are of variable height to allow routing of connections between cells. Connections between cells on the same row or opposite rows are routed through adjacent channels. Connections between non-adjacent rows are routed through feed-through cells placed in the intermediate row(s) [1, 2, 4, 8, 12, 13, 14]. Figure 1.2 shows the configuration of a standard cell layout.

1.3 Multi-Objective Placement Problem

Modern placement programs usually require the optimization of several objectives and the satisfaction of several constraints. Typical objectives are the overall area of the circuit i.e., the functional area and the wiring area

¹NP-Hard Problems are a class of problems that cannot be solved by a deterministic algorithm of polynomial complexity [10].

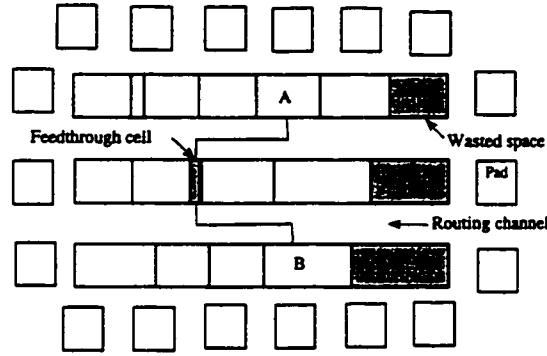


Figure 1.2: Configuration of a VLSI standard cell layout.

[2, 3], the total wire length [4], the delay of the critical paths [5, 6, 7], the routability of the interconnections [8], or a combination of some or all of these. Some of the objectives might be conflicting. For example, to have the smallest possible area, one might have to accept a larger delay. A similar conflict might exist between routability and wire length. In order to combine conflicting objectives in the evaluation function, either weighted sum or fuzzy evaluation should be used. In this work, interconnection length, overall area, and critical path delay will be used to quantify the goodness of a specific solution. The objective of the search is to define the *most acceptable* solution and to allow the designer to express her/his preferences for each criterion. In this work, fuzzy algebra is used to evaluate the overall cost of the solution.

1.3.1 Interconnection Length

One of the most important characteristics of a proposed solution is its wire length. This is due to the rapid improvement in the switching delay of the transistors. As a result, the interconnection delay is becoming the bottleneck in VLSI technology. Another reason is that the functional area is getting much smaller compared to the interconnection area. These factors led to considering wire length as a critical measure in any proposed layout configuration.

Different models are used to estimate the length of a given *net* which is a set of points or pins that have to be in the same voltage level (connected points). Half-perimeter bounding box, minimum Steiner tree, and minimum spanning tree are among those models [1, 8].

Steiner tree approximation is fast and accurate enough in modeling actual wire length and it has been used in this work [15]. To apply this model, the bounding box, which is the smallest rectangle bounding a net, is found for each net. The average vertical distance Y and horizontal distance X of all cells in the net are computed from the origin which is the bottom left corner of the bounding box of the net. A central point (X, Y) is placed at the computed average distances. If X is greater than Y then the vertical line crossing the central point is considered as the bisecting line. Otherwise, the horizontal line is considered as the bisecting line. Steiner tree approximation of a net is the length of the bisecting line added to the summation of perpendicular distances to it from all cells belonging to the net. Steiner tree approximation is computed for each net and the summation of all Steiner trees is considered as the interconnection length of the proposed solution.

$$X = \frac{\sum_{i=1}^n x_i}{n} \quad Y = \frac{\sum_{i=1}^n y_i}{n} \quad (1.1)$$

where n is the number of cells contributing to the current net.

$$Steiner\ Tree = B + \sum_{j=1}^k P_j \quad (1.2)$$

Where B is the length of the bisecting line, k is the number of cells contributing to the net and P_j is the perpendicular distance from cell j to the bisecting line.

$$Interconnection\ Length = \sum_{l=1}^m Steiner\ Tree_l \quad (1.3)$$

Where m is the number of nets.

1.3.2 Area

In standard cell placement, cells (or blocks) of fixed heights are placed in rows. It is the width of these rows that varies with the proposed solution according to the type and number of cells placed in the row. Heights of routing channels are initially estimated and assumed to be fixed because routing is not in the scope of this work. The overall area of a given solution is estimated as follows

$$A = (n_rows \times row_height + (n_rows - 1) \times channel_height) \times longest_width \quad (1.4)$$

n_rows gives the number of rows used in the solution under test. The number of channels is 1 less than the number of rows. When n_rows and $(n_rows - 1)$ are multiplied by row_height and $channel_height$ respectively, they give the overall height of the layout. Multiplied by the longest row, it gives the area of the bounding rectangle of the proposed solution. Since row_height is constant and $channel_height$ is assumed to be constant, $width$ of the longest row will be measured as it is the only thing that differs from solution to another².

1.3.3 Critical Path Delay

A digital circuit comprises a collection of paths. A path is a sequence of nets and blocks from a source to a sink. A source can be an input pad or a memory cell output, and a sink can be an output pad or a memory cell input. The longest path (*critical path*) is the dominant factor in deciding the clock frequency of the circuit. A critical path makes a problem in the design if it has a delay that is larger than the largest allowed delay (period) according to the clock frequency.

The delay of any given path is computed as the summation of the delays of the nets v_1, v_2, \dots, v_k belonging to that path and the switching delay of the

²From here on, in this thesis, width will be used in place of area because the height of the layout is assumed to be constant.

cells driving these nets. The delay of a given path π is given by

$$T_{\pi} = \sum_{i=1}^{k-1} (CD_{vi} + ID_{vi}) \quad (1.5)$$

where CD_{vi} is the switching delay of the driving cell and ID_{vi} is the interconnection delay that is given by the product of the load factor of the driving cell and the capacitance of the interconnection net, i.e.,

$$ID_{vi} = LF_{vi} \times C_{vi} \quad (1.6)$$

$SLACK_{\pi}$ of path π is given by

$$SLACK_{\pi} = LRAT_{\pi} - T_{\pi} \quad (1.7)$$

where $LRAT_{\pi}$ is the latest required arrival time and T_{π} is the path delay [16, 17]. If T_{π} is greater than $LRAT_{\pi}$, then the path π will have a negative $SLACK$ which is an indicator of a **long path** problem. Upper bounds can be applied to nets belonging to the critical path as constraints not to allow them to exceed a certain limit beyond which the $SLACK$ will be negative.

1.3.4 Overall Solution Quality Evaluation

The overall quality (or cost) of a multi-objective problem is a vector quantity. In this work, since we are using three criteria to represent a solution cost, the cost is a tri-valued vector. To represent this vector as a scalar quantity for comparison between solutions, weighted sum can be used. The main disadvantage in using weighted sum scheme is the difficulty in finding the appropriate combination of weights. The objective of the search is to find the *most acceptable* solution with respect to the evaluation criteria. This makes fuzzy algebra a suitable way for defining the goodness of a given solution with respect to different criteria [11, 15, 18, 19, 20]. The evaluation function should allow expressing designer preferences with respect to each evaluation criterion. For this, the scheme we are using for fuzzification is called *Fuzzy Goal Based Cost Measure* which was proposed in [15].

Let there be p values in the solution cost vector. For a given solution x , the cost is given by the vector $C(x) = (C_1(x), C_2(x), \dots, C_p(x))$. Let $O = (O_1, O_2, \dots, O_p)$ be a vector representing the optimum quantities for all cost criteria i.e., $O_i \leq C_i(x) \forall i, \forall x \in \Pi$, where Π is the set of possible solutions. $G = (g_1, g_2, \dots, g_p)$ is a user-specified goal vector representing the acceptable limits for all cost criteria in the cost vector. A solution x is considered acceptable by the user if $C_i(x) \leq g_i \times O_i, \forall i$. Since the acceptable limit has to be greater than the optimum value, $g_i > 1.0 \forall i$.

Figure 1.3 shows the region of acceptable solutions for a tri-valued cost vector given O_1, O_2, O_3, g_1, g_2 and g_3 . The projection of that figure on our cost function that has O_{wl}, O_{delay} and O_{width} as the optimum values for wire length, delay and longest width respectively and g_{wl}, g_{delay} and g_{width} as the goals for those criteria is shown in Figure 1.4.

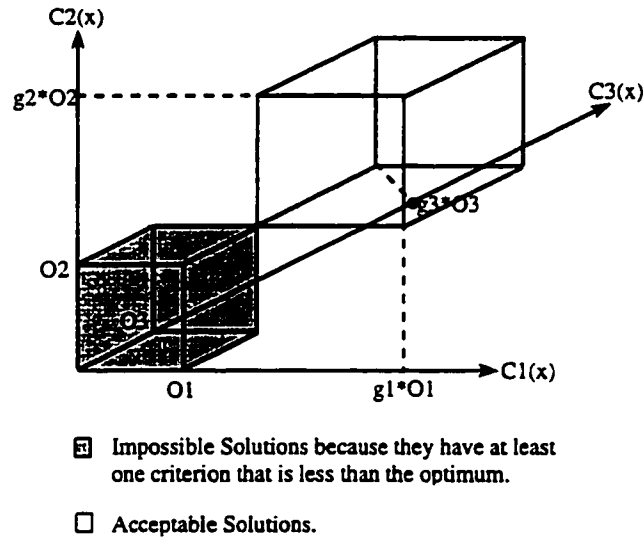


Figure 1.3: Range of acceptable solutions for a tri-valued cost vector.

In this work, the rule for determining membership in the fuzzy set is taken as in [15]. The rule is as follows:

Rule1: If a solution is *within acceptable wire length* AND *within acceptable circuit delay* AND *within acceptable width*, THEN it is an acceptable solution.

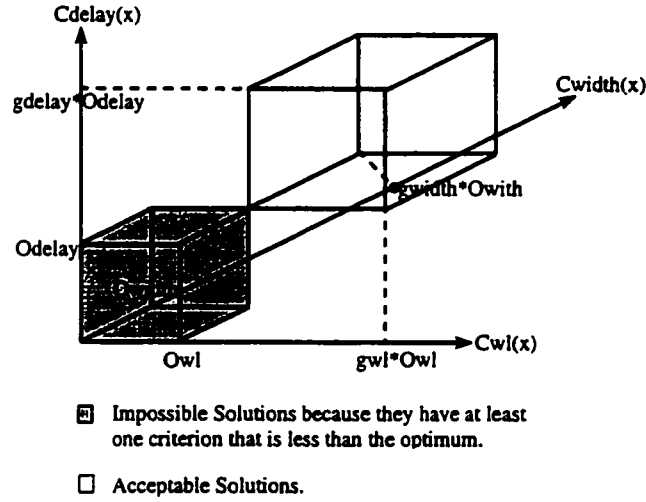


Figure 1.4: Range of acceptable solutions for a the proposed cost vector.

Using fuzzy algebraic notation, while adopting the andlike ordered weighted averaging (OWA) operator of Yager [21], the above rule is expressed as follows,

$$\mu(x) = \beta \times \min(\mu_1(x), \mu_2(x), \mu_3(x)) + (1 - \beta) \times \frac{1}{3} \sum_{i=1}^3 \mu_i(x) \quad (1.8)$$

where, $\mu(x)$ is the membership value for solution x in the fuzzy set *acceptable solutions*. μ_i for $i \in \{1, 2, 3\}$ (or $i \in \{wl, delay, width\}$) represents the membership values for solution x in fuzzy sets *within acceptable wire length*, *within acceptable circuit delay* and *within acceptable width* respectively. β is an averaging constant in the range $[0, 1]$. The solution that results in the maximum value for Equation 1.8 is considered as the best solution found.

The membership functions used for wire length, delay and width are the same. They are computed as follows:

$$\mu_i = \begin{cases} 1 & \text{if } C_i(x)/O_i \leq 1 \\ \frac{g_i - C_i(x)/O_i}{g_i - 1} & \text{if } 1 < C_i(x)/O_i \leq g_i \\ 0 & \text{if } C_i(x)/O_i > g_i \end{cases} \quad (1.9)$$

Where, $i \in \{1, 2, 3\}$ (or $i \in \{wl, delay, width\}$). The membership function is shown in Figure 1.5.

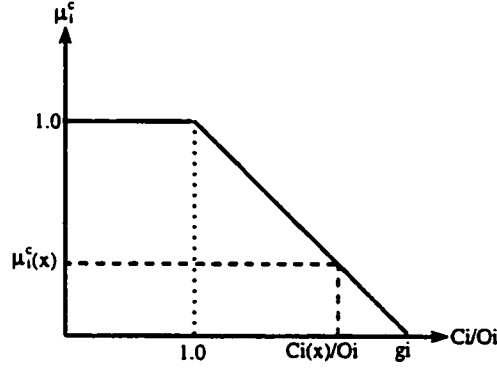


Figure 1.5: The membership function *within acceptable criterion i*.

The lower bounds for wire length, delay and width were computed in [15] using the following formulae:

$$O_1 = \sum_{i=1}^n OWL_{vi} \quad \forall v_i \in \{v_1, v_2, \dots, v_n\} n \text{ nets in the circuit} \quad (1.10)$$

$$O_2 = \sum_{j=1}^k CD_{vj} \quad \forall v_i \in \{v_1, v_2, \dots, v_k\} k \text{ nets in path } \pi \quad (1.11)$$

$$O_3 = \left\lceil \frac{\sum_{i=1}^n Width_i}{\# \text{ of cell rows in layout}} \right\rceil \quad (1.12)$$

Where, O_1 , O_2 and O_3 are taken as lower bounds for wire length, delay and width respectively.

The value g_i , where $i \in \{1, 2, 3\}$ (or $i \in \{wl, delay, width\}$), which is specified by the user allows increasing or decreasing the contribution of the criterion i to the overall cost because $\mu_i(x)$ is directly proportional to g_i .

1.4 Heuristics Applied to VLSI Placement

The number of blocks to be placed in a VLSI circuit is very large (can be in thousands or more) and this makes a brute force approach an impractical way for solving such a problem.

Heuristics applied to VLSI placement problem can be classified into *deterministic heuristics*, which move towards the solution by making deterministic decisions at each step, and *stochastic heuristics* which make controlled random decisions. Heuristics can be also classified as *constructive heuristics*, which construct the solution by placing one or few cells at a time, and *iterative heuristics* which start from an initial solution and modify it in several iterations to improve it with respect to an objective function.

Among the constructive deterministic heuristics applied to placement problem is the *maximum connectivity strategy* [1] and the *min-cut algorithm* [9, 22, 23]. *Force directed algorithm* is an iterative deterministic algorithm that was applied to placement [24]. Among the iterative stochastic heuristics are *Simulated Annealing* [25, 26, 27], *Genetic Algorithm* [28, 29], *Simulated Evolution* [30, 15] and *Tabu Search* [11, 31].

Constructive heuristics might trap the search at a solution which does not meet the constraints. For large problems, they usually lead to a solution that is far from the global optimum. In such a situation, numerous tedious manual modifications are needed and that is impractical in case of VLSI design [32]. Iterative heuristics, on the other hand, try to escape from local optima searching for a global one by accepting to climb hills in the search space through making non-improving moves at some stages of the search. In this thesis, we investigate *Tabu Search* as one of these iterative heuristics.

In this work, *Tabu Search* is applied to VLSI standard cell placement to optimize the wire length, the delay, and the total area of the final layout. Some standard benchmark circuits are used to test the performance of the proposed algorithm. The algorithm is parallelized on a network of stations including heterogeneous architectures and machines of varying speeds. Parallel Virtual Machine (PVM) is used to parallelize the algorithm. PVM is a system that makes a network of stations look like a single distributed machine for parallel computing using message passing [33] (see Appendix A). A diversification strategy for *Tabu Search* is proposed to make each process investigate a different area of the search space.

1.5 Organization of the Thesis

In this chapter, VLSI placement was introduced and defined and its complexity discussed. Different VLSI placement methodologies were addressed. The multi-objective placement problem was introduced. Three criteria for placement solution quality were discussed. Interconnection length, circuit delay and overall delay of the solution were explained. Combining conflicting objectives in one evaluation function using weighted sum and fuzzy logic was presented. Various heuristics that were applied to VLSI placement problem were presented and classified. The outline of the rest of the thesis was given.

Chapter 2 discusses *Tabu Search* and its main characteristics and parameters. The literature available on *Tabu Search* for VLSI placement problem and on *Tabu Search* Parallelization is reviewed in Chapter 3. The proposed *Parallel Tabu Search Algorithm for VLSI Standard Cell Placement* problem is explained in Chapter 4. Experimental results of the proposed methods and comparisons with other earlier approaches are presented and discussed in Chapter 5. Chapter 6 concludes the thesis.

Chapter 2

Tabu Search

Tabu Search (TS) is a general iterative metaheuristic that is used for solving combinatorial optimization problems. The heuristic is based on selected concepts from AI [32]. The rules used in *Tabu Search* are broad enough to make it applicable separately or as a guide for other heuristic procedures applied to combinatorial optimization problems [34].

A key feature of TS is that it imposes restrictions on the search process preventing it from moving in certain directions to drive the process through regions desired for investigation [35, 36]. An important component that enables TS to achieve the mentioned feature is the use of an adaptive flexible memory that distinguishes TS from other memoryless optimization heuristics. Various ways for using this memory are possible and applied for different objectives [34, 37, 36].

Tabu Search is a generalization of a local search. It searches for the best move in the neighborhood of the current solution. As opposed to local search, TS does not get trapped in local optima because it accepts bad moves if they are expected to lead to unvisited solutions [32].

2.1 Basic Tabu Search Algorithm

In its very basic operation, TS works as follows. It starts with an initial solution s that is selected randomly or using any constructive algorithm. It defines a subset $V^*(s)$ of its neighborhood $N(s)$. The algorithm evaluates the solutions in $V^*(s)$ and finds the best (in terms of the evaluation function) among them, call it s^* to be considered as the next solution. If the memory component (tabu list) used does not define the move leading to s^* as *tabu*, it is accepted as the new solution even if it is worse than the current solution in terms of the evaluation function. However, if the move leading to s^* is defined as *tabu* by the memory component, the solution is not accepted until it has one or more features that makes the algorithm override its tabu status to accept it. *Aspiration criterion* is used to check whether the *tabu* solution is to be accepted or not. The basic description of *Tabu Search* is shown in Figure 2.1 [32]. A flowchart of the basic algorithm is also shown in Figure 2.2.

```

X      : Set of feasible solutions.
s      : Current solution.
s*     : Best admissible solution.
C      : Objective function.
N(s)   : Neighborhood of  $s \in X$ .
V*     : Sample of neighborhood solutions.
TL     : Tabu list.
AL     : Aspiration Level.

1.      Start with an initial feasible solution  $s \in X$ .
2.      Initialize tabu lists and aspiration level.
3.      For fixed number of iterations Do
4.          Generate neighbor solutions  $V^* \subset N(s)$ .
5.          Find best  $s^* \in V^*$ .
6.          If move  $s$  to  $s^*$  is not in TL Then
7.              Accept move and update best solution.
8.              Update tabu list and aspiration level.
9.              Increment iteration number.
10.         Else
11.             If  $C(s^*) < AL$  Then
12.                 Accept move and update best solution.
13.                 Update tabu list and aspiration level.
14.                 Increment iteration number.
15.             EndIf
16.         EndIf
17.     EndFor

```

Figure 2.1: Algorithmic description of tabu search (TS).

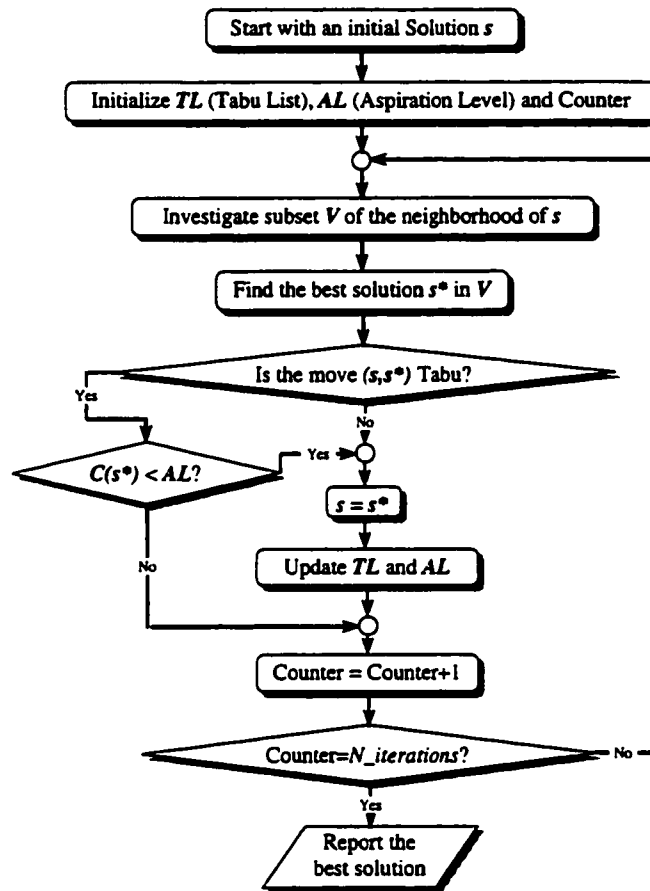


Figure 2.2: Basic tabu search algorithm flowchart.

2.2 Tabu Search Parameters

2.2.1 Candidate List

Examining all neighbors of the current solution is impossible in case of large neighborhoods. For this reason, a *candidate list*, which contains only a subset of all possible moves from the current solution, is used to select the next move to be made. It can be constructed by making random moves from the current solution.

Various strategies for constructing and examining candidate lists are given in [37, 38]. Some of these are:

- **Aspiration Plus:** In this strategy, moves from the current solution are examined until an empirically determined threshold of quality is achieved by a move. The strategy then continues for *plus* more moves. The best move among all examined ones is chosen as the new solution. In order to make sure that neither too few nor too many moves are tested, two limits *Min* and *Max* are put on the number of moves to be tested. The values for *Min* and *Max* are also empirically determined. The operation of this strategy is shown in Figure 2.3. As shown in the figure, moves are tried until a quality greater than 'aspiration' is found. A number of additional moves between the *Min* and the *Max* values is examined and the best is taken.
- **Elite Candidate List:** In this strategy, a master list is built by examining a large number of moves and selecting the *k* best moves among them, where *k* is a parameter of the implementation. At each iteration, the current best move from the master list is chosen. This continues until the quality of the move becomes less than a case-dependent threshold. At that point, a new master list is constructed and the process repeats. The strategy relies on the concept that good moves might have a good effect if not on the current solution, then on the coming ones. The operation of this strategy is shown in Figure 2.4. The master list

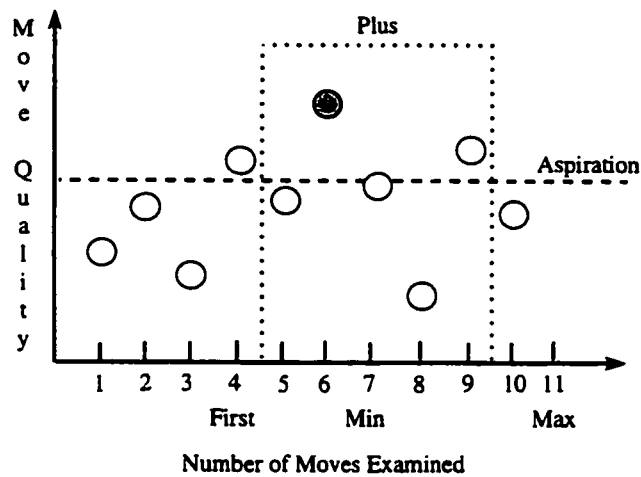


Figure 2.3: Aspiration plus strategy.

shown in the figure is constructed first. The best moves in that list are made one after another. When the next move to be made leads to a solution of a quality worse than the threshold, the list is rebuilt and the process continues.

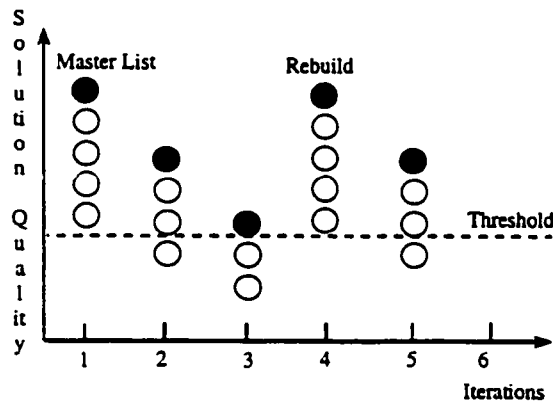


Figure 2.4: Elite candidate list strategy.

- **Successive Filter Strategy:** This strategy tries to improve the search efficiency by examining the effects of separate components of the move

operation on the solution quality. A single component of the operation is considered at a time and the best moves with respect to that component are chosen to join the candidate list. By doing this, a smaller set of possible combinations of changes are to be examined. For example, if several criteria are to be optimized by the move, moves improving a criterion at a time join the candidate list successively.

- Sequential Fan Candidate List:** This strategy is most suitable if parallel processing is used to construct the list. It works by generating p best alternative moves. Then a stream of moves is constructed from each of these moves. Again the best p moves of each stream are considered. The overall p best complex moves among all examined are the candidates to join the list. p and the depth of the operation are case-dependent. The operation of this strategy is shown in Figure 2.5. As shown in the figure, p moves are made first. From those moves, streams of moves are tried and only p streams of moves are maintained.

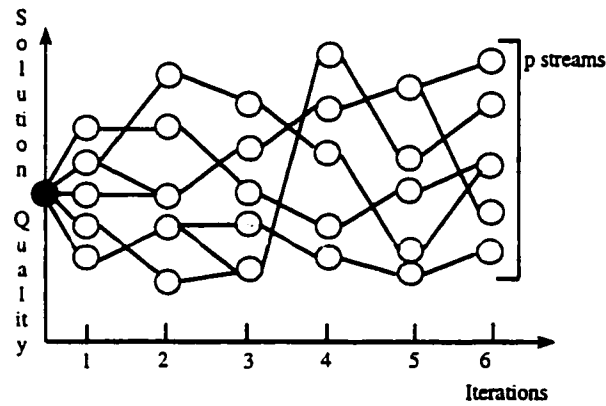


Figure 2.5: Sequential fan candidate list strategy.

- Bounded Change Candidate List:** In this strategy, the domain of moves is restricted such that no move is accepted if it causes a change that exceeds a certain limit in a given component of the solution. Such

a strategy allows intensifying the search in a specific area of the search space if the designer expects it to be a promising area.

2.2.2 Moves and Move Attributes

In order to construct the candidate list, neighbor solutions must be generated. These solutions are generated from the current solution by applying what is called a *move*. Swapping two elements in the solution configuration can be considered as a move. It can also be taken as dropping some edges from the graph representing the current solution (or adding some edges). A move can be composed of multiple modifications.

In order to utilize the memory concept employed by *Tabu Search*, the move attributes (or the reverse move attributes) are to be stored in a data structure that corresponds to the memory. Storing complete solutions is not feasible when the solution representation is large or complex. The infeasibility is in the storage required and the time needed for future manipulations. A list of possible attributes to be stored in the memory could be:

- Complementing the value of a selected binary variable x_i that represents a component of the current solution.
- Having a change in the cost that is equivalent to $Cost(s^*) - Cost(s)$, where s^* is the new solution and s is the current solution.
- Another function that depends on the problem under investigation might be used as basis for the difference mentioned above.
- Any combination of the above can also be used as an attribute [32, 39].

2.2.3 Evaluation Function

The whole process of *Tabu Search* Algorithm is performed in order to minimize/maximize a given function while satisfying given constraints of the

solution. The targeted function is called *Cost Function* or *Evaluation Function*. This cost might be the implementation cost like the interconnection length and overall area or the operation cost like the critical path delay as in the case of VLSI placement problem.

The cost function might have one or multiple objectives to minimize (or maximize) simultaneously. It might also minimize one variable under some constraints of another according to the user specification and application.

2.2.4 Tabu List

At any given point of time during the operation of the algorithm, there are n possible directions for future investigation of the search space, where n is the number of possible moves at that point and many directions might be overlapping. One of the objectives of *Tabu Search* is to prevent cycling back to previous solutions. Another objective might be to stay around the same neighborhood of solutions that have specific features (intensification). Moving to diverse solutions from the current one might be a third objective during (diversification) phase. In order to achieve these objectives, TS should use some sort of history.

For the purpose of not cycling back to recently visited solutions, reverse attributes of the accepted moves are made tabu for a specific number of coming iterations called Tabu Tenure or Tabu List size. To implement this, a queue of attributes can be used, or, an array whose entries are the remaining iterations during which the attribute will remain tabu where the array size is the number of attributes.

Tabu tenure depends primarily on the size of the problem as well as the objective of the search whether it is intensification, diversification or normal continuation of the search process. Generally, the size can be determined using experimental runs.

If the objective varies during various stages of the search, then a dynamic tabu list (variable tabu tenure) can be used [37]. The concept of a

variable tabu tenure showed better performance in general even when the objective of the search was constant as in [36, 40, 41]. In case of a variable tabu tenure, empirically determined bounds are used to control the change in the tenure.

2.2.5 Aspiration Criteria

Saving on storage and computation time achieved by storing only attributes of accepted moves is paid for in the accuracy. By preventing the complement of an attribute to occur right after that attribute, cycling back is prevented. However, preventing the reverse attribute of the n^{th} recent move might cause a state that was never encountered to be considered *tabu*. Of course, this contradicts with the algorithm initial objective that is defined as trying to investigate the unseen parts of the search space. In order to give the algorithm a chance of not losing good solutions because of this misuse of the tabu list, an aspiration criterion is used. This criterion is used to override the tabu status of a specific move attribute if the move leads to a solution that is desirable as far as the objective of the search at that stage is concerned.

The following are some of the possible aspiration criteria and their corresponding objectives:

1. **Global Objective Aspiration:** If the objective of the search is to continue normal operation searching for the best solution, then the aspiration criterion can be taken as achieving a cost that is better than all of those solutions encountered so far.

$$AC = Cost(s^*) < Best_Cost \quad (2.1)$$

2. **Regional Objective Aspiration:** The previous aspiration criterion can take a regional shape if the target is the best solution in the current neighborhood of size R .

$$AC = Cost(s^*) < Best_Cost(R) \quad (2.2)$$

3. **Aspiration by Search Direction:** If the current objective of the search is to traverse the search space towards a local optimum i.e., intensification, then aspiration by search direction can be used. In this case, if the tabu move is improving i.e., $Cost(s^*) < Cost(s)$ and the reverse of that move that caused the tabu status was also improving then the solution is considered to be satisfying the aspiration criterion. To implement this, a bit per entry in the tabu list can be used. If the move accepted is improving, the bit is set to 1. When the reverse attribute is found tabu but improving the bit is checked. If it is 1, the attribute is accepted. Otherwise, it is rejected.

$$AC = (improving == 1) \&\& (Cost(s^*) < Cost(s)) \quad (2.3)$$

4. **Aspiration by Influence:** If the current objective of the search is to go to a diverse point in the search space, then aspiration by influence can be used. In this method, a tabu move is considered to be satisfying the aspiration criterion if its reverse attributes that established the tabu status had a low influence and occurred before a considerable change in the current solution. To implement this aspiration technique, a bit per entry in the tabu list is used to represent influence. If the accepted attribute causes a change greater than L in the cost of the current solution, then influence bit is set. An array *Latest* of size 2 is used to store the latest iteration that had influence of level 0 and 1.

$$AC = ((influence(e) == 0) \&\& (tabu_start(e) < latest(L))) \quad (2.4)$$

where e is the tabu attribute and $tabu_start(e)$ is the iteration where e joined the tabu list. This can be generalized to include various levels of influence by increasing that size of the array *Latest* and satisfying that $influence(e) < L$ such that $tabu_start(e) < Latest(L)$ [32, 37, 39].

2.3 Tabu Search Classes

According to the way memory is used, *Tabu Search* can be classified into three categories:

1. *Short Term Memory Tabu Search*
2. *Intermediate Term Memory Tabu Search*
3. *Long Term Memory Tabu Search*

1. Short Term Memory Tabu Search

The *Tabu Search* algorithm where a list of attributes of n recently accepted moves is maintained to prevent accepting any of them in the next iteration until and unless it satisfies the aspiration criterion is called short term memory class. It is called so because it uses the memory as a recent short history to prevent cycling back to visited points. This class shows good performance where the search space is not of huge size. Otherwise, it might be trapped in a specific proximity of the search space.

2. Intermediate Term Memory Tabu Search

The main function of the intermediate term memory tabu search is to intensify the search by aggressively forcing some common features of good solutions in the new solution.

The algorithm records the common features between the most recent best m solutions or the most recent m local optima and forces the new solution to satisfy those features in order to be considered for acceptance. A move leading to a solution that does not satisfy these features may not be included in the candidate list.

3. Long Term Memory Tabu Search

It might be the objective during some stage of the search process to move to a diverse point in the search space. The motivation behind

this is the observation that the search has been trapped in a certain region i.e., several features have been repeated in most of the recent solutions. It might also be that the search has not been giving good results after running for a comparatively long time. A third motivation, in case of parallelization, might be to make sure that a set of processors are not searching in the same areas of the search space. One of the distinct features of tabu search is the fact that its memory component can be modified such that it supports this objective by making deterministic diversification steps. The frequently repeating features could be avoided by storing them in a special memory component.

In the diversification stage, the evaluation function can be modified to include a term that penalizes frequent moves as follows

$$F(s^*) = \begin{cases} Cost(s^*) & \text{if } Cost(s^*) \leq Cost(s) \\ Cost(s^*) + \alpha \times Freq(M(s^*)) & \text{if } Cost(s^*) > Cost(s) \end{cases} \quad (2.5)$$

where F is the evaluation function, α is a positive constant that depends on the range of the evaluation function values, the number of iterations, the history, etc. The value of α should be chosen such that cost and frequency are appropriately balanced. M is the move attribute that leads to the new solution s^* .

2.4 Summary

In this chapter, *Tabu Search* as a stochastic iterative heuristic to be applied to VLSI placement was presented. The basic algorithm was introduced. Various parameters of *Tabu Search* were discussed. *Candidate List* strategies were presented and explained. Different moves and move attributes were discussed. The evaluation function and the Tabu List were mentioned. Aspiration Criteria of different types were

presented. Finally, *Tabu Search* classes and their objectives were discussed.

Chapter 3

Literature Review

Section 3.1 of this chapter reviews the heuristics applied to VLSI placement problem. In Section 3.2, the application of *Tabu Search* to VLSI placement problem is reviewed in the literature. In Section 3.3, the parallelization of Tabu Search algorithm is addressed.

3.1 Heuristics Applied to VLSI Placement

Heuristics that have been applied to placement problem include *Maximum Connectivity strategy* which is a constructive deterministic strategy that works by placing a cell at each iteration. The selected cell for placement should have the maximum connectivity to the already placed cells. The complexity of this strategy is $O(n^3)$ [1]. The main disadvantage of it is that it can be easily trapped at a local optimum.

Another constructive deterministic heuristic that has been applied to placement is called *Min-cut Placement* [9, 22, 23]. This algorithm is based on the observation that minimizing the number of horizontal and/or vertical lines that are cut when the layout is partitioned in gate-array placement or standard cell placement improves the routability and reduces the number of feedthrough cells. This heuristic suggests partitioning the layout at each step and dividing the cells between the partitions such that the number of lines

cut by the partitioning is reduced. This heuristic also suffers from the local view of an optimum solution rather than a global view.

Force-directed Placement is a technique that belongs to the class of deterministic iterative heuristics. This heuristic starts from an initial solution and then starts taking a cell at a time and moving it to a more appropriate location according to its connectivity [24].

Iterative stochastic heuristics have been giving better results than constructive deterministic ones in placement problem. A reason for that is the global view of the optimum solution and the ability to escape from local optima through the controlled randomness introduced in these heuristics.

Among the iterative stochastic heuristics applied to placement is *Simulated Annealing* [25, 26, 27]. Simulated Annealing starts from an initial solution and keeps modifying the solution with random moves. It accepts a large portion of moves at the beginning of the search and then it becomes restricted to accept only improving moves towards the end.

Another heuristic which was applied to placement is *Genetic Algorithm* [28, 29]. It starts from a set of initial solutions called *initial population*. It then tries to combine features from good solutions to come up with better generations of solutions.

Simulated Evolution is another heuristic that was applied to placement [30, 15]. It depends on goodness of particular cells. At each iteration, a group of cells that are relatively badly placed are selected to be moved to better locations.

A common feature between all of the mentioned stochastic iterative heuristics is that they are memoryless. They don't utilize any memory structure to look at previous moves. On the other hand, *Tabu Search*, which was also applied to placement [31, 11], utilizes some memory to make decisions at various stages of the search process. This memory structure can be used to prevent reverses of recent moves by putting their attributes in a tabu list such that the algorithm does not cycle back to solutions that have been already investigated. It can be used to force the new solution to have different

features than previously seen solutions (*diversification*). It can also be used to force the new solution to have some features that have been seen in recent good solutions (*intensification*). It is up to the user to specify what it is that is required from *Tabu Search* at various stages of the search process.

3.2 Tabu Search in VLSI Placement

VLSI Placement is an NP-hard problem as discussed earlier. *Tabu Search* is a heuristic that was invented to be applied to NP-hard problems.

In [42], performance driven VLSI placement with global routing algorithm for Macro-Cell placement was presented. Quad partitioning approach was applied using *Tabu Search* Algorithm. The objective of the implementation was to minimize the delay.

Their algorithm obtains a placement for macro-cell layout with consideration of global routing. The overall delay is the main goodness criterion they used for a given solution. The overall delay is computed as the summation of the module delays and the interconnection delays in the path. Quad-partitioning is applied using *Tabu Search* and used weighted sum to find the overall cost of the solution considering imbalance of area usage, number of edges crossing boundaries, imbalance of edges passing boundaries, the largest net delay and the number of long paths in a region. Their results showed improvement in delay and run time compared to [43].

In [44], *Tabu Search* was applied to capacitor placement problem in a radial distribution system. Their design considered operating constraints of capacitor placement problem, load growth, the upper and lower bound constraints of voltage at different load levels, etc. The objective was to minimize the investment cost of capacitors and the system energy loss. The algorithm they applied was short term memory tabu search and the candidate list was generated randomly.

Their results showed improvements in energy and voltage loss. Their results were compared to previous results generated using Simulated Anneal-

ing [45, 46, 47] and the observation was that *Tabu Search* achieved the same quality of solution in much less time.

In [48], Short term memory tabu search was applied to placement of analog LSI chip designs. The objective was to minimize the overall interconnection length and layout area under the constraint of making routing an easy next step. *Tabu Search* was applied separately in one experiment and imposed on Genetic Algorithm in another experiment. Short term memory tabu search did not show good results when compared to Genetic Algorithm with dynamic adjustment of mutation rate.

In [11], *Tabu Search* was applied to quad-partitioning VLSI Macro-cell placement problem. The objective was to minimize the interconnection length which in turn minimizes the delay. The results showed that using fuzzy cost function provided up to 43% improvements in the cost.

The cost function applied was a fuzzy cost function that included delay and interconnection length. The results were compared to [42] and [43] and significant cost improvement over both was observed.

3.3 Tabu Search Parallelization

Saving in time to achieve the same quality of solution is the main goal behind distributing *Tabu Search* on a set of processors. Another goal can be achieving a better solution quality in the same time. The ideal improvement in investigation time is given by the following formula

$$ideal_speedup = \frac{T_n + T_u}{T_n/p + T_u} \quad (3.1)$$

where T_n is the sequential time for achieving the computation to be parallelized, T_u is the non-parallelizable part and p is the number of processors [34]. One way of parallelization is to distribute the component that requires the most CPU time. In *Tabu Search*, this is neighborhood examination component.

If TS behaves like a memoryless algorithm i.e., if the interarrival of a solution of a specific quality is exponentially distributed, then the probability of providing the desired solution quality during time t is given by

$$p(t) = 1 - e^{-\lambda t} \quad (3.2)$$

with $\lambda > 0$. In this case, it is *probabilistically* equivalent to perform one search for time t or p searches for time t/p [49]. If this is the case, many independent searches can be performed starting each one from a different initial solution.

According to Toulouse et. al taxonomy [37], a possible parallelization strategy of *tabu search* is to distribute the component that requires the most CPU time on available machines (*functional decomposition*). Another strategy is to perform many independent searches (*multi-search threads*). A third strategy, is to decompose the search space among processes (*domain decomposition*). A different taxonomy by Crainic et. al., classifies TS on the basis of three dimensions. The first dimension is **control cardinality** where the algorithm is either *1-control* or *p-control*. In a *1-control* algorithm, one processor executes the search and distributes numerically intensive tasks on other processors. In a *p-control* algorithm, each processor is responsible for its own search and the communication with other processors. **Control and communication type** is the other dimension where the algorithm can follow a *rigid synchronization (RS)*, a *knowledge synchronization (KS)*, a *collegial (C)*, or a *knowledge collegial (KC)* strategy. The third dimension is **search differentiation** where the algorithm can be *single point single strategy (SPSS)*, *single point different strategies (SPDS)*, *multiple points single strategy (MPSS)*, or *multiple points different strategies (MPDS)* [37].

In [50], a mechanism for parallelizing *tabu search* by dividing neighborhood examination among a number of slaves for solving flow shop sequencing problem was presented. A master process is used to send an initial solution to all slaves. At each iteration, each one of the slaves examines part of the neighborhood and reports the best move to the master process. The master

process chooses the best move among all and sends it to all slaves as the next move to be performed if it is not tabu. The process then continues for a fixed number of iterations or until no improvement is observed for a given number of iterations. The algorithm presented uses *domain decomposition* strategy. It is a *1-control*, *RS*, *SPSS* algorithm.

Garcia et. al [51] presented a parallel implementation of *tabu search* for vehicle routing problem. In their work, a master process applies *Tabu Search* and calls slaves which (with the master) investigate the neighborhood of the current solution. Each process identifies its best move and sends it to the master. The master process selects a set of the best moves and broadcasts them to all slaves. Processes exchange only sequence of moves rather than exchanging complete solutions which causes redundant communication overhead. The algorithm uses *domain decomposition* strategy. It is a *1-control*, *RS*, *SPSS* algorithm.

In [52, 53], evolution principles were included to improve parallel Tabu Search. In the given strategy, short term memory *tabu search* was applied on a set of machines. After a specific number of iterations, each machine exchanges best solutions with its neighbors. At each machine, If the received solution is better than the local best, it replaces it. The algorithm uses *multi-search threads* strategy. It is a *p-control*, *C*, *MPSS* algorithm.

Niar and Freville [54] proposed a parallel *Tabu Search* algorithm for the 0-1 multidimensional knapsack problem. In their algorithm, they had a master process that generates initial solutions for slaves. The initial solution for process i is taken as its previous best solution except in two cases: (1) if the quality of the best solution is less than a fraction α of the overall best solution or, (2) if the best solution of process i has not been modified for a given number of iterations. For each slave, the master generates a different strategy where the strategy is represented by the tabu list size, the number of local iterations and the number of successive drops at each iteration. The algorithm uses *multi-search threads* strategy. It is a *p-control*, *RS*, *MPDS* algorithm.

In [55], a parallel *tabu search* algorithm for voltage and reactive power control in power systems was presented. Two schemes were implemented in that work. In the first one, the neighborhood was decomposed for parallel processing at each iteration. This is a *domain decomposition strategy*. The algorithm is *1-control RS SPSS* search. In the second scheme, *tabu search* was replicated with various tabu lengths at different processes. This is a *multi-search threads* strategy. The algorithm is *p-control, RS, SPDS* search.

In this work, *tabu search* algorithm is parallelized on a network of heterogeneous stations using Parallel Virtual Machine (PVM) [33]. The proposed algorithm uses two strategies simultaneously. It parallelizes the candidate list construction and examination at the low level (*functional decomposition*), and also coordinates several concurrent searches synchronously (*multi-search threads*). The algorithm also imposes some restriction on processes such that *domain decomposition* is probabilistically applied. The algorithm falls into *p-control* class at the higher parallelization level and into *1-control* class at the lower level. It belongs to the *rigid synchronization* category on the **control and communication type** dimension. It is a *multiple points single strategy (MPSS)* search on the **search differentiation** dimension.

3.4 Summary

In this chapter, a literature review was presented for three topics. The heuristics applied to VLSI placement problem were discussed. The application of *tabu search* to VLSI placement problem was reviewed. Parallelization of *tabu search* algorithm was introduced and some previous work addressed. The proposed parallel algorithm was identified.

Chapter 4

Parallel TS for VLSI Standard-Cell Placement

In this chapter, we describe our parallelization of the TS algorithm for VLSI standard-cell placement. First, we present a sequential version of TS for the placement problem in Section 4.1. Section 4.2 addresses parallelization of the algorithm. Applying the algorithm in a heterogeneous environment is presented in Section 4.3. Diversification of the search in the algorithm is investigated in Section 4.4.

4.1 Proposed Tabu Search Algorithm

As an essential starting step for this work, *tabu search* was applied in its very basic operation to VLSI standard-cell placement.

The algorithm starts by reading a randomly generated initial solution and storing it in a data structure that stores each cell with its size, the row where it is initially located and its index in the row i.e., the number of cells placed before it in the same row. It then starts from that solution as an initial solution s . It considers it as the best solution and sets the aspiration level to its cost. For a fixed iteration count, that is proportional to the number of cells in the circuit, the algorithm runs the simple short-term memory *tabu*

search.

The algorithm constructs a *candidate list* by examining a number of moves. A *move* is taken as swapping the locations of two cells that are generated randomly. The move attribute stored in the *tabu list* is the numbers of the swapped cells. The change in the cost is computed after relocating the two cells. N_v cells are tried for swapping and the best swap is considered for the next move. A compound move is made where N_v other moves are tested for d times, where d is the desired move depth, and the best move is taken each time. N_v and d are functions of the circuit size. By doing this, the algorithm applies *Sequential Fan Candidate List Strategy* with one stream (see Section 2.2.1). When the algorithm is parallelized it will be generating p streams, where p is the number of processes investigating the neighborhood of the current solution s . If the move gives improvement over the cost of the current solution, then the process does not need to proceed further in depth. It just considers the reached solution as the best it found and gives it to the algorithm as the next neighbor of the current solution.

If any of the cells to be swapped has been moved in the recent history, then the move is considered *tabu*. The algorithm checks if the move is *tabu* by considering only the two cells that were swapped first in the compound move. If the move is found *tabu*, the *aspiration criterion* is checked. If the move satisfies it, it is accepted. Otherwise, it is rejected and the process repeats.

The *tabu tenure* (tabu list size) used is a parameter of the circuit size. An array of size equal to the number of cells in the circuit is used. Whenever a swap is accepted, the swapped cells entries in the array are updated with the value of the *tabu tenure*. All other non-zero entries are decremented by one. When a *tabu* status of a cell is to be checked, its entry in the array is read. If it is non-zero, then the associated move is considered *tabu*.

The aspiration criterion used in the algorithm is *Aspiration by Objective* (see Section 2.2.5). If the *tabu* move leads to a solution whose cost is better than all of those seen so far, then the algorithm overrides the *tabu* status

and the move is accepted. This is a quite restricted criterion especially for huge search spaces. It is to be relaxed to a *Regional Aspiration by Objective* when the algorithm is parallelized as discussed in Section 4.2.

The cost function used by the algorithm is the same as the one proposed by Ali [15]. It computes the wire length, the circuit delay, and the width of the layout. It then computes the overall cost using *Fuzzy Goal Based Cost Measure*. The optimum values for wire length, delay and width are computed for each circuit as explained in Section 1.3.4. The goal values are taken as 2.0-3.0, 3.0-4.0, and 1.1-1.5 for wire length, delay and width respectively. It was experimentally found that the best combination of weights for weighted sum cost computation is 0.6, 0.1 and 0.3 for wire length, delay and width respectively [15]. Those weights are used in the experiment where weighted sum is applied.

4.2 Parallelization of the Algorithm

Tabu search is parallelized on a network of stations using PVM (see Appendix A). The purpose of parallelization in this work was saving some of the time required to achieve a certain solution quality (search speedup) and to achieve a better quality of solution in the same search time.

The algorithm is parallelized on two levels simultaneously. The higher one is at the *tabu search* process level where a master starts a number of *Tabu Search Workers* (*multi-search threads*) and provides them with the same initial solution. The lower level is the *Candidate List* construction level where each TSW starts a number of *Candidate List Workers* (*functional decomposition*). The paradigm of the parallel algorithm is shown in Figure 4.1. The algorithm falls into *p-control* class at the higher parallelization level and into *1-control* class at the lower level. It belongs to the *rigid synchronization* category on the *control and communication type* dimension. It is a *multiple points single strategy (MPSS)* search on the *search differentiation* dimension because TSWs diversify from the initial solution at each global iteration using

the diversification scheme proposed by Kelly et. al [56] (see Section 4.4).

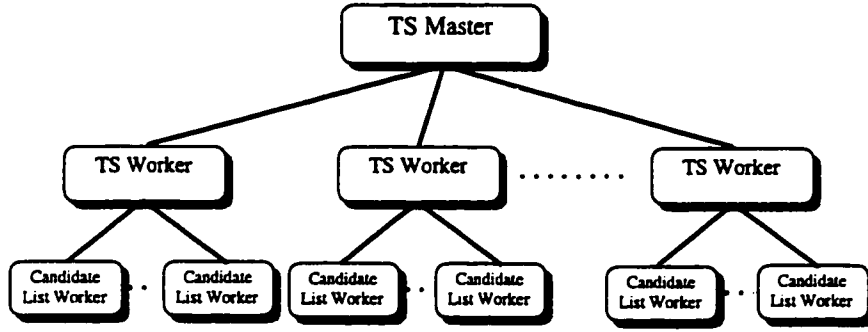


Figure 4.1: Paradigm of tabu search parallel implementation.

Because of memory component used by TS, the algorithm has to exchange information other than the best solution. This is unlike memoryless iterative algorithms such as SA, GA, and SE where best solution is all that is required to be exchanged. The algorithm proposed communicates the best solution as well as the associated *tabu list* between the master and the TSWs.

The parallel algorithm works as follows. The master starts the process by initializing the data structures, the tabu list and the aspiration level. It also reads the initial solution from a file. After that, it starts a number of *Tabu Search Workers* (TSWs) to perform *Tabu Search* starting from the given initial solution. The implementation has an option to start from the initial solution or from a previous best solution. It sends all parameters of the search to the workers and tells each worker about a range of cells within which that worker should diversify from the initial solution (diversification is discussed in Section 4.4). It broadcasts the same initial solution to all workers. Each TSW then initializes the aspiration level and the tabu list. After that it performs a diversification step. The TSW then starts a number of *Candidate List Workers* to investigate the neighborhood of the current solution. It sends the parameters and the initial solution to each CLW. It also gives each CLW a range of cells to search the neighborhood with respect to those cells as follows. The TSW divides the number of cells n by the k CLWs it has started. Each CLW gets the n cells of the circuit and is informed

about the $\frac{n}{k}$ cells belonging to its range.

For every move it makes, the CLW has to choose one of the cells from its range and the other as any other cell. The probability that it take the other cell from its range is $\frac{1}{k}$. The probability that two *Candidate List Workers* performing the same move is the probability that the 1st CLW chooses its second cell as a specific cell from the range of the other CLW and the other CLW does the same. The probability that two CLWs select the same two cells is given by $\frac{1}{n-1} \times \frac{1}{n-1} = \frac{1}{(n-1)^2}$. The probability that more than two CLWs select the same two cells is 0.

Note that we should not restrict each CLW to choose both cells from its range because that will cause a big portion of the neighborhood to be ignored without investigation. In our case, to choose two cells for swapping from n cells, we have $\binom{n}{2} \times \binom{n-1}{1} = (n-1) \times \frac{n}{2} = \frac{n(n-1)}{2}$ distinct choices for each CLW. For k CLWs, we have $k \times \frac{n(n-1)}{2} = \frac{k n(n-1)}{2}$ choices. If we restrict each one of k CLWs to choose only from a range of $\frac{n}{k}$ cells, then the number of choices will be $\binom{n/k}{2} \times k$. Take a small case where the number of cells $n = 100$ and the number of CLWs $k = 4$. Then $100 \times (100 - 1) = 9900$ and $\binom{100/4}{2} \times 4 = 1200$. This implies that the number of ignored swaps is $9900 - 1200 = 8700$ which makes 88% of the whole neighborhood.

The restriction imposed by our scheme that one cell must be taken from the range belonging to the CLW implies that the probability that the two selected cells are taken from the same range is given by $\frac{k-1}{n-1}$. Let us assume that the restriction is not imposed. In this case, the probability that the two selected cells belong to the range of the CLW is given by $\frac{k}{n} \times \frac{k-1}{n-1} = \frac{k(k-1)}{n(n-1)}$. This means that by imposing our restriction, the probability of choosing the two cells from the range of the CLW is increased by the factor $\frac{n}{k}$. Equivalently, the probability of choosing one cell from outside the range is reduced by $\frac{n}{k}$. Outstanding moves might peacefully reside in the area of reduced probability of selection. This explains why it is not always good in our scheme to keep increasing the degree of parallelism. The effect might not only be in the efficiency but also in the quality of the solution as some experiments show in

Chapter 5.

Each CLW makes a compound move of a predetermined depth and keeps computing the gain. If the current cost is improved before reaching the maximum depth, the move is accepted without further investigation. Otherwise, the move is completed to the maximum depth hoping to climb the hill that is currently faced. At each point towards the maximum depth, a number N_v of neighbors is checked. After finding the compound move that improves the cost the most (or degrades it the least), the *Candidate List Worker* sends its best solution to the TSW that started it. The TSW gets the best solution from the CLW that achieves the maximum cost improvement (or the least cost degradation). It then checks if the move is tabu as mentioned in Section 4.1. If the move is not tabu, it is accepted. Otherwise, the cost of the new solution is checked against the *aspiration criterion*. If it satisfies it, it is accepted. Otherwise, it is rejected and the process continues for a number of local iterations between the TSW and its CLWs. At the end of the local iteration count, each TSW sends its best cost to the master process. The master asks the TSW who got the overall best solution to report it. It then broadcasts that solution to all TSWs who perform their diversification steps and the process continues for a number of global iterations. An algorithmic descriptions of the master and the worker processes of the parallel implementation of *Tabu Search* are given in Figures 4.2 and 4.3 respectively. An abstract scenario of the complete process is shown in Figure 4.4.

4.3 Applying the Algorithm in a Heterogeneous Environment

Normally, a network of workstations is composed of heterogeneous machines. Heterogeneity can be of various types like machine architecture, data for-

N_i : Number of iterations.
 X : Set of feasible solutions.
 bs : Current best solution.
 bc : Current best cost.
 TL : Tabu list.
 N_w : Number of workers.

1. Start with an initial feasible solution $bs \in X$.
2. Initialize TL and bc .
3. Spawn N_w workers to perform Tabu Search.
4. Send(bs, TL, bc) to all workers.
5. **For** N_i **Do**
6. Wait for best cost from all workers.
7. Ask for bs and TL from the worker that has the overall best.
8. Receive(s, TL).
9. Update bc .
10. Send(bs, TL, bc) to all workers except sender.
11. Increment iteration number.
12. **EndFor**

Figure 4.2: Algorithmic description of master process of parallel TS.

N_i : Number of iterations.
 X : Set of feasible solutions.
 s : Current solution.
 s^* : Best admissible solution.
 bs : Current best solution.
 C : Objective function.
 $N(s)$: Neighborhood of $s \in X$.
 V^* : Sample of neighborhood solutions.
 TL : Tabu list.
 AL : Aspiration Level.

1. Receive(s, TL, AL) from master.
2. **For** N_i **Do**
3. Perform a diversification step.
4. Apply short term memory TS for a fixed number of iterations.
5. Send AL to master.
6. **If** the master asks for bs **Then**
7. Send(bs, TL) to master.
8. **Else**
9. Receive(bs, TL, AL) from master.
10. $s = bs$.
11. **EndIf**
12. **EndFor**

Figure 4.3: Algorithmic description of worker process of parallel TS.

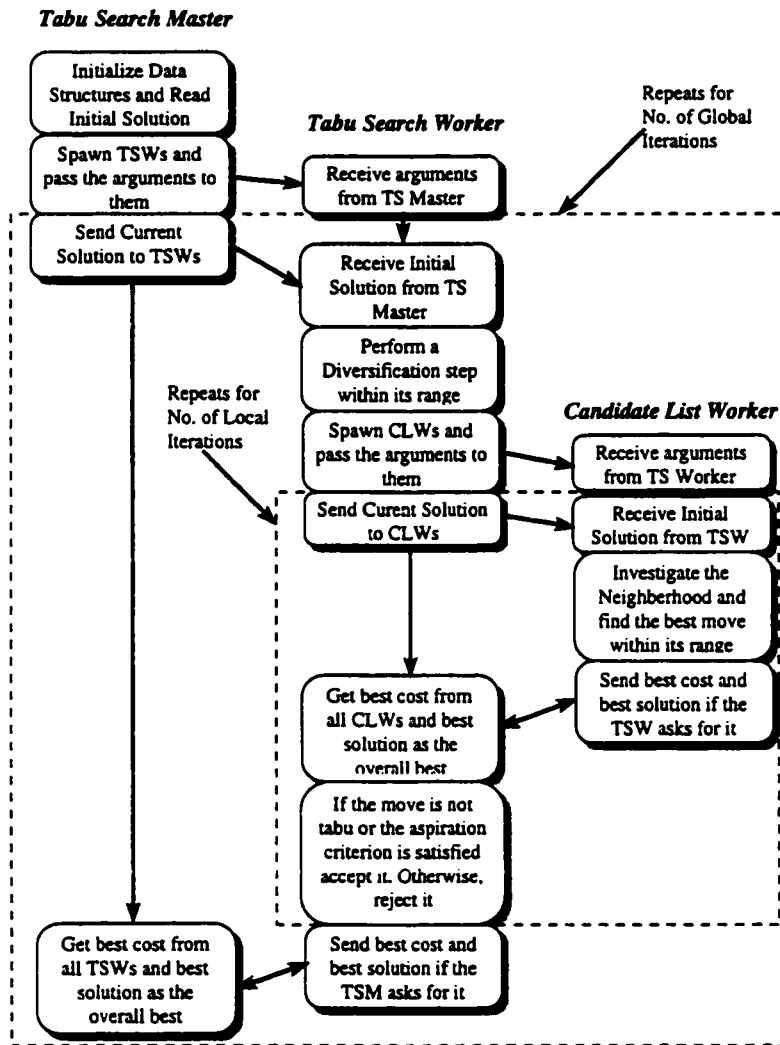


Figure 4.4: A scenario of the *master*, the *TSW* and the *CLW* processes.

mat, computational speed, network type, machine load, and network load. PVM can take care of machine architecture heterogeneity and data format conversion.

In our implementation of parallel *tabu search*, we account for speed and load heterogeneity by letting the master receive the best cost from any *Tabu Search Worker* that has finished the local iterations. Once the number of *Tabu Search Workers* who gave their best cost to the master reaches half the total number of TSWs, the master sends a message to all other *Tabu Search Workers* asking them to report whatever best cost they have achieved. *Tabu Search Workers* check for such a message in their buffers frequently (every 10 iterations). Once they receive the message, they kill the currently running *Candidate List Workers* and report to the master with their best achieved costs.

The same principle applies in the communication between *Tabu Search Workers* and their own *Candidate List Workers* who check for a message from their parents frequently. That message either kills them, if it is the TS master who is asking the TSW to report, or asks them for their best achieved solutions if half of the CLWs have reported their best. By applying this principle, machine load, machine speed and network load heterogeneity are accounted for.

Our experiments are tested on three different speed levels of machines and four different architectures. These architectures are IPX/SPARC, Sparc-Station 10, LX/SPARC and UltraSparc 1. All machines have the same operating system (which is Solaris 2.5).

4.4 Diversification of the Search Process

Diversification is the process of driving the search into regions that have not been visited before. As mentioned in Section 2.3, one of the motivations behind diversification is to make sure that a set of processes in a parallel implementation are not applying *tabu search* in the same region of the search

space.

A simple strategy for diversification is to penalize frequent moves as shown in Equation 2.5. This strategy is acceptable if the search is sequential. In case of several parallel processes, this strategy will give the same evaluation function value in all processes. In that case, all processes may end up searching in the same region.

In this work, we applied a deterministic diversification strategy that was first applied by Kelly et. al., to Quadratic Assignment Problem (QAP) [56]. In this strategy, local search with pair-wise swapping is applied until a local minimum is reached. After that, one of the following two forms of diversification is used:

1. **First Order Diversification:** This diversification scheme is applied to take the search to a point that is maximally diverse from the recent local minimum. This diversity is taken with respect to distance i.e., the number of moves needed to go from one solution to another. If the most recent minimum solution is given by the permutation $\Pi_{min} = \{\pi_{min}(1), \pi_{min}(2), \dots, \pi_{min}(n)\}$ and the current solution has the permutation $\Pi_{cur} = \{\pi_{cur}(1), \pi_{cur}(2), \dots, \pi_{cur}(n)\}$, then all swaps $\pi_{cur}(x) \leftrightarrow \pi_{cur}(y)$ such that $\pi_{cur}(x) = \pi_{min}(x)$ or $\pi_{cur}(y) = \pi_{min}(y)$ are considered. The swap that leads to highest improvement (or least degradation) in the objective function is performed. Such diversifying moves are made until no more moves belonging to the specified category exist.
2. **Second Order Diversification:** Since local search is a greedy approach that considers only the highest improvement in cost at each iteration, moving from a solution to another solution that has a close or worse objective function value has a low probability. This diversification scheme is presented to allow moving between such solutions. The diversity in this scheme is taken with respect to difficulty of reaching from a solution to another. To apply this scheme, they use a matrix M whose entries m_{ij} count the number of times object i occupied location

j in the history; m^* represents the number of local optima encountered so far. Another matrix M^* whose entries are $m_{ij}^* = m_{ij}/m^*$ is maintained too. Two possible uses of M and M^* are presented in [56].

Restarting: In this technique, a cutoff rule is used to stop first-order diversification. Another starting solution is generated by solving a linear assignment using the coefficients $c_{ij} = m_{ij}$. After that, first-order diversification resumes until meeting the cutoff rule condition again.

Periodic Second-Order Evaluation: In this technique, the first-order diversification is terminated when a given number of local optima is encountered. In the next sequence of moves, the objective function is changed to minimize the quantity $m_{iy} + m_{kx}$ where $i = \pi_{cur}(x)$ and $k = \pi_{cur}(y)$. The idea is to move objects out of high frequency locations.

In this work, the first order diversification is applied because it can be modified to give a chance to every TSW to diversify in a different direction. In our work, every time a TSW receives a solution from the master it diversifies from it according to the range that was assigned to it by the master. Within that range, it performs swaps to a predetermined diversification depth. At every swap, it makes N_v trials and accepts the most improving (or least degrading) one.

The first cell to be swapped has to be from the diversification range but the second does not have to. A condition for the swap to be made is that the new locations of the cells have to be different from their locations in the initial solution.

4.5 Summary

In this chapter, we presented a detailed description of our parallel TS algorithm for standard cell placement. Parallelization is achieved at two levels. The parallelization scheme used for the algorithm was addressed with a full

scenario of the parallel process. In order to minimize subspaces searched by various concurrent processes, each process is confined to work with a subset of the cells. Applying the algorithm in a heterogeneous environment was presented and the scheme used to account for speed heterogeneity was discussed. Diversification of the search in the algorithm was investigated and the proposed scheme for diversification in different directions was explained.

Chapter 5

Experimental Results

In this chapter, we present and discuss various experiments that are performed using the proposed parallel *tabu search* algorithm for VLSI standard cell placement. In Sections 5.1 and 5.2, we study the effect of the degree of low-level and high-level parallelization on the algorithm performance, namely quality of best solution and speedup. The definition of speedup for non-deterministic algorithms such as TS is different from that used for deterministic constructive algorithms. For this category of algorithms, speedup is defined as follows

$$Speedup_{(n,x)} = \frac{t_{(1,x)}}{t_{(n,x)}} \quad (5.1)$$

where $t_{(1,x)}$ is the time needed to hit an x -quality solution using one *Candidate List Worker* (or *Tabu Search Worker*) and $t_{(n,x)}$ is the time needed to hit the same solution quality using n CLWs (or TSWs). $Speedup_{(n,x)}$ in this case can be greater than n because investigating the neighborhood with n CLWs (or making n independent searches) might cause the stochastic search to hit an x -quality solution more than n times faster. The efficiency is defined as follows

$$Efficiency_{(n,x)} = \frac{Speedup_{(n,x)}}{n} \quad (5.2)$$

where n is the number of CLWs (or TSWs) used to hit an x -solution quality. $Efficiency_{(n,x)}$ in this case can be greater than one. In Section 5.3, we

compare the results of an implementation that accounts for heterogeneity to an implementation that does not. In Section 5.4, we see the effect of diversification performed by *Tabu Search Workers* by comparing the results to a run of the algorithm where diversification is not performed. In Section 5.5, an experiment is conducted to check whether the algorithm is memoryless or not. In Section 5.6, we compare the performance of the algorithm using fuzzy cost evaluation to the performance using weighted sum technique. In Section 5.7, we compare our results to the results of previous work done on the same circuits using Simulated Evolution.

In our experimentation, we used four *ISCAS-89* benchmark circuits. Table 5.1 shows the number of cells, the number of IOs, the number of cell rows, the layout height, the routing channel height, the optimum wire length, the optimum delay, and the optimum width for all circuits used.

Circuit			Layout					
Name	Cells	IOs	Rows	LH	Avg. RCH	O_{wl}	O_{delay}	O_{width}
highway	56	11	3	284	55.0	7156	3.91	512
fract	149	24	5	556	66.5	28207	7.97	784
c499	283	73	6	750	80.4	43583	8.17	1176
c532	395	43	7	703	49.5	64674	17.83	1152
c880	784	86	9	1034	64.0	123616	16.8	1824
c1355	1451	73	13	1557	66.9	273138	13.62	2304
struct	1952	64	15	2102	88.0	432096	13.54	3280
c3540	2243	72	17	2480	93.4	500157	23.26	3096

Table 5.1: Characteristics of circuits and layouts used. (LH = layout heights and $Avg. RCH$ = average routing channel height in microns).

5.1 Effect of Degree of Low-level Parallelization

In this experiment, different numbers of CLWs are tried starting from 1 to 4 for each circuit. The change in the best solution quality is monitored as the number of CLWs is changed. All other algorithm parameters are fixed. The number of TSWs is 4 in all experiments. 12 machines are used as a parallel virtual machine. The number of global iterations, number of local iterations, N_v , move depth, and T-tenure are all fixed as in Table 5.2.

Name	Cells	GI	LI	N_v	Move Depth	Div_Depth	T_Tenure
highway	56	500	37	7	3	7	7
fract	149	500	61	12	3	18	7
c499	283	160	70	17	4	70	8
c532	395	120	100	20	4	100	9
c880	784	120	200	28	5	200	9
c1355	1451	80	120	9	2	60	9
struct	1952	80	120	9	2	60	10
c3540	2243	80	120	9	2	60	10

Table 5.2: Parameters of parallel experiments. (GI = Global Iterations and LI = Local Iterations).

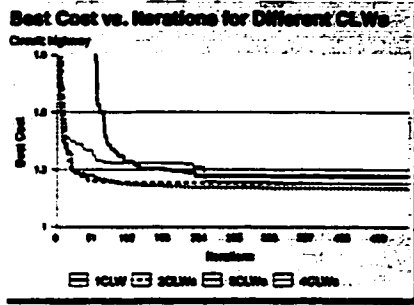
Figure 5.1 shows the effect of changing the number of CLWs on the best solution quality for all circuits. For most of the circuits, it is clear that increasing low level parallelization degree is beneficial. The cost used here is the fuzzy cost mentioned in the introduction based on the optimum values shown in Table 5.1 for wire length, delay and width. For *highway*, the circuit size is small. That makes adding CLWs beyond 2 not useful. Using 4 CLWs gives bad solution quality. A reason for this is the restriction imposed on choosing cells when more CLWs are used as mentioned in Section 4.2. For

c880, adding CLWs beyond 2 did not change much the quality. The reason is that the starting solution quality was good enough where adding CLWs did not have much to do.

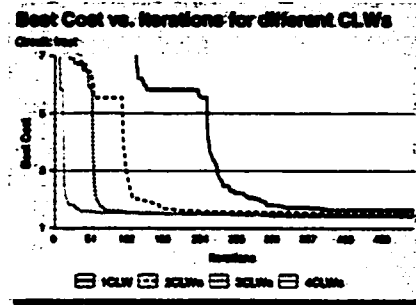
Figure 5.2 shows the time needed to achieve a specific solution quality for all circuits. This quality is chosen according to the best quality achieved by the worst strategy. Adding more CLWs resulted in reaching better solutions in less time except for *highway* and *c499* where increasing the number of CLWs to more than 2 was causing communication overhead more than speeding up the search.

Figure 5.3 shows the speedup achieved in reaching a specific solution quality for all circuits. It is clear from the figure that in most of the experiments, as the number of CLWs increases from 1 to 4, the speedup increases. The sharpness of the speedup increase depends on the circuit size and the goodness of the initial solution. For *fract* and *c532*, the initial solution is too far from the best reached. As a result, increasing the number of CLWs results in a sharper change in the speedup. For *struct*, the same behavior was observed because the circuit size is large. For *c880* and *c1355*, the circuit size is moderate and the initial solution is not that far from the best reached as in the case of *fract* and *c532*. As a result, the rate of change in the speedup goes down as the number of CLWs is increased. In all experiments except *highway* and *c499*, the critical point, where the speedup starts to degrade, is not reached but it is clear in some curves that it is being approached. For *highway* and *c499*, the critical point occurred at 2 CLWs because the 2 circuits are small.

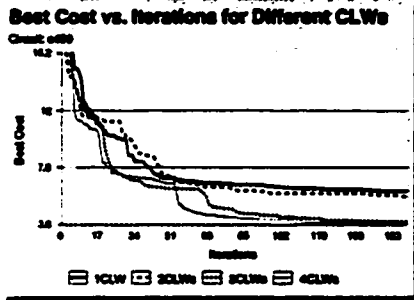
Figure 5.4 shows the efficiency of using more CLWs in reaching a specific solution quality for all circuits. The highest efficiency point differs from a circuit to another according to its size, complexity and initial solution. For *highway*, *c499*, and *c880*, the highest efficiency point occurred at 2 CLWs. For *c1355*, and *c3540*, it occurred at 3 CLWs. For *fract*, *c352*, and *struct*, it occurred at 4 CLWs or more.



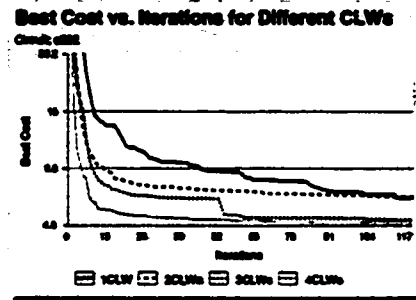
(a)



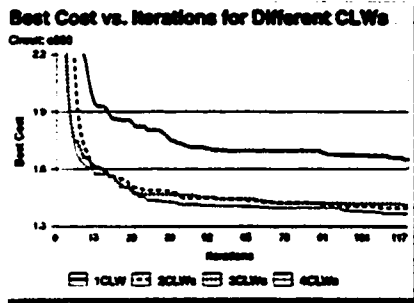
(b)



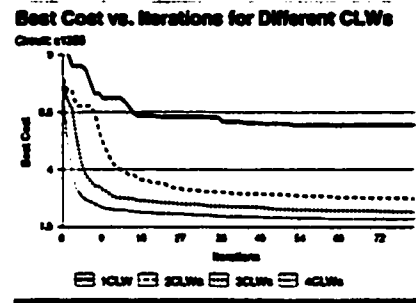
(c)



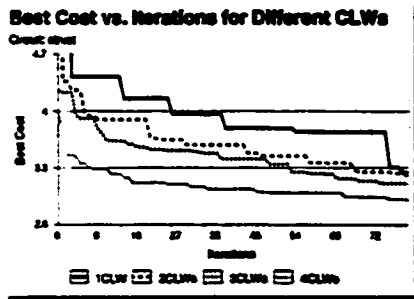
(d)



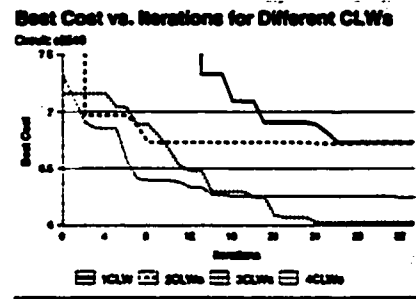
(e)



(f)

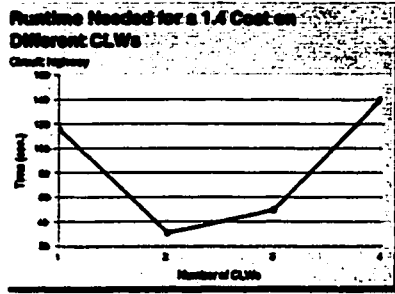


(g)

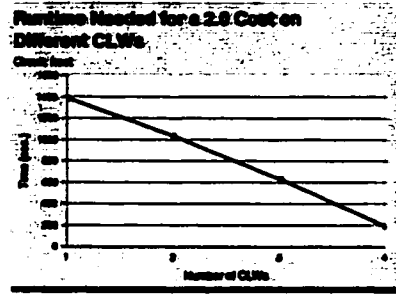


(h)

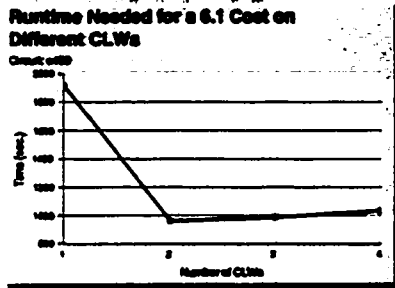
Figure 5.1: Effect of number of CLWs on the solution quality.



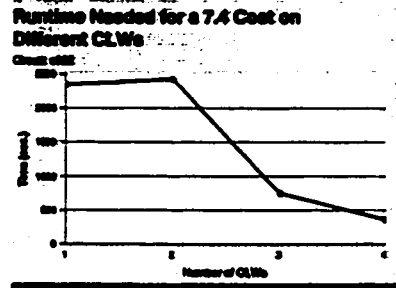
(a)



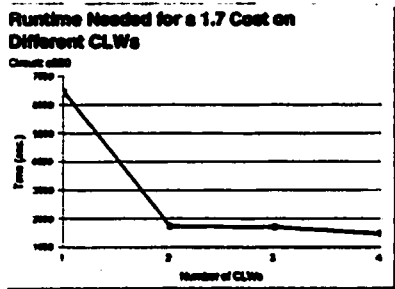
(b)



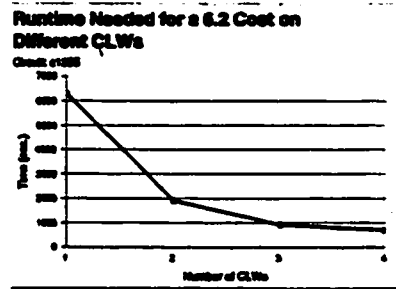
(c)



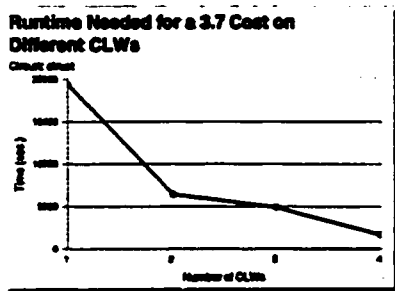
(d)



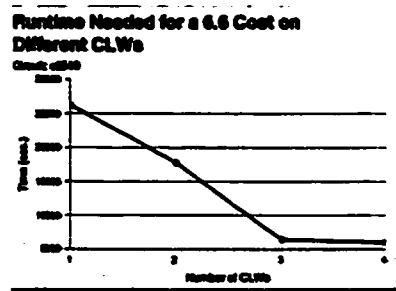
(e)



(f)

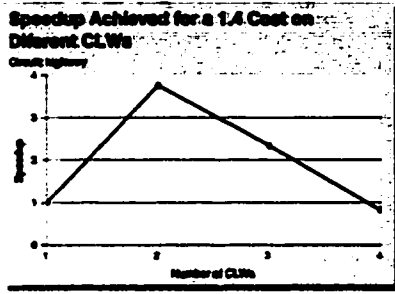


(g)

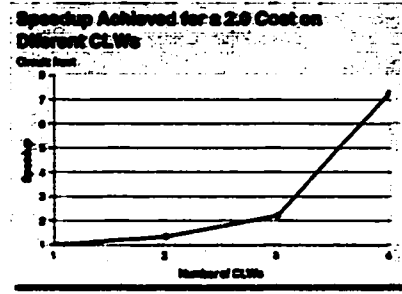


(h)

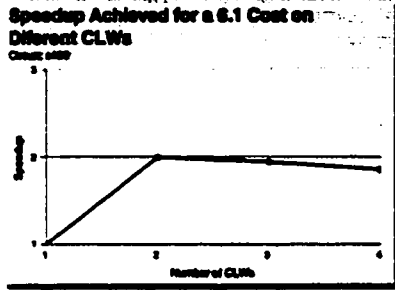
Figure 5.2: Runtime needed to achieve a solution of cost less than x for different numbers of CLWs.



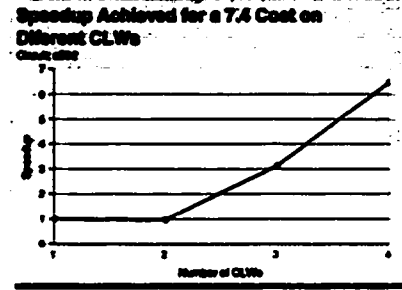
(a)



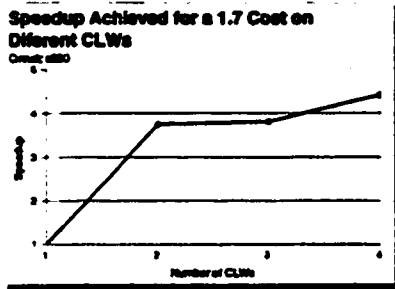
(b)



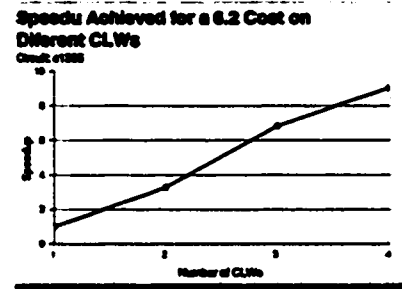
(c)



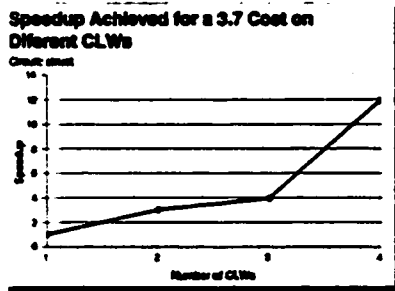
(d)



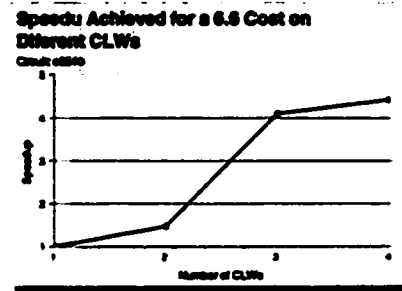
(e)



(f)

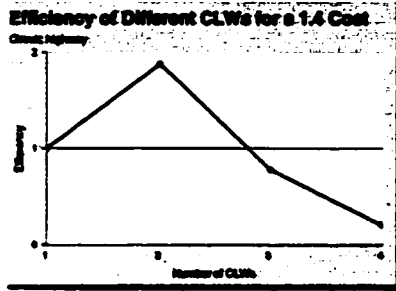


(g)

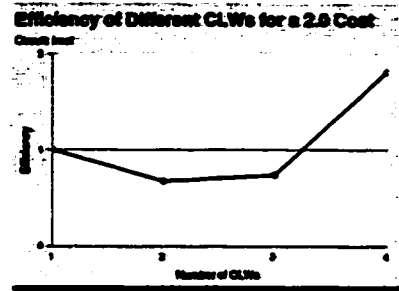


(h)

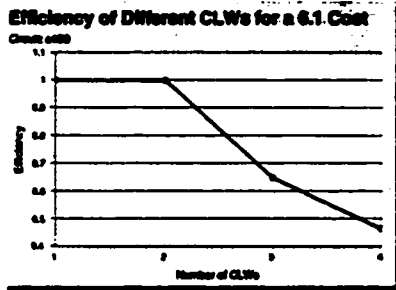
Figure 5.3: Speedup achieved in reaching a solution of cost less than x for different numbers of CLWs.



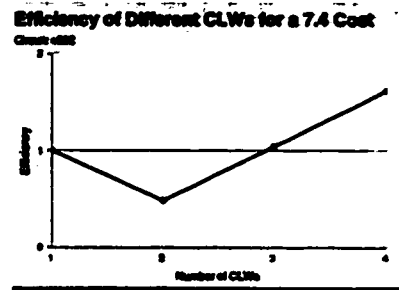
(a)



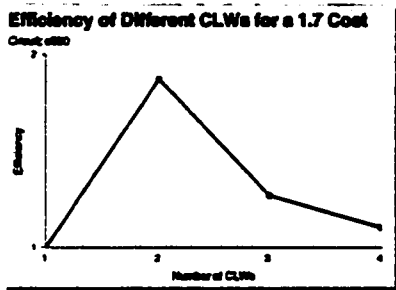
(b)



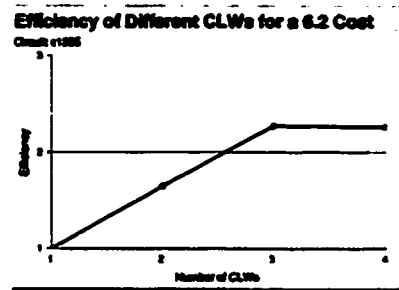
(c)



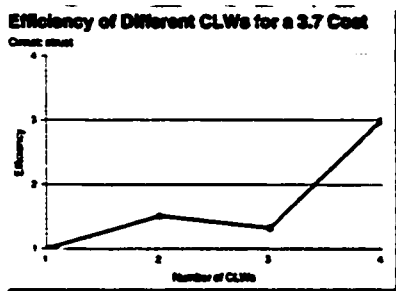
(d)



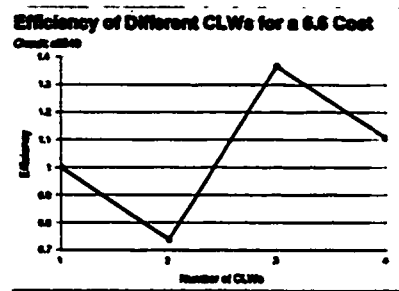
(e)



(f)



(g)



(h)

Figure 5.4: Efficiency of using more CLWs in reaching a solution of cost less than x .

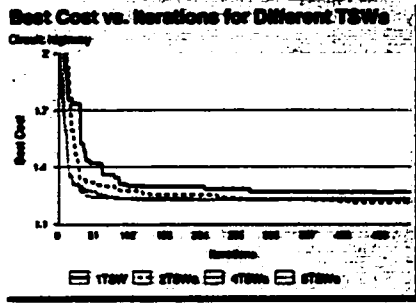
5.2 Effect of Degree of High-level Parallelization

In this experiment, Different numbers of TSWs are tried starting from 1 to 8 for each circuit. The change in the best solution quality is monitored as the number of TSWs is changed. The number of CLWs is fixed to one in all experiments. 12 machines are used as a parallel virtual machine. The number of global iterations, number of local iterations, N_v , move depth, and T-Tenure are all fixed to the values shown in Table 5.2.

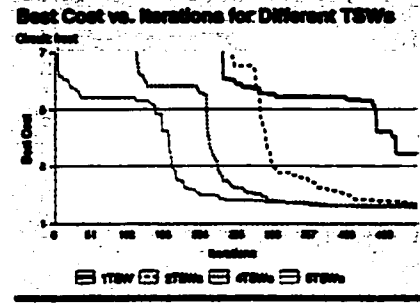
Figure 5.5 shows the effect of changing the number of TSWs on the best solution quality for all circuits. For *fract*, *c499*, *c1355*, *struct*, and *c3540*, it is clear that using more TSWs gives better solution quality in all iterations. For *highway*, the circuit size is small. This makes adding *Tabu Search Workers* beyond 4 not useful. For *c532*, Using 8 TSWs started to give good solutions. However, as iterations continued, it gave similar solutions to 4 TSWs. That is because the circuit needed more investigation and less diversification. For *c880*, adding TSWs beyond 2 degraded the quality. The reason is that the starting solution quality was good enough where adding TSWs caused the search to diversify to undesired regions.

Figure 5.6 shows the time needed to achieve a specific solution quality for all circuits. This quality is chosen according to the best quality achieved by the worst strategy. Adding more TSWs proved to be beneficial with respect to runtime except for *highway*, *fract*, *c532*, and *c3540* where increasing the number of TSWs to more than 4 was causing communication overhead more than speeding up the search. For *c880*, even two TSWs were not worth being added because the starting solution was good.

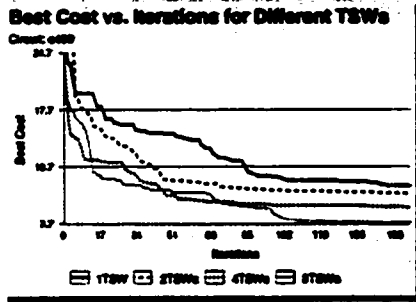
Figure 5.7 shows the speedup achieved in reaching a specific solution quality for all circuits. For *highway*, *fract*, *c532*, and *c3540*, the critical point, occurred at 4 TSWs. Adding more TSWs degraded the speedup. For *c880*, one TSW was enough. For the other three circuits, the critical point was approached but not yet reached.



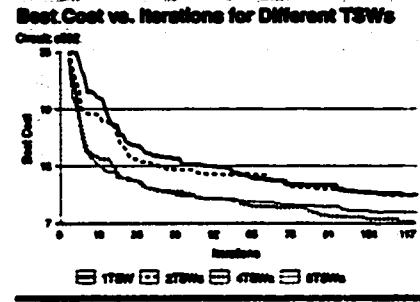
(a)



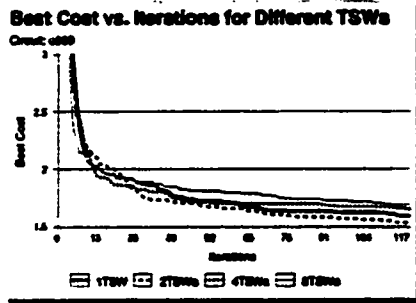
(b)



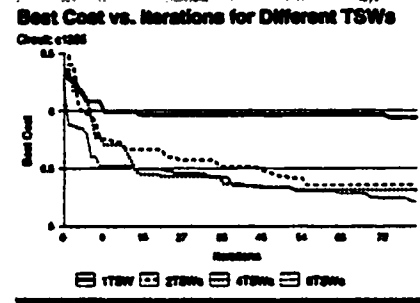
(c)



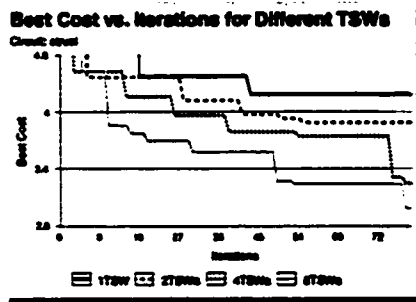
(d)



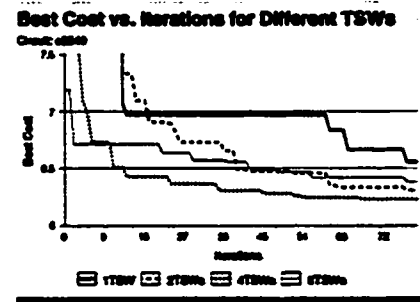
(e)



(f)

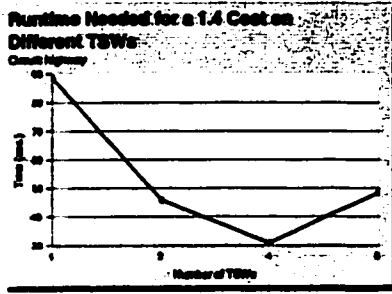


(g)

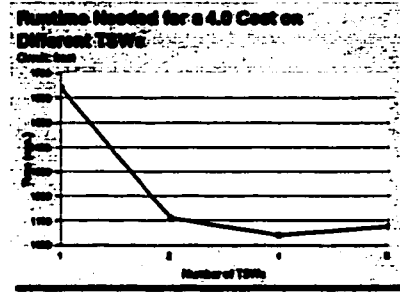


(h)

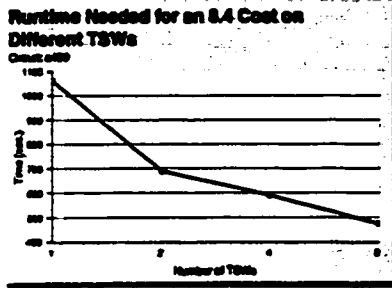
Figure 5.5: Effect of number of TSWs on the solution quality.



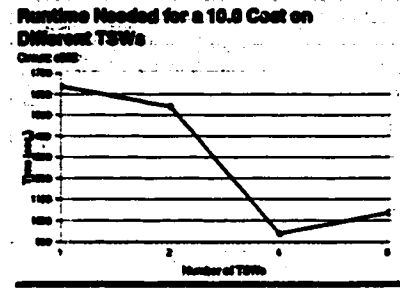
(a)



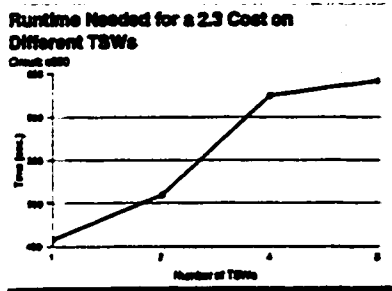
(b)



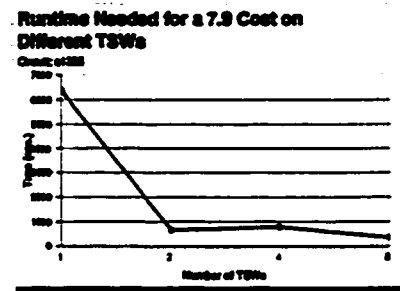
(c)



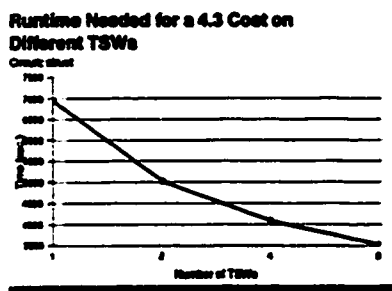
(d)



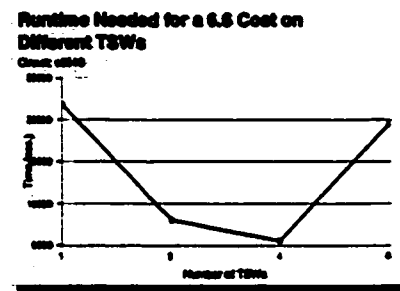
(e)



(f)

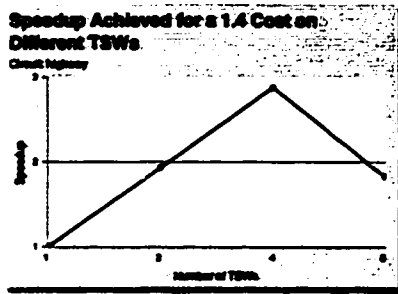


(g)

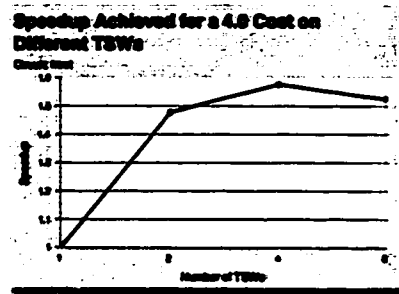


(h)

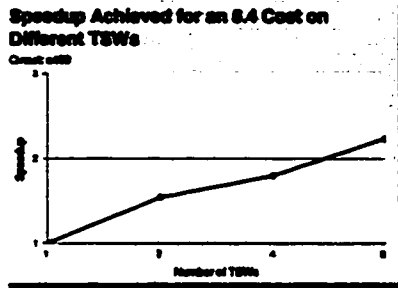
Figure 5.6: Runtime needed to achieve a solution of cost less than x for different numbers of TSWs.



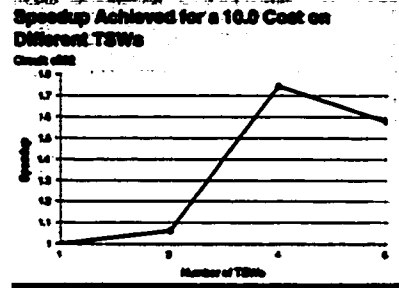
(a)



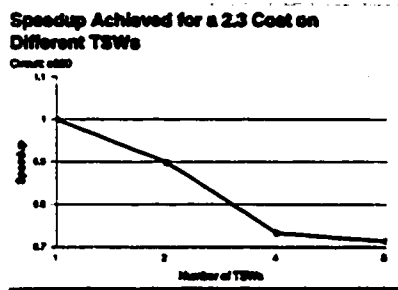
(b)



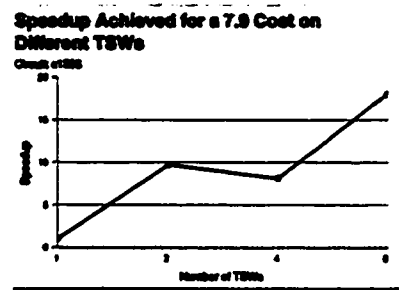
(c)



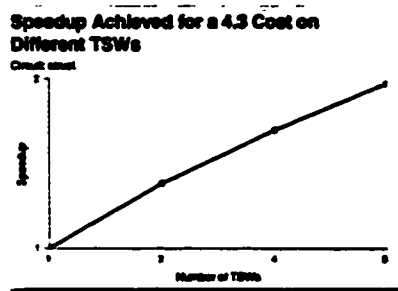
(d)



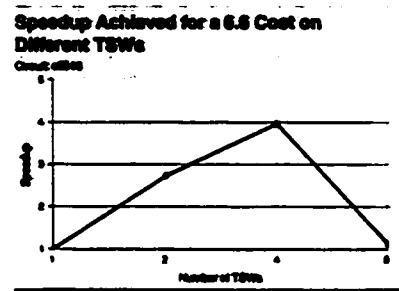
(e)



(f)



(g)



(h)

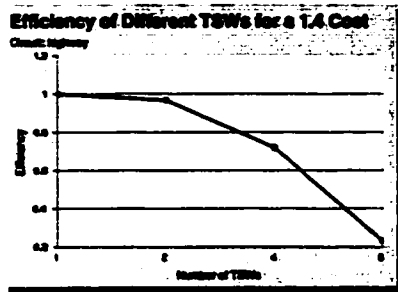
Figure 5.7: Speedup achieved in reaching a solution of cost less than x for different numbers of TSWs.

Figure 5.8 shows the efficiency of using more TSWs in reaching a specific solution quality for all circuits. The highest efficiency point differs from a circuit to another according to its size, complexity and initial solution. For *c1955*, and *c3540*, the highest efficiency point occurred at two TSWs. For the other circuits, the highest efficiency occurred at one TSW.

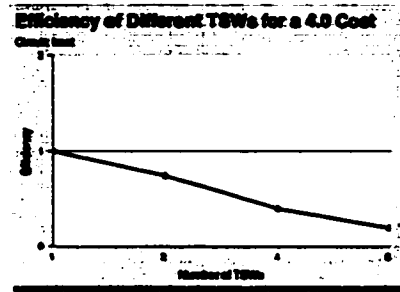
Generally speaking, increasing the number of CLWs performed better than increasing the number of TSWs because the CLW is an inner loop for all TSWs running. As a result, the speedup critical point was approached using low-level parallelization faster than high-level parallelization.

5.3 Accounting for Speed and Load Heterogeneity

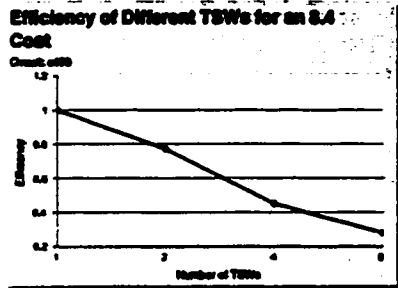
In this experiment, we try to see the effect of accounting for speed and load heterogeneity of various machines by performing two runs. In the first one, we run our algorithm that accounts for speed and load heterogeneity by making the master ask for best solutions from all TSWs once half of them complete all assigned iterations and report their best to the master. TSWs do the same by asking their CLWs to submit their best solutions once half of them report their best to the parent. In all experiments we use twelve machines to make the Parallel Virtual Machine. These machines include seven high-speed machines, 3 medium-speed machines and 2 low-speed machines. PVM takes care of distributing processes between machines. In both runs, we use 4 TSWs and 4 CLWs per TSW. The run that does not account for heterogeneity is supposed to give better solutions because the parent waits for all of its children to give their best solutions and does not force any one to stop searching because others have finished. However, the time needed to run the experiment that accounts for heterogeneity is expected to be far less than the time needed by the other experiment. As a result of that, if the solution quality is comparable, then the experiment is worth being performed



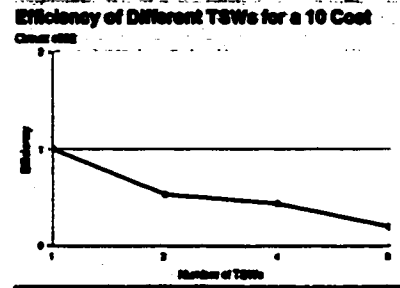
(a)



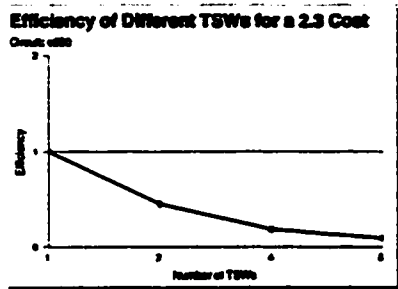
(b)



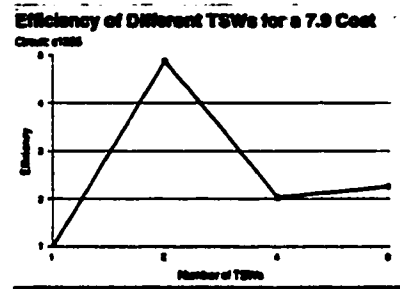
(c)



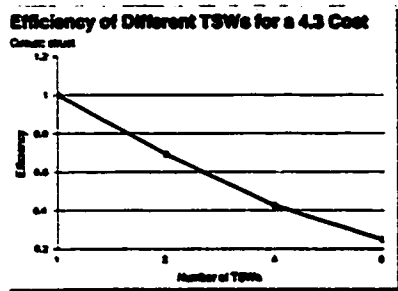
(d)



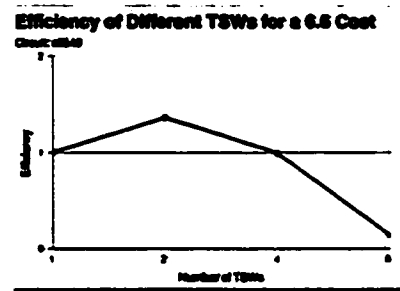
(e)



(f)



(g)



(h)

Figure 5.8: Efficiency of using more TSWs in reaching a solution of cost less than x .

to speed up the search.

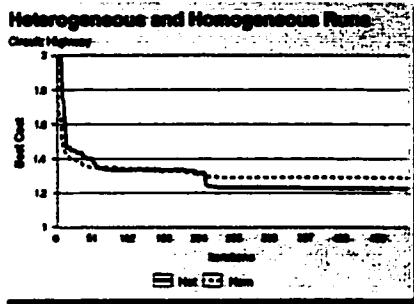
Figure 5.9 shows the best quality of solution achieved versus global iterations for the run that accounts for heterogeneity and the one that does not for seven circuits. For most of the circuits, such as *highway*, *c532*, *c880*, *c1355* and *struct*, there is no big difference in the solution quality between the homogeneous and the heterogeneous runs. In *fract*, the solution quality of the heterogeneous run starts to be worse but as iterations proceed, both runs converge to the same quality. For *c499*, there is a solution quality difference that should be compensated for by the runtime needed in order for the heterogeneous run to outperform the homogeneous run.

Table 5.3 shows the runtime needed for heterogeneous and homogeneous runs for the seven circuits to complete. Based on the runtime for the heterogeneous and the homogeneous runs, another set of curves given in Figure 5.10 show the best cost achieved versus run time in seconds for heterogeneous and homogeneous runs for the seven circuits.

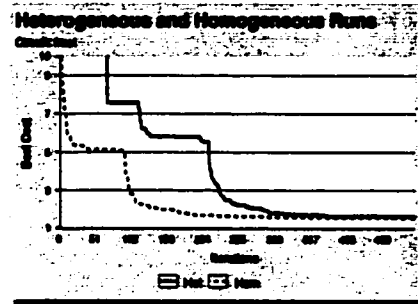
Circuit	Hom. Runtime	Het. Runtime	Improvement
highway	2316	1631	1.42
fract	11194	5626	1.99
c499	5722	3060	1.87
c532	8615	4839	1.78
c880	32361	19550	1.66
c1355	42560	27822	1.53
struct	76954	43332	1.78

Table 5.3: Runtime of homogeneous and heterogeneous runs in seconds for seven circuits.

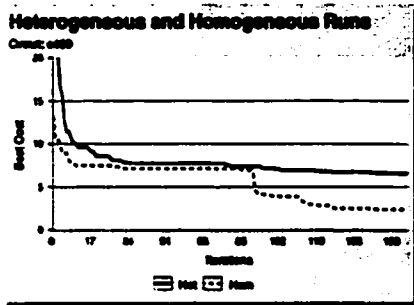
Figure 5.10 shows that towards the end of the experiment, the heterogeneous run is doing either better than or at least as good as the homogeneous run. It never performs worse. For some circuits like *highway*, *c532*, *c880*



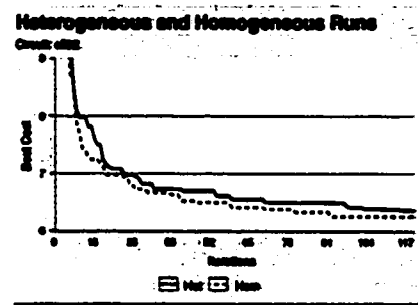
(a)



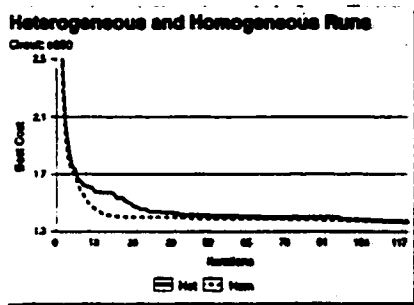
(b)



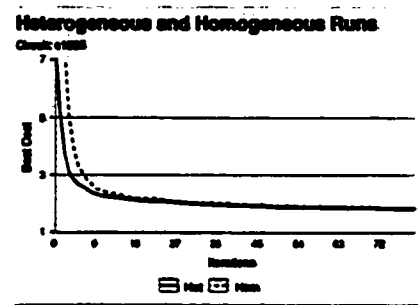
(c)



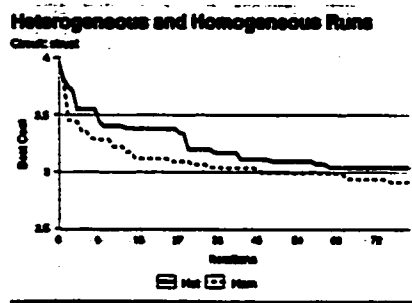
(d)



(e)



(f)



(g)

Figure 5.9: Heterogeneous vs. homogeneous runs.

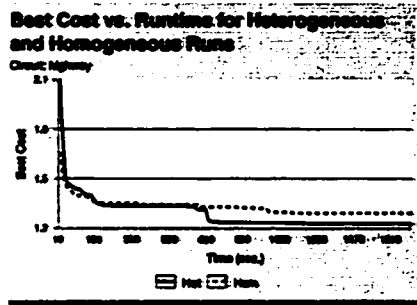
and *c1955*, the heterogeneous run keeps performing better than the homogeneous run throughout the execution. For *c499*, the heterogeneous run starts by performing worse and afterwards it outperforms the homogeneous run. For *fract* and *struct*, the homogeneous run outperformed the heterogeneous run initially. However, they converged to the same quality finally. The reason is that more investigation of the neighborhood is required in these two circuits and that could be clearly seen from Figure 5.1.

In the following experiment, we try to see whether it is useful or not to include the slow machines in the parallel virtual machine because the master keeps stopping them once the other machines report their best solutions. To see the contribution of the slow machines in our strategy, we conduct an experiment where one high-speed, one medium-speed, and one low-speed machine are used as a parallel virtual machine. A single TSW is spawned on each machine with one CLW per TSW. Once a TSW reports its best solution to the master, the master asks all others to stop and report their best solutions to it. By monitoring the number of solutions reported by each TSW within various cost ranges, we can tell which machine is contributing more to the search with useful results. Figure 5.11 shows the results of the experiment ran on *c499* for 500 global iterations. The results show that the contribution of the three machines is almost equal in all cost ranges. This is an expected result of the broadcasting step performed by the master at each global iteration.

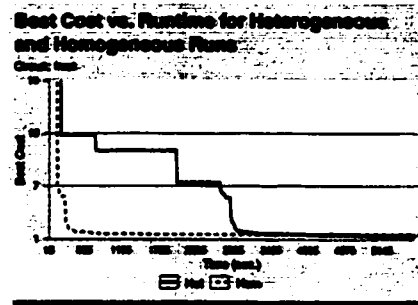
5.4 Effect of Diversification

In this experiment, we try to see the effect of the diversification step performed by the TSWs at the beginning of each global iteration. As mentioned in Section 4.4, the diversification step is performed to make every TSW investigate a different region of the search space.

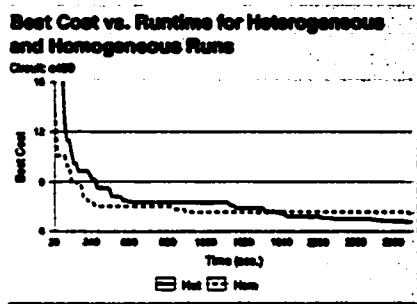
Each TSW performs a number of successive moves up to a predefined diversification depth at each global iteration. The diversification depth used



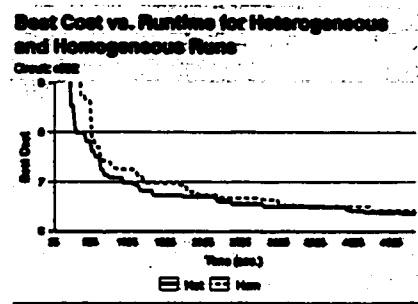
(a)



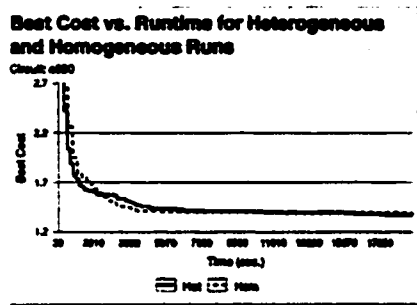
(b)



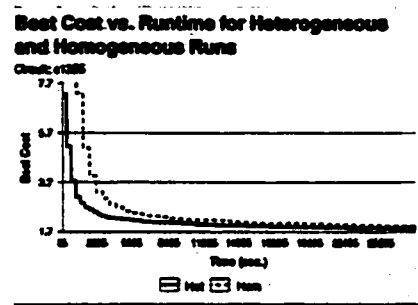
(c)



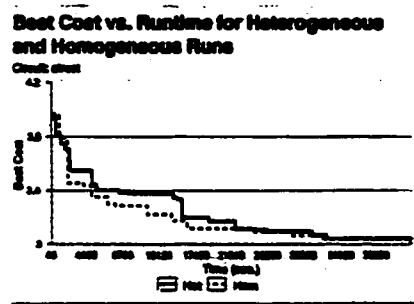
(d)



(e)



(f)



(g)

Figure 5.10: Best cost vs. runtime for heterogeneous and homogeneous runs.

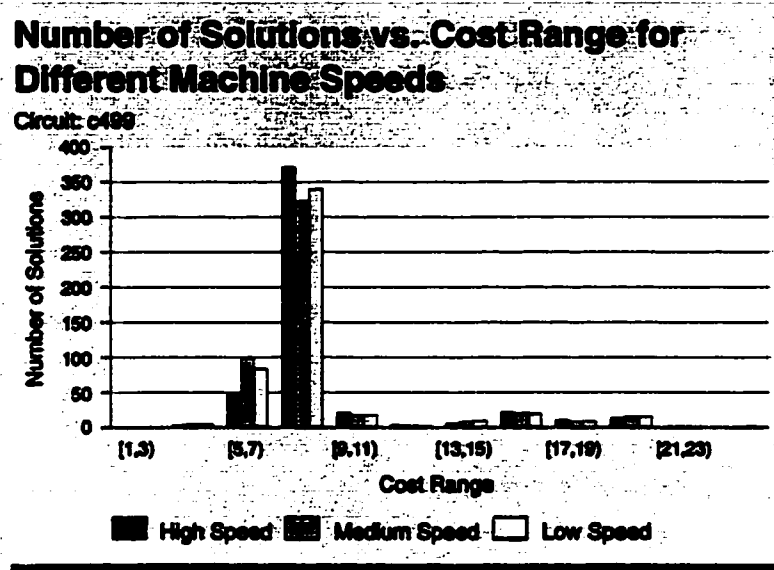
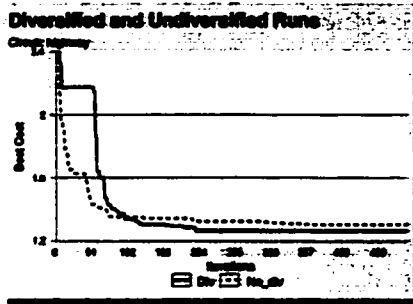


Figure 5.11: Number of solutions provided by machines of different speeds within various solution ranges.

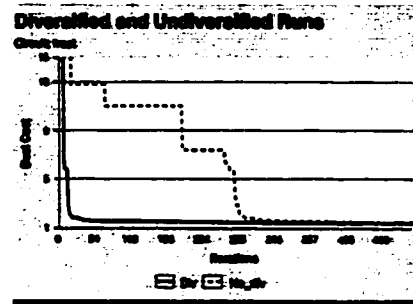
in all experiments for each circuit is shown in Table 5.2. At each move, the TSW tries N_s different swaps and picks the most improving (or least degrading) one. One of the cells to be swapped has to be from the range belonging to the TSW.

Figure 5.12 shows a comparison between two runs of four TSWs and one CLW per TSW. In one run, diversification is done to the diversification depth mentioned in Table 5.2. In the other run, no diversification is performed. It is clear from the figure, that the diversified run outperforms the undiversified run significantly. For *highway* and *struct*, the undiversified run starts to outperform the diversified run because it is more greedy. However, in the long term, the diversified run always performs better.

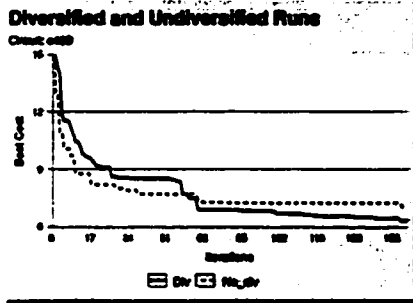
Some diversification is always useful as shown in Figure 5.12. However, too much diversification without enough local investigation might mislead the search by making it jump from place to another without enough investigation any where. The only way to decide how much local investigation versus



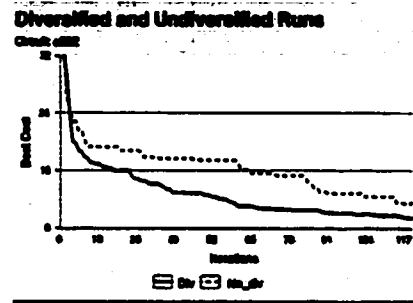
(a)



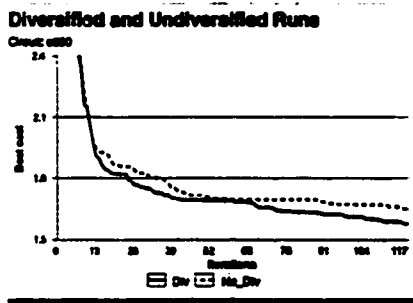
(b)



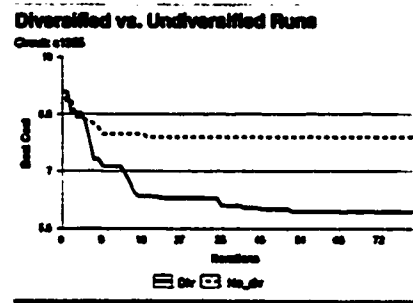
(c)



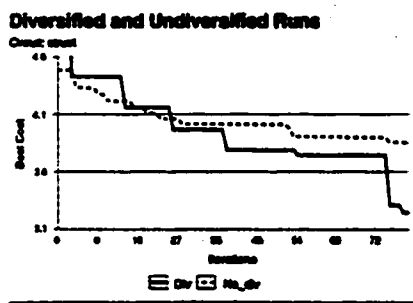
(d)



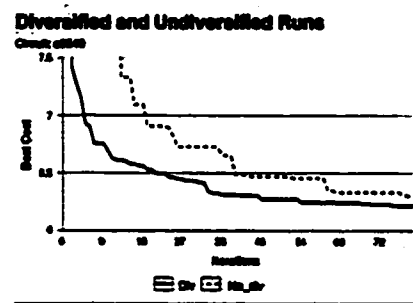
(e)



(f)



(g)



(h)

Figure 5.12: Effect of diversification.

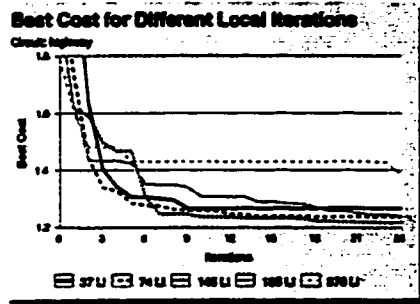
diversification is enough is through experiments.

Figure 5.13 shows the results of an experiment where the number of global iterations is decreased as the number of local iterations is increased for all circuits. As we decrease the number of global iterations and increase the number of local iterations, we make more diversification and less local investigation. It is clear from the figure that no general conclusion can be made about the best number of global iterations versus local iterations. It all depends on the problem instance itself. This experiment is used as a guide for the most suitable number of local and global iterations that should be used to continue searching for the best achievable solution.

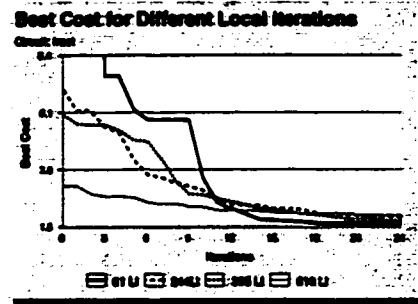
5.5 Interarrival of a q -Quality Solution

In this experiment, we try to see whether TS is a memoryless algorithm or not by performing a large number of runs (100 runs). In all of these runs, only one TSW and one CLW are used. This means that it is a sequential TS run. In each run we make 200 global iterations. Over all runs, we find the number of runs where a solution of a specified quality, call it q , was achieved in less than a number of iterations t .

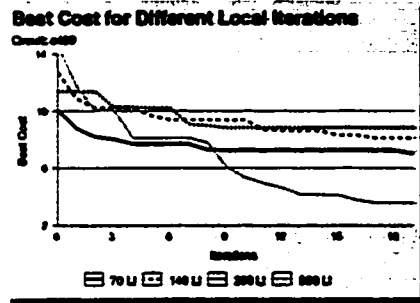
Figure 5.14 shows the fraction of runs where a specific solution quality was achieved within a specific number of iterations or less. The curves are taken for *highway* and *fract*. For *highway*, the curve almost fits to the CDF of an exponential plot with a λ of 0.04. For *fract*, it almost fits to the CDF of an exponential plot with a λ of 0.03. This means that the interarrival time of a solution of quality q is exponentially distributed. A conclusion out of this is that TS is memoryless algorithm over a long run if the points are taken on a compressed scale.



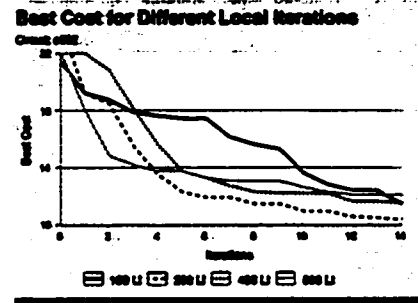
(a)



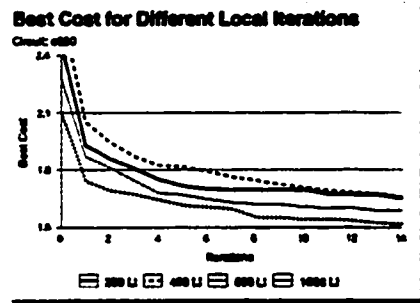
(b)



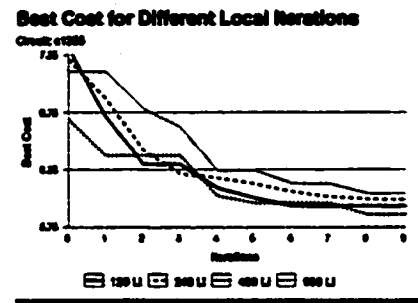
(c)



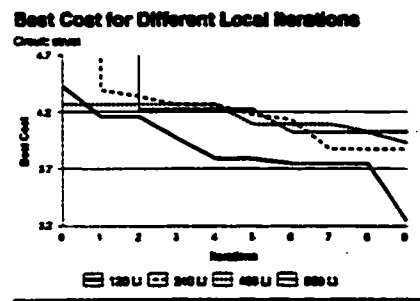
(d)



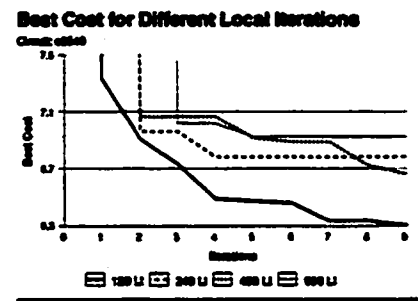
(e)



(f)

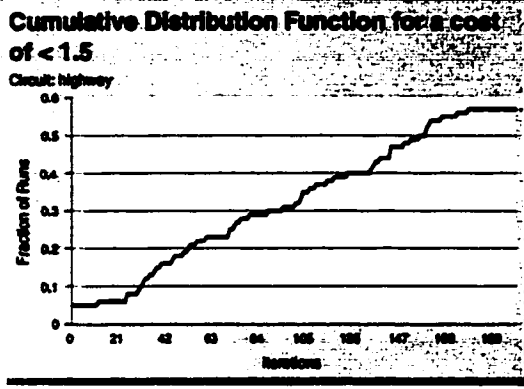


(g)

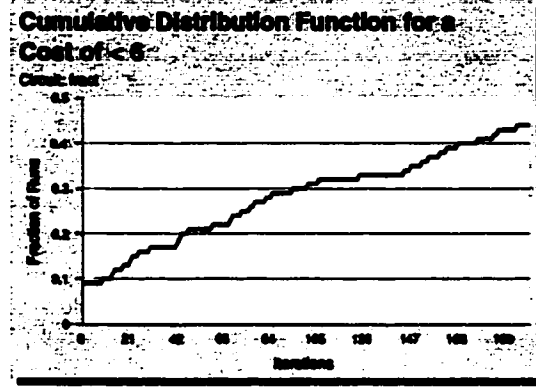


(h)

Figure 5.13: Local vs. global iterations.



(a)



(b)

Figure 5.14: CDF for getting a solution of quality $< q$.

5.6 Fuzzy Cost Evaluation vs. Weighted Sum

In this experiment, we try to compare the results obtained from fuzzy cost evaluation with those obtained from weighted sum evaluation. As mentioned in Section 1.3.4, weighted sum evaluation has the disadvantage of difficulty in deciding the weights to be used. A previous work that used the same circuits and the same evaluation function found experimentally that the best weights combination is 0.6, 0.1 and 0.3 for wire length, delay and width respectively [15]. The same combination is used in this work. For the fuzzy evaluation, the goal vector values were changed according to the initial solution in order to limit the number of iterations spent in searching within solutions that are outside the acceptable range. Table 5.4 shows a comparison of the best solution achieved using four TSWs and one CLW per TSW in a run that used fuzzy evaluation and a run that used weighted sum for all circuits. It shows the wire length in microns, the delay in nanoseconds and the width in microns.

It is clear from the table that since we give very high weight to the wire length in the weighted sum run, it gives most of the time better wire length than fuzzy evaluation. Since low weights are given to the delay and the width, the fuzzy evaluation run is outperforming the weighted sum run in

Circuit	Fuzzy Evaluation			Weighted Sum			% Gain		
	WL	Del.	W	WL	Del.	W	WL	Del.	W
highway	8448	5.78	520	7948	6.12	544	-6.28	5.55	4.41
fract	40331	14.90	792	42917	14.58	792	6.03	-2.19	0.00
c499	85376	16.50	1264	82546	17.48	1304	-3.43	5.61	3.07
c532	118790	36.37	1368	105677	38.13	1360	-12.41	4.62	-0.59
c880	229800	35.64	2048	286023	37.44	2248	19.66	4.81	8.90
c1355	1028118	44.77	2576	910075	41.88	2488	-12.97	-6.90	-3.54
struct	1359976	36.41	3312	1462027	34.65	3344	6.98	-5.08	0.96
c3540	2345763.2	85.807	3200.0	2186335	94.64	3256	-7.29	9.33	1.72

Table 5.4: Best solution achieved by fuzzy evaluation run vs. weighted sum run.

delay and width. This shows the difficulty in deciding the weights for the weighted sum run. If we compute the total gains from Table 5.4, They are -9.71 for wire length, 15.75 for delay, and 14.93 for width.

5.7 Comparison with Previous Work

In this section, we compare the best results obtained for each circuit with results obtained from a previous work done by Ali [15]. The purpose is to verify that the results obtained by the proposed algorithm are comparable to previous work results such that we make sure that the algorithm is working fine as far the placement solutions provided for the problem are concerned. The purpose of this section is not to compare the performance of *Simulated Evolution* to parallel *Tabu Search* for standard cell placement, rather it is to make sure that the solutions provided by the proposed algorithm fall within close limits achieved before. In his work, Ali performed three experiments. In his 1st experiment, he used *Classical Simulated Evolution* (CSE). In the second experiment, he used Fuzzy Allocation (Fa_SE). In the last experiment, he fuzzified the evaluation step of the algorithm (Fe_SE). In all experiments, he used a Fuzzy cost function. Tables 5.5, 5.6 and 5.7, show the best layout

found by our *Parallel Tabu Search* vs. those found by the mentioned three experiments for all circuits.

Circuit	Parallel Tabu Search			Classical SE			% Gain		
	WL	Del.	W	WL	Del.	W	WL	Del.	W
highway	7544	5.55	520	9919	6.15	520	23.94	9.76	0.00
fract	32451	13.46	792	37285	14.58	800	12.96	7.68	1.00
c499	49676	14.13	1208	59278	14.51	1200	16.20	2.62	-0.67
c532	77944	34.78	1168	72789	38.55	1184	-7.08	9.78	1.35
c880	136289	30.70	1848	135509	30.92	1848	-0.58	0.71	0.00
c1355	281065	26.63	2336	335589	28.41	2344	16.25	6.27	0.34
struct	631770	28.81	3376	685328	26.65	3312	7.81	-8.11	-1.93
c3540	842566	50.16	3184	844069	54.03	3152	0.18	7.16	-1.02

Table 5.5: Best solution achieved by parallel tabu search vs. classical simulated evolution.

Circuit	Parallel Tabu Search			Fuzzy Evaluation SE			% Gain		
	WL	Del.	W	WL	Del.	W	WL	Del.	W
highway	7544	5.55	520	9503	6.74	528	20.61	17.66	1.52
fract	32451	13.46	792	35635	13.50	808	8.94	0.30	1.98
c499	49676	14.13	1208	60964	14.83	1192	18.52	4.72	-1.34
c532	77944	34.78	1168	75216	35.80	1200	-3.63	2.85	2.67
c880	136289	30.70	1848	137838	29.03	1852	1.12	-5.75	0.22
c1355	281065	26.63	2336	349953	28.23	2368	19.68	5.67	1.35
c3540	842566	50.16	3184	883503	52.59	3128	4.63	4.62	-1.79

Table 5.6: Best solution achieved by parallel tabu search vs. simulated evolution with fuzzy evaluation.

It is clear from the three tables that the proposed parallel *Tabu Search* algorithm is providing solutions comparable to previous work results if not better.

	Parallel Tabu Search			Fuzzy Allocation SE			% Gain		
Circuit	WL	Del.	W	WL	Del.	W	WL	Del.	W
highway	7544	5.55	520	7735	5.56	520	2.47	0.18	0.00
fract	32451	13.46	792	31528	13.62	784	-2.93	1.17	-1.02
c499	49676	14.13	1208	56506	14.13	1200	12.09	0.00	-0.67
c532	77944	34.78	1168	80779	37.96	1160	3.51	8.38	-0.69
c880	136289	30.70	1848	137309	29.46	1872	0.74	-4.21	1.28
c1355	281065	26.63	2336	290221	27.05	2320	3.15	1.55	-0.69
struct	631770	28.81	3376	667850	28.8	3336	5.40	-0.03	-1.20
c3540	842566	50.16	3184	750153	46.01	3152	-12.32	-9.02	-1.02

Table 5.7: Best solution achieved by parallel tabu search vs. simulated evolution with fuzzy allocation.

5.8 Summary

In this chapter, we presented and discussed various experiments that were performed using the proposed parallel TS algorithm. The effect of degree of low-level parallelization and high-level parallelization on the algorithm performance and speed was studied and the results were analyzed. The effect of accounting for speed and load heterogeneity on the algorithm speed and performance was studied and analyzed. The effect of diversification performed by TSWs was presented by comparing the results of a diversified run to a run of the algorithm where diversification was not performed. Performance of the algorithm using fuzzy cost evaluation was compared to the performance using weighted sum technique. For the purpose of verification, experimental results were compared to results of previous work done on the same circuits using *Simulated Evolution*.

Chapter 6

Conclusions

In this thesis, VLSI standard cell placement was addressed as one of the VLSI physical design problems. The problem was formally defined and its complexity was discussed. Various VLSI placement styles were presented. Standard cell layout methodology was explained as the style adopted in the work. Heuristics applied to VLSI placement were classified.

The criteria according to which a specific placement solution is evaluated were presented and discussed. Interconnection length was discussed and the model used to approximate it was explained. Area was estimated as proportional to the width of the longest row. The critical path delay was defined and formulated. Two evaluation schemes of the overall solution quality were presented and compared. The fuzzy goal based scheme was applied in this work.

Tabu Search was presented, its parameters were discussed, and their effects were explained. These parameters are the candidate list, moves and move attributes, the evaluation function, the tabu list, and the aspiration criteria.

A comprehensive literature review of the heuristics applied to VLSI placement was conducted. The application of TS to VLSI placement was separately investigated in the literature. Parallelization of TS algorithm was introduced, reviewed in the literature, and classified according to two known

taxonomies.

The algorithm proposed for the problem was presented and the parameters chosen were justified. The parallelization scheme applied was explained and accordingly the algorithm was classified. The algorithm was parallelized at two levels simultaneously such that it is a *multi-search threads* algorithm and a *functional decomposition* algorithm. A restriction scheme was presented to make it also a *domain decomposition* algorithm probabilistically.

The motivation and the strategy of applying the algorithm in a heterogeneous environment were discussed. The scheme used to make it account for heterogeneity in speed and load was explained. The heterogeneous environment where the algorithm was run was presented.

The idea of diversification of tabu search was presented. The motivation and strategy used in the proposed algorithm to diversify the search were discussed. The main idea of the scheme used was given by Kelly et. al. The scheme was modified to satisfy the objective of diversification in our work. This objective is to have different *tabu search* processes investigating different areas of the search space.

The benchmark circuits used as well as the parameters used for all experiments were given. A definition for 'Speedup' and 'Efficiency' was presented for this category of problems. Two experiments were conducted to see the effect of low-level and high-level parallelization degrees. It was found most of the time beneficial in terms of solution quality and runtime to parallelize up to some critical point where increasing the parallelization degree started to cause communication overhead that was not compensated for by the parallelization. The low-level parallelization produced slightly better results than the high-level parallelization.

An experiment was conducted to see the effect of accounting for speed and load heterogeneity in the machines used for parallelization. The experiment showed that the run that accounted for heterogeneity had improvement in the time required to perform the search and accordingly produced similar solutions quality in less time than the run which did not account for hetero-

geneity. Another experiment was run to see if the slow machines are ever contributing to the search process. It was observed that as far as the solution quality is concerned, machines of different speeds contributed equally to the search process.

The effect of the diversification scheme used was studied using another experiment where the results of a diversified run were compared to results of an undiversified run with the same parameters. It was found that diversification was beneficial for all circuits. The diversified run produced solutions better than those produced by the undiversified run.

To check whether the algorithm is memoryless or not, an experiment was conducted over a large number of runs. It was observed that the interarrival time of a q -quality solution is almost exponential which supports that the algorithm is memoryless over a long run and a compressed scale.

In another experiment, the results of a run where weighted sum was used were compared to results of another experiment where fuzzy cost evaluation was applied. The experiment showed the difficulty in choosing the appropriate weights for the weighted sum run. It also showed overall improvement in the solutions when fuzzy cost evaluation was used.

A final experiment was conducted to compare the best results we got for each circuit with results obtained from a previous work. The purpose was to verify that the results obtained by the proposed algorithm are comparable to previous work results such that we make sure that the algorithm is working fine as far the placement solutions provided for the problem are concerned. It was found that the proposed parallel *Tabu Search* algorithm was providing solutions comparable to previous work results.

Appendix A

Parallel Virtual Machine

A.1 Introduction

In 'Distributed Computing', several interconnected computers work together to solve a large problem [33]. Using a Network of Workstations as an environment for distributed computing has been addressed and investigated for the following main factors:

- Cost effectiveness compared to an expensive large multiprocessor supercomputer alternative [57].
- Utilization of the computation power of workstations that remain idle for a large fraction of time.
- High usability of workstations since they have different interfaces and programming tools which might suit more users than a single hard-to-use supercomputer [58].
- Availability of high speed transmission links that have capacities in the order of gigabits/sec [59].

Due to these factors, the need for a powerful system to allow use of a network as for distributed computing has emerged. A difficulty in that process is that most of the time, a network is composed of heterogeneous machines.

This heterogeneity can be of various types such as machine architecture, data format, computational speed, network type, machine load and network load [33]. In spite of this difficulty, a team from Oak Ridge National Laboratory, the University of Tennessee, and Emory University, developed a Parallel Virtual Machine (**PVM**) system that gives the user the chance to write her/his program to run on several heterogeneous machines interconnected by a network. The user writes her/his application as a collection of communicating (tasks) and lets PVM handle all of the message passing, format conversion and task scheduling problems [33, 60].

A.2 Parallel Virtual Machine (PVM)

PVM consists of two main parts, a message passing library and a daemon.

A.2.1 PVM Daemon (*pvmd*)

This is the PVM engine that should be running on all machines that comprise the virtual machine. It serves as a message router and controller [33, 61, 58].

At startup of any task, *pvmd* of the task configures itself as a master or a slave according to command line configuration. It establishes communication channel(s) with all other running daemons.

Master *pvmd* handles virtual machine configuration. Other *pvmds* handle all tasks running under them. It is also the responsibility of the *pvmd* to detect any fault and try to recover from it if possible [33].

A.2.2 PVM Library (PVML)

PVML is a set of subroutines allowing a task to communicate with *pvmd* through TCP connection and with other tasks through UDP sockets [33, 61, 58].

It consists of two libraries written in C. One supports programs written in C and the other supports programs written in FORTRAN [33, 61, 60].

The functions provided by LPVM allow a task for spawning one or more other tasks, killing tasks, getting information about the virtual machine, adding and deleting hosts to and from the machine, packing messages to consider format conversion, sending messages, receiving and unpacking messages and defining and handling groups of tasks [33].

A.2.3 Architectural Description

From the user standpoint, PVM is a general-purpose flexible parallel computer. It handles all conversion of formats and heterogeneity with all of its types. An application program consists of components each of which might initiate instances that utilize PVM facilities [58].

PVM can be accessed by the user at three different levels; *transparent level* where instances are automatically located at the most appropriate sites, *architectural level* where a user can specify the architecture required to execute his task, and *low level* where a specific machine can be specified for executing the task [60].

Bibliography

- [1] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice*. McGraw-Hill Book Co., Europe, 1995 (also co-published by IEEE press).
- [2] E. Horvath, R. Shankar, and A. Pandya. A parallel algorithm for standard cell placement. In *Seattle International Joint Conference on Neural Networks*, pages 896–897, July 1991.
- [3] A. Rajanala and A. Tyagi. An area estimation technique for module generation. In *IEEE International Conference on Computer Design, VLSI in Computers and Processors*, pages 459–462, September 1990.
- [4] M. Pedram and B. Preas. Interconnection length estimation for optimized standard cell layouts. In *IEEE International Conference on Computer Aided Design*, pages 390–393, November 1989.
- [5] J. Chandy and P. Banerjee. A parallel circuit-partitioned algorithm for timing driven cell placement. In *IEEE International Conference on Computer Design, VLSI in Computers and Processors*, pages 621–627, October 1997.
- [6] T. Koide, M Ono, S. Wakabayashi, and Y. Nishimaru. Par-popins: A timing-driven parallel placement method with the elmore delay model for row based VLSIs. In *Asia and South Pacific Design Automation Conference*, pages 133–140, January 1997.

- [7] W. Donath, R. Norman, B. Agrawal, S. Bello, S. Han, J. Kurtzberg, P. Lowy, and R. McMillan. Timing driven placement using complete path delays. In *27th ACM/IEEE Design Automation Conference*, pages 84–89, June 1990.
- [8] P. Cheung, C. Yeung, S. Tse, C. Yuen, and W. Ko. A new optimization cost model for VLSI standard cell placement. In *IEEE International Symposium on Circuits and Systems*, pages 1708–1711, June 1997.
- [9] M. A. Breuer. A class of min-cut placement algorithms. *Proceedings of 14th Design Automation Conference*, pages 284–290, October 1977.
- [10] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Company, New York, 1979.
- [11] Carol A. Mackey and Jo Dale Carothers. Performance-driven macrocell placement. In *IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pages 427–433, March 1996.
- [12] E. I. Horvath. A parallel force directed based VLSI standard cell placement algorithm. In *IEEE International Symposium on Circuits and Systems*, pages 2071–2074, May 1993.
- [13] Jin-Tai Yan. An efficient heuristic approach on minimizing the number of feedthrough cells in standard cell placement. In *5th Great Lakes Symposium on VLSI*, pages 128–131, March 1995.
- [14] S. Mohan and P. Mazumdar. Wolverines: Standard cell placement on a network of workstations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(9):1312–1326, September 1993.
- [15] Syed Hussain Ali. Fuzzy simulated evolution algorithm for VLSI placement. Master’s thesis, Computer Engineering Department, King Fahd University of Petroleum and Minerals, December 1998.

- [16] Sadiq M. Sait and Habib Youssef. Timing-influenced general-cell genetic floorplanner. *Microelectronics Journal*, 28(2):151–166, 1997.
- [17] H. Youssef, S. Sait, and K. Al-Farra. Timing-influenced force directed floorplanning. In *European Design Automation Conference Euro-DAC 95*, pages 156–161, September 1995.
- [18] Q. Kang, R. Lin, and E. Shragowitz. Fuzzy logic approach to VLSI placement. *IEEE Transactions on VLSI Systems*, 2(4):489–501, December 1994.
- [19] M. Razaz. A fuzzy c-means clustering placement algorithm. In *IEEE International Symposium on Circuits and Systems-ISCAS*, volume 3, pages 2051–2054, May 1993.
- [20] Adiche Hakeem. Fuzzy logic based iterative algorithms for floorplanning. Master's thesis, Computer Engineering Department, King Fahd University of Petroleum and Minerals, September 1997.
- [21] Ronald Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions Systems, man, and Cybernetics*, 18(1):183–190, January 1988.
- [22] M. A. Breuer. Min-cut placement. *Journal of Design Automation and Fault Tolerant Computing*, 1(4):343–382. October 1977.
- [23] U. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. *Proceedings of 16th Design Automation Conference*, pages 1–10, 1979.
- [24] M. Hanan and J. M. Kurtzberg. Placement techniques. in M. A. Breuer, Editor, *Design Automation of Digital Systems*, Prentice-Hall Inc, Englewood Cliffs, New Jersey, pages 213–282, 1972.

- [25] C. Sechen and A. L. Sangiovanni-Vincentelli. Timberwolf3.2: A new standard cell placement and global routing package. *Proceedings of 23rd Design Automation Conference*, pages 432–439, 1986.
- [26] A. Casotto, F. Romeo, and A. L. Sangiovanni-Vincentelli. A parallel simulated annealing algorithm for the placement of macro-cells. *IEEE Transactions on Computer Aided Design*, CAD-6(5):838–847, September 1987.
- [27] S. A. Kravitz and R. A. Rutenbar. Placement by simulated annealing of a multiprocessor. *IEEE Transactions on Computer-Aided Design*, CAD-6(4):534–549, July 1987.
- [28] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, CAD-6:956–964, November 1987.
- [29] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer Aided Design*, 9(5):500–511, May 1990.
- [30] R. Kling and P. Bannerjee. ESP: A new standard cell placement package using simulated evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.
- [31] L. Song and A. Vannelli. VLSI placement using tabu search. *Microelectronics Journal*, 17(5):437–445, 1992.
- [32] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms and their Applications in Engineering*. IEEE Computer Society Press, 1999.
- [33] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, Massachusetts, London, England, 1994.

- [34] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.
- [35] F. Glover. Tabu search: A tutorial. Technical report, Graduate School of Business Administration, University of Colorado at Boudler, February 1990.
- [36] F. Glover. Tabu search fundamentals and uses. Technical report, Graduate School of Business Administration, University of Colorado at Boudler, June 1995.
- [37] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, USA, 1997.
- [38] F. Glover. Candidate list strategies and tabu search. Technical report, Graduate School of Business Administration, University of Colorado at Boudler, July 1989.
- [39] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill Book Co., Europe, 1995.
- [40] F. Glover. Tabu search- Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
- [41] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.
- [42] A. Lim, Y. Chee, and C. Wu. Performance driven placement with global routing for macro cells. In *Proceedings of the Second Great Lakes Symposium on VLSI*, pages 35–41, February 1992.
- [43] I. Lin and David Du. Performance-driven constructive placement. In *IEEE 27th Design Automation Conference*, pages 103–106, June 1990.
- [44] *Solution to Capacitor Placement Problem in a Radial Distribution System Using Tabu Search Method*, November 1995.

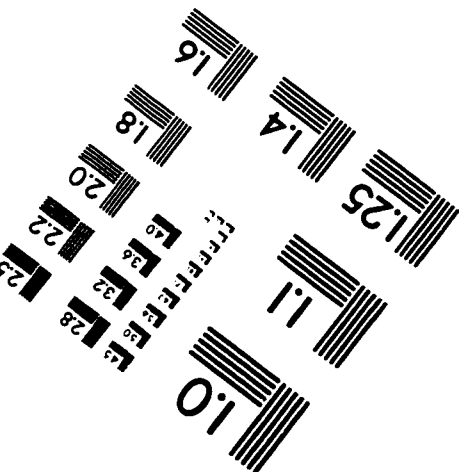
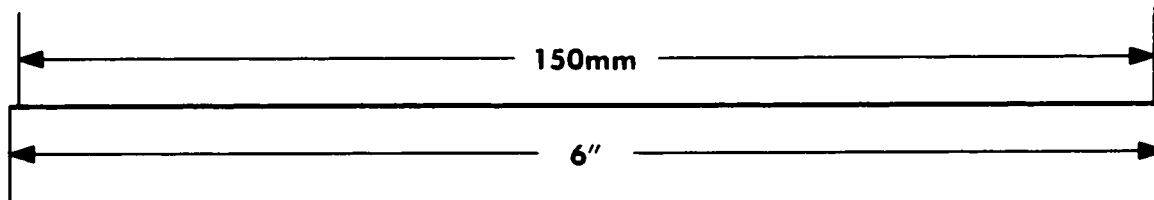
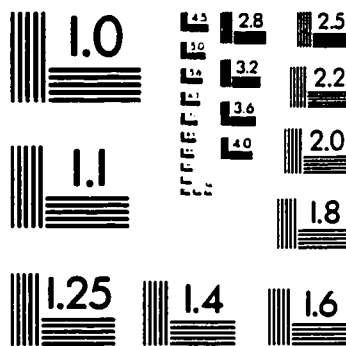
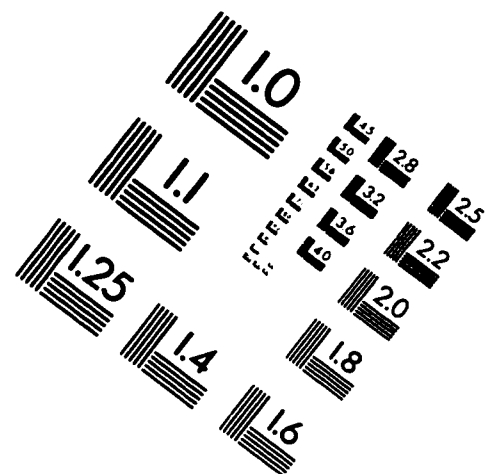
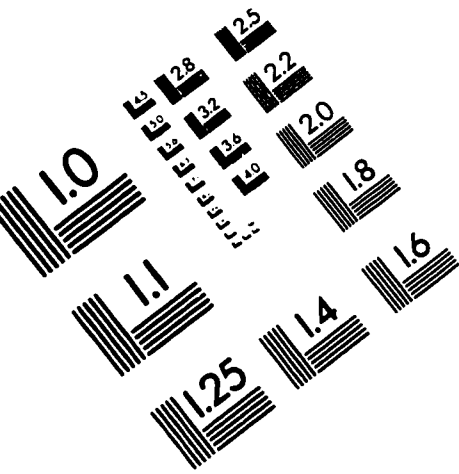
- [45] M. E. Baran and F. F. Wu. Optimal capacitor placement on radial distribution system. *IEEE Transactions on Power Delivery*, 4(1):725–734, January 1989.
- [46] M. E. Baran and F. F. Wu. Optimal sizing of capacitors placed on a radial distribution system. *IEEE Transactions on Power Delivery*, 4(1):735–743, January 1989.
- [47] H. Chiang, J. Wang, O. Cockings, and H. Shin. Optimal capacitor placement on distribution systems: Part I and part II. *IEEE Transactions on Power Delivery*, 5(2):634–649, January 1990.
- [48] K. Handa and S. Kuga. Polycell placement for analog LSI chip designs by genetic algorithms and tabu search. In *IEEE International Conference on Evolutionary Computation*, pages 716–721, November 1995.
- [49] E. Taillard. Robust tabu search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1991.
- [50] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 417:65–74, 1990.
- [51] Bruno-Laurent Garica, Jean-Yves Potvin, and Jean-Marc Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033, November 1994.
- [52] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro. Improving search by incorporating evolution principles in parallel tabu search. In *Proc. of the first IEEE Conference on Evolutionary Computation-ICEC'94*, pages 823–828, June 1994.
- [53] I. De Falco, R. Del Balio, and E. Tarantino. An effective parallel heuristic algorithm for the mapping problem. In *Proc. of Australian New Zealand Intelligent Information Systems Conference- ANZIIS'94*, pages 160–164, November 1994.

- [54] Smail Niar and Arnaud Freville, editors. *A Parallel Tabu Search Algorithm For The 0-1 Multidimensional Knapsack Problem*. 11th International Parallel Processing Symposium, Apr. 1997.
- [55] H. Mori and T. Hayashim. New parallel tabu search for voltage and reactive power control in power systems. In *Proc. of the 1998 IEEE International Symposium on Circuits and Systems- ISCAS'98*, pages 431–434, May 1998.
- [56] J. P. Kelly, M. Laguna, and F. Glover. A study of diversification strategies for the quadratic assignment problem. *Computers Ops Research*, 21(8):885–893, 1994.
- [57] M. Lewis and R. Cline. PVM communication performance in a switched FDDI heterogeneous distributed computing environment. In *IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 13–19, October 1993.
- [58] G. Geist and V. Sunderam. The evolution of the PVM concurrent computing system. In *Compcon Spring '93, Digest of Papers*, pages 549–557, February 1993.
- [59] P. Crandall, E. Sumithasri, and M. Clement. Performance comparison of desktop multiprocessing and workstation cluster computing. In *5th IEEE International Symposium on High Performance Distributed Computing*, pages 272–281, August 1996.
- [60] G. Geist and V. Sunderam. The PVM system: Supercomputer level concurrent computation on a heterogeneous network of workstations. In *6th Distributed Memory Computing Conference*, pages 258–261, May 1991.
- [61] Rod Fatoohi. Performance evaluation of communication networks for distributed computing. In *4th International Conference on Computer Communications and Networks*, pages 456–459, September 1995.

Vita

- Ahmad Abdul-Jabbar Al-Yamani.
- Born in Madinah, Saudi Arabia.
- Received Bachelor of Science (B.S.) degree in Computer Engineering from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia in June 1997.
- Joined the Department of Computer Engineering at KFUPM as a graduate assistant in July 1997.
- Completed Master of Science (M.S.) in Computer Engineering at KFUPM in May 1999.

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989

© 1993, Applied Image, Inc., All Rights Reserved

