

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

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600



Optimization of Mixed CMOS/BiCMOS Circuits Using Tabu Search

BY

Mounir M. Zahra

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

July 1998

UMI Number: 1391850

UMI Microform 1391850
Copyright 1998, 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, SAUDI ARABIA
COLLEGE OF GRADUATE STUDIES

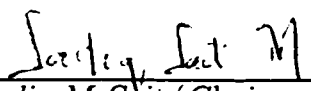
This thesis, written by

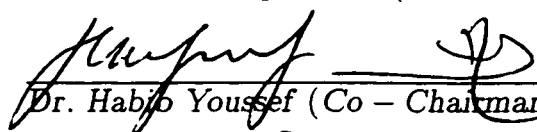
MOUNIR MAHMOUD ZAHRA

under the direction of his Thesis Advisor and approved by his Thesis Committee,
has been presented to and accepted by the Dean of the College of Graduate Studies,
in partial fulfillment of the requirements for the degree of


MASTER OF SCIENCE IN COMPUTER ENGINEERING

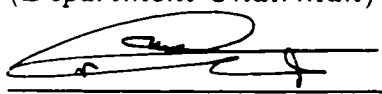
Thesis Committee


Dr. Sadiq M. Sait (Chairman)


Dr. Habib Youssef (Co – Chairman)


Dr. Mostafa Abd - El - Barr (Member)


Dr. Khalid M. Al - Tawil
(Department Chairman)


Dr. Abdallah M. Al - Shehri
(Dean, College of Graduate Studies)

2016/98
Date



Dedicated to

My Parents,

Wife, and Brothers

Acknowledgments

In the name of Allah, Most Gracious, Most Merciful. Read in the name of thy Lord and Cherisher, Who created. Created man from a {*leech-like*} clot. Read and thy Lord is Most Bountiful. He Who taught {the use of} the pen. Taught man that which he knew not. Nay, but man doth transgress all bounds. In that he looketh upon himself as self-sufficient. Verily, to thy Lord is the return {of all}.

(The Holy Quran, Surah 96)

All praise to Almighty Allah, who bestowed on me strength and patience to accomplish my goals. The acknowledgment is due to the King Fahd University of Petroleum & Minerals for providing me with tremendous support to this research work. I would like to express my indebtedness and sincere appreciation to Dr. Sadiq M. Sait, the chairman of the thesis committee, and Dr. Youssef Habib, co-chairman, who have been a constant source of guidance and encouragement throughout this work. I also appreciate the cooperation and support extended by Dr. Mostafa Abd Elbar who served as a committee member. I am thankful to all my friends and colleagues, whose kindness and understanding attitude made my stay, as a graduate student, a pleasant experience. Last but not the least, thanks are due to my parents, wife, and brothers for their prayers, guidance and moral support throughout my academic career.

Contents

Acknowledgments	iii
List of Figures	ix
List of Tables	xi
Abstract (English)	xiii
Abstract (Arabic)	xiv
1 Introduction	1
1.1 Importance of VLSI Circuits Optimization	1
1.2 Objectives and Scope of the Thesis	2
1.3 Difficulties and Limitations	4
1.4 Outline of the Thesis	5
2 Background Material and Literature Review	7
2.1 Introduction	7

2.2	VLSI Optimization	8
2.3	Critical Path Problem	11
2.3.1	Critical Path Algorithms	14
2.4	False Path Problem	15
2.4.1	Sources of False Paths	16
2.4.2	False Paths Detection Algorithms	17
2.5	Optimization Techniques	21
2.6	Summary	24
3	Problem Definition and Proposed Solution Technique	25
3.1	Introduction	25
3.2	Problem Definition	26
3.2.1	Is COP in NP ?	29
3.2.2	Is COP NP -Complete?	31
3.3	The Proposed Solution Technique	33
3.3.1	Phase I	35
3.3.2	Phase II	36
3.3.3	Phase III	37
3.4	Summary	38
4	Timing Analysis and False Path Problem	39
4.1	Introduction	39

4.2	Timing Analysis and α -Critical Approach	40
4.2.1	Description of α -Critical Algorithm	42
4.2.2	Example	44
4.3	False Path Problem	44
4.3.1	False Path Detection Algorithm	48
4.3.2	Implementation Details	58
4.4	Experimental Results	60
4.5	Summary	65
5	Circuit Optimization Problem Using Tabu Search	66
5.1	Introduction	66
5.2	Algorithm Description	67
5.2.1	Tabu Restriction	68
5.2.2	Aspiration Criteria	70
5.2.3	How the Algorithm Works	70
5.3	COP Formulation	72
5.3.1	Initial, Current and Best Solution	72
5.3.2	Generation of Moves	75
5.3.3	Tabu List	76
5.3.4	Restriction Criteria	77
5.3.5	Aspiration Criteria	77

5.3.6	Evaluation Function	78
5.3.7	Example	81
5.4	Intermediate and Long Term Memory	84
5.4.1	Intermediate Term Memory: Intensification Strategy	84
5.4.2	Long Term Memory: Diversification Strategy	88
5.5	Proposed Frequency-based Diversification Strategy	91
5.6	Evolutionary Tabu Search	92
5.6.1	Simulated Evolution Overview	93
5.6.2	Evolutionary Tabu Search	95
5.7	Summary	98
6	Implementation, Results and Discussion	100
6.1	Introduction	100
6.2	Implementation Details	101
6.2.1	Inputs/Outputs of the Program	101
6.2.2	Data Structures	105
6.2.3	Implementation Method	108
6.3	Optimization Tools	109
6.4	Experiments	112
6.5	Results and Discussion	118
6.5.1	Circuit Delay Improvement Results	119

6.5.2	Results of Short Term TS vs. Results of Long Term TS	121
6.5.3	Results of TS with AS_1 vs. Results of TS with AS_2	124
6.5.4	Results of Classical TS vs. Results of Evolutionary TS	126
6.5.5	Tabu Search Behavior	130
6.6	Summary	143
7	Conclusion and Future Work	144
7.1	Conclusion	144
7.2	Future Work	147
7.2.1	Other Mixed Technologies	147
7.2.2	Selection of Critical Paths for Optimization	148
7.2.3	Some Tabu Search Issues	149
	Appendix A	150
	Bibliography	159
	Vita	164

List of Figures

2.1	Delay vs. load for CMOS and BiCMOS Nand gate.	12
2.2	Logic dependency is a source of false paths [PCM89].	18
3.1	Nondeterministic COP algorithm.	30
3.2	General structure of the proposed solution technique.	34
4.1	A simple combinational circuit.	45
4.2	Another simple combinational circuit.	47
4.3	The false path detection algorithm (cont.).	53
4.4	The false path detection algorithm [DYG89].	54
4.5	Critical paths generation process for a circuit in SLIF format.	62
5.1	The relationship between memory functions of Tabu Search [Glo90]. .	69
5.2	Tabu list visualized as a window over accepted moves.	71
5.3	Block diagram of tabu search procedure.	73
5.4	Algorithmic description of Tabu Search (TS).	74
5.5	A simple combinational circuit.	82

6.1	Modified VPNR format of CKT.	102
6.2	Fanout description of CKT.	103
6.3	Sensitizable paths of CKT.	104
6.4	The adopted TS based circuit optimization algorithm.	110
6.5	Mixed CMOS/BiCMOS circuit optimization tools (cont.).	113
6.6	Mixed CMOS/BiCMOS circuit optimization tools.	114
6.7	Effect of T_SIZE on delta delay for short term memory.	132
6.8	Effect of CAN_SIZE on delta delay for short term memory.	133
6.9	Behavior of short term memory TS in terms of current and best cost.	135
6.10	Behavior of long term memory TS in terms of current and best cost.	136
6.11	Comparison between the behavior of short and long term memory TS.	137
6.12	Comparison between the behavior of Classical TS and Evolutionary TS in terms of current solution cost.	139
6.13	Comparison between the behavior of Classical TS and Evolutionary TS in terms of best solution cost.	141
6.14	Solutions distribution.	142

List of Tables

4.1	<i>Early_arrive_signals</i> and <i>Late_arrive_signals</i> of the path $P = C - 1 - E1 - 2 - F - 3 - H - 5 - I - 6 - Y$	57
4.2	Results of applying α -critical algorithm on some selected benchmark circuits.	63
4.3	Some application results of the adopted false path detection algorithm.	64
5.1	Iteration # 1.	85
5.2	Iteration # 2.	86
6.1	Sample of CMOS library.	106
6.2	Circuit delay improvement results for some circuits.	120
6.3	Best results of short term memory of TS with AS_1	122
6.4	Best results of long term memory of TS with AS_1	123
6.5	Best results of long term memory of TS with AS_2	125
6.6	Best results of long term memory of Evolutionary TS.	127

6.7 Comparison between Classical TS and Evolutionary TS in terms of quality and performance.	129
---	-----

Abstract

Name: Mounir Zahra
Title: Optimization of Mixed CMOS/BiCMOS Circuits
Using Tabu Search
Major Field: Computer Engineering
Date of Degree: 1998

This thesis studies the problem of optimizing mixed CMOS/BiCMOS circuits using *Tabu Search* (TS) approach. Recently, many techniques have been proposed to improve the performance of VLSI circuits. Some of these techniques include: transistor sizing, buffer insertion, and optimal selection of different templates of the same technology.

For further improvement, a mixed technology design can be developed to take advantage of each technology. One possibility is to mix CMOS and BiCMOS cells within the same circuit in order to take advantage of the high speed and high driving capabilities of BiCMOS, and the regularity and low power consumption of CMOS.

The problem is formulated as a constrained combinatorial optimization problem. The proposed solution technique comprises three phases. In the first phase, critical paths of the input circuit are generated because circuit delay is determined by longest *sensitizable* paths only. Second phase involves the application of the false paths detection algorithm. Since some critical paths may be false, it is important to eliminate them in order to get the longest sensitizable paths. In the third phase, TS algorithm is applied on the longest sensitizable paths. This is the core phase of the proposed approach where TS is used as a heuristic technique for optimal selection of gates that need to be implemented in BiCMOS to maximize the circuit performance with minimum increase in power consumption and area.

As a part of the third phase, two new strategies have been proposed. First, frequency-based diversification strategy has been developed to be used in long term memory of TS. Second, we applied some concepts of *Simulated Evolution* algorithm in generating neighbor solutions to the current solution.

The achievement of the proposed technique is of two parts. First, it has been shown that TS is very effective and efficient in producing very good solutions to circuit optimization problem. Second, remarkable increase in speed (%10 - 30%) with negligible increase in the power and area (less than 5%) of mixed CMOS/BiCMOS circuits has been accomplished.

Although the technique has been applied to BiCMOS and CMOS technologies, it is generally applicable to other technologies as long as it is feasible and practical to mix them in one circuit.

خلاصة الرسالة

الاسم: منير محمود زمر

عنوان الرسالة : الوصول إلى المثالية لدوائر أشباه الموصلات المتتامة المؤكدة والثانية (سيموس، بايسيموس) باستخدام بحث "تابو"

مجال التخصص: هندسة حاسب آلي .

تاريخ الرسالة : يوليو ١٩٩٨ م.

في الأونة الأخيرة تم اقتراح العديد من الأساليب التقنية لتحسين أداء الدوائر المدمجة الضخمة، ومن هذه الأساليب:

- ١- التحكم بحجم الترانزستور .
- ٢- استخدام الحواجز بين الدوائر أو البوابات.
- ٣- الاختيار الأمثل لوحدة بناء مختلفة من نفس نوع التقنية.

وللحصول على دوائر أكثر أداءً ، من الممكن تصميم دوائر مدمجة من عدة تقنيات ، وذلك للاستفادة من ميزات كل منها. ومن ذلك دمج تقنية سيموس وتقنية بايسيموس في دائرة واحدة للاستفادة من ميزات السرعة العالية والقدرة على التحريك لدى تقنية بايسيموس، وميزات استهلاك الطاقة المنخفض والتناسق لدى تقنية سيموس.

تم في هذه الرسالة صياغة هذه المشكلة كأحدى مشاكل الوصول إلى المثالية في ترتيب ومعالجة عناصر حسابية في مجموعات بشكل مقيد. وتم اقتراح أسلوب تقني فعال لتطوير وتحسين أداء دوائر مدمجة من تقنية سيموس، بايسيموس بأقل زيادة ممكنة في المساحة والطاقة المستهلكة، ويتكون الأسلوب المقترح من ثلاث مراحل. المرحلة الأولى والثانية تتضمن توليد أطول الطرق الصحيحة التي تحدد سرعة الدائرة حيث يتم توليد الطرق الحرجة أولاً ثم تطبيق خوارزمية الكشف عن الطرق الخاطئة لعزلها من الطرق الحرجة. وتتضمن المرحلة الثالثة تطبيق طريقة "تابو" لإيجاد أطول الطرق الصحيحة، وتمثل المرحلة الرئيسية في عملية التحسين كلها، حيث يتم استخدام طريقة "تابو" للبحث كأسلوب تنقيبي في عملية الاختيار الأمثل لبعض البوابات التي يمكن تنفيذها من تقنية بايسيموس، وذلك لزيادة سرعة الدائرة بأقل ما يمكن من الطاقة المستهلكة والمساحة. وكجزء من هذه المرحلة تم اقتراح واستخدام طريقتين جديدتين : الأولى في عملية توسيع مجال البحث المستخدمة في الذاكرة الطويلة المدى من طريقة "تابو" والثانية في تطبيق بعض مفاهيم خوارزمية "التقدم المتحاكي" في عملية توليد الحلول المجاورة للحل الحالي .

باستخدام وتطبيق هذا الأسلوب تم تحقيق إنجازين هامين:

- ١- بيان فعالية أسلوب "تابو" في إيجاد الحلول الجيدة والمطلوبة لمشكلة الوصول إلى المثالية للدوائر المدمجة.
- ٢- الحصول على زيادة عالية في سرعة الدوائر المدمجة (حوالي ١٠% - ٣٠%) بزيادة طفيفة في المساحة والطاقة المستهلكة (أقل من ٥%).

وكما تم تطبيق هذا الدمج على تقنية سيموس وبايسيموس فيمكن تعميمه على تقنيات أخرى مع الأخذ بالاعتبار ملائمة وإمكانية دمجها عملياً .

Chapter 1

Introduction

1.1 Importance of VLSI Circuits Optimization

Nowadays, CMOS technology has qualified to be the most appropriate choice in VLSI applications because of its low DC power dissipation and its high package density. However, the demand for superior performance, which has motivated the research and development of new technology, was behind the emergence of BiCMOS technology. BiCMOS is a combination of CMOS and Bipolar technologies which take advantage of their qualities; namely high speed and high driving capabilities of Bipolar, and small area and low power consumption of CMOS. This kind of optimization is achieved in the logic level resulting in an optimization of the entire network [EBE93].

Many techniques have been proposed to improve the performance of VLSI circuits

at the physical level. One of these techniques is to optimize the logic blocks that constitute these circuits. In the standard cells design, the optimization can be made by an optimal selection of functional cells from a cell library. This type of library is composed of different versions of cells of different characteristics such as size and delay. However all the cells use the same technology (e.g., CMOS) [LMSK90].

For further improvement of the performance of VLSI circuits, a mixed technology design can be developed to take advantage of the individual technology. One possibility is to mix CMOS and BiCMOS cells within the same circuit in order utilize the high speed and high driving capabilities of BiCMOS, and the regularity and low power consumption of CMOS.

1.2 Objectives and Scope of the Thesis

The objective of the thesis is to develop a methodology and right tools to optimize mixed CMOS/BiCMOS circuits in terms of delay, power and area. Although the scope of the work is directed to CMOS and BiCMOS technologies, other technologies can be included taking into consideration the feasibility and practicality of mixing these technologies. The input circuits are in standard cell format. The basic idea is as follows. Given a circuit consisting of CMOS cells only, some of those cells are selected and replaced by their equivalent BiCMOS cells in such a way that the entire delay of the circuit is decreased with a minimum increase in power and area.

Several questions can be raised while achieving this objective: What are the cells that have to be selected? Do we have to consider all the cells in this process? Is the selection problem *NP*-hard? What is the most efficient method for selection to produce the objective optimized circuit? How are the delay and power of a given circuit calculated? These questions and others are answered in this thesis. For ease of reference, we will refer to the problem of cells selection for optimizing a given circuit as *Circuit Optimization Problem (COP)*.

Although the idea of our work is based on the work done in [BAB94], the approach and methodology are different. The proposed approach is more efficient and faster because the search space is restricted to a portion of the circuit rather than the entire circuit. The results are better and more accurate than those found in [BAB94].

We focused on delay and power for optimization. Area of layout is not considered because it is not affected much in the process as the difference in area between CMOS and BiCMOS cells is slight compared to other technologies. When other technologies of large area scale are meant for this type of optimization, our method has to be slightly modified to incorporate the area factor.

1.3 Difficulties and Limitations

Given below is a summary of difficulties and limitations encountered as normal in a research work.

1. Lack of resources related to this field of research, particularly in the BiCMOS technology.
2. Development and usage of the tools have been done on different platforms, hence the tools have to be modified slightly each time the platform is changed.
3. Non-availability of BiCMOS cell library.
4. Non-availability of power model for BiCMOS technology in the literature. In order to overcome this problem, capacitance is used as proportional to the power. This point will be explained later.
5. Limitation of the tools that have been used for generating critical paths of a given circuit. The tools work only for small circuits and do not consider the elimination of false paths.
6. Difficulties in obtaining benchmark circuits in VPNR format which is the input to critical paths tools. For this issue, different tools have been used to convert the available ISCAS'85 benchmark circuits into VPNR format.

1.4 Outline of the Thesis

This thesis describes in detail the work that has been done to address COP. It covers all the research, processes, algorithms, tools, methodologies, implementation, results, analysis and discussions. Throughout the thesis, each chapter begins with an introduction that serves as a link to the preceding one and gives an overview of that chapter. Also, each chapter concludes with a summary of the main ideas and outcomes. Whenever essential, the ideas and points in reference are followed by a figure or an illustration. The thesis is organized as follows. Chapter 2 presents material background and literature review on most of related research previously made. In Chapter 3, the idea of our work is explained and formal definition of COP is presented. Also the proposed solution process and technique that have been adopted are provided in this chapter. The proposed solution consists of three phases. The details of phase I and phase II are presented in Chapter 4 while details of phase III are presented in Chapter 5. Chapter 4 highlights the timing analysis and critical path problem and gives a detailed description of the algorithm used in our work. In addition, this chapter discusses the problem of false paths as well as the adopted false path detection algorithm supported with an illustrative example. Chapter 5 deals with *Tabu Search* technique which is the main part of the proposed solution approach to COP. Explanation of tabu search and all its memory components is presented. In addition, in this chapter, we explained the process of modeling COP

to fit with tabu search technique. Methods of implementing and applying TS on COP, experiments, results, and discussions are provided in Chapter 6. Finally, our conclusions and future work are discussed in Chapter 7.

Appendix A provides a detailed information on the adopted tools such as execution procedure, format of input circuits, and limitations of the tools. In addition, it provides the CMOS and BiCMOS libraries and other tables used in this work.

Chapter 2

Background Material and Literature Review

2.1 Introduction

This chapter provides background about some materials, ideas, methodologies, and processes related to the work. It is an important introduction to the subsequent chapters which discuss COP and the proposed approach for its solution. In Section 2 of this chapter, an overview regarding optimization of VLSI circuits is presented with emphasis on the earlier works in this field. Section 3 addresses critical path and false path problems and reviews some well known algorithms proposed to solve such problems. The last section deals with some of the recent techniques developed for solving common optimization problems such as quadratic assignment and graph

coloring.

2.2 VLSI Optimization

A combinational circuit is required to operate as fast as possible and in no case slower than a given speed limit. That is, the delay of a circuit should not be longer than the system clock period. If a circuit is verified to be slower than the clock period, its performance needs to be improved. However, most of the time the improvement in performance is achieved at the expense of extra area and extra power. Therefore, many techniques have been proposed to optimize circuits in terms of those three parameters, especially in VLSI circuits where area and power are important issues.

According to the design level, optimization techniques can be divided into three categories:

1. At the **structural level**, the internal structure of gates and their interconnections are modified to improve circuit performance. For example, the technique of converting a ripple-carry adder into carry-lookahead adder by the use of Shannon factorization belongs to this category [CDL93].
2. At the **physical level**, performance-driven placement of gates and performance-driven routing of wires are aimed at minimizing the delay of the longest paths [CDL93].

3. At the **gate/circuit level**, techniques for transistor sizing, buffering and powering are used to improve gate speed while the topology of the whole circuit is retained [CDL93].

Due to their ability of retaining circuit topology, gate/circuit level optimization techniques are usually the first to be employed in optimization process. Physical or structural level techniques are applied when gate/circuit techniques cannot achieve the goal [CDL93].

One of the main techniques applied at the gate/circuit level, is the optimization of logic blocks constituting the VLSI circuits in terms of speed and area. When using standard cells approach, the optimization can be performed by an optimal selection of different templates (gates or cells) in the same technology. These templates differ in area, driving capabilities, intrinsic delay, and capacitive loading [LMSK90].

For further improvement of performance of VLSI circuits with minimum area and power penalty, a mixed design of different technologies can be adopted. One possible choice is to mix CMOS and BiCMOS technologies in order to take advantage of their qualities, namely high speed and high driving capability of BiCMOS, and regularity and low power consumption of CMOS. In terms of manufacturing process, this mixture is feasible in the sense that the CMOS process is part of the BiCMOS process. The CMOS-based BiCMOS process is a CMOS baseline process to which a bipolar transistor is added [EBE93]. Therefore, for developing mixed design circuits, initially all cells are exposed to CMOS process. Then bipolar transistors

will be added only to those cells that are selected to be BiCMOS.

In [BAB94], the above approach for optimizing standard cells circuits has been used. The technique aims at improving circuit performance by making for each gate, a choice between CMOS or BiCMOS cells depending on their load capacitance. The comparison between CMOS and BiCMOS gate delay driving a capacitive load CL shows that the two curves intersect at the crosspoint CX as shown in Figure 2.1. Based on this feature, the selection technique is applied as follows:

1. Initially, all gates are considered as CMOS in order to increase circuit density.
2. Output gates are replaced by BiCMOS gates because the inter-block wiring capacitance is assumed to be large.
3. The remaining gates are selected by comparing the value of their respective output load capacitance to the corresponding crossover capacitance CX . This load is equal to the sum of all input capacitance of the driven gates and the interconnection load.
4. Since all gates driven by the gate under check must have already been selected, a Breadth First Search (BFS) is adopted. The BFS algorithm reported in [BAB94] successively selects gates according to their decreasing logical level, from the primary outputs to the primary inputs. The BFS algorithm associates with each gate a counter representing the number of driven gates. Each time a gate is selected, the counters of all gates driving it are decremented.

The algorithm has been tested on ISCAS'85 benchmark circuits and achieved speed improvement by about 11% to 41% as compared to their pure CMOS version. The overhead in terms of area and power is shown to be small [BAB94].

Although the reported approach achieved a good improvement in performance, it suffers from several problems as follows:

1. Traversing all the nodes is not a necessary step because it is time consuming.
2. Replacing all output nodes produces overhead in area and power because some of them are not on time critical paths.
3. This approach is local; that is, it performs the optimization on a single node. It does not have a global view of the circuit, hence it is expected to give a local optimum solution.
4. It is net oriented not path oriented; hence it does not show the actual delay of the circuit.

2.3 Critical Path Problem

The actual delay of a circuit is determined by the delay of its longest *sensitizable path*. A sensitizable path is the path which can be activated by at least one input vector. The paths which cannot be activated by any input vector are called *false paths*. A path is *critical* if its total delay is greater than the clock period. Thus, the

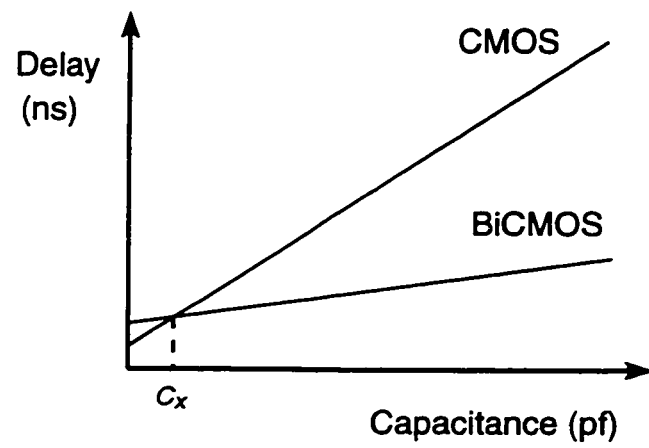


Figure 2.1: Delay vs. load for CMOS and BiCMOS Nand gate.

problem of finding and estimating the delay of critical paths is called the *critical path problem* [CDL93]. Only the sensitizable paths are called “critical paths” by some authors. In this work, the term “critical paths” refers to both sensitizable and false paths.

Timing simulation and timing analysis are two popular approaches to verify the delay characteristics of combinational circuits. A timing simulation algorithm simulates each input vector on the circuit so as to identify all the sensitizable paths. After finding all the sensitizable paths, the simulation algorithm returns the delay of the longest paths as the actual delay of the circuit. Due to the very large computation time required to simulate all the vectors which grows exponentially with the complexity of the circuit, researchers focus more on the timing analysis approach.

Timing analysis ignores the logic properties of the circuit elements and checks only the timing behavior of the circuit. This fact makes timing analysis very time efficient as compared to simulation. Ignoring the functionality of the circuit elements, however, is responsible for the difficulty of the false path problem.

In most of timing analysis techniques, the circuit is modeled as acyclic directed graph in which three famous algorithms are used to trace the paths. These algorithms include Depth First Search (DFS) with/without pruning, Breadth First Search (BFS) and PERT-like trace [AF95].

2.3.1 Critical Path Algorithms

One approach to critical path prediction is to report the most critical path in the circuit. This approach is also referred to as the *Block Oriented Approach* [YDG89]. However, reporting only the most critical path often fails to give sufficient information to the designer to correct the timing violations because there might be more than one longest path having the same delay or the subsequent longest paths may violate the timing constraints as well.

To get more information, another approach called *Path Enumeration* is taken up in which all the paths are enumerated and then critical paths violating some timing constraints are reported. Although this approach gives more information than the previous one, it suffers from the large computation due to the fact that the total number of paths grows exponentially with the size of the design [YDG89].

A third approach is to find the K most critical paths. In this approach, all the paths which are greater or smaller than a given *threshold* are reported. The threshold is usually the clock period but it can be of any timing limit. A variation of this approach is reported in [YDG89]. In this algorithm, if the given threshold is too small, the number of paths being reported will increase rapidly. Therefore, it is preferable to choose a proper threshold and limit the number of paths to the K most paths [YDG89].

The algorithm reported in [You90] also provides the K most critical paths in

descending order. In this algorithm, a score is computed for each enumerated path. The score is a function of parameters that are correlated with the total path delay. Examples of these parameters include: load factors, number of nets on the path, etc. The K paths with worst scores are the K most critical paths. Linear regression model is used to predict the delay of interconnections at the net level as well as at the path level. However, the regression approach does not produce desirable results due to the large prediction errors of the model.

A variation of the above approach has been proposed in [AF95]. This approach is called *α -critical* and described in detail in Chapter 4. The time complexity of these algorithms is proportional to the reported number of critical paths [AF95].

2.4 False Path Problem

The timing analysis based techniques ignores the functionality of the circuit. These techniques aim at finding the longest paths either sensitizable (true) or unsensitizable (false). The presence of false paths has many undesirable effects:

- The loss of accuracy. There is no theoretical limit between the longest true path and the longest false path. Therefore the loss of accuracy can be large and results in loss of confidence in the timing analysis tool.
- Waste of optimization effort. Since the longest propagation path determines the length of the clock period, a large difference between the longest false path

and the longest true path will result in unnecessary conservative designs, or alternatively wasted power and silicon area. Thus, eventually optimization efforts will be spent on false paths rather than on the true paths.

2.4.1 Sources of False Paths

1. *Incompatible transitions:* In this very simple case, a false path results from the combination of incompatible transition. For example, in subsequent inverters, the addition of the delays associated with the $1 \rightarrow 0$ transitions instead of alternating the $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions [BMCM90] is a source of false paths.
2. *Incorrect signal flow:* Timing verifiers that operate at switch level encounter this problem. Due to the bi-directional nature of MOS transistors, the intended signal flow in structures such as barrel shifter is not always obvious [BMCM90].
3. *Synchronization:* Whether the synchronization is simply performed within the clocking scheme or in a more complex way, it implies that signals are latched and have to wait for next synchronization point. When transparent latches are used, depending on their arrival time, signals might be allowed to continue to propagate. Therefore, the propagation conditions of these latches may or may not be compatible with those of the path that lead to them. In the latter case, ignoring these conditions will lead some timing verifiers to incorrectly

view these latches and subsequent combinational logic as part of the path [BMCM90].

4. *Logic dependency*: The most explicit source of false paths comes from some logic that depends on the output of other logic. For example, in the circuit shown in Figure 2.2, the path *a d i f y* can not be activated because both multiplexers can not select 1-input at the same time [PCM89].

2.4.2 False Paths Detection Algorithms

Recently a number of techniques have been proposed to detect the false paths. Some of them aim at finding the sensitizable paths directly, hence avoiding false paths reporting. Other techniques detect the false paths and remove them from reporting to the designer. Each of these techniques uses a path sensitization criterion. All of the reported criteria can be classified into three main conditions: *Static*, *Dynamic* and *Viable*. This section gives an overview regarding most of the false paths detection techniques according to these three conditions.

Static Sensitization:

The first techniques to detect false paths are based on D-Algorithm which is used extensively in testing. One important assumption of the D-Algorithm is that, except the signal to be propagated, all other signals in the circuit are assumed to be always *stable*. A signal *s* becomes *stable* when it has the logical value determined by the

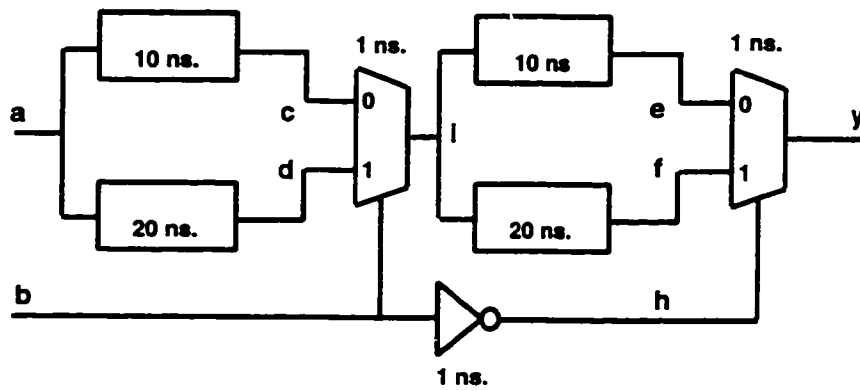


Figure 2.2: Logic dependency is a source of false paths [PCM89].

Boolean expression of s and the logical values of primary inputs. This assumption, however, is not applicable when the D-Algorithm is used for the detection of false paths because it will give incorrect results. First it will report some paths as false paths while in reality they are not. Second, it can underestimate the sensitizable path length. In Chapter 4, we will show one example that demonstrates these effects. Some of the algorithms that use static conditions are reported in [Rot66], [BI86], and [BMCM90].

Dynamic Sensitization:

Since the algorithms based on static sensitization generate inaccurate results as mentioned above, some researchers propose other techniques that release the assumption of the D-algorithm.

In [DYG89], the authors propose an algorithm to detect whether a given path is false or not by using an input independent approach which takes the *stable* time of signals into account. They show that the time required for a signal to be stable depends highly on both the stable times and logical values of its input signals. The detail of this algorithm will be presented in Chapter 4.

Another approach that computes the longest dynamically sensitizable paths has been proposed in [PCM89]. In this approach, propagation conditions for the signals that take the dynamic behavior of the circuit into account are used. The propagation condition for a certain signal explicitly depends upon arrival time of its predecessors.

Two approaches proposed in [HPS93] use what is called *Timed Boolean Cal-*

culus. The dynamic behavior of a given logic circuit is modeled as timed Boolean expressions. Then for each term in the expressions, its corresponding sensitizability function is computed. The terms whose sensitizability functions are not satisfiable are removed and the maximal delay for each node are determined from the remaining terms. For a given path, if the maximal delay of one of its internal node is greater than the maximal delay of any of the successor nodes, then that path is claimed to be false.

Chen and Du [CD93] have proposed three sensitization criteria; namely *exact*, *loose*, and *dynamic*. In exact criterion, all input vectors that can activate a given path are found. So, if no vectors are found, then the path is claimed to be false. This criterion is called exact because it can yield exactly the same critical path delay as the gate level timing simulation approach. Loose criterion is used to make the comparison between different criteria easier in the sense that some proposed criteria are meant for sensitization of long paths only, not for general path sensitization problem. Under dynamic criterion, a lead (signal on the path under test) is considered to dominate its succeeding gate if it is either a lowest earliest controlling input or a noncontrolling input with all its side inputs being noncontrolling inputs too. Using this criterion, at least one critical path is identified.

Viability:

The flaw in the dynamic sensitization condition is the absence of perfect knowledge of the delay of gates. In such case, both the exact value of any node at any time

before the node has settled to a final value and the time at which a node settles to a final value is problematic. That means the above mentioned methods use criteria which are non-robust.

MaGeer and Brayton [MB89] developed a technique that computes the longest viable path in combinatorial circuits. Their technique is based on two conditions: correctness and robustness. These two requirements are derived from the idea of *Boolean Difference*. A path is defined to be sensitizable if there exists at least one input vector to allow a transition to propagate through the path. The path sensitization criterion associates the path with a Boolean expression, which represents the set of input vectors that activate the path. If the associated Boolean expression of the path is computed to equal logic 0, the path is claimed to be a false path.

2.5 Optimization Techniques

The problem of optimizing mixed technology design is considered as one of combinatorial problems. For a mixed CMOS/BiCMOS design, there exist 2^n solutions for a circuit with n gates. Therefore, it can not be solved by polynomial-time algorithms. *Heuristic* techniques can be applied which generate optimal and near optimal solutions to a wide variety of *NP*-complete problems. Colin Reeves defines a heuristic as “a technique which seeks good (i.e., near-optimal) solutions at a reasonable computational cost without being able to generate either feasibility or optimality, or

even in many cases to state how close to optimality a particular feasible solution is” [Ree95].

Heuristic techniques can be classified into *deterministic* and *nondeterministic*. A deterministic algorithm progresses toward the solution by making deterministic decisions. On the other hand a non-deterministic algorithm makes random decision to find the solution. Therefore, deterministic algorithms always produce the same solution for a given input instance, while non-deterministic algorithms may produce different solutions. Another classification of heuristic algorithms is based on the initial input: *Constructive* and *Iterative*. A Constructive heuristic algorithm receives the problem description as input and then constructs a solution to the problem. An iterative heuristic algorithm receives the problem description and an initial solution as input. Then it attempts to modify the initial solution so as to improve a cost function. The iterative procedure is applied repeatedly until no cost improvement is possible. Usually, a constructive technique is first applied to find a certain solution and then an iterative technique is applied to produce a better solution.

Recently, several heuristic techniques have been proposed to find optimal and near optimal solutions to a wide variety of classical and practical optimization problems. Some of these well known techniques include *Simulated Annealing*, *Genetic Algorithm* and *Tabu Search* [SY95]. They have been applied in areas ranging from scheduling to telecommunications, and from character recognition to neural networks.

Simulated Annealing is an *adaptive* heuristic technique which was first introduced by Kirkpatrick, Gelatt and Vecchi in 1993. It was first used for trying to obtain a good crystal structure of a metal through a process of heating it to high temperature to break the chemical bond and then slowly cooling it to get a certain structure. By varying the cooling rate, time of the annealing process, and other parameters, we get different results.

Comparing the annealing process to optimization process, the aim of getting global optimum solution is analogous to the aim of getting a good crystal structure.

Genetic Algorithm derives its name from the natural process of evolution to get the optimum characteristics of an individual from a large population of different characteristics. The idea of this algorithm is as follows. Given a set of strings of symbols, during each iteration, the strings in the current solution are evaluated using some measures of *fitness*. Based on the fitness, two strings at a time are selected and a number of genetic operations are applied on the selected strings to generate a new string. These operations include *crossover*, *mutation* and *inversion*. The process is repeated until an optimum string is obtained [SY95].

Tabu Search is a *metaheuristic* which can be used as an independent search technique or as a higher level heuristic procedure for solving problems. It is basically designed to guide other methods to escape the trap of local optimality. TS operates by incorporating flexible memory functions to forbid transitions (*moves*) between solutions that reinstate certain attributes of past solutions. Attributes that are not

permitted to be reinstated are called *tabu*, and are maintained in short term memory on a list called *tabu list*. After a specified duration they are removed from the list and become free for re-insertion.

In a variety of problems, TS has found solutions superior to the best previously obtained by alternative methods. In other cases, it has demonstrated advantages in ease of implementation or in the ability to handle additional considerations such as constraints not encompassed by an original problem formulation [Glo90].

Some of the main applications of TS include: bandwidth packing [LG93a], quadratic assignment problem [BT94], graph coloring [Hd87], graph partitioning [LC91], VLSI placement [SV92], and scheduling and allocation in high-level synthesis [Ali94].

2.6 Summary

In this chapter, some background on VLSI optimization, critical and false paths problems, and some of the optimization techniques have been addressed. In addition, we have presented some proposed techniques very popular in each area. Our proposed approach to the COP, as will be shown later, is strongly based on these concepts and algorithms.

Chapter 3

Problem Definition and Proposed Solution Technique

3.1 Introduction

The COP is an *NP*-Complete problem. However, this claim needs to be proved based on the theories of *NP*-hard and *NP*-complete problems. In this chapter, a formal definition of the COP is given. Then we prove that this problem is *NP*-complete and therefore requires a heuristic technique for its solution. Finally, the proposed technique of generating an optimized mixed CMOS/BiCMOS circuit is presented.

3.2 Problem Definition

The objective of this work is to find an optimal or near-optimal solution to the problem of mixing CMOS/BiCMOS gates in one circuit such that the overall delay is minimized with minimum increase in the power and area. Only the longest sensitizable paths, that the delay of a circuit depends on, are considered in the search. Moreover, the BiCMOS gate has less delay than the corresponding CMOS only if its fanout load capacitance CL is greater than a certain threshold CX as mentioned in [BAB94]. Therefore only those nodes with $CL > CX$ will be considered.

Given a circuit with m nodes and K sensitizable paths, we should first extract all nodes that are included in the sensitizable paths and satisfy the inequality $CL > CX$. Let A be the set of such nodes and let n be the size of this set.

The input for the COP is:

- Set $A = (g_1, g_2, \dots, g_n)$
- Vector $\mathcal{D} = (\Delta D_1, \Delta D_2, \dots, \Delta D_n)$

$$\Delta D_i = D_{g_i}^c - D_{g_i}^b$$

where:

ΔD_i is the circuit delay gain due to changing gate i implementation from CMOS to BiCMOS. Obviously, for all $g_i \in A$, $\Delta D_i \geq 0$. The value “0” is

because g_i may be swapped without any improvement in the delay when it is not a part of longest path.

$D_{g_i}^c$ and $D_{g_i}^b$ is the circuit delay when gate g_i is implemented in CMOS and BiCMOS respectively.

- Vector $\mathcal{C} = (\Delta C_1, \Delta C_2, \dots, \Delta C_n)$

$$\Delta C_i = C_{g_i}^b - C_{g_i}^c$$

where:

ΔC_i is the total capacitance increase of the circuit due to changing gate i implementation from CMOS to BiCMOS.

$C_{g_i}^c$ and $C_{g_i}^b$ is total capacitance of the circuit when g_i is CMOS and BiCMOS respectively.

Due to non-availability of power dissipation model for BiCMOS, we expressed the changes in power in terms of changes in capacitance. The power of CMOS and BiCMOS gates is proportional to their capacitive load [EBE93].

The problem is to find a subset S of A such that when the nodes in S are implemented by BiCMOS lead to maximum reduction in T_{max} , while satisfying a threshold constraint on capacitance, that is,

$$\begin{cases} \text{maximize} & \sum_{i \in S} \Delta D_i \\ \text{subject to} & \sum_{i \in S} \Delta C_i \leq C_T \end{cases} \quad (3.1)$$

The term $\sum_{i \in S} \Delta D_i$ reflects the total circuit delay gain due to changing a set S of gates from CMOS to BiCMOS. C_T is a user specified threshold which represents the maximum allowable total capacitance increase.

The output is:

- S where $S \subseteq A$.
- $\Delta D = \sum_{i \in S} \Delta D_i$
- $\Delta C = \sum_{i \in S} \Delta C_i$

If T_{max} is the initial delay of the circuit, then the final delay:

$$D_F = T_{max} - \Delta D \quad (3.2)$$

Let C_I be the initial total capacitance. Then the final capacitance is,

$$C_F = C_I - \Delta C \quad (3.3)$$

Next, we shall show that the decision version of COP is *NP*-Complete.

3.2.1 Is COP in NP ?

In order to prove that COP is in NP , we need to find a nondeterministic algorithm that could be used to solve the problem in polynomial time [HS90]. Before doing this, let us present some definitions and concepts. $Choice(A)$ is a function that arbitrarily chooses one of the elements of a set A . *Success* and *Failure* are two signals to indicate a successful and unsuccessful completion of the algorithm respectively. The assignment statement $X \leftarrow Choice(1 : n)$ could result in X being assigned any value of the integers in the range $[1, n]$. There is no rule to specify how this choice is to be made. Whenever there is a set of choices leading to a successful completion then one such set of choices is always made and the algorithm terminates successfully. A nondeterministic algorithm terminates unsuccessfully if and only if there exists no set of choices leading to a success signal [HS90].

For COP, let “0” and “1” represent the choice between CMOS and BiCMOS, and \mathcal{X} represents a set of these choices. Let M be a maximum objective reduction in the delay. Then we can formulate a nondeterministic algorithm for COP as shown in Figure 3.1. As you can see from the figure that there is no rule of how to guess a solution that leads to a successful termination of the algorithm. The time complexity of this algorithm is $O(n)$.

```

algorithm  $COP(n, M, C_T, \mathcal{X}, \mathcal{D}, \mathcal{C})$ 
  for  $i \leftarrow 1$  to  $n$  do
     $X_i \leftarrow \text{choice}(0,1)$ 
  repeat
    if  $\sum_{1 \leq i \leq n} (\Delta D_i \times X_i) < M$  or  $\sum_{1 \leq i \leq n} (\Delta C_i \times X_i) > C_T$  then failure
    else success
    endif
  endCOP

```

Figure 3.1: Nondeterministic COP algorithm.

3.2.2 Is COP *NP*-Complete?

The COP is an *NP*-Complete problem if it is *NP*-Hard and belongs to the *NP* class of problems. In the previous section, we proved that COP is *NP*. Now let us try to prove that it is *NP*-Hard. To do so, the following steps need to be carried out: [Baa91]

1. Select an *NP*-Complete problem Π .
2. Show that Π is reducible to COP by finding a polynomial function $T(x)$ that transforms (reduces) Π to COP

Let Π denotes the Knapsack problem which is known to be *NP*-Complete problem [HS90]. The definition of Knapsack problem is as follows: A set of n items is available to be packed into a knapsack with capacity C units. Item i has a profit p_i and uses up s_i units of capacity. The problem is to determine the subset I of items which should be packed in order to maximize:

$$\sum_{i \in I} p_i \tag{3.4}$$

such that

$$\sum_{i \in I} s_i \leq C$$

Here the solution is represented by the subset $I \subseteq \{1, \dots, n\}$. Now let us show

that Π is reducible to COP. As we can see there is a correspondence between the input/output of COP and the input/output of Π . That is:

- Both of the problems have an input of n items.
- Item i in COP has ΔD_i gain in the delay (profit) which corresponds to the profit p_i of item i in Π .
- Item i in COP uses ΔC_i units of capacity which corresponds to the capacity s_i of item i in Π .
- The objective of both problems is to find out a subset of items that maximize the total gain (profit).
- Both problems COP and Π are subjected to some constraints of a given capacity threshold, C_T and C respectively.

Let $T(x)$ be a polynomial reducible function from Π to COP. Then from the previous correspondence we can deduce the following:

$$T(C) = C_T \tag{3.5}$$

$$T(s_1, s_2, \dots, s_n) = \Delta C_1, \Delta C_2, \dots, \Delta C_n \tag{3.6}$$

$$T(p_1, p_2, \dots, p_n) = \Delta D_1, \Delta D_2, \dots, \Delta D_n \tag{3.7}$$

It is clear that $T(x)$ is one-to-one function of $O(n)$ time complexity. This means that Π is reducible to COP. Hence COP is NP -Hard problem. Since COP is NP -Hard and at the same time it belongs to NP class of problems, then it is NP -Complete.

The above result justifies searching for a heuristic solution to this problem. The heuristic solution adopted is described next.

3.3 The Proposed Solution Technique

The proposed solution technique consists of three phases:

1. Generate critical paths of the input circuit.
2. Eliminate false paths from the generated critical paths.
3. Apply TS algorithm to select a subset S of BiCMOS gates among those covered by the sensitizable critical paths so as to minimize delay while satisfying a threshold constraint on circuit capacitance.

The general structure of the process is shown in Figure 3.2. In the following sections each phase will be described in more details.

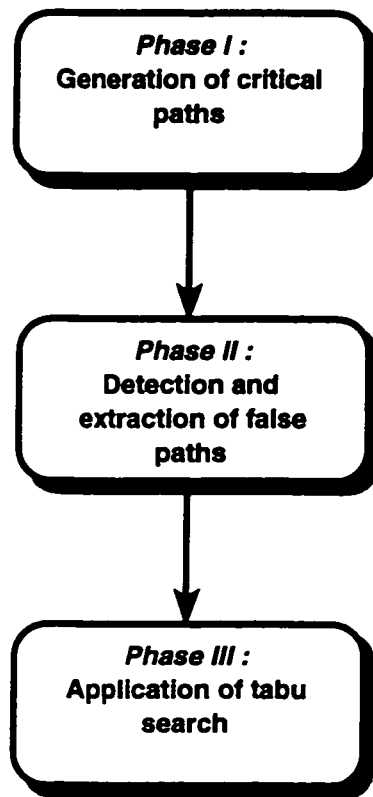


Figure 3.2: General structure of the proposed solution technique.

3.3.1 Phase I

As we have seen in Section 2.3.1, it is not feasible and even not necessary to consider all the paths in the optimization process. Instead, optimizing only the longest sensitizable paths eases the problem and decreases the computation time considerably. Many techniques have been proposed to find the longest sensitizable paths as described earlier. Since our main goal emphasizes on the output of the optimization process, we can choose any sensitization method which generates reasonable results. Nevertheless, α -critical algorithm [AF95], which generates critical paths including false paths, has been selected because of the following reasons:

- It can easily be interfaced with the adopted false path checking algorithm.
- The algorithm considers the delay of interconnections resulting in more accurate timing prediction.
- It is flexible in generating as many critical paths as required by the user.
- The tools implementing this algorithm are available to us.

The detailed description of α -critical algorithm and its application on some benchmark circuits are presented in the following chapter.

3.3.2 Phase II

After generating all critical paths for a given circuit, these paths are checked via false path checking algorithm to eliminate the false paths. This step is necessary to speed up the optimization process by minimizing the input population as well as providing accurate timing. As mentioned earlier, many techniques have been proposed for this purpose. We have chosen the algorithm reported in [YDG89] for the following reasons:

- It is easy to interface with the α -critical algorithm.
- It is based on dynamic criterion by which more accurate results are generated.
- It is easy to implement.
- The algorithm focuses on the general false path problem, which detects whether a given path (not necessarily the longest one) is false or not. In this case, any number of paths having the same delay can be detected. This capability is not available in most of the other techniques.

The detailed description of this algorithm and its application on some benchmark circuits are presented in Chapter 4.

3.3.3 Phase III

The main contribution of our work is included in this phase which covers the process of optimal selection of gates required to be implemented in BiCMOS technology. The input of this process is the output of the second phase in which the longest sensitizable paths are generated. The goal is to replace some of the CMOS gates by BiCMOS gates in order to optimize the circuit for delay and capacitance. The output is a mixed CMOS/BiCMOS circuit with optimum cost. Since this process involves a lot of processing and data manipulation, the required optimization algorithm has to be correct, effective, and efficient. TS algorithm has proven to be efficient and powerful in dealing with very complex problems. In addition, we have chosen TS for the following reasons:

- It is modular and easy to implement.
- Its ability to handle additional considerations such as constraints not encompassed by the original problem formulation.
- It is able to escape the trap of local optimality.

The details of this algorithm and how it has been applied on COP are presented in Chapter 5.

3.4 Summary

In this chapter, the formal definition of COP has been presented. We have proven that the decision version of this problem belongs to the class of *NP*-Complete problems which require heuristic techniques to find sub-optimal solutions. Our proposed solution technique to this problem consists of three phases. The first and second phase aim at finding the K longest sensitizable paths for a given circuit because circuit delay is determined by the longest sensitizable paths. The third phase is the core phase which includes the application of TS algorithm.

Chapter 4

Timing Analysis and False Path Problem

4.1 Introduction

This chapter covers the timing analysis and false path problem. In Section 2 of this chapter, some timing analysis concepts and a detailed description of α -critical path algorithm, which represents the first phase in the proposed solution technique to COP, are presented and illustrated with an example. The second phase of our proposed solution technique is the elimination of false paths from the generated critical paths. This phase is covered in Section 3 where some aspects of false path problem and drawbacks of D-algorithm to detect false paths are explained with an illustrative example. In addition, this section describes in detail the adopted false

path detection algorithm and the method of implementation. The experimental results of applying both α -critical path and false path detection algorithms are presented and discussed in Section 4. Then we conclude in Section 5. We begin with the discussion on some timing analysis concepts that have been taken from [AF95] and summarized for completeness.

4.2 Timing Analysis and α -Critical Approach

The delay of the circuit is determined by its longest sensitizable paths. Therefore, to verify and optimize the circuit timing, the focus should be on predicting the timing critical paths only. A path π is classified as critical if its total delay, T_π , is very close to its latest required arrival time $LRAT_\pi$. If T_π exceeds $LRAT_\pi$, path π becomes a long path. The path delay consists of two components: the logic delay which is known prior to layout, and the interconnect delay which is unknown. In VLSI designs, the interconnect delay is a major part of the overall path delay. Therefore it is very important for pre-layout timing analysis to predict the interconnect delay requirements. The interconnect capacitance is a key element in the total interconnect delay [AF95]. The α – *critical* algorithm aims at predicating the interconnect delay requirements of a given circuit by estimating the delay of the longest paths in the circuit. Before we describe this algorithm, we recall from [AF95] some definitions and equations proposed to compute the path delay.

Let $\pi = \{v_1, v_2, \dots, v_p\}$ be a path in the circuit graph, where v_1 and v_p are the source and sink cells. The total delay on π is given by,

$$T_\pi = \sum_{i=1}^{p-1} (CD_{v_i} + ID_{v_i}) \quad (4.1)$$

where, CD_{v_i} is the switching delay of cell v_i and ID_{v_i} is the interconnect delay of the net driven by cell v_i .

The switching delay may be expressed as follows,

$$CD_{v_i} = BD_{v_i} + LF_{v_i} \times AcL_{v_i} \quad (4.2)$$

where, BD_{v_i} is the base (intrinsic) delay of cell v_i in nanoseconds, LF_{v_i} is the load factor of the output pin of the driving cell v_i , expressed in units of time per unit capacitance, and AcL_{v_i} is the summation of input capacitance of fan-out gates of cell v_i .

The interconnection delay may be expressed as follows,

$$ID_{v_i} = LF_{v_i} \times C_{v_i} \quad (4.3)$$

where, C_{v_i} is the total interconnect capacitance (area + fringe) of the net driven by cell v_i .

The interconnect capacitance C_{v_i} is estimated using data from past designs as follows. The average and standard deviation of net length for different types of nets (2-pin, 3-pin, ..., m -pin) are collected from past designs of similar complexity¹.

¹this classification helps reduce the sample variance around the mean

These are transformed into interconnect capacitances. Let $\overline{C_{v_i}}$ and s_{v_i} be the estimated expected interconnect capacitance and standard deviation of the net driven by cell v_i . Then, the expected interconnect delay $\overline{TD_{v_i}}$ of net v_i and its corresponding variance $S_{v_i}^2$ are estimated as follows:

$$\overline{TD_{v_i}} = LF_{v_i} \times \overline{C_{v_i}} \quad ; \quad S_{v_i}^2 = LF_{v_i}^2 \times s_{v_i}^2 \quad (4.4)$$

Under the assumption of statistical independence between the nets, the expected delay and variance on any path π can be expressed as follows,

$$T_\pi = \sum_{i=1}^{p-1} (CD_{v_i} + \overline{TD_{v_i}}) \quad ; \quad S_\pi^2 = \sum_{i=1}^{p-1} S_{v_i}^2 \quad (4.5)$$

Let T_{\max} be the expected delay of the longest path in the circuit, that is,

$$T_{\max} = \max_{\pi \in \Pi} (T_\pi) \quad (4.6)$$

where Π is the set of all paths in the circuit graph G .

4.2.1 Description of α -Critical Algorithm

The α -critical approach is based on the following definition:

Definition: A given path Π , with overall delay T_Π , is α -critical iff:

$$T_\Pi + \alpha \times \sqrt{S_\Pi^2} \geq T_{\max} \quad (4.7)$$

For a user specified α , the α -critical approach enumerates all paths which satisfy Equation 4.7. The parameter α is interpreted as a confidence level. $T_\Pi + \alpha \times \sqrt{S_\Pi^2}$

means that we are $\alpha \times \sqrt{S_{\Pi}^2}$ ns confident that path Π is critical. The higher α is, the larger the number of reported paths will be, and the higher is the probability of capturing all the critical paths [AF95].

The input circuit to the algorithm is modeled as an acyclic directed graph $G = (V, E)$, where V is the set of vertices representing the signal nets and E is the set of edges representing the nodes. To trace all vertices and enumerate the paths, *Depth First Search* with pruning is applied. The algorithm starts with backward trace from primary outputs to primary inputs. During this step, for each vertex v the maximum delay $MAXD_v$ and the maximum variance $MAXV_v$ between the vertex and the primary outputs are computed. Then forward trace is performed to enumerate the critical paths. During this step, when the search reaches a vertex v along the partial path Π with partial delay T_{Π} and variance S_{Π}^2 , the following criticality test is made:

$$T_{\Pi} + MAXD_v + \alpha \times \sqrt{S_{\Pi}^2 + MAXV_v} \geq T_{max} \quad (4.8)$$

If the test is positive, meaning that the partial path Π is α -critical, the search continues. If not, this means that the partial path Π is not α -critical. Hence all paths starting with this partial path are not α -critical too. Therefore, it is not necessary to continue the search. In this case, the algorithm stops and backtrace to the predecessor vertex (*pruning the search*). Once all the α -critical paths are found, they are sorted in descending order and then reported as an output [AF95].

4.2.2 Example

Let us give a simple example that demonstrates how the algorithm works. Figure 4.1 shows a simple combinational circuit. Assume that for 2-pin and 3-pin nets, the average interconnect capacitance, \bar{c} , is 0.04966 pF and 0.06428 pF respectively, and the standard deviation, s , is 0.03819 pF and 0.02698 pF respectively. The values of base delay, load factor and input capacitance of the gates are taken from Table 7.2.3 in Appendix A. Using the equations stated earlier with the confidence level set to 25, we obtain the following critical paths:

- $P_1 = C - 1 - E - 2 - F - 3 - H - 5 - I - 6 - Y$. $T_{p_1} = 14.056$ ns, $S_{p_1}^2 = 0.128$
- $P_2 = D - 2 - F - 3 - H - 5 - I - 6 - Y$. $T_{p_2} = 10.958$ ns, $S_{p_2}^2 = 0.112$
- $P_3 = B - 3 - H - 5 - I - 6 - Y$. $T_{p_3} = 7.853$ ns, $S_{p_3}^2 = 0.090$

4.3 False Path Problem

The timing analysis based techniques ignores the functionality of the circuit. These techniques aim at finding the longest paths either sensitizable (true) or unsensitizable (false). The presence of false paths has many undesirable effects including loss of accuracy and waste of optimization effort. Therefore, it is essential to eliminate false paths from the critical ones. Many techniques have been proposed to serve this purpose. However, most of them are based on the D-algorithm approach which is

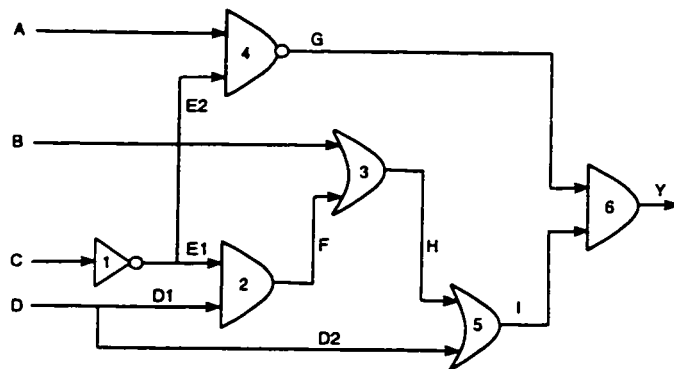


Figure 4.1: A simple combinational circuit.

applied mainly in testing a circuit for stuck_at_fault. During a signal propagation, all other signals are assumed to be stable. When this algorithm is used for false path detection, stableness assumption remains. For most of the time this assumption may be true, especially in detecting the single longest false path. When general path is considered or several longest paths with same delay exist in the circuit, this assumption may not work. The following example demonstrates the idea. Figure 4.2 shows a simple combinational circuit in which the delay of all gates is assumed equal 1 ns, the delay of all signals is assumed equal 0 ns, and all the primary inputs are stabilized at 0 ns. Let $P_1 = B1 - 1 - F - 3 - G - 4 - H - 5 - Y$ and $P_2 = C1 - 1 - F - 3 - G - 4 - H - 5 - Y$. Both P_1 and P_2 are the longest paths with the same delay 4 ns. Using D-algorithm to detect whether path P_1 is false or not, we set signals $C1 = 1, A = 0, D = 0$ & $E = 1$ respectively to allow the signals propagate along path P_1 . Then new values of signals are induced from those signals. Since $C1 = 1 = C = C2$, $C2$ is set to 1. Since gate 2 is NOR and one of its inputs is 1, then E becomes 0. However the initial value of E is 1 which makes E *inconsistent*. Therefore we claim P_1 to be false. Using a similar argument, we can conclude that P_2 is false too. Based on that the worst delay of the circuit is 3 ns instead of 4 ns.

Now let us apply the input ($A = 1, B = 0, C = 0, D = 0$), we get $F = 0$ & $E = 1$ at 1 ns, $G = 1$ at 2 ns, and $H = 1$ at 3 ns. Since $E = 1$ is a non-control value to gate 5, the output Y of gate 5 can not be decided until H is stabilized. Signal H is

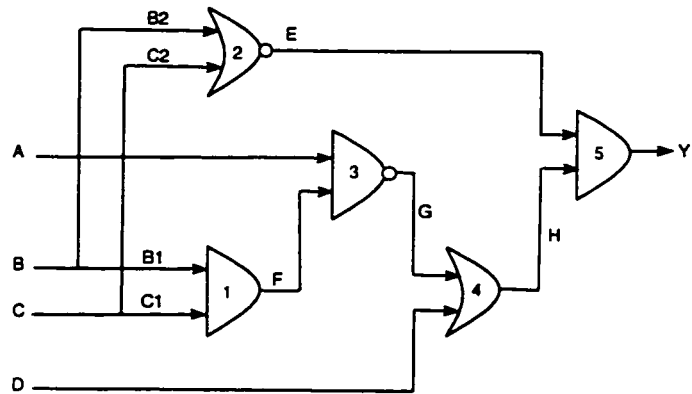


Figure 4.2: Another simple combinational circuit.

stabilized at 3 ns, hence signal Y is stabilized at 4 ns. This condition shows that the worst delay of the circuit is 3 ns which conflicts with the result of the D-algorithm.

4.3.1 False Path Detection Algorithm

From the above example, it is clear that D-algorithm approach does not take the *stability* of the signals into consideration, hence it gives inaccurate results. The approach reported in [DYG89] handles this issue in a very efficient way. The approach is based on the idea that the *stability* of a signal is highly dependent on the *stable time* and the logic values of its input signals. The *stable time* of a signal s is the minimal time for the signal to reach its logic value.

In this work we have implemented this algorithm but with minor modifications. The first modification is the use of a variable signal delay; i.e., for each signal there is a certain delay which is calculated using the model used in [AF95], whereas the original algorithm considers an average signal delay. Other modifications have been made in the way of handling generation of new events in the events propagation phase as will be shown later. Before we discuss this algorithm, it is important to present some definitions from [DYG89] with examples that help one in understanding how the algorithm works.

Definitions

In order to take the *stableness* of signals into consideration during detection of false paths, the following definitions have been proposed in [DYG89]:

Definition 1: For any signal s_i along path P , $P = s_0, g_0, s_1, g_1, \dots, g_{k-1}, s_k$, two special sets of signals for the signal s_i , $0 \leq i \leq k$, with respect to all input vectors can be defined as follows:

$$\begin{aligned} \text{Early_arrive_signals}(s_i, *) &= \{s_j \mid s_j \text{ is an input signal to the gate } g_i \\ &\text{and the maximal time for } s_j \text{ to be stabilized} < \text{the minimum} \\ &\text{signal delay of } s_i \text{ along path } P\} \end{aligned} \quad (4.9)$$

$$\begin{aligned} \text{Late_arrive_signals}(s_i, *) &= \{s_j \mid s_j \text{ is an input signal to the gate } g_i \\ &\text{and the minimal time for } s_j \text{ to be stabilized} > \text{the maximum} \\ &\text{signal delay of } s_i \text{ along path } P\} \end{aligned} \quad (4.10)$$

The symbol (*) in the above definition is to show that the set of *Early_arrive_signals* and the set of *Late_arrive_signals* are vector independent, hence the algorithm is vector independent too.

The above definition can be clarified easily through an example. In Figure 4.2, let us assume that all primary inputs are stabilized at 0 ns, the propagation delay of each gate is 1 ns and all signal delays are 0 ns. Then, along the path $P = B - 1 - F - 3 - G - 4 - H - 5 - Y$, some of the timings are:

minimal time for F to be stabilized = $\min(\text{minimal time of } B, \text{minimal time of } C)$
+ propagation delay of gate (1) + signal delay of F = $\min(0,0) + 1 + 0 = 0 + 1 + 0 = 1$ ns.

maximal time for F to be stabilized = $\max(\text{maximal time of } B, \text{maximal time of } C)$
+ propagation delay of gate (1) + signal delay of F = $\max(0,0) + 1 + 0 = 0 + 1 + 0 = 1$ ns.

maximum signal delay of F along path P = maximum signal delay of B along path P + propagation delay of gate (1) + signal delay of F = $0 + 1 + 0 = 1$ ns.

minimum signal delay of F along path P = minimum signal delay of B along path P + propagation delay of gate (1) + signal delay of F = $0 + 1 + 0 = 1$ ns.

maximum signal delay of G along path P = maximum signal delay of F + propagation delay of gate (3) + signal delay of G = $1 + 1 + 0 = 2$ ns.

minimum signal delay of G along path P = minimum signal delay of F + propagation delay of gate (3) + signal delay of G = $1 + 1 + 0 = 2$ ns.

Also, along the path P , the following sets of signals with respect to signal A and G are defined:

$Early_arrive_signals(A, *) = \Phi$, because there is no input signal whose maximal time to be stabilized is less than the minimum signal delay of A along P (0 ns).

$Late_arrive_signals(A, *) = \{F\}$, because the minimal time for F (1 ns) to be stabilized is greater than the maximum signal delay of A along P (0 ns).

$Early_arrive_signals(G, *) = \{D\}$, because the maximal time for D (0 ns) to be

stabilized is less than the minimum signal delay of G along P (2 ns).

$Late_arrive_signals(G, *) = \Phi$, because there is no input signal whose minimal time to be stabilized is greater than the maximum signal delay of G along P (2 ns).

Definition 2: A path $P = s_0, g_0, s_1, g_1, \dots, g_{k-1}, s_k$, is a possible *sensitizable path* if each signal s_i , $0 \leq i \leq k$, satisfies the following two conditions:

1. Non of the signals in $Early_arrive_signals(s_i, *)$ set has a *control* value of g_i .
2. If $Late_arrive_signals(s_i, *) \neq \Phi$, s_i should have a *control* value of g_i .

Algorithm Description

Based on the above definitions, the false_path_detection (FPD) algorithm can be described as follows. As shown in Figure 4.4, the algorithm receives the path P and returns $false_path = \text{true}$ if P is a false path, or $false_path = \text{false}$ if P is a possible sensitizable path. This algorithm is divided into two phases. The first phase is the *events generation* phase and the second is the *events propagation* phase. The value assigned to the signal forms an *event* and it is denoted as $(signal, value)$. In the first phase, the path P is traversed, starting from the primary input and stops at the primary output, and each signal is set to a certain logic based on the definitions stated earlier. All the events are stored in the queue *QUEUE*. In the second phase, these setting events are propagated as in the D-algorithm. During the propagation of the event $(s, value)$, if the signal s is found to be **inconsistent** then the algorithm stops and designates P as a false path. Otherwise, the *NEW_EVENTS_CREATION*

procedure is called to check whether new events have been created or not. If new events are created, they are inserted in *QUEUE* and the propagation process continues. After all events are propagated and none of them cause any signal to become **inconsistent** (assigned more than one logic value), then path *P* is claimed to a possible sensitizable path [DYG89].

All the steps of FPD are straightforward except the step of new events creation. This step is derived from D-algorithm. During a signal propagation, the values assigned to this signal can induce values to the input signals of its driving gate and to the output signals of its driven gates depending on the gate type, the signal value, and other conditions. The original algorithm considers only basic cells, AND, NAND, OR, NOR, and NOT. We have modified the propagation phase to handle other types of cells such as XOR, XNOR, and most of the other block structures of standard cells. Let us call those cells as non-basic cells. Therefore, there are five cases for the new events to be generated as follows: (all the terms refer to a gate *g*, *s* is the output of gate *g* and s_1, s_2, \dots, s_k are the input signals to gate *g*)

- Gate *g* is an AND (NAND) gate, the event (*s*,1) (event (*s*,0)) can generate the following events: (s_i , value = *non-control* value of gate *g*), $1 \leq i \leq k$. A similar case holds for OR (NOR) gate and NOT.
- Gate *g* is an AND (NAND) gate, the event (s_i , value = *control* value of gate *g*) creates new event (*s*,0), (event (*s*,1)). A similar case holds for OR (NOR)

ALGORITHM FALSE_PATH_DETECTION(P , $false_path$)

Notation

P : the path to be checked, $P = s_0, g_0, s_1, g_1, \dots, s_i, g_i, \dots, s_k$

s_0 : primary input

s_k : primary output

$e = (s_i, value)$: the format of the event e where $value$ is the logic value assigned to signal s_i

$QUEUE$: the queue to store the events

new_events : set of events

$LIST$: the data structure to store the logic values assigned to signals

$signal_state$: the state of signal: **consistent** or **inconsistent**

$CHECK_LIST_CONSISTENCY(signal_state)$: procedure to check signal consistency in $LIST$

$NEW_EVENTS_CREATION(e, new_events)$: procedure to check whether the event e creates new_events

$PUSH(e)$: procedure to push the event e in $QUEUE$

$POP(e)$: procedure to pop the event e from $QUEUE$

$ADD_TO_LIST(e)$: procedure to add the event e to $LIST$

begin

The Events Generation Phase

Initialize $QUEUE$

for each s_i along the path P **do**

begin

for each $s_j \in Early_arrive_signals(s_i, *)$ **do**

begin

$PUSH(s_j, value = non_control \text{ value of gate } g_i)$

end

if $Late_arrive_signals(s_i, *) \neq \Phi$ **then**

begin

$PUSH(s_i, value = control \text{ value of gate } g_i)$

end

end

Figure 4.3: The false path detection algorithm (cont.).

The Events Propagation Phase

```

Initialize LIST
while QUEUE is not empty do
  begin
    POP(sj, value)
    ADD_TO_LIST(sj, value)
    CHECK_LIST_CONSISTENCY(signal_state)
    if signal_state = inconsistent then
      begin
        false_path  $\leftarrow$  true
        Exit
      end
    else
      begin
        NEW_EVENTS_CREATION((sj, value), new_events)
        if new_events  $\neq \Phi$  then
          PUSH(new_events)
        end
      end
    end
  false_path  $\leftarrow$  false
end{ Algorithm}

```

Figure 4.4: The false path detection algorithm [DYG89].

and NOT.

- Gate g is an AND gate whose input signals have values $value_1, value_2, \dots, value_k$ except s_i , then the event $(s_i, \text{non-control value of gate } g)$ creates new event $(s, \text{value} = \text{AND}(value_1, value_2, \dots, value_k))$. A similar case holds for NAND, OR, NOR.
- Gate g is an AND gate whose input signals have *non-control* values except s_i , then the event $(s, 0)$ creates the event $(s_i, \text{control value of gate } g)$. A similar case holds for NAND, OR, NOR.
- Gate g is a non-basic cell, two conditions are applied. If all input signals are already assigned values except s_i , then the event $(s_i, 0 \text{ or } 1)$ creates the event $(s, \text{output value of the logical function of gate } g)$. If all input signals have values except s_i , then the event $(s, 0 \text{ or } 1)$ creates the event $(s_i, 0 \text{ or } 1 \text{ based on the logical function of gate } g)$.

Example

The following example gives better understanding of how the FPD algorithm works.

Consider the circuit shown in Figure 4.1.

We will assume that all primary inputs are stabilized at 0 ns, each gate delay equals 1 ns and each signal delay equals 0 ns. Let $P = C - 1 - E1 - 2 - F - 3 - H - 5 - I - 6 - Y$ be the path to be checked. Before we start executing the algorithm, it

is preferable to get the two sets, *Early_arrive_signals* and *Late_arrive_signals* for each signal along the path P . Using the Equations 4.9 and 4.10, Table 4.1 shows the two sets for each signal along path P .

Then the algorithm starts with the event generation phase. Initially $QUEUE = []$. Then for each signal along the path P , the algorithm checks its *Early_arrive_signals* and *Late_arrive_signals*. For signal C , no early or late arrive signals exist, hence no action. Signal $E1$ has only $D1$ as an early arrive signal. Therefore, the event $(D1, non-control \text{ value of gate } 2 = 1)$ is pushed in $QUEUE$. Now $QUEUE = [(D1, 1)]$. For signal F , B is an early arrive signal, and therefore the event $(B, 0)$ is pushed in $QUEUE$ to become $[(D1, 1), (B, 0)]$. Signal $D2$ is the only early arrive signal of H . Therefore, $(D2, 0)$ is pushed in $QUEUE$. Signal I has signal G as an early arrive signal. Therefore, $(G, 1)$ is pushed in $QUEUE$. Since Y has no any early or late arrive signals, no action is taken. Now $QUEUE = [(D1, 1), (B, 0), (D2, 0), (G, 1)]$. After the completion of events generation phase, the event propagation phase starts. Initially, $LIST$ is set to $[]$. Then the events in $QUEUE$ are popped according to FIFO strategy. The event $(C1, 1)$ is popped first and stored in $LIST$. Since no events exist in $LIST$ other than $(D1, 1)$, consistency check is not applicable and the algorithm jumps to execute new events creation step. However, the event $(D1, 1)$ does not create any new event based on the conditions of new events creation criteria. Next, the event $(B, 0)$ is popped from $QUEUE$ and stored in $LIST$. Now $LIST$ has two events $[(D1, 1), (B, 0)]$, and $QUEUE$ has two events $[(D2, 0), (G, 1)]$.

Signal	Early_arrive_ signals	Late_arrive_ signals
C	-	-
E1	D1	-
F	B	-
H	D2	-
I	G	-
Y	-	-

Table 4.1: *Early_arrive_signals* and *Late_arrive_signals* of the path $P = C - 1 - E1 - 2 - F - 3 - H - 5 - I - 6 - Y$.

Obviously, no inconsistent events exist in *LIST* and no new events can be created as a consequence of adding $(B, 0)$. Next, $(D2, 0)$ is popped and stored in *LIST* to become $[(D1, 1), (B, 0), (D2, 0)]$. By checking consistency in *LIST*, it is clear that the two events $(D1, 1)$ and $(D2, 0)$ are **inconsistent** because $D1 = D2 = D$ which is assigned two different values. Then, the algorithm assigns *false_path* signal as **true** and exits.

4.3.2 Implementation Details

Below the used data structures and the implementation method are described.

Data Structures

- The input circuit is modeled as two single linked lists. In the first list, each element consists of a gate number, gate type (AND, OR, etc.), list of gate inputs, gate delay and signal delay. In the second list, each element consists of a gate number and its list of fanout gates.
- The library of gates is stored in a linked list in which each element consists of a gate type, base delay, load factor and input capacitance.
- Linked list of signals with maximum and minimum arrival time.
- The input path is modeled as a linked list of gates. Each element consists of a gate number, its type, list of early arrive signals, list of late arrive signals,

and maximum and minimum delay of signals along the path.

- A queue to store the events that are generated during the event generation phase. The queue element consists of a signal number and its assigned value.
- A Linked list to store the logic values assigned to each signal during the event propagation phase.

In all data structures, each signal is represented by its driving gate number.

Implementation Method

The algorithm has been implemented through four steps:

1. The input circuit, gates library, and other related information are read into the data structures mentioned above. Although this may require extensive memory, it is necessary to speed up the process.
2. The circuit is traversed to calculate the maximum and minimum arrival time and identify the early and late arrive signals for each signal. From the definitions mentioned earlier, it is obvious that prior to determining these timing parameters and signals for a given gate, all its driving gates have to be processed first. To achieve that, we should traverse the circuit from its primary inputs to its primary outputs more than once. In each traversal, those gates whose inputs have been already processed will only be subjected to processing.

Since the circuit is traversed in the order of its occurrence in the netlist, the number of traversals is less than or equal to the depth of the circuit.

3. The input path is read into its corresponding data structure. Then it is traversed to generate the events to be stored in the queue.
4. The events in the queue are propagated along the path, consistency check is made, and new events are generated according to the algorithm.

Steps 3 and 4 are repeated for all input paths. We have implemented this algorithm in the C language and called the program as *FPFIND*.

4.4 Experimental Results

The α -critical algorithm has been implemented using C language as described in [AF95]. The algorithm has been tested on different combinational and sequential circuits. In order to use these timing analysis tools, the given circuit should be described in VPNR² format. Since the ISCAS'85 benchmark circuits we used for testing are in SLIF format, we had to convert them into VPNR. This task was accomplished through two steps. First, we developed a C program (TRANS) that translates SLIF format to RNL³ format. Second, we used OASIS package to convert the RNL format into VPNR.

²refers to Vanilla Place aNd Route circuit description

³refers to Net List circuit description

Figure 4.5 shows the process of generating the critical paths of a given circuit in SLIF format using various tools that have been described earlier. A detailed description of these tools and how they are used are presented in Appendix A. In order to test functionality and performance of all developed tools, nine different benchmark circuits were used for this purpose.

Table 4.2 lists these circuits indicating some of their characteristics and shows the results of applying α -critical algorithm. As seen from the table, we have used small confidence level values (≤ 3) for large circuits and high confidence level values (100) for small circuits (highway and fract) in order to generate reasonable number of critical paths. Some circuits (such as c432 and c499) have large number of critical paths even for small values of confidence factor. For those circuits marked with “*”, we used 500 as a sealing value to limit the reported critical paths.

Table 4.3 shows the results of applying false path detection algorithm on the selected benchmark circuits. As clear from the table, some circuits have a large number of false paths and some do not have false paths at all. This is based on the critical paths that have been inspected. Also it is shown that most of the detected false paths of the circuits (except for c3540) did not affect the maximum circuit delay because these circuits are well-designed [CD93]. Nevertheless, these results match with those reported in [CD93]. In addition, these results are very good for speeding up the optimization process by reducing the number of paths meant for processing specially in the case of c6288 and c3540 circuits which have large number

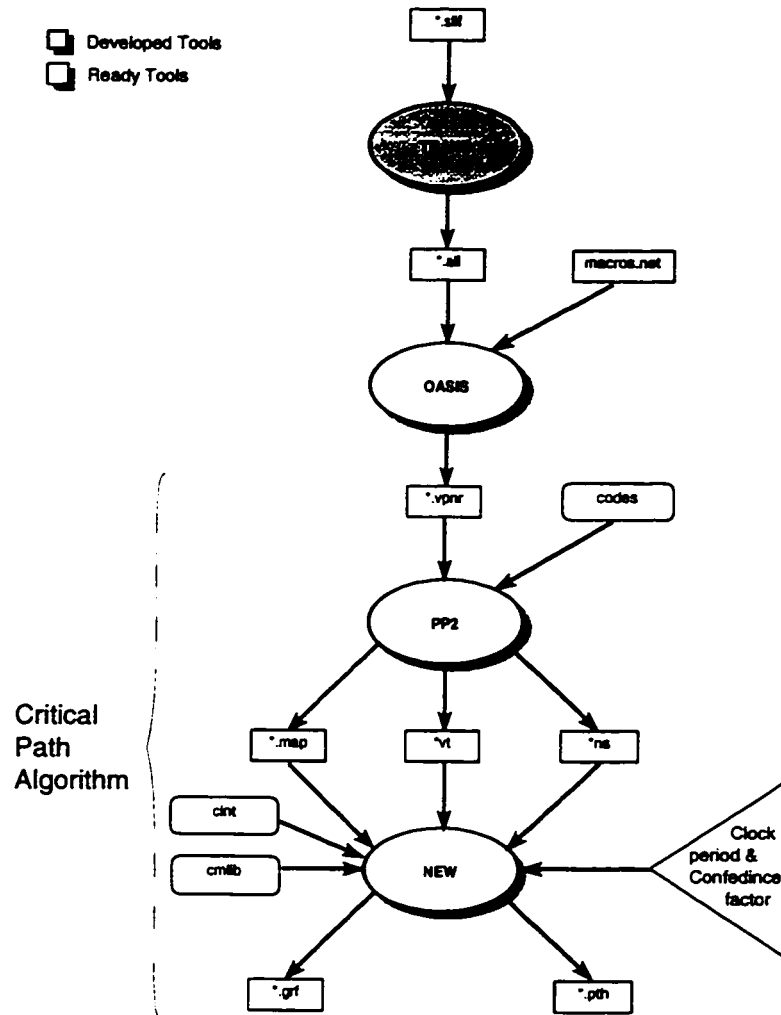


Figure 4.5: Critical paths generation process for a circuit in SLIF format.

Circuit Name	No. of Nodes	Confidence Factor	No. of critical Paths	Max Delay (ns)	Min Delay (ns)
c432*	395	1	500	171.911	171.911
c499*	283	1	500	65.344	65.344
c880	784	1	175	125.506	115.444
c1355	1453	1	151	109.860	94.467
c3540*	2243	3	500	192.188	172.800
c6288*	6672	1	500	675.646	675.646
struct*	1952	1	500	121.894	113.508
highway	54	100	61	32.348	22.946
fract	149	100	169	76.574	3.734

Table 4.2: Results of applying α -critical algorithm on some selected benchmark circuits.

Circuit Name	No. of Nodes	No. of Crit. Paths	No. of False Paths	% of False Paths	Ckt. Delay after FPD
c432	395	500	120	24%	171.911
c499	283	500	0	0	65.344
c880	784	175	0	0	125.506
c1355	1453	151	4	2.65	109.860
c3540	2243	500	460	92	185.201
c6288	6672	500	412	82	675.646
struct	1952	500	0	0	121.894
highway	54	61	0	0	32.348
fract	149	169	0	0	76.574

Table 4.3: Some application results of the adopted false path detection algorithm.

of gates.

4.5 Summary

The α -critical algorithm effective in generating critical paths and estimating circuit timing. It considers the net delays based on the data of interconnection capacitance of previous designs. It is because of this capability, we adopted this algorithm as a first phase of our proposed optimization process. We applied this algorithm on some benchmark circuits to generate critical paths that will be used as an input to the second phase to eliminate false paths from the generated critical paths. The integration of α -critical algorithm and false paths detection algorithm to extract the K longest sensitizable paths has produced a reasonably an efficient approach. Both techniques are very important for minimizing the effort and time for circuit optimization process. In addition, more accurate results are obtained using a combination of these algorithms

Chapter 5

Circuit Optimization Problem

Using Tabu Search

5.1 Introduction

The main contribution of this work is the application of Tabu Search (TS) technique on COP as an independent optimization technique. In order to accomplish this, COP has to be modeled to fit the requirements of TS technique. Since TS is a heuristic, experiments with different parameter values are recommended to find out the most optimal solution. In this chapter TS is explained in detail with emphasis on how COP has been formulated. A Detailed description of TS is presented in Section 2 of this chapter. In Section 3, the problem formulation is explained. Intermediate and long term memory structures of TS are discussed in Section 4. Section 5 talks about

our proposed diversification strategy. Use of some *Simulated Evolution* concepts with TS in COP is presented in Section 6. Finally, we conclude in Section 7.

5.2 Algorithm Description

TS was proposed by F. Glover for finding good solutions to combinatorial optimization problems [Glo90]. This technique is conceptually simple and elegant. It is a *meta-heuristic* which can be superimposed on other procedures to prevent them from becoming trapped at a locally optimal solution. The method can be used to guide any process that employs a set of *moves* for transforming one solution into another, and provides an evaluation function for measuring the attractiveness of these moves.

TS is based on three primary ideas:[Glo90]

1. It uses flexible attribute-based memory structures in order to get benefit of the search history. This feature allows TS to exploit the historical information more thoroughly than the techniques using rigid memory such as branch and bound, and A^* search, or the memoryless techniques such as simulated annealing. The memory structures include: *Short Term*, *Intermediate Term*, and *Long Term*.
2. It controls the search process by applying two mechanisms: *tabu restrictions* and *tabu aspiration criteria*. They constrain and free the search in attempting at finding better solutions. This strategy uses the short term memory

structure.

3. It implements two strategies: *intensification* and *diversification* by incorporating memory functions of different time span. *Intensification strategies* refer to procedures for reinforcing move combination and solution features historically found good , while *diversification strategies* refer to driving the search into new regions [Glo90]. The intensification and diversification are applied in the intermediate and long term memory structures respectively.

The link between different types of memory and their associated strategic components in TS are illustrated in Figure 5.1.

5.2.1 Tabu Restriction

TS goes from one trial solution to another by making moves. Several candidate moves are generated to give different solutions. The best solution is selected for current iteration and it may not be better than the best solution found so far. If this strategy is applied alone, it is possible to reach a local optimum, ascend, and then come back to the same local optimum. Thus there is a possibility of cycling. To avoid this trap, tabu restriction is used. In tabu restrictions, some attribute of moves are made tabu (forbidden) to avoid move reversal.

Tabu restrictions are enforced by a *tabu list* which stores the move attributes to avoid move reversal. Tabu list looks like a window on accepted moves (Figure 5.2).

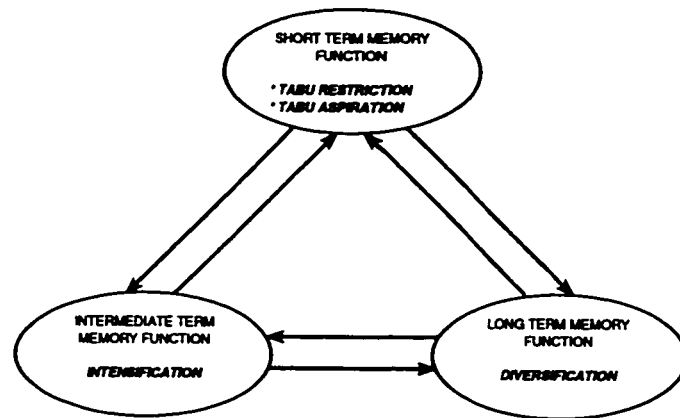


Figure 5.1: The relationship between memory functions of Tabu Search [Glo90].

Once an attribute is removed from the list, it is released from its tabu status, and the move containing this attribute again becomes free to be selected. The window or tabu list size can be any number depending on the problem.

5.2.2 Aspiration Criteria

Aspiration criteria of TS free the search process to allow some moves to be reselected again. They override the tabu restriction if the reverse move produces a better solution than the previous one. If a move is made tabu in iteration i and its reversal comes in iteration j , where $j > i + c$, then it is possible that the reverse move takes the search into a new region because of the effect of c intermediate moves.

Many aspiration criteria can be identified for this purpose. The simplest aspiration criterion is to override the tabu status if the reversal produces a solution better than the best found so far.

5.2.3 How the Algorithm Works

Figure 5.3 shows a simplified description of TS. Initially the current solution is the best one. A list of candidate solutions is generated from the current solution by making some moves. The size of this list is a trade-off between quality and performance. The best among these solutions is selected and if it is not tabu, then it becomes the current solution for next iteration. Otherwise, its aspiration criterion is checked. If it passes the aspiration criterion then it becomes the current solution.

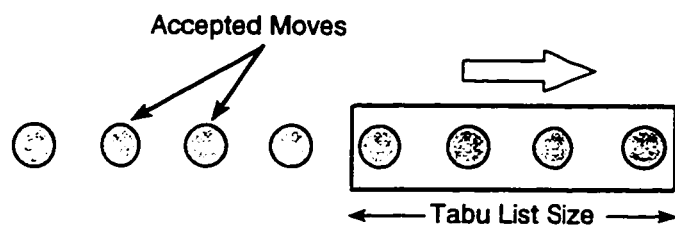


Figure 5.2: Tabu list visualized as a window over accepted moves.

otherwise moves are regenerated to get another set of new solutions. The best solution is updated to the current solution if it is better than the best found so far. A more detailed algorithmic description of TS is given in Figure 5.4.

5.3 COP Formulation

To use TS for any particular problem, one has to perform the following tasks:

- Choose a proper initial solution.
- Define a neighborhood for a given solution.
- Generate moves.
- Formulate and maintain tabu list.
- Define a proper restriction criterion.
- Define a proper aspiration criterion.
- Formulate the cost function to evaluate alternative solutions.

Below we explain how each task can be handled in our case.

5.3.1 Initial, Current and Best Solution

Although in theory the initial solution can be any feasible solution, it is found that TS may take longer if it is given a poor initial solution [Ali94]. In our case, the initial

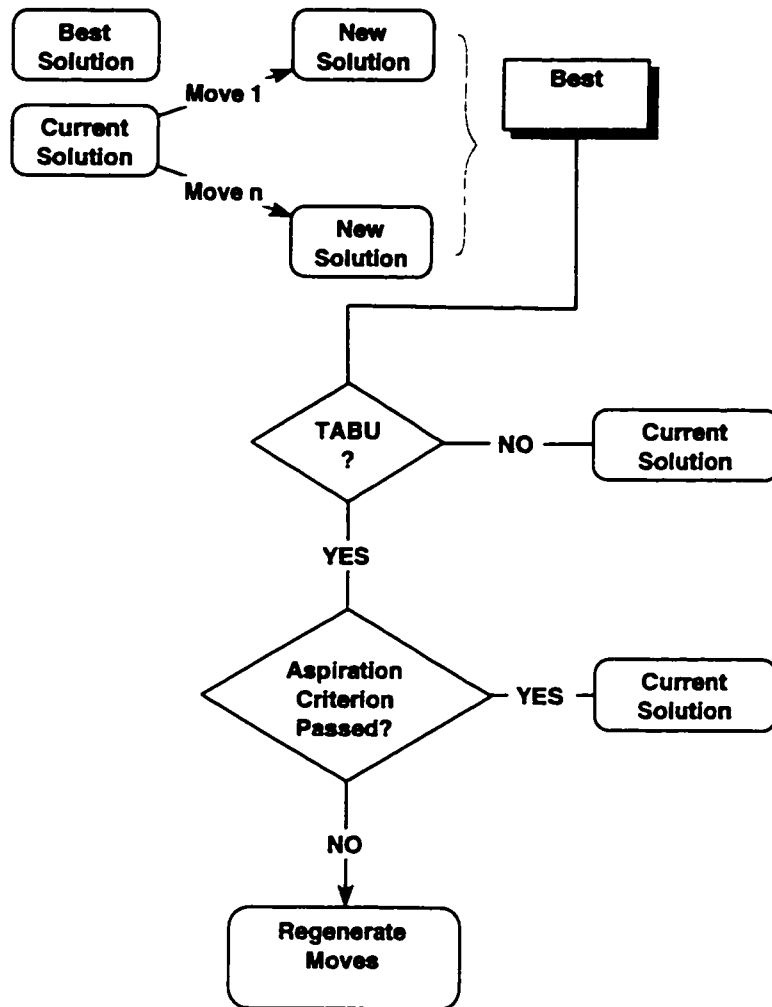


Figure 5.3: Block diagram of tabu search procedure.

X :Set of feasible solutions.
 s :Current solution.
 s' :Best admissible solution.
 c :Objective function.
 $N(s)$:Neighborhood of $s \in X$.
 $S(s)$:Sample of neighborhood solutions.
 T :Tabu list.
 AL :Aspiration level.

1. Start with an initial feasible solution $s \in X$.
2. Initialize tabu lists aspiration level.
3. **FOR** fixed number of iterations **DO**
4. Generate neighborhood solutions $S(s) \in N(s)$.
5. Find best $s' \in S(s)$.
6. **IF** move s' to s is not in T **THEN**
7. Accept move and update best solution.
8. Update tabu list and aspiration level.
9. Increment iteration number.
10. **ELSE**
11. **IF** AL is satisfied **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 5.4: Algorithmic description of Tabu Search (TS).

solution is a set A of CMOS nodes covered by the K longest sensitizable paths. As TS proceeds, in each iteration we will generate a number of current solutions by swapping some CMOS gates by BiCMOS gates randomly. Then each solution is evaluated according to the cost function. A solution that is better than the best solution found so far will be assigned as the best solution. TS stops if one of the following is satisfied:

- TS executed exactly m iterations.
- A user specified objective improvement in circuit delay is achieved.
- The circuit delay has been reduced by ΔD where,

$$\Delta D = T_{max} - T_{min} \quad (5.1)$$

where $T_{max} = \max_{\pi \in \mathcal{P}}(T_{\pi})$, $T_{min} = \min_{\pi \in \mathcal{P}}(T_{\pi})$, and \mathcal{P} is a set of sensitizable paths.

5.3.2 Generation of Moves

One possible move in the COP is to swap a CMOS gate by its equivalent BiCMOS gate or a BiCMOS gate by its equivalent CMOS gate. Each move will generate one neighbor solution. Another way of generating neighbor solution is by making more than one move. This approach has fewer chances of finding the global optimum as the solution may be disturbed too much. The selection of the gate to swap follows a random strategy.

Another method of generating moves is based on some characteristics and attributes of nodes such as its location, type and delay. One of the techniques used for this purpose is *Simulated Evolution* (SE) which consists of evaluating the goodness (fitness) of each node in the set. The goodness of a node in our case can be identified by how much is its contribution to the circuit delay. More details about SE and how it is integrated with TS for generating moves are presented in Section 5.6. A move e can be:

illegal: when the move is improving but violating the constraints.

legal: when the move is non-improving and not violating the constraints.

Good: when the move is improving and not violating the constraints.

The constraints are presented later in this section.

5.3.3 Tabu List

Formulation of the tabu list is one of the main steps in using TS to solve a particular problem. Since we have only one type of move we will use only one tabu list. Each entry in the tabu list comprises the following information:

- Gate number in the path. As will be shown later each gate is represented by a number in the circuit description.
- Gate type: AND, NAND, ... etc.
- Cost associated with this move.

- Frequency of this move. The frequency is defined as the number of times the move has been admitted since the starting of the search process.
- Gain bit (0 or 1): this field is used only with one of the adopted aspiration criteria as will be explained later.

The tabu list size is an important parameter. Some researchers use the *magic number* 7 [Ali94]. However, the size depends on the problem under consideration. Therefore, in our implementation of TS, we have experimented with several sizes and chosen the one that produced the best solution.

5.3.4 Restriction Criteria

The adopted tabu restriction for COP is to forbid reversing a move that is already in the tabu list. To achieve this, the gate number in the current move is compared with each gate number in tabu moves. If both are equal, this means that this gate has been selected recently for swapping and it is forbidden to swap it again. The gate in tabu list can be CMOS or BiCMOS.

5.3.5 Aspiration Criteria

After all the candidate moves are generated for iteration i , the best is selected, which may not be better than the current solution. If this best move is not tabu, it is accepted and becomes the current solution for the next iteration. If the move is

tabu, its aspiration criterion is checked.

We carried out experiments with two aspiration criteria. The first one is to override the tabu restriction in case the swap produces a new current best solution. The second criterion is called *Aspiration by Search Direction*. In this criterion, if an improving (non-improving) move e is made and it is tabu, then the reverse move \bar{e} is accepted if it is also improving (non-improving) move. A gain bit G is used where $G(e)$ is set to a value “1” (improving) or “0” (non-improving) depending on the move. The value of $G(e)$ is updated in each iteration. That is, aspiration level of a move e is satisfied if both the move and its reversal lead to a solution with a higher cost or both lead to a solution with a cost lower than the current solution. For convenience, we shall call the first aspiration criterion AS_1 and the second aspiration criterion AS_2 .

5.3.6 Evaluation Function

In order to select the best solution among several solutions that are generated by certain moves, we have to evaluate each solution according to some cost function.

The evaluation function should be formulated in such a way as to incorporate all the parameters meant for optimization. The aim is to minimize circuit delay with minimal increase in power. Since power is proportional to capacitance, the evaluation function should be a measure of the effect of a move in the delay and capacitance.

Let $E(s)$ be the cost function of a solution s . Then,

$$E(s) = f(\text{delay}, \text{capacitance}) \quad (5.2)$$

As mentioned earlier, the delay of a circuit is determined by the delay of its longest sensitizable path. However, Since there are no changes in interconnection delays when a certain node is swapped from CMOS to BiCMOS or vice versa, only the circuit delay D_{p_l} , which is the summation of switching delays of those gates constituting the longest path P_l , would be affected. We adopted the same switching delay model as in Equation 4.2.

Let i be the gate number. Then its switching delay CD_i is expressed as:

$$CD_i = BD_i + LF_i \times AcL_i \quad (5.3)$$

Let F_i be the number of fan-out gates of gate i . If C_k is the input capacitance of a loading gate k , then:

$$AcL_i = \sum_{k=1}^{F_i} C_k \quad (5.4)$$

Now, we can compute D_{p_l} . Let m be the number of gates in the longest path. Then:

$$D_{p_l} = \sum_{i=1}^m CD_i \quad (5.5)$$

The total capacitance of the circuit is given by:

$$C_t = \sum_{i=1}^n C_i \quad (5.6)$$

where n is the total number of gates in the circuit and C_i is the input capacitance of gate i .

As stated earlier in Chapter 3, the objective is as follows:

$$\begin{cases} \text{maximize} & \sum_{i \in S} \Delta D_i \\ \text{subject to} & \sum_{i \in S} \Delta C_i \leq C_T \end{cases} \quad (5.7)$$

Recall that ΔD_i is the circuit delay gain due to changing gate i implementation from CMOS to BiCMOS. Therefore, in order to compute ΔD_i , we can simply use Equation 5.5. The capacitance constraint may be expressed as a penalty in the cost function when there is an illegal move. Then, the evaluation function of a neighbor solution s' to a solution s can be expressed as:

$$E(s') = E(s) + \Delta D_i - X \quad (5.8)$$

where

$$X = \begin{cases} \text{penalty} & \text{if the move is illegal} \\ 0 & \text{otherwise} \end{cases}$$

The penalty X differs from problem to problem depending on the circuit delay as will be shown later.

For the initial solution s_0 , $E(s_0) = 0$. Then as TS proceeds, the cost of the subsequent solutions will be expressed in terms of the cost of the current solution plus the gain in the delay subject to the penalty value.

The evaluation function in Equation 5.8 is meant only for short term memory component and does not incorporate any historical information that can be used in long term memory. In the next section we will show how a record of history of moves is used to diversify the search to improve results.

5.3.7 Example

Consider the simple combinational circuit shown in Figure 5.5 Let us use the following data:

Number of sensitizable paths = 3.

Tabu list size = 3.

Candidate list size = 3.

Capacitance constraint = 10% of the total capacitance.

By applying phase 1 and phase 2 of our approach using the data presented in Appendix A, we find that the three longest sensitizable paths are:

P_1 : 4,9,10,11,13,6 with $D_{p_1} = 10.987$ ns.

P_2 : 2,7,10,11,13,6 with $D_{p_2} = 10.467$ ns.

P_3 : 3,7,10,11,13,6 with $D_{p_3} = 10.467$ ns.

Therefore, the circuit delay is 10.987 ns and its total capacitance is 2.075 pF. If we

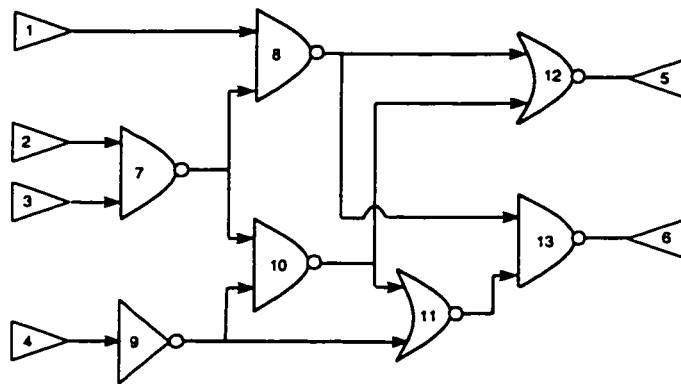


Figure 5.5: A simple combinational circuit.

examine the nodes satisfying $CL > CX$ among all the nodes within P_1 , P_2 , and P_3 , we will find the following set: $A = 2, 3, 4, 7, 9, 10, 11, 6$.

Now let us apply TS on this circuit for two iterations only. Initially, the tabu queue is empty and the current solution consists of CMOS gates only with cost = 0. The best cost initially = 0. For convenience, the current solution is represented in a tabular form which consists of the set A of nodes and their types (CMOS or BiCMOS).

Iteration #1: In each iteration, 3 neighboring solutions are generated by selecting 3 nodes randomly. Table 5.1(a) shows a list of 3 solutions generated by random selection of 3 nodes: 4, 7 and 9. The “New Cost” in the table indicates the solution cost after swapping the corresponding gate from CMOS to BiCMOS. For example, according to Equation 5.3, the CMOS delay of gate number 9 is 3.823 ns. If we swap this gate to BiCMOS, its delay becomes 1.796 ns. Since node number 9 is in the longest sensitizable paths, then $\Delta D = 0.52ns$. The increase in the circuit capacitance = 0.102 pF which does not violate the capacitance constraints. Therefore, according to the evaluation function (Equation 5.8), the cost of this solution = 0.52. As shown in the table, from the list of admissible candidates, the swap of gate 9 marked with “*” corresponds to the best move, and therefore it is chosen as the move to new current solution and inserted in the tabu list. The swap of gate 7 produces solution with cost = 0 because it is not in the longest path of the current solution and therefore it has no contribution to the circuit delay. The new solution

produced by the accepted move is given in Table 5.1 (c). The cost of the current solution and best solution is 0.52. Now the delays of the three longest paths P_1 , P_2 , and P_3 are 8.96 ns, 10.465 ns, and 10.465 ns respectively.

Iteration #2: As in the previous iteration, once again 3 new neighbors are generated by selecting 3 gates randomly as shown in Table 5.2(a). The best move is the swap of gate 11. This move increases the cost to 2.51. Therefore the best solution becomes 2.51. Since this move is not tabu, it is accepted and inserted in the tabu list as shown in Table 5.2(b). The current solution now in Table 5.2(c) consists of two BiCMOS nodes 9 and 11 with circuit delay = 8.477 ns.

5.4 Intermediate and Long Term Memory

In many combinatorial problems, especially large and hard problems, application of short term memory alone does not produce a good solution. Several studies reported in [Glo90] show that the intermediate and long term memory functions can be very important for obtaining best results.

5.4.1 Intermediate Term Memory: Intensification Strategy

If we look at a tabu list in the short term memory of TS, we can find that it has an intensification role by temporarily “locking in” certain locally attractive attributes.

Current Cost = 0
 Best Cost = 0

gate	ΔD	ΔC	New Cost
4	0.17	0.002	0.17
7	0	1.36	0
9	0.52	1.796	0.52

(a) 3 Neighbors of Current Solution *

9		
BiCMOS		

(b) Tabu Queue

2	3	4	7	9	10	11	6
CMOS	CMOS	CMOS	CMOS	BiCMOS	CMOS	CMOS	CMOS

(c) Current Solution

Table 5.1: Iteration # 1.

Current Cost = 0.52

Best Cost = 0.52

gate	ΔD	ΔC	New Cost
2	0.17	0.002	0.69
10	1.66	1.36	2.18
11	1.985	0.167	2.51

*

(a) 3 Neighbors of Current Solution

9	11	
BiCMOS	BiCMOS	

(b) Tabu Queue

2	3	4	7	9	10	11	6
CMOS	CMOS	CMOS	CMOS	BiCMOS	CMOS	BiCMOS	CMOS

(c) Current Solution

Table 5.2: Iteration # 2.

Another way of intensifying the solution using the tabu list is to change the attribute of the move when aspiration criterion is satisfied rather than re-inserting it again into the list. By this way the move may be kept in the list for shorter period and thereby getting more opportunity for re-selection. However this action is not sufficient to direct the solution to most preferable attributes due to a short record of move history [AF95].

Many intensification strategies have been reported in the literature. *Target Analysis* (TA) is a procedure for creating improved problem-solving methods in artificial intelligence. The main idea of TA is the use of *learning* process of which rules are best to solve a particular class of problems. This capability of TA can be beneficial to heuristic techniques such as TS as a means of creating more effective forms of intensification and diversification [Glo90].

Another technique of intensification reported in [SY98] operates as follows. After a number of iterations, number of best trial solutions are selected and their features are recorded and compared. Then the common features are taken as new attributes of good solutions. Then to find out such good solutions, those moves which do not reveal the new attributes are penalized.

Although an intensification is an improving method to produce better solutions, it is not as important as diversification due to the fact that short term memory component most of the time directs the search into a regional space creating the necessity to transfer the search to another area (diversification) rather than restrict-

ing the search to the same region (intensification). In our case, we used only one move attribute (technology type) which can hardly be used to propose an efficient and useful intensification technique. However, using Simulated Evolution to generate moves is considered as a form of intensification. Only the gates having high goodness will have higher probability to be selected than other gates.

5.4.2 Long Term Memory: Diversification Strategy

As a tabu list has intensification role, it has diversification role too by compelling new choices to include or exclude attributes that are not among those recently discarded or incorporated.

In spite of this, as mentioned earlier, there is often a need to apply other methods that use some historical data to diversify the search. Because of its importance in producing better optimal solutions than those produced by other techniques, diversification component of TS attracted many researchers to develop different diversification strategies for solving different combinatorial problems. Gendreau modeled TS method to solve the maximum clique problem [KLG94]. Two tabu lists are used, one for move selection and the second for avoiding any move that would lead to a solution visited in the past T iterations, where T is set to values as large as 150 for solution attempts of 300 iterations. Fiechter applied parallel TS on large traveling salesman problem as reported in [KLG94]. To diversify the search, he used super-moves which correspond to reallocating key portions of the tour to an extent

that a re-starting mechanism is not necessary.

Woodruff and Spearman introduced the use of what is called *diversification parameter* d [KLG94]. This parameter is equal to the reciprocal of a Lagrangian multiplier. The low values of d results in nearly infinite costs for constraint violation while high values allow searching through infeasible regions. In addition, the parameter d is given a role similar to that of the temperature in simulated annealing to control the amount of randomization.

Frequency-based diversification strategies has been proposed in solving different optimization problems. In attempting to optimize the quadratic assignment problem, Skorin-Kapov used $n \times n$ matrix to record the number of times a pair of objects exchange locations. The frequencies are weighted to modify the distance between every pair of locations and force the construction to diverse solutions during a re-starting phase [KLG94].

Glover and Laguna used frequency based strategy in number of ways. In [LG93b], frequency counts are used to modify the selection of moves when there are no improving moves available. Applied to single machine scheduling problem, the frequency count is multiplied by a penalty parameter and added to the cost function of every non-improving move. Then the move with the least penalized value is selected. This strategy can successfully avoid long term cycling and find improved solutions. Frequency counts have also been used in bandwidth packing problem where the frequency count is a function of the candidate list size, the maximum tabu list size

(since dynamic sizes are used) and the current iteration number [LG93a]. Another frequency-based strategy proposed by Glover and Laguna is used in quadratic assignment problem [KLG94]. Two simple memory components are applied: recency-based memory that achieves a first order form of diversification and frequency-based memory that achieves a second order form of diversification. After the construction of initial solution and applying a series of moves to lead to a local optimum, the algorithm enters a first order diversification stage in which two concepts are employed. First, two solutions are increasingly diverse if their separation distance increase which is defined as the minimum number of moves required to reach one solution from another. The second concept is related to the difficulty of getting one solution from another. Using these two concepts, a set of new swaps is made after hitting a local optimum. Then, from among these swaps, the best solution is chosen and the algorithm switches back to short term memory phase. After a selected number of local optima are generated during the first order phase or a selected cutoff rule is satisfied, the algorithm enters the second order phase. In this phase, the evaluation function is replaced by a new one which incorporates the move frequency term to favor the exchanges of moves with low frequencies. Then the algorithm switches back to the first order phase.

Other reported diversification strategies include: use of learning process of Target Analysis [Glo90], [LG93a], application of some features of Strategic Oscillation [Glo90], and Influential Diversification [HG94].

5.5 Proposed Frequency-based Diversification Strategy

To apply frequency-based diversification strategy, we have to keep track of the number of times a certain move has been selected so far. Therefore, in our approach we assign a frequency counter for each gate which is incremented each time the gate is selected.

We proposed and adopted the following diversification method:

1. When the short term TS algorithm hits a local optimum, the following actions are taken:

- (a) All BiCMOS gates are swapped to CMOS. Denote the number of those gates as `NUM_OF_BiCMOS`

- (b) Search for least frequent CMOS gates and swap them by BiCMOS type.

The search and swap process continues till the objective load threshold is reached or till the number of swapped gates is equal to `NUM_of_BiCMOS`. By doing this, the search process is transferred to another region where the search might lead to a better solution.

2. Re-start the short term memory component and continue till a local optimum is hit, then repeat step 1.

Determining when the algorithm hits a local optimum is not easy because we don't know for how long the algorithm should run. One way to overcome this problem is to inspect the best solution for a fixed number of iterations x ; if there is no change in the best solution cost for the last x iterations, then we state that the algorithm has reached a local optimum. However, the value of x depends on the size of the circuit and the number of sensitizable paths. Therefore, we have to experiment with several values of x in order to obtain the best result.

In step 1(b), the stopping criterion has two parts: either a load threshold is hit or NUM_OF_BiCMOS is reached. The latter makes sense because as the number of BiCMOS nodes of the new solutions equals the number of BiCMOS nodes in recent solution, then the achieved delays and loads of both solutions may be close. The load criterion is used to prevent the load of a new starting solution from exceeding the objective load. By applying this method, better solutions have been produced as shown in the next chapter.

5.6 Evolutionary Tabu Search

As mentioned in Section 5.3.2, moves can be generated based on evolutionary aspects of *Simulated Evolution* heuristic. Before explaining how this can be done, let us give a brief overview about SE.

5.6.1 Simulated Evolution Overview

Simulated Evolution (SE) is one of the iterative heuristic techniques for solving combinatorial optimization problems. It was proposed by Kling and Banerjee in 1987 [KB87]. SE simulates the natural evolution process where organisms try to adapt to their environment by developing or changing some features such location, size and color. The main idea of SE is that the selection of components to change to improve the solution is done according to a stochastic rule. The components not located in a proper manner need to change their locations to improve the solution while those components already well located have a high probability to stay in their locations [SY98].

The algorithm starts with the initialization phase where various parameters are set to the desired values. Some of these parameters include: number of iterations and a selection bias B . Then the algorithm enters the iterative phase which consists of three steps: *Evaluation*, *Selection* and *Allocation*. The three steps are executed repeatedly until the stopping criteria are met. One possible stopping criterion is to run the algorithm until no improvement is observed for a number of iterations or the objective improvement in solution cost is achieved. Another possible stopping criterion is to run the algorithm for a prefixed number of iterations. Let us explain each step briefly [SY98].

Evaluation:

In this step, the goodness of each element e_i in the population P is evaluated.

Goodness is defined as follows:

$$g_i = \frac{O_i}{C_i} \quad (5.9)$$

Where O_i is an estimate of the optimal cost of the element e_i , and C_i is the actual cost of e_i in its current location. Accordingly, the O_i 's do not change from generation to generation, and therefore, are computed only once while the C_i 's have to be recomputed at each call to evaluation step.

The above equation assumes a minimization problem and it should be in the range of $[0,1]$. The goodness measure must be strongly related to the target objective of the given problem.

Selection:

After evaluating goodness of all individuals in the population, some of them are selected to be allocated in new locations. The selection is based on a selection function F_S which has two parameters: *goodness* g_i and *Selection Bias* B . Values of B are recommended to be in the range of $[-1;0.1]$. In many cases a value of $B = 0$ would also be a reasonable choice.

The higher the goodness value of the element, the more likely that it will not be selected and hence the higher is the probability of the element to remain in its current location.

Allocation:

In the *Allocation* step, locations of selected elements in S are altered according

to an allocation function F_A . The choice of a suitable allocation function is problem specific. The allocation function may be a nondeterministic function which involves a choice among a number of alternative moves for each element. The order and type of alteration of elements are problem specific. This is why, in many cases, *Sorting* step is important to achieve better solutions. Since the goodness of the elements are so tightly coupled with the target objective, superior alterations are supposed to gradually improve the individual goodnesses. Hence, *Allocation* allows the search to progressively converge towards an optimal configuration where each element is optimally located [SY98].

5.6.2 Evolutionary Tabu Search

In our approach of applying TS to COP, we used two functions of SE as an alternative stochastic method for generating moves. The two functions are: *Evaluation* and *Selection*. Let us show how each function is formulated for COP.

Evaluation:

Let $A = (g_1, g_2, \dots, g_n)$ where each g_i satisfies $CL_i > CX_i$. For each g_i in A , we compute ΔD_i which is the gain delay due changing gate i implementation from CMOS to BiCMOS, that is,

$$\Delta D_i = D_{g_i}^c - D_{g_i}^b \quad (5.10)$$

This computation is done only once. In this case,

$$O_i = \Delta D_i \quad (5.11)$$

Let T be the current delay of the circuit and T'_i be the delay of the circuits after swapping gate i . Then the actual circuit delay gain (cost) is

$$C_i = T'_i - T \quad (5.12)$$

Let G_i be the goodness of gate i . Since COP is a maximization problem, then the goodness function should be derived in such a way as to show that if the gate goodness is high, its fitness should also be high so that the gate will most likely not get swapped. Therefore G_i is defined as follows:

$$G_i = 1 - 0.5 \times (1 + \frac{C_i}{O_i}) \quad (5.13)$$

For example, if swapping gate i from CMOS to BiCMOS produces maximum gain in the circuit delay, then $C_i = O_i$ resulting in G_i to be 0. In this case, gate i is not in its optimal state and it needs to be swapped.

Using the above equations and definitions, the evaluation step of SE can be applied as follows:

FOR EACH $g_i \in A$ **DO**

$$C_i = T'_i - T$$

$$G_i = 1 - 0.5 \times (1 + \frac{C_i}{O_i});$$

END FOR EACH

Selection:

After computing the goodness of all gates in A , we select from A a subset R of size N for the purpose of generating N moves. The selection of those gates is made as follows:

REPEAT

select gate g_i randomly;

generate a *Random* number between “0” and “1”;

IF $Random \leq Min(1, 1 - G_i + B)$ **THEN** $R = R \cup g_i$;

ENDIF;

UNTIL $|R| = N$;

The bias B is used only when gate g_i is already BiCMOS and needs to be swapped to CMOS. This is because when a low goodness CMOS gate is swapped to BiCMOS, its goodness becomes high. Therefore, in order for a gate g_i to be re-selected as a mechanism to escape from the trap of local optima, the bias B is used to maximize the gate re-selection probability. Since the value of B is problem dependent, we experimented with different values.

The expected advantage of generating moves based on SE approach is that the search will be biased to drift towards better solutions faster than generating moves randomly. However, the evaluation and selection steps are done during every itera-

tion which means that each iteration in this approach takes much more time than the iteration in the classical approach. Therefore, SE based approach usually takes longer time than the classical approach. For convenience, we shall call the TS based on generating moves randomly as *Classical Tabu Search (CTS)* and the TS based on the SE to generate moves as *Evolutionary Tabu Search (ETS)*.

5.7 Summary

Tabu Search algorithm is a heuristic technique that can be applied as a stand alone or can be superimposed on other techniques to solve particular combinatorial optimization problem. In this thesis, we applied TS for COP as a stand alone technique where the solution is constructed by swapping some CMOS gates by BiCMOS gates from a given description of a circuit which consists of CMOS gates only. Since the problem has been formulated to be similar to Knapsack problem, the objective is to maximize the reduction in circuit delay subject to capacitance increase constraints. Accordingly, the evaluation function has been derived to satisfy the objectives. In optimization problems where the search space is very large as in our case, applying short term memory component of TS is not sufficient to find a good solution. Rather, the search has to be diversified to cover other regions where the global optimal solution may exist. For this purpose, we proposed and applied frequency-based diversification strategy as previously explained. In addition, we proposed Evolu-

tionary Tabu Search technique which applies some features of Simulated Evolution on the move generation process to intensify the search.

Chapter 6

Implementation, Results and Discussion

6.1 Introduction

In this chapter, we provide the implementation details of the TS heuristic for solving COP. We also present experimental results of several benchmark circuits. Section 2 of this chapter presents the implementation details including input and outputs of the programs, data structures that have been used and implementation method. Section 3 describes the tools and languages used in the whole optimization process. Then we discuss the experimental results of the TS with different parameter values (tabu list size, candidate list size, number of iterations, etc.,) in Section 4. The results of short TS versus long term TS, TS with AS_1 versus TS with AS_2 , and CTS versus

ETS are presented and discussed in detail in Section 5. Section 6 summarizes the contents of this chapter.

6.2 Implementation Details

6.2.1 Inputs/Outputs of the Program

The TS program for COP receives three types of input files:

1. A modified VPNR format circuit description where each node is described by a unique number, its equivalent AHPL¹ type and its inputs. This description is generated using the tools developed in [AF95]. For example, Figure 6.1 shows the modified VPNR description of the circuit CKT shown in Figure 5.5.
2. A fanout file which consists of all nodes where each node is represented by a number (given in modified VPNR file) and its fanout nodes. Also this file is generated by the tools reported in [AF95]. Figure 6.2 shows a fanout description of CKT.
3. A set of sensitizable paths generated by α -critical path and false_paths_detection algorithms. Each path is represented by its number, delay, and the nodes making this path. For example, the sensitizable paths of CKT are given in Figure 6.3.

¹refers to A Hardware Programming Language

```

domain begin fadd swap=0
profile top (0,0) (0,0);
profile bottom (0,0) (0,0);
iolist
1 0 4018 IN1 T:(0,100) pintype=pi
2 0 4018 IN2 T:(0,100) pintype=pi
3 0 4018 IN3 T:(0,100) pintype=pi
4 0 4018 IN4 T:(0,100) pintype=pi
5 1 4019 OUT5 B:(0,100) pintype=po
6 1 4019 OUT6 B:(0,100) pintype=po
;
row 1
7 4202 ai2s INSI7 2 3 7
8 4202 ai2s INSI8 1 7 8
9 4105 i1s INSI9 4 9
10 4202 ai2s INSI10 7 9 10
11 4205 oi2s INSI11 9 10 11
12 4202 ai2s INSI12 8 10 5
13 4205 oi2s INSI13 8 11 6
;
domain end fadd

```

Figure 6.1: Modified VPNR format of CKT.


```
INPUTPADs= 4 OUTPUTPADs= 2 FLIPFLOPS= 0
1 > 8 ;
2 > 7 ;
3 > 7 ;
4 > 9 ;
7 > 8 10 ;
8 > 12 13 ;
9 > 10 11 ;
10 > 11 12 ;
11 > 13 ;
12 > 5 ;
13 > 6 ;
5 > ;
6 > ;
```

Figure 6.2: Fanout description of CKT.

13
 PATH 1
 Dcell 10.987 Dnet -0.987 LRAT 10.000 v=0.076
 4 9 10 11 13 6 ;
 PATH 2
 Dcell 10.467 Dnet -0.467 LRAT 10.000 v=0.072
 3 7 10 11 13 6 ;
 PATH 3
 Dcell 10.467 Dnet -0.467 LRAT 10.000 v=0.072
 2 7 10 11 13 6 ;
 PATH 4
 Dcell 8.265 Dnet 1.735 LRAT 10.000 v=0.049
 4 9 10 12 5 ;
 PATH 5
 Dcell 7.921 Dnet 2.079 LRAT 10.000 v=0.048
 3 7 8 13 6 ;
 PATH 6
 Dcell 7.921 Dnet 2.079 LRAT 10.000 v=0.048
 2 7 8 13 6 ;
 PATH 7
 Dcell 7.745 Dnet 2.255 LRAT 10.000 v=0.045
 2 7 10 12 5 ;
 PATH 8
 Dcell 7.745 Dnet 2.255 LRAT 10.000 v=0.045
 3 7 10 12 5 ;
 PATH 9
 Dcell 7.745 Dnet 2.255 LRAT 10.000 v=0.045
 2 7 8 12 5 ;
 PATH 10
 Dcell 7.745 Dnet 2.255 LRAT 10.000 v=0.045
 3 7 8 12 5 ;
 PATH 11
 Dcell 7.403 Dnet 2.597 LRAT 10.000 v=0.066
 4 9 11 13 6 ;
 PATH 12
 Dcell 4.705 Dnet 5.295 LRAT 10.000 v=0.037
 1 8 13 6 ;
 PATH 13
 Dcell 4.529 Dnet 5.471 LRAT 10.000 v=0.034
 1 8 12 5 ;

Figure 6.3: Sensitizable paths of CKT.

The specification of each gate type is given in a library. Two libraries are used: one for CMOS gates and the other for BiCMOS gates. Table 6.1 shows a sample of CMOS library, where each line shows a gate in AHPL code, its base delay, load factor, input capacitance, and its CX value. For other mixed technologies, similar library structure can be used. For BiCMOS, the same structure is used with the difference that the BiCMOS gate is represented by CMOS AHPL code with addition of “0” as a least significant digit. For example, the INV BiCMOS gate is represented as 41050 (see Table A.4 in Appendix A).

The output of the circuit optimization program consists of the following:

- Estimate of the reduction gain in the circuit delay (real value in ns).
- Estimate of the total increase in overall circuit load capacitance (real value in pF).
- Set of nodes (node number and its AHPL code) that have to be swapped to BiCMOS to get the optimized circuit.

6.2.2 Data Structures

We have adopted the following data structures to implement TS algorithm for COP:

1. **Node List:** is a dynamic list of elements where each element consists of a node number, its AHPL code and its selection frequency when the diversification

Gate Type	Base Delay (ns)	Load Factor	Input Capacitance (pF)	<i>CX</i>
4105	0.315	4.525	0.255	0.105
4103	0.32	2.05	0.455	0.234
4202	0.585	3.87	0.340	0.226
4302	0.88	3.715	4.15	0.355
4402	1.35	3.805	0.435	0.532
4102	0.275	0.579	1.670	0.751

Table 6.1: Sample of CMOS library.

strategy is applied. This list is constructed from the modified VPNR input file.

2. **Fanout List:** is a dynamic list of elements where each element consists of a node number and a set of fanout nodes. This is constructed from the fanout input file.

3. **Initial/Current Solution:** is an array of the longest sensitizable paths. Each element in the array consists of the following:

- Path number.
- Path delay.
- Dynamic list of the path nodes. Each element of the list consists of a node number, node delay, and node type.

The initial solution is constructed by reading the input sensitizable paths file and using other constructed data structures. This structure is also used for current solution where only the type and delay of the selected node are updated. We have used a static array for this structure because of two reasons. First, the average number of sensitizable paths (≈ 500) considered for optimization is not large enough to justify the use of a more complex dynamic structure. Second, the use of a static structure simplifies the implementation, and speeds up data manipulation and searching.

4. **Candidate List:** is an array of N elements representing N moves (swap of a gate from CMOS to BiCMOS or vice versa). Each element consists of a node number, type, node delay, new solution cost, and the move frequency when diversification strategy is applied.
5. **Tabu List:** is a queue of tabu moves. Each element represents a move and consists of move attributes including node number, its type, solution cost associated with this move, and the move frequency when diversification strategy is applied.
6. **Best Solution:** is a dynamic list of elements consisting of a set of nodes that need to be implemented in BiCMOS in order to obtain the best solution. The best cost (delay reduction and capacitance increase) is also recorded in this list.
7. **Admissible Solution:** is a record consisting of information about the admissible solution to be used for updating the best solution, updating the tabu list, and constructing the new current solution.

6.2.3 Implementation Method

The implementation of the algorithm proceeds in three stages as follows:

- *Stage 1: Initialization:* In this stage all three input files (modified VPNR circuit description, fanout description, and sensitizable paths) are read. Also

CMOS and BiCMOS libraries are read. While reading, *Node List*, *Fanout List* and *Initial Solution* data structures are constructed.

- *Stage 2: Circuit Optimization:* In this stage tabu search algorithm is processed. User defined data such as number of iterations, objective delay reduction percentage, capacitance threshold percentage, tabu list size, and candidate list size should be provided.
- *Stage 3: Output Generation:* The output of the program is written to a file that contains general information about the circuit, user defined inputs, and the *Best Solution* list.

Figure 6.4 shows a flowchart of the adopted TS based circuit optimization algorithm. The parts in the figure having a dashed outline correspond to those functions which are processed in the long term memory phase only.

6.3 Optimization Tools

In order to obtain an optimized mixed CMOS/BiCMOS circuit, different tools have to be used based on the given circuit description. Figure 6.6 shows all tools involved in the optimization process for ISCAS'85 benchmark circuits. Below is a brief description of each tool:

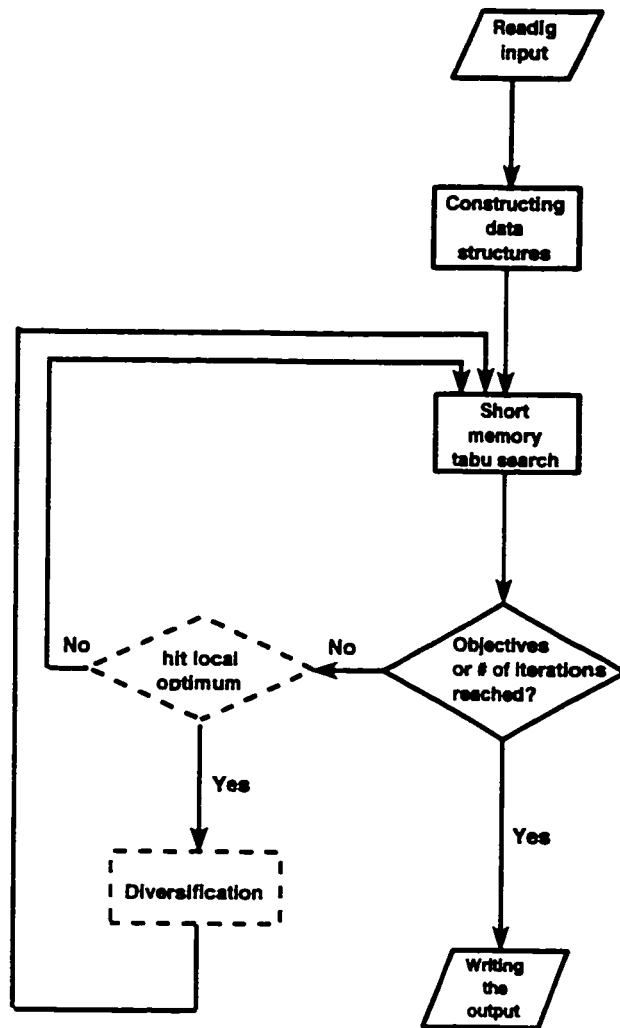


Figure 6.4: The adopted TS based circuit optimization algorithm.

- **TRANS:** this tool is used to convert the ISCAS'85 circuit format (SLIF) to RNL format.
- **OASIS:** this tool has been developed by MCNC and is heavily used on most of VLSI research and development processes because it has the capability of constructing circuit layout from the high level description (RNL). We have used this tool only to convert RNL circuit description to VPNR description. This step can be eliminated if a tool is available to convert SLIF format directly to VPNR format.
- **PP2:** this tool has been developed by Al-Farrah K. as reported in [AF95]. It is used just to assign a positive integer number and AHPL code to each node in the circuit to ease referencing and manipulation.
- **NEW:** this tool has also been developed by Al-Farrah. This is a Timing Analyzer of a given circuit to generate critical paths of a circuit based on α -critical algorithm described earlier.
- **FPFIND:** as previously described, this tool is used to eliminate false paths from given critical paths generated by the Timing Analyzer.
- **MIXER-S & MIXER-L:** both tools form the heart of the whole optimization process. MIXER-S is the implemented circuit optimization algorithm based on TS short term memory and MIXER-L is the implementation of long term

memory of TS based on the proposed diversification strategy. Both tools have different versions based on the aspiration criterion (AS_1 or AS_2) and move generation method (CTS or ETS).

The figure also shows all other necessary inputs and outputs of these tools. A more detailed description of the tools and how they are used is given in Appendix A. All the tools mentioned above (except OASIS) have been developed using C++ programming language. The development has been made on PC and SUN SPARC stations. However the results have been obtained using SUN SPARC workstations only.

6.4 Experiments

Our approach for COP using TS can be applied for any two technologies as long as it is feasible to mix them in one circuit. We applied this approach on optimization of mixed CMOS/BiCMOS circuits as discussed before, hence standard cell libraries for both technologies have been used. To simplify the timing computation, a modified version of the standard CMOS 1.0 library has been used. The following modification have been made:

- Only the average of rising and falling values have been considered.
- Input capacitance has been made to be five times greater than the original input capacitance in order to show the effect of the optimization for small

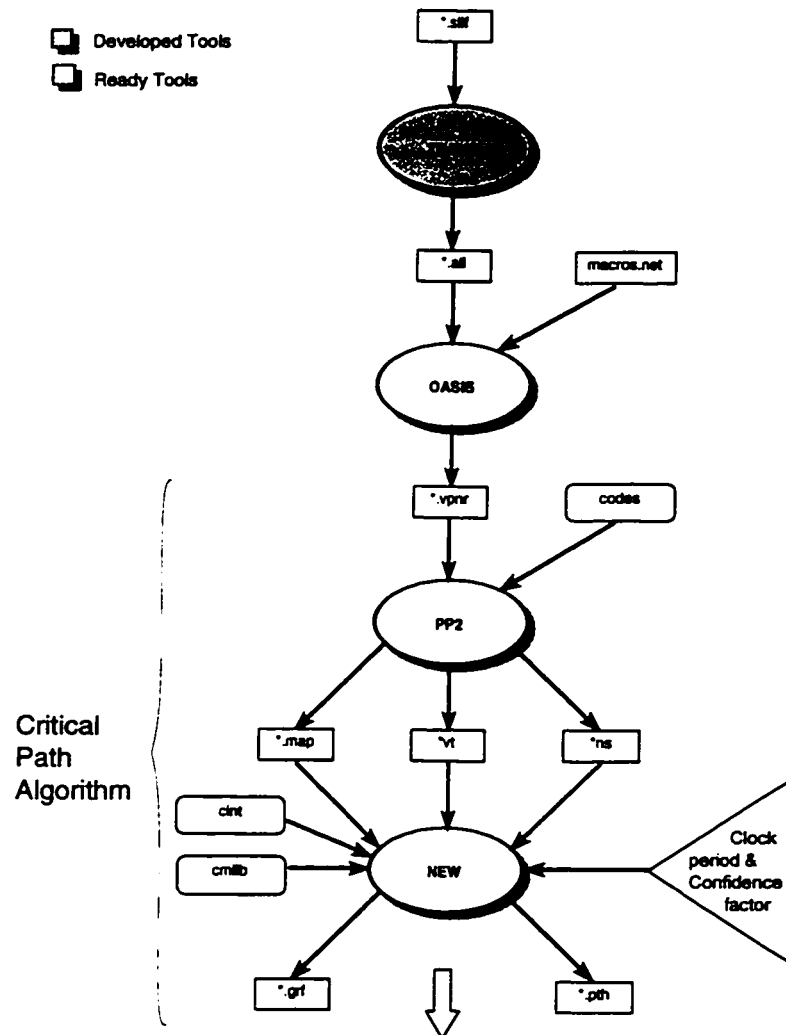


Figure 6.5: Mixed CMOS/BiCMOS circuit optimization tools (cont.).



circuits where the fan out of each gate is very small.

- CX parameter has been added.

Since BiCMOS technology library was not available during the work, we have created a new library based on the following facts from CMOS and BiCMOS specifications [EBE93], [BAB94]:

- Base delay of BiCMOS cell $>$ Base delay of its equivalent CMOS cell.
- Load factor of BiCMOS cell $<$ Load factor of its equivalent CMOS cell.
- Input capacitance of BiCMOS cell $>$ Input capacitance of its equivalent CMOS cell.

The question is: By how much is it greater or smaller? Based on the library used in [BAB94], the following equations have been derived:

- $BiCMOS_Base_Delay = CMOS_Base_Delay \times 2$
- $BiCMOS_Load_Factor = CMOS_Load_Factor / 3$
- $BiCMOS_Input_Capacitance = CMOS_Input_Capacitance \times 1.4$

Based on the above values, CX_i of a gate g_i has been calculated as follows. CX is the capacitive load value of a gate when its CMOS delay is equal to its BiCMOS delay. That is, at CX_i :

$$CMOSDelay(g_i) = BiCMOSDelay(g_i) \quad (6.1)$$

Using Equation 4.2 for the calculation of gate delay,

$$BD_i + LF_i \times CX_i = BD_i \times 2 + \frac{LF_i}{3} \times CX_i \quad (6.2)$$

Then,

$$CX_i = \frac{3}{2} \times \frac{BD_i}{LF_i} \quad (6.3)$$

The modified CMOS library and the new BiCMOS library are shown in Appendix A.

We carried out different experiments with different values of objective delay, objective load, tabu list size, candidate list size, etc. We can classify these experiments into two categories:

1. Experiments to show the actual gain in terms of circuit delay and total load.

For these experiments we used the following data:

- Capacitance threshold = 5%.
- Objective delay reduction = the difference between maximum and minimum delay as shown in Table 4.2.
- Number of iterations = 2000.

2. Experiments to show the difference, in terms of quality and performance, between short and long term of TS, TS with AS_1 and TS with AS_2 , and between Classical TS and Evolutionary TS. For these experiments, the following data have been used:

- Capacitance threshold = 10%.
- Objective reduction in the delay = 30%.
- Maximum number of iterations = 2000.

Reference to equation 5.8, the penalty value has to be identified. The purpose of the penalty is to penalize a solution which violates the capacitance constraints so as to forbid selection of such solution. To achieve that, the penalty value has to be larger than the maximum solution cost of the inspected circuit. Although this choice does work, it needs to be provided for each circuit. In our experiments on the selected benchmark circuits, we have used 1000 as the penalty value based on the knowledge that the maximum cost function of all those circuits can not exceed a value 1000.

In attempting to find the best results and observe the behavior of TS, experiments have been conducted with varying values of tabu list size T_SIZE and candidate list size CAN_SIZE . The following data have been used:

$T_SIZE = 4, 5, 6, 7, 8, 9$

$CAN_SIZE = 10, 12, 14, 16, 18, 20$.

In the long term TS, according to our diversification strategy, the algorithm should switch to diversification step when a local optimum is hit. Based on several experiments on the benchmark circuits, we identified that the local optimum is hit when there is no change in the best solution value for the last x iterations. The value of x varies with different circuits. For each circuit, we have experimented with $x = 100, 200, 300, 400, 500$ and recorded the best results.

Several versions of TS have been implemented to show the difference between the proposed strategies:

- Short term *Classical Tabu Search* with AS_1
- Long term *Classical Tabu Search* with AS_1
- Long term *Classical Tabu Search* with AS_2
- Long term *Evolutionary Tabu Search* with AS_1

6.5 Results and Discussion

In this section, the results of applying the circuit optimization tools on the selected benchmark circuits are presented and analyzed. First, we show the delay improvement results of applying long term TS. Next we present and discuss the results of applying the different strategies mentioned in the previous section. Finally, a detailed discussion regarding TS behavior for COP is presented.

6.5.1 Circuit Delay Improvement Results

Table 6.2 shows a summary of the results of circuit delay improvement obtained by applying long term TS with AS_1 on some of the benchmark circuits. Three circuits, c499, c432, and c6288 are not considered because there is no difference between the maximum and minimum delay of their sensitizable paths. From this table, we can observe that considerable improvement in the performance of mixed CMOS/BiCMOS circuits as compared to pure CMOS versions has been achieved. The improvement ranges approximately from 7% to 24%. The overhead in the capacitance (which reflects overhead in the power) is very small, about 0.1% to 3%. In addition, the increase in the total area is minimal due to the small number of selected BiCMOS gates compared to the overall number of gates. It is clear that for the first four circuits, the objective delay reduction has been achieved whereas it has not been achieved for “highway” and “fract” because of the low load threshold constraint. More speed improvement could be achieved if more sensitizable paths are considered (which means higher objective delay reduction) and the load threshold is increased.

Circuit Name	Max Delay(ns)	Delay Red.	%	Total Cap.(pF)	Cap. Incr.	%	No. of BiCMOS
c880	125.506	10.062	8.02	215.111	0.244	0.11	3
c1355	109.860	15.393	14.01	399.564	0.606	0.2	6
c3540	185.201	12.4	6.7	683.401	0.204	0.3	2
struct	121.894	8.386	6.89	750.081	1.102	0.15	5
highway	32.438	7.939	24.47	15.15	0.355	2.34	12
fract	76.575	16.994	22.19	43.405	1.311	3.02	12

Table 6.2: Circuit delay improvement results for some circuits.

6.5.2 Results of Short Term TS vs. Results of Long Term TS

In this section and the following sections, all results have been obtained by conducting the second category of experiments where the objective delay may exceed the range between maximum and minimum delay of the inspected sensitizable paths. Therefore the results in this case show the delay improvement within the inspected sensitizable paths and do not reflect the actual improvement in the circuit delay.

Table 6.3 shows a summary of the best results obtained by applying short term TS with AS_1 . From this table, we can observe that a very large improvement of about 10% to 29% of the circuit delay has been achieved. The overhead of capacitance (which reflects overhead in the power) is very small, about 0.4% to 6%. In addition, the increase in the total area is minimal due to the small number of selected BiCMOS gates as compared to overall number of gates. These results are constrained by the chosen capacitance threshold which is 10% of the total capacitance of the set A (recall that A is the set of nodes that are covered by the selected sensitizable paths and satisfy $CL > CX$). More speed improvement could be achieved if the capacitance threshold is increased.

As we have explained earlier, long term memory phase of TS is very important to escape from local optima in which the short term memory phase may fall. The results shown in Table 6.4 illustrate the effect of this feature. Most of the results

Circuit Name	No. of Nodes with $CL > CX$	T_SIZE	CAN_SIZE	No. of BiCMOS
c432	55	6	20	19
c499	41	7	18	13
c880	79	4	16	22
c1355	265	9	16	65
c3540	59	4	16	21
c6288	267	5	20	77
struct	67	9	16	26
highway	29	7	20	7
fract	93	4	20	24

Circuit Name	Max Delay(ns)	Delay Red.	% of Delay Red.	Total Cap.(pF)	Cap. Incr.	% of Cap. Incr.
c432	171.911	43.342	25.2	109.260	1.514	1.4
c499	65.344	9.875	15.1	101.717	1.388	1.4
c880	125.506	25.212	20.1	215.111	2.122	0.99
c1355	109.860	26.677	24.3	399.564	7.266	1.8
c3540	185.201	48.207	26.03	683.401	1.732	0.25
c6288	657.646	68.234	10.4	1983.395	7.854	0.4
struct	121.894	25.899	21.25	750.081	2.534	0.34
highway	32.438	8.109	25	15.15	0.762	5.0
fract	76.575	22.3	29.1	43.405	2.642	6.1

Table 6.3: Best results of short term memory of TS with AS_1 .

Circuit Name	Nodes with $CL > CX$	T_SIZE	CAN_SIZE	No. of BiCMOS	Hit Local Opt. Iter.
c432	55	6	18	19	500
c499	41	7	18	21	200
c880	79	9	16	23	400
c1355	265	4	20	63	400
c3540	59	6	20	18	500
c6288	267	7	10	166	100
struct	67	4	20	26	100
highway	29	7	16	17	100
fract	93	7	16	23	500

Circuit Name	Max Delay(ns)	Delay Red.	% of Delay Red.	Total Cap.(pF)	Cap. Incr.	% of Cap. Incr.
c432	171.911	43.342	25.2	109.260	1.478	1.0
c499	65.344	13.357	20	101.717	1.384	1.0
c880	125.506	25.212	20.1	215.111	2.122	0.99
c1355	109.860	26.677	24.3	399.564	7.266	1.8
c3540	185.201	48.645	26.3	683.401	1.796	0.26
c6288	657.646	98.103	14.9	1983.395	7.856	0.4
struct	121.894	26.106	21.4	750.081	2.534	0.34
highway	32.438	8.815	27.2	15.15	0.762	5.0
fract	76.575	22.497	29.38	43.405	2.691	6.2

Table 6.4: Best results of long term memory of TS with AS_1 .

produced by long term phase show higher speed improvement than what is produced by short term phase especially in the case of c499 and c6288 where 20% and 15.2% improvement has been achieved as compared to 15.1% and 10.4% improvement respectively. Also it is shown from the table that best results for each circuit are obtained at different diversification factor (number of iterations where the proposed diversification strategy is applied). This is expected due to the fact that those circuits have different sizes and search spaces.

6.5.3 Results of TS with AS_1 vs. Results of TS with AS_2

Table 6.5 shows the best results obtained by applying long term memory TS with AS_2 . If we compare the results of TS using AS_1 with those of TS using AS_2 , we find that TS with AS_2 performs better than TS with AS_1 for the circuits c6288, highway, and fract, while it generates almost the same results as TS with AS_1 for the other circuits. In the case of “fract”, the delay reduction objective has been achieved which means that the run time (2000 iterations) is enough to reach the stated objectives. Let us look at the AS_2 again to explain why TS with AS_2 produces better results. Using AS_2 , a move e is accepted if it is tabu and both e and its reverse move \bar{e} are improving or both are non-improving. This means that the solution tries to follow a certain direction to seek a better solution than the current one. By following a certain direction during the search, TS tries to climb the hill to escape from local optima. Of course for some cases, TS with AS_2 does not produce better results than

Circuit Name	Nodes with $CL > CX$	T_SIZE	CAN_SIZE	No. of BiCMOS	Hit Local Opt. Iter.
c432	55	5	20	19	500
c499	41	4	18	10	200
c880	79	5	18	21	300
c1355	265	4	20	62	200
c3540	59	5	14	18	400
c6288	267	5	20	175	100
struct	67	7	20	26	200
highway	29	9	20	12	400
fract	93	6	18	23	300

Circuit Name	Max Delay(ns)	Delay Red.	% of Delay Red.	Total Cap.(pF)	Cap. Incr.	% of Cap. Incr.
c432	171.911	43.342	25.2	109.260	1.516	1.4
c499	65.344	12.276	18.8	101.717	1.434	1.4
c880	125.506	25.257	20.1	215.111	2.144	1.0
c1355	109.860	26.712	24.3	399.564	7.164	1.8
c3540	185.201	48.645	26.3	683.401	1.796	0.26
c6288	657.646	99.712	15.2	1983.395	7.906	0.4
struct	121.894	26.130	21.4	750.081	2.534	0.34
highway	32.438	8.542	26.3	15.15	0.762	5.0
fract	76.575	22.975	30.0	43.405	2.667	6.1

Table 6.5: Best results of long term memory of TS with AS_2 .

TS with AS_1 because the TS algorithm is nondeterministic, hence several runs have to be conducted to get the best.

6.5.4 Results of Classical TS vs. Results of Evolutionary TS

Table 6.6 shows a summary of the best results obtained by applying long term *ETS*. If we compare these results with results of *CTS* (Table 6.4), we find that *ETS* performs better than *CTS* in four cases; namely in c880, c1355, struct, and fract. On the other hand, *CTS* produced better results than *ETS* in case of c432, c499, c6288, and highway while both of the strategies produced the same results in case of c3540.

This reflects the fact that the Evolutionary TS could generate good results because it tries to select and replace only those gates which have low fitness; i.e., those that are good to replace. However it may get trapped in a local optimum sometimes as is clear from some results that are worse than those of random generation strategy. Let us explain the reason behind this kind of behavior.

In the selection step of Simulated Evolution, a node i is selected if the following inequality is satisfied:

$$Random \leq Min(1, 1 - G_i + B) \quad (6.4)$$

If the goodness G_i is low, the probability of selecting the node i will be high. Ac-

Circuit Name	Nodes with $CL > CX$	T_SIZE	CAN_SIZE	No. of BiCMOS	Hit Local Opt. Iter.
c432	55	4	20	19	500
c499	41	4	20	20	100
c880	79	5	20	22	500
c1355	265	6	20	65	500
c3540	59	5	16	19	400
c6288	267	4	20	135	500
struct	67	8	18	26	500
highway	29	5	16	13	500
fract	93	6	14	21	100

Circuit Name	Max Delay(ns)	Delay Red.	% of Delay Red.	Total Cap.(pF)	Cap. Incr.	% of Cap. Incr.
c432	171.911	43.308	25.2	109.260	1.516	1.4
c499	65.344	12.463	19	101.717	1.43	1.4
c880	125.506	25.273	20.1	215.111	2.16	1.0
c1355	109.860	27.016	24.6	399.564	7.228	1.8
c3540	185.201	48.645	26.3	683.401	1.796	0.26
c6288	657.646	71.13	10.8	1983.395	7.902	0.4
struct	121.894	26.54	21.8	750.081	2.534	0.34
highway	32.438	8.720	26.9	15.15	0.777	5.1
fract	76.575	22.976	30	43.405	2.675	6.16

Table 6.6: Best results of long term memory of Evolutionary TS.

cording to our goodness function, if a CMOS gate with low goodness is selected for swapping, then its goodness will be high. Then the probability of re-selecting this gate again (as a mechanism of TS to escape from local optima) will be very low. Therefore, once some of the gates are swapped to BiCMOS, they are unlikely to be selected again resulting in a local optimal solution. In order to avoid that, we used the bias B only in the case when a gate is BiCMOS to increase its selection probability even if its goodness is high. But the question is what value of B should be used to achieve this purpose?. We experimented with different values from 0.1 to 0.7 with interval of 0.1. By doing so we achieved some good results as shown in the table with the values 0.5 - 0.7. However, the use of bias B only still is not enough to avoid trap of local optimality. Therefore, we have to modify our goodness function or selection criterion in such a way as to enable TS to escape from local optimality as the Classical TS does.

As another method of measuring the quality and performance of Evolutionary TS versus Classical TS, we calculated the average delay reduction for several executions of both algorithms. These results are shown in Table 6.7 for some of benchmark circuits. For each circuit, both *CTS* and *ETS* have been executed for 2000 iterations for different values of tabu list size, candidate list size, and the iteration value where a local optimum is hit as mentioned in the previous section. Then the average value has been computed for each.

It is obvious that *ETS* produces better solutions (i.e., more delay reduction)

Circuit Name	Classical TS		Evolutionary TS	
	Average Delay Reduction (ns)	Run Time (s)	Average Delay Reduction (ns)	Run Time (s)
c432	42.39	313	42.42	544
c499	7.47	133	7.49	471
c880	23.87	167	24.11	566
c1355	23.82	379	24.47	1578
highway	7.72	52	7.83	35
fract	21.44	99	22.72	135

Table 6.7: Comparison between Classical TS and Evolutionary TS in terms of quality and performance.

than *CTS* on the average. This means that to achieve a certain objective solution, it is better to apply *ETS* for a few trials. However, *ETS* takes more time to finish executing 2000 iterations or to achieve objectives than what *CTS* takes. There are two reasons behind this. First, the step of computing the goodness of all nodes each iteration (not present in the Classical TS) in Evolutionary TS is an expensive step. Second, each iteration in Evolutionary TS takes much more time than the time spent in each iteration in Classical TS. Let us compare the move generation step in both approaches to clarify the point. In Classical TS, in each iteration N moves are generated by selecting only N nodes randomly and then the necessary processing and evaluation is done. In Evolutionary TS, in order to generate N moves, we randomly select a node and check if its goodness satisfies the inequality 6.4 or not. If the inequality is satisfied, then the node is accepted as one of candidates; otherwise another node has to be selected randomly again and the process is repeated. Therefore, generating N moves in this case requires selection of M nodes where $M \geq N$. Obviously this step is more expensive than its equivalent step in Classical TS.

6.5.5 Tabu Search Behavior

In this section, a discussion on TS behavior in terms of the effect of tabu list size, candidate list size and diversification strategy on the solution cost are presented. The behavior of Classical TS versus Evolutionary TS is also addressed. First, we discuss the effect of tabu list size and candidate list size on the solution cost, hence

on the delay reduction. In order to observe this behavior, the following experiment has been conducted. One benchmark circuit c499 has been chosen for this purpose. Then the results for different tabu list sizes as well as different candidate list sizes have been obtained. The average solution cost of $CAN_SIZE = 10, 12, 14, 16, 18, 20$ for a particular T_SIZE has been computed. Finally, the results have been plotted in Figure 6.7 for short term phase. The same has been made for CAN_SIZE versus cost for the average values of $T_SIZE = 4, 5, 6, 7, 8, 9$. The plot of the effect of candidate list size on the solution cost is shown in Figure 6.8. If we look at Figure 6.7, we find the highest values occur at $T_SIZE = 4$ and $T_SIZE = 7$. From these results, it is difficult to deduce what the best tabu list size is. However, from the results shown in previous tables, we can conclude that for COP applied on the selected benchmark circuits, good values for T_SIZE are from 4 to 7. Nevertheless, tabu list size depends heavily on the type of the problem, problem size, and other factors as reported in previous research. But generally the tabu list size should not be very small. If not, the feature of using memory component to keep the history of selected moves will not be applicable. Also it should not be very large as the idea of move reversal for hill climbing will not be applied properly.

For the candidate list size effect, most of the best results occurred at CAN_SIZE between 16 to 20 as shown in Figure 6.8 as well as in the tables of best results. This means that for COP, it is better to inspect large number of neighbor solutions which constitute a very small percentage of the whole solution space.

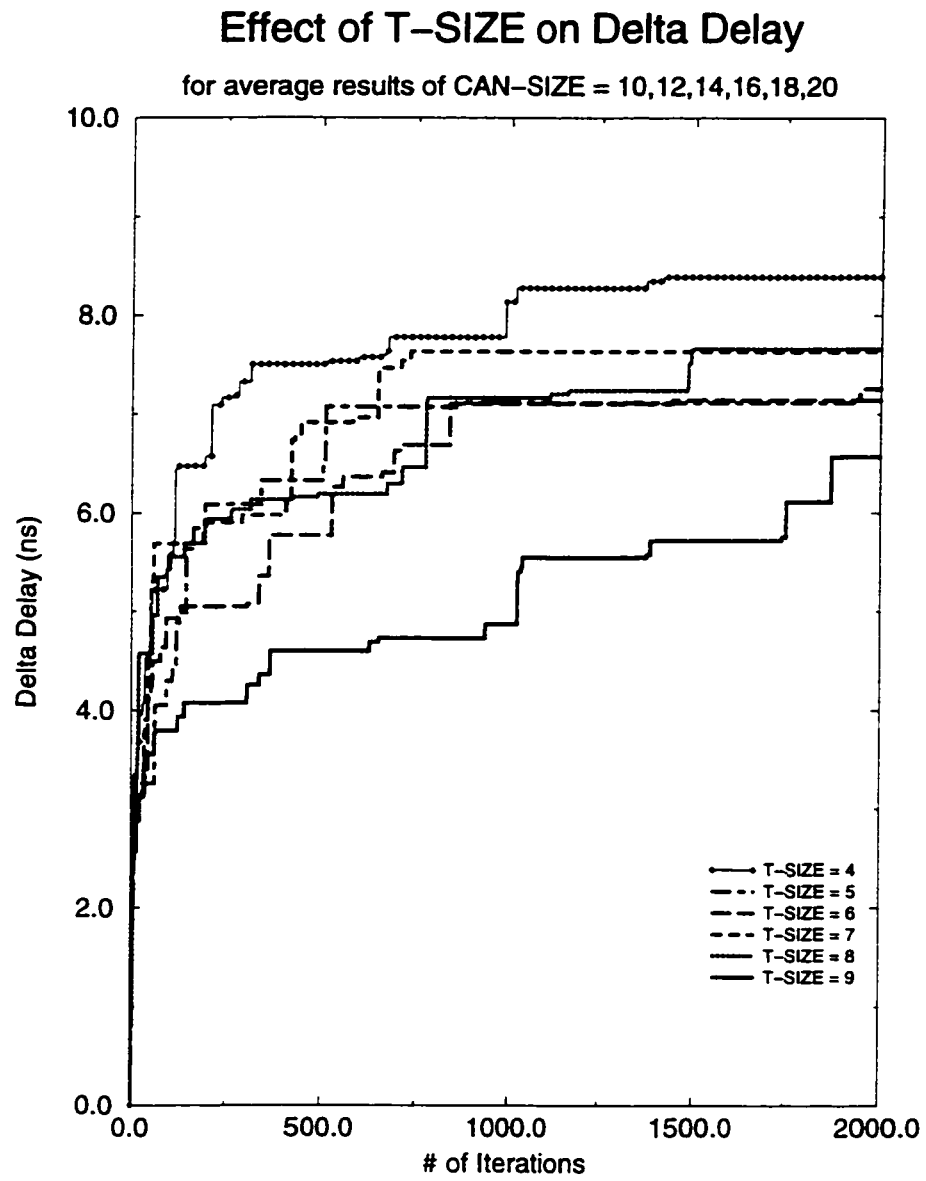


Figure 6.7: Effect of T_SIZE on delta delay for short term memory.

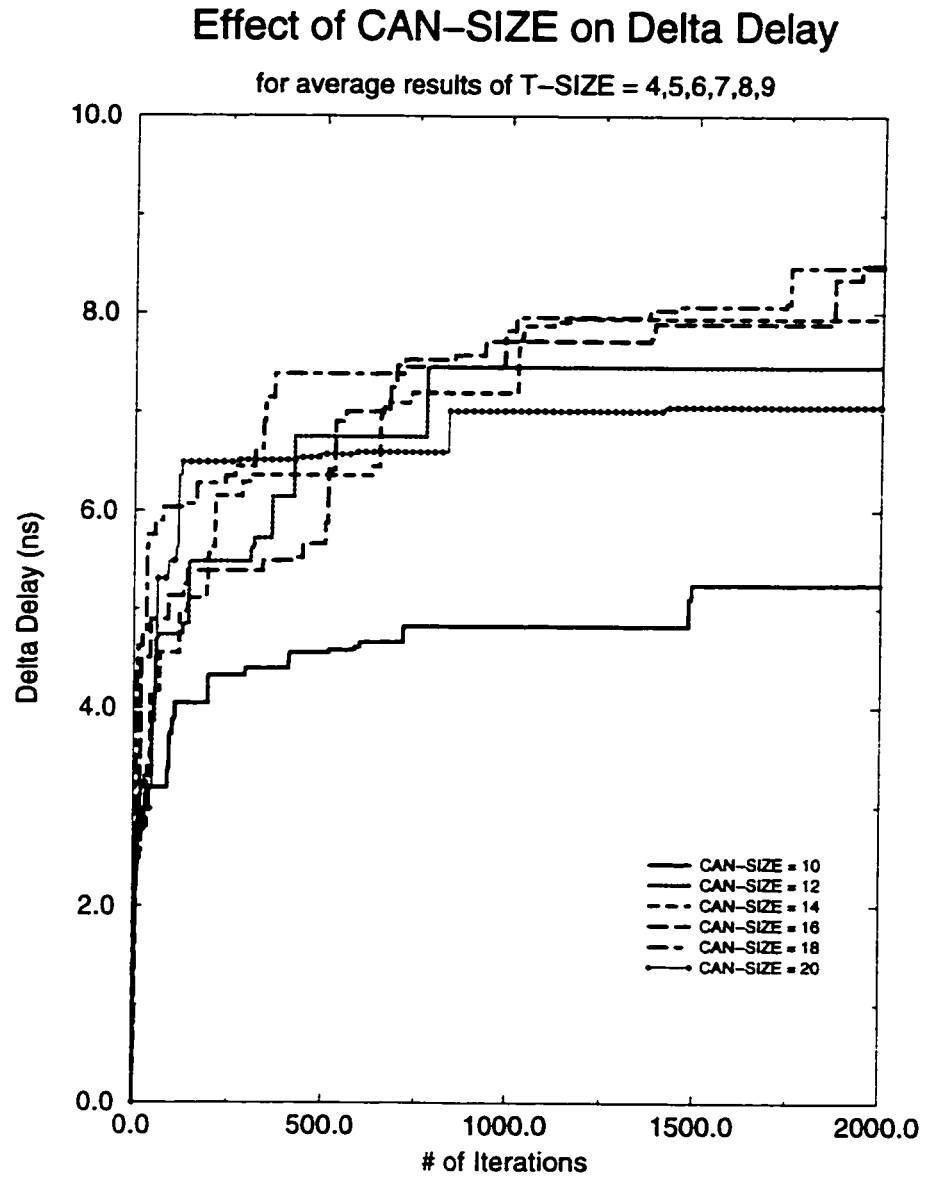


Figure 6.8: Effect of CAN_SIZE on delta delay for short term memory.

To show the behavior of short term memory phase of TS in terms of current solution cost and the best solution cost, we chose as an example the results obtained for the circuit c499 at $T_SIZE = 5$ and $CAN_SIZE = 14$. The plot of these results are shown in Figure 6.9. Figure 6.10 shows the behavior of long term memory phase of TS in terms of the current and best cost for the same circuit.

From Figure 6.9, it is obvious that after small number of iterations (short term), the algorithm reaches a local optimal solution, as expected, and gets trapped at that level for the rest of running time. Actually we can see also that when an illegal move is made, the solution cost is penalized which hardly improves again. The solution of this trap is to diversify the search as shown in Figure 6.10. From this figure we can see that when the solution hits a local optimum (no change of best solution for the last 200 iterations), the proposed diversification strategy drives the search to another region where the cost of the new solution gets improved. It is clear that after 409 iterations where that current solution cost was approximately 4, the diversification produced better solution with cost around 6. As diversification is a procedure of long term memory, the longer TS runs, the better would be the result. This fact is clear in the figure where after 1700 iterations, the solution got also improved.

For easy comparison between the behavior of short term and long term phases, we combined the plots in Figure 6.9 and Figure 6.10 in one plot as shown in Figure 6.11. The penalty value we have chosen is 1000 as mentioned before. However, in the plots, it has been scaled to 30 just to give better layout. Some of the values of solution cost

Behavior of Short Term Memory of TS

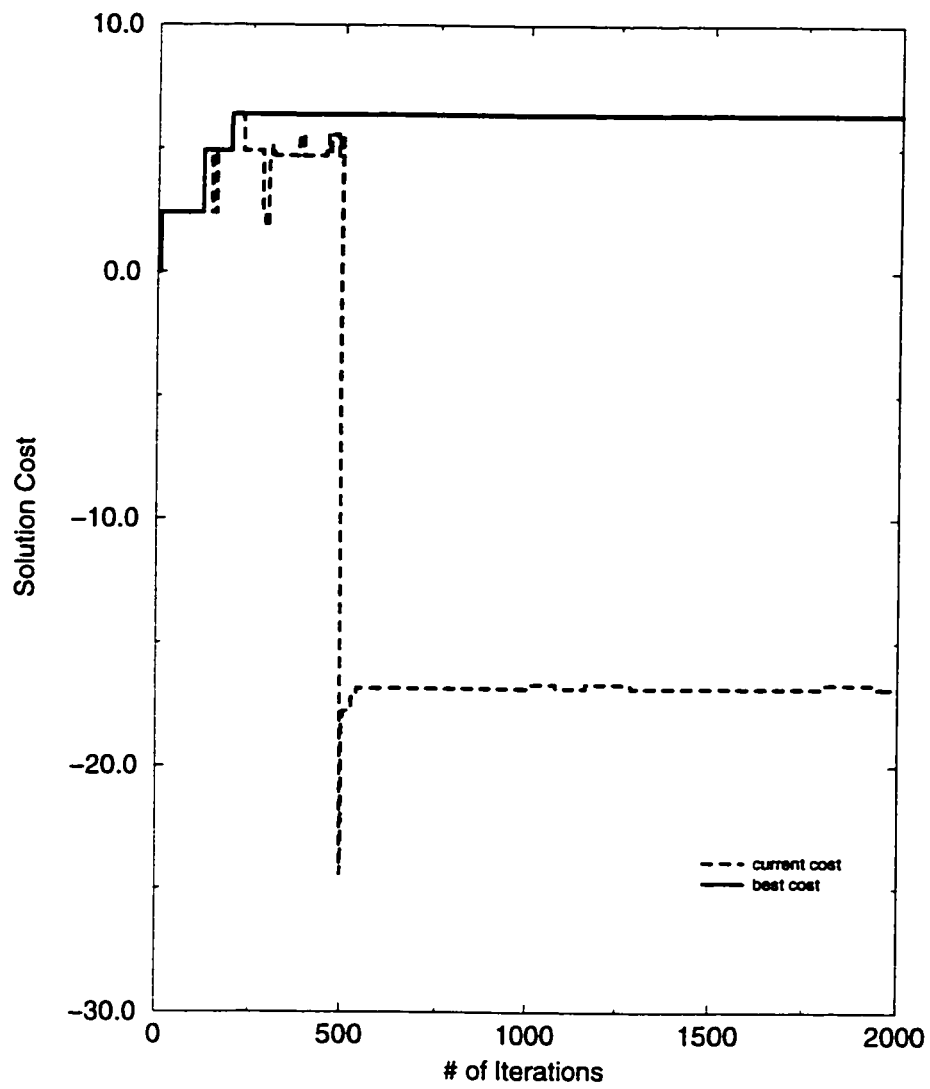


Figure 6.9: Behavior of short term memory TS in terms of current and best cost.

Behavior of Long Term Memory of TS

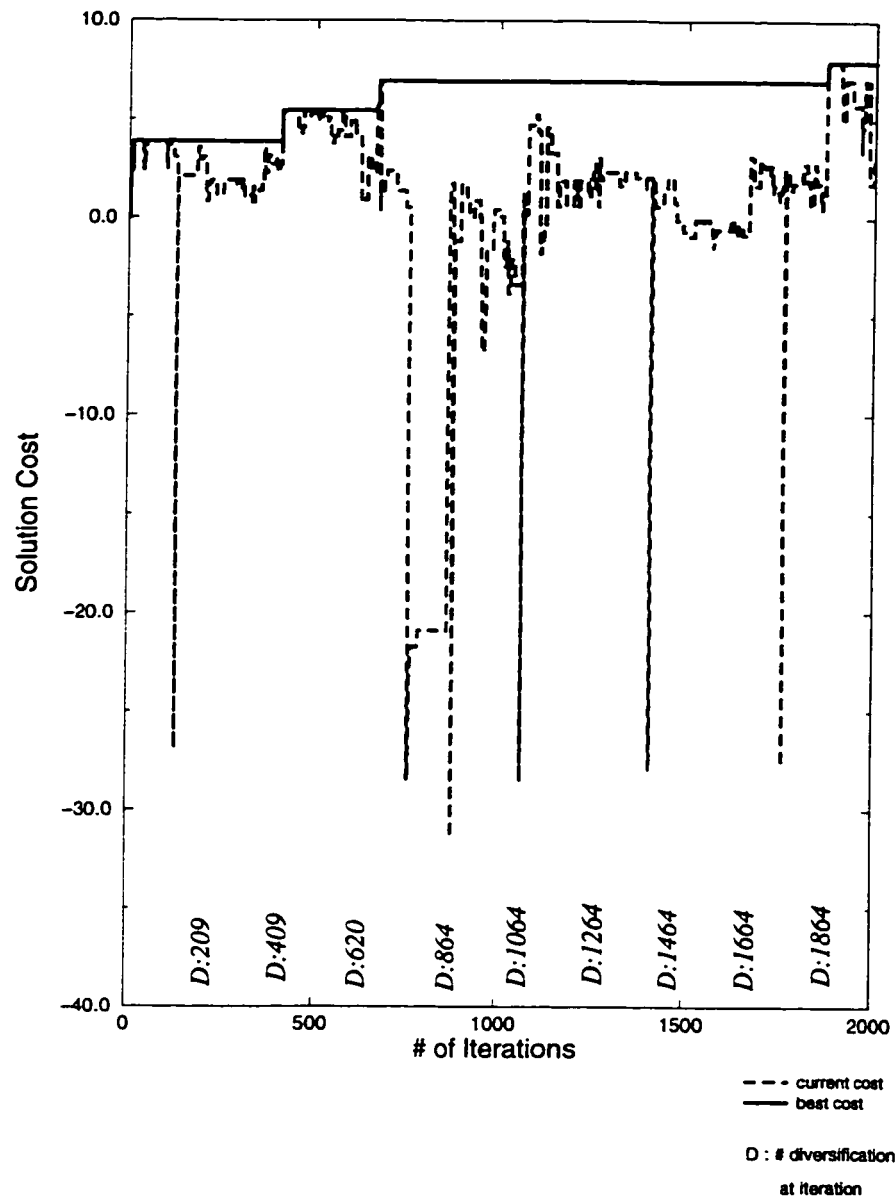


Figure 6.10: Behavior of long term memory TS in terms of current and best cost.

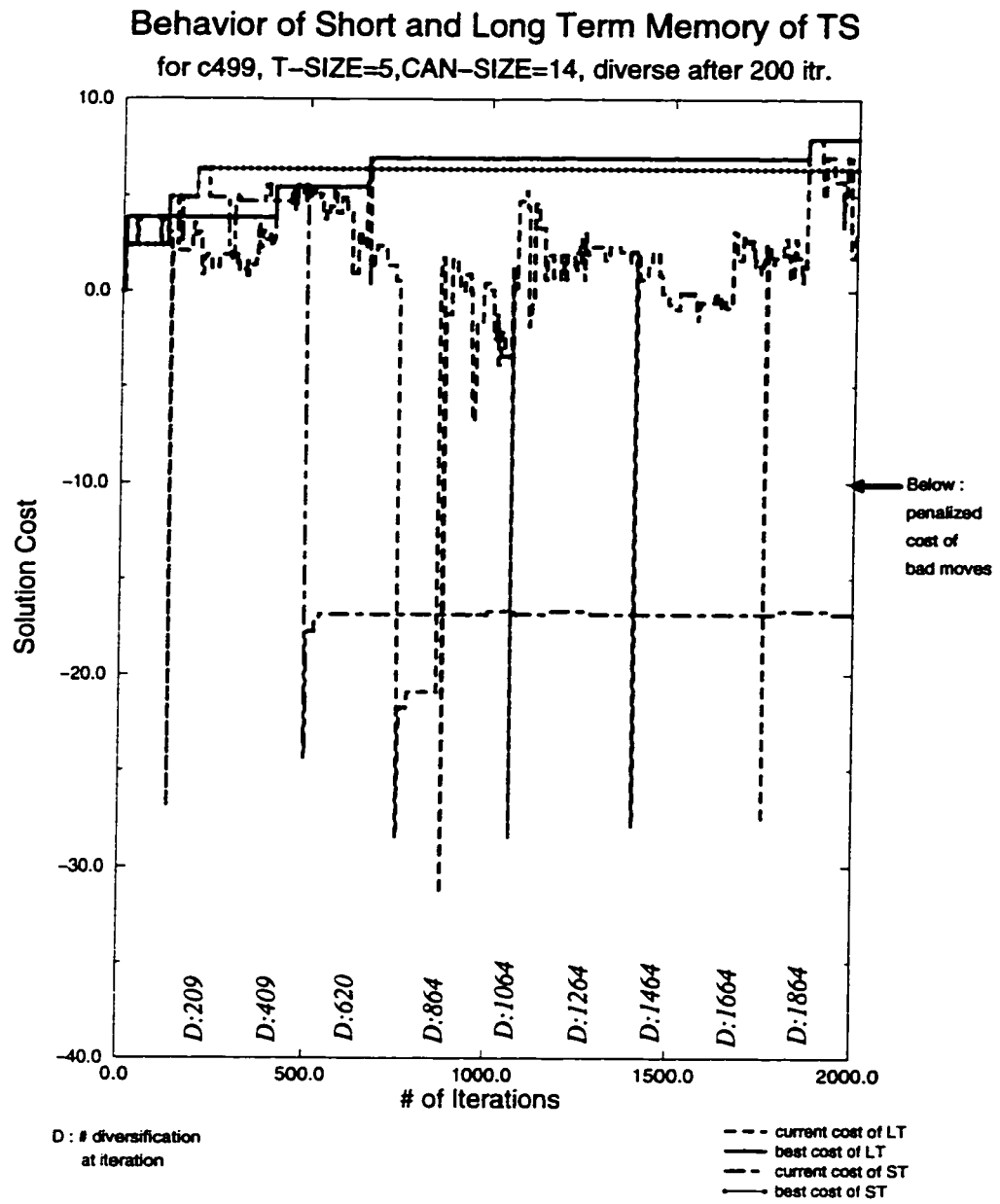


Figure 6.11: Comparison between the behavior of short and long term memory TS.

are negative even though their associated moves are not penalized. This is because of the fact that at some instances, the current solution may consist of BiCMOS gates which are not on the longest paths and their driving CMOS gates are on the longest paths making the CMOS delay of those gates greater than their original delay, hence the overall delay is increased.

In order to compare the behavior of *CTS* versus *ETS*, both algorithms have been applied on c499. The data of current cost and best cost for 2000 iterations have been collected and plotted in Figure 6.12 and Figure 6.13 respectively. As clear from Figure 6.12 that *ETS* finds good solutions (in this example, good delay reduction is around 7 ns.) quickly in a few iterations (in this example, around 200) because it examines small set of gates having low goodness. This means that some gates with low goodness are swapped to BiCMOS to get better solution. Then, *ETS* tries to look for any gate with low goodness to swap. But since most of the gates now are having high goodness, *ETS* will swap some of these gates producing worse solutions than before. Therefore, it will get trap at local optimal solution. On the other hand, *CTS* requires more number of iterations to find good solutions because it examine all gates in the critical paths. However, after finding a good solution (delay reduction around 6 ns in this case), it continues in generating good solutions and even better solutions found so far. This is clear from the figure where *ETS* produced good solutions with the following reduction in the delay, around 5 ns after 500 iterations, around 6.5 ns after 700 iterations, around 5 ns after 1200 iterations,

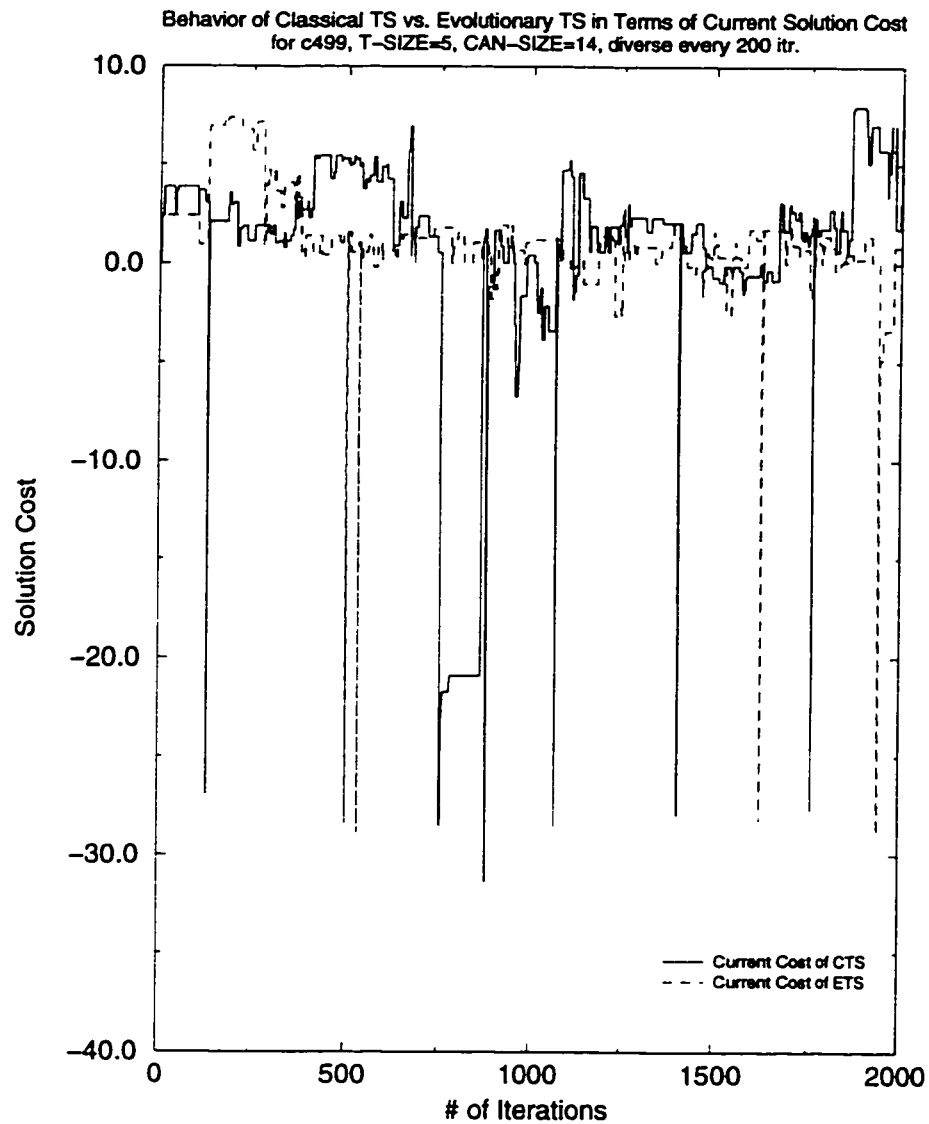


Figure 6.12: Comparison between the behavior of Classical TS and Evolutionary TS in terms of current solution cost.

and around 7.5 ns after 1800 iterations. This reflects the fact that *CTS* is capable of escaping from local optimum trap. Figure 6.13 gives more clearer idea about the quality of both *CTS* and *ETS*. Obviously, *ETS* jumps quickly to a good solutions and stuck there while *CTS* gradually produces better solutions as it runs for more time.

Figure 6.14 shows the distribution of solutions for 2000 iterations for the circuit c499 at $T_SIZE = 5$ and $CAN_SIZE = 14$. For each interval in the X-axis, it is shown how many solutions have achieved the designated delay reduction within that interval. Obviously, more than 80% of the solutions have delay reduction greater than 4.0 which represents 50% of the maximum delay reduction. This means that TS tries to achieve the objective as quick as possible. In addition, from the figure we can see that most of the solutions occur between the values 5.5 and 7.0, and a few occur between the values 7.0 and 8.0. This distribution, particularly for this circuit, reflects the fact that the former values in delay reduction can be easily achieved through the short term TS whereas the latter values require long term phase. Again we can see for almost all solutions that *ETS* has visited, they almost have the same cost meaning that *ETS* quickly hits a good solution. Therefore, in general, TS is very effective and efficient in generating a desired solution for COP.

One last comment on the behavior of TS for COP is related to the running time. The running time is dependent on a number of factors including circuit size, number of iterations, move generation strategy, tabu list size, and candidate list size. In our

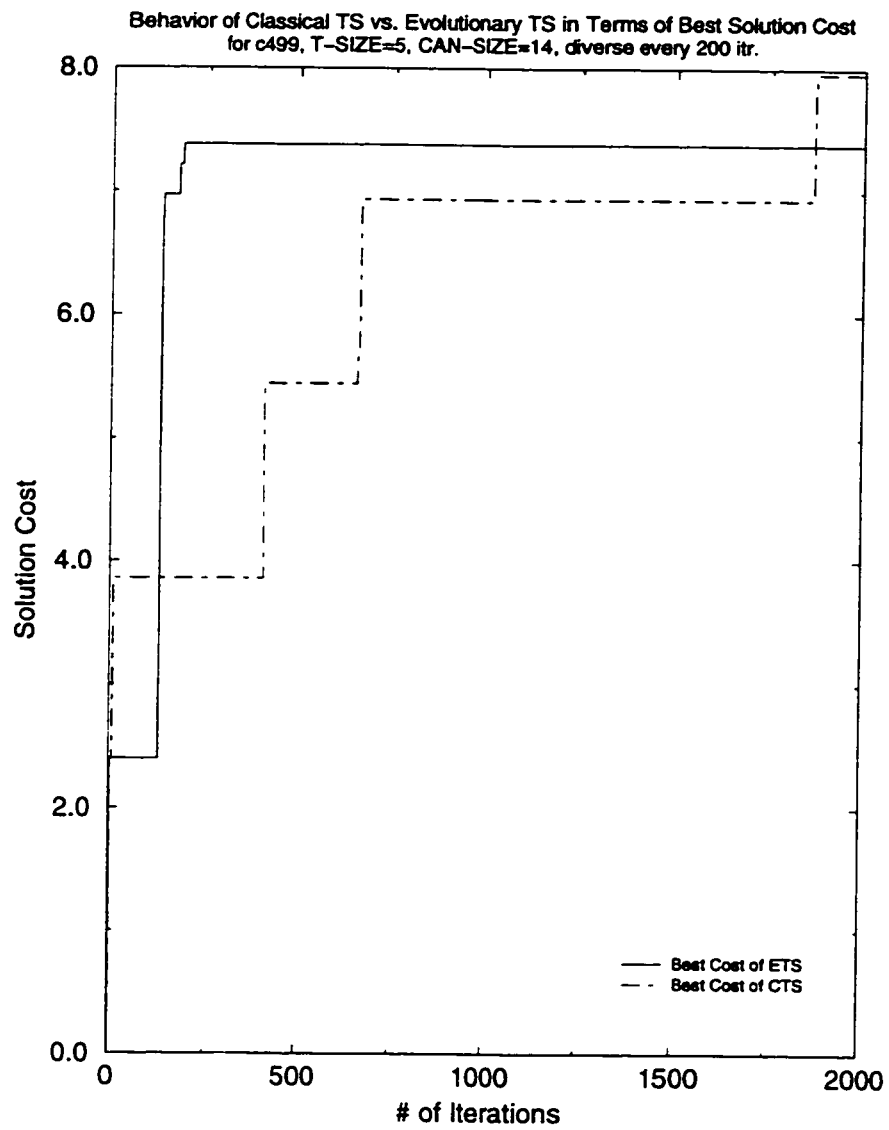


Figure 6.13: Comparison between the behavior of Classical TS and Evolutionary TS in terms of best solution cost.

Number of Solutions

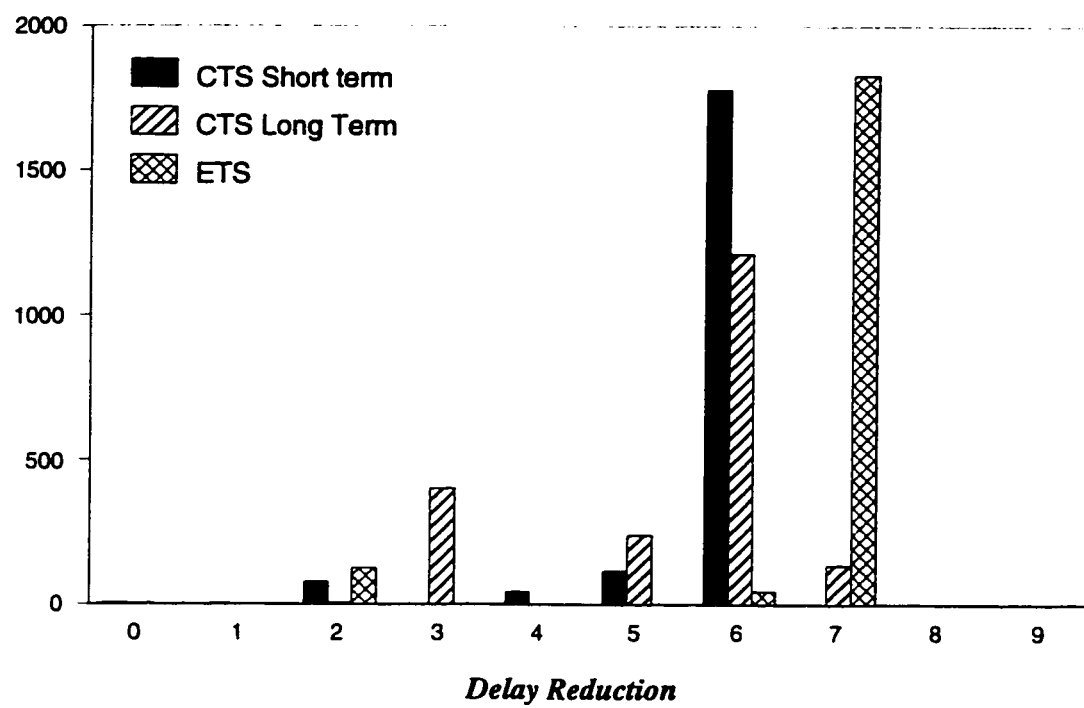


Figure 6.14: Solutions distribution.

experiments we applied TS for 2000 iterations. Therefore, for 2000 iterations, the Classical TS short term takes around 10 - 120 seconds while the Classical TS long term takes around 50 - 600 seconds. Evolutionary TS in the long term takes much more time, around 50 - 1600 seconds, than Classical TS as clear from Table 6.7.

6.6 Summary

The proposed tabu search based algorithm for solving COP has been implemented using C programming language (MIXER-S and MIXER-L). The integration of these tools with the format conversion tools (TRANS and OASIS) and longest sensitizable paths generation tools (NEW and FPFIND) forms an easy and efficient approach for optimizing mixed CMOS/BiCMOS circuits. To find out a good or a near optimal solution for COP using this approach, we experimented with different values of tabu list size, candidate list size, and diversification factor because specific values for these parameters are unknown to produce an optimal solution. The best results have been presented with our observations and discussions on the difference between classical aspiration and aspiration by search direction, and the difference between random and SE based move generation methods.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The Circuit Optimization Problem is one of *NP*-Complete class problems. Basically the problem is to optimize mixed technology circuits for performance and power. Merging different technologies into one circuit takes the advantage of both technologies. In mixed CMOS/BiCMOS circuits, the new circuit takes the advantages of the high speed and high driving capabilities of BiCMOS, and the regularity and low power consumption of CMOS.

In this thesis, we have proposed an efficient technique to achieve high performance mixed CMOS/BiCMOS circuits with very small overhead in terms of area and power. The technique consists of three phases:

1. Generate the critical paths of the input circuit. α -critical path algorithm has

been adopted for this purpose.

2. Eliminate false paths from the generated critical paths. D-algorithm based technique has been adopted to accomplish this phase.
3. Apply TS algorithm to select a set of BiCMOS gates among those covered by the sensitizable critical paths. Basic components and mechanisms have been used and two new techniques have been proposed including:
 - An effective frequency-based diversification method used in long term memory phase.
 - Simulated Evolution based move generation technique as an alternative to the random generation method.

Applying the above approach on different benchmark circuits, we came up with the following conclusions:

- The proposed solution technique produces very good results for COP in a reasonable running time, 50s-100s for small circuits, 200s-400s for medium circuits, and 500s-800s for large circuits.
- Significant reduction in circuit delay (10% to 30%) with very small capacitance overhead (0.25% to 6.2%) for mixed CMOS/BiCMOS circuits have been achieved. In addition the number of BiCMOS gates, in most of the cases, is minimal (1.0% to 5.0%) which reflects the small area overhead.

- The integration of α -critical paths algorithm, the adopted false paths detection algorithm and tabu search is a very easy, efficient and accurate process. The accuracy has been accomplished through consideration of the interconnections delays in both false path and critical path algorithms.
- The proposed technique is modular and independent of the technology type.
- TS is very effective and efficient in achieving a good or a desired solution for COP.
- The proposed diversification strategy is very efficient in getting out of local optima trap and producing effective results.
- The use of *Search by Direction* aspiration criterion produces better solutions than the classical one.
- Generally, the proposed Evolutionary Tabu Search produces local optimal solutions. However, it produces better average results compared to Classical TS. In addition, Evolutionary TS takes much longer time than Classical TS. Therefore, to obtain a certain objective within few executions regardless of the time, Evolutionary TS may be applied.

7.2 Future Work

7.2.1 Other Mixed Technologies

The proposed optimization technique presented in this thesis is independent of the technologies to be merged. We have selected CMOS/BiCMOS type of mix because of the feasibility of this process in terms of manufacturing. Other type of technologies or different templates of same technology can be mixed using this approach as long as the merge is feasible and practical.

Our future work in this field may include the following:

- Applying the same technique using actual values of CMOS and BiCMOS libraries; not the approximated values as we have done in this thesis. This will produce more accurate results which can be used for further research.
- Including power and area in the optimization process. This requires a power model for BiCMOS technology and accurate computation of CMOS and BiCMOS areas.
- Investigating the possibility of merging ECL with CMOS or ECL with BiCMOS or all using our approach. ECL logic family has the lowest propagation delay of any family and used mostly in systems which require very high speed operation. However, its power dissipation is the worst of all other logic families. Therefore, mixing this technology with others like CMOS or BiCMOS

may give a very good performance with reasonable overhead in power and area.

7.2.2 Selection of Critical Paths for Optimization

In our approach, we select the longest K sensitizable paths for optimization. This choice is right but it restricts the optimization to the boundary values within those paths. If more delay reduction is targeted, then more paths need to be considered which is costly in terms of effort and run time, especially that the value of K is randomly selected. Several techniques have been proposed to select small number of critical paths for efficient optimization as much as possible. One of these techniques is reported in [CDL93]. In this paper, the researchers proposed a technique which considers selection of sensitizable paths as well as false paths. They deduced that a long false path may need to be selected because after optimizing the long sensitizable path, a long false path may become a long sensitizable path. In addition, they show that a long sensitizable path may not need to be selected in some cases when a part of this path is shared with another long path having the same delay.

Accordingly, as a future work we may replace phase I and phase II of current approach by an efficient method of paths selection for better optimization.

7.2.3 Some Tabu Search Issues

There are some issues related to generation of moves and node selection methods that need to be addressed and discussed in the future. The first issue is related to the method of generating the neighbor solutions. The proposed method is to make one single move to produce one neighbor solution. Another way that can be applied in the future is to make more than one move at a time. However this may perturb the search too much in a way that approaching the optimal solution will take longer time.

Determination of tabu list size and candidate list size is another issue of concern. We have experimented with different sizes to obtain the best results. However, this depends on the problem size. So if we have bigger circuits, the selected sizes may not work. One possible solution that can be adopted in the future is to derive functions for tabu list size and candidate list size in terms of some parameters and attributes such as problem size, number of critical paths, and capacitance threshold.

Appendix A

A.1 Limitations of the Tools

NEW: This is a timing analyzer which generates critical paths based on α -critical algorithm. The limitations of this tool are as follows:

- It uses static data structure which does not work for large circuits.
- It does not work in the case of a circuit having output nodes used as input to some internal nodes. To overcome this limitation, new dummy nodes have to be introduced and connected to the output nodes in the circuit to serve as new output nodes.
- It reads from ASCII files having lines of width 80 characters only. To overcome that, the lines with more than 80 characters should be truncated.

FPPIND: This is a program to detect false paths in a circuit out of critical paths.

It works with any combinational circuit. For sequential circuit, the circuit has to be

converted first to combinational in which input of flip flops used as primary output and output of flip flop used as primary input.

MIXER-S & MIXER-L: These are the implementation of mixed CMOS/BiCMOS circuits optimization algorithm using tabu search short term phase and long term phase respectively. Both tools are limited to be used for CMOS/BiCMOS technology even though the algorithm is independent of the technology. This is because of the delay model and libraries of CMOS/BiCMOS technology. In addition, both tools use the format of the files generated by NEW program.

A.2 Libraries

As shown in Figure 6.6, the following libraries are used in the optimization process:

codes: this is a library of standard cells and their equivalent AHPL codes. It is shown in Table A.1.

cint: this library has been obtained from [AF95] and it consists of an estimated mean interconnection capacitance and standard deviation for different nets sizes. It is used in NEW and FPFIND for the calculation of the delay of signals. It is shown in Table A.2.

cmllib: this is a modified standard CMOS 1.0 library. It consists of standard cell characteristics including: average base delay, average load factor, input capacitance, and capacitive load crossover CX . The complete library that have has used is shown

in Table A.3.

clib: it is similar to **cmlib**, but does not have *CX* parameter. It is used only by the **NEW** program.

biclib: this is a new library constructed as explained before, that consists of BiCMOS characteristics of some cells. The complete library that has been used is shown Table A.4.

table: this table consists of controlling and non controlling values of some standard cells. It is used as look up table in **FPFIND** in the signal propagation phase.

A.3 How to Use the Tools

All the tools used in this work run under DOS and UNIX. They are available in CCSE Design Automation Laboratory in KFUPM. For the proper use of tools, the following should be considered:

- The input circuit should be in **VPNR** format or in **SLIF** format.
- Same libraries mentioned above with the same names should be used.
- The versions of **MIXER-S** and **MIXER-L** with built-in fixed data use the following data: *T_SIZE* = 4,5,6,7,8,9, *CAN_SIZE* = 10, 12, 14 ,16 ,18,20, and *I_NUM* = 2000. If different values of *T_SIZE* ,*CAN_SIZE*, and *I_NUM* need to be applied, interactive versions are used as explained below.

- Limitations of the tools have to be taken into account.
- The libraries required for a particular program should be in the same directory of that program.

The following procedure explains the use of the tools:

1. Execute PP2 to convert the VPNR circuit (for example *test*) to *testvt*. The VPNR circuit should be in the same directory of PP2. Another file is generated *test.map* which consists of the original names of nodes and their assigned numbers.
2. Execute NEW and enter the following: *testvt* and *test.map*. After some time, the program asks to enter the clock period and confidence factor. When it is done, *testvt.pth* and *testvt.grf* are generated. *testvt.pth* includes the generated critical paths. *testvt.grf* consists of fan out nodes of each node.
3. Then FPFIND is executed to get sensitizable paths file *testvt.sen*. Only the *testvt* file should be entered because the other files will be read automatically.
4. Once the three files *testvt*, *testvt.grf* and *testvt.sen*, are generated, use either MIXER-S or MIXER-L. There are six versions for each tool:
 MIXER-S-A1: For short term *CTS* with AS_1 and built-in fixed parameters.
 MIXER-S-A1-I : For short term *CTS* with AS_1 and interactive input data.
 MIXER-S-A2: For short term *CTS* with AS_2 and built-in fixed parameters.

MIXER-S-A2-I: For short term *CTS* with AS_2 and interactive input data.

MIXER-S-E: For short term *ETS* with AS_1 and built-in fixed parameters.

MIXER-S-E-I: For short term *ETS* with AS_1 and interactive input data.

MIXER-L-A1: For long term *CTS* with AS_1 and built-in fixed parameters.

MIXER-L-A1-I : For long *CTS* with AS_1 and interactive input data.

MIXER-L-A2: For long term *CTS* with AS_2 and built-in fixed parameters.

MIXER-L-A2-I: For long *CTS* with AS_2 and interactive input data.

MIXER-L-E: For long term *ETS* with AS_1 and built-in fixed parameters.

MIXER-L-E-I: For long *ETS* with AS_1 and interactive input data.

If the interactive version is used, the program will prompt to feed in all necessary data such as input files, tabu list size, etc. When the execution is completed, the output will be written to the specified output file.

AHPL Code	Standard Cell Name
4105	i1s
4444	i3s
4103	i2s
4202	ai2s
0244	ai2sf
4302	ai3s
4402	ai4s
4102	i8s
4205	oi2s
4305	oi3s
4405	oi4s
4204	exors
0344	exorf
4104	xors
4004	exnors
4106	dr2s
4009	dsr2s
4006	dsr2s
4201	a2s
4301	a3s
4401	a4s
4203	o2s
4303	o3s
4403	o4s
4422	oai22s
4522	oai221s
4622	oai2211s
4431	oai31s
4532	oai32s
4633	oai33s
4321	oai21s
4421	oai211s
4521	oai2111s
5521	soi2111s
5522	soi221s
5422	soi22s
5622	soi222s
5621	soi2211s
5321	soi21s
5532	soi32s
5633	soi33s
5421	soi211s

Table A.1: Names of standard cells and their equivalent AHPL codes.

Net Size	Mean Capacitance (pF)	Standard Deviation
2	0.0496613614	0.0381922414
3	0.0642885151	0.0269860004
4	0.1045077916	0.0474687684
5	0.1239954827	0.0564270349
6	0.1325196250	0.0574357545
7	0.1486822222	0.0440815179
8	0.261911	0.0319537001
9	0.240657	0.0
10	0.2624445	0.0377625
11	0.267	0.04
12	0.27	0.044
13	0.0001	0.0001
15	0.00001	0.00001
16	0.3	0.05
17	0.35	0.05
18	0.001	0.001
19	0.001	0.001
20	0.35	.05
21	0.35	.05
22	0.35	0.05
23	0.35	0.05
25	0.35	0.05

Table A.2: The library of estimated interconnection delays [AF95].

Gate	BD	LF	IC	CX
4105	0.315	4.525	0.255	0.105
4444	0.315	0.515	4.755	0.917
0244	0.00315	0.00605	0.04755	0.781
0344	0.00315	0.00605	0.04755	0.781
4103	0.32	2.05	0.455	0.234
4202	0.585	3.87	0.340	0.226
4302	0.88	3.715	0.415	0.355
4402	1.35	3.805	0.435	0.532
4102	0.275	0.579	1.670	0.751
4205	0.76	4.105	0.435	0.278
4204	1.7	3.735	0.635	0.683
4104	1.7	3.735	0.635	0.683
4305	1.25	5.02	0.455	0.371
4405	1.4	6.265	0.425	0.335
4106	3.15	6.47	0.20	0.730
4006	3.055	4.6	0.235	0.996
4009	3.055	4.6	0.235	0.996
4201	1.33	3.985	0.380	0.500
4301	1.8	4.40	0.340	0.615
4401	1.85	4.58	0.380	0.606
4203	1.35	4.73	0.35	0.428
4303	1.759	3.825	0.405	0.700
4403	2.45	4.785	0.395	0.768
4018	0.0	1.0	0.005	0.0
4019	0.0	1.0	0.005	0.0
4020	0.0	1.0	0.005	0.0
4050	0.88	3.10	0.470	0.425
4004	0.845	4.95	0.640	0.256
4422	1.3	4.125	0.445	0.473
4522	2.0	4.475	0.465	0.670
4431	1.6	4.95	0.505	0.485
4532	1.55	4.795	0.520	0.488
4633	1.7	4.815	0.530	0.353
4321	0.985	4.130	0.445	0.358
4421	1.465	3.755	0.540	0.585
5521	2.05	6.56	0.475	0.469
5221	1.15	4.03	0.475	0.428
5422	2.35	5.26	0.475	0.670
5621	2.35	6.15	0.0505	0.573
5321	0.97	3.86	0.495	0.377
5532	1.80	4.455	0.475	0.606
5633	1.75	4.30	0.510	0.610
5421	1.45	5.02	0.485	0.433

Table A.3: CMOS library.

Gate	BD	LF	IC
41050	0.63	1.505	0.357
44440	0.63	0.172	6.657
02440	0.0063	0.00200	0.06657
03440	0.0063	0.00200	0.06657
41030	0.64	0.6833	0.665
42020	1.170	1.290	0.476
43020	1.76	1.2380	0.581
44020	2.7	1.268	0.609
41020	0.55	0.183	2.415
42050	1.52	1.368	0.602
42040	3.4	1.245	0.889
41040	3.4	1.245	0.889
43050	2.5	1.673	0.637
44050	2.8	2.088	0.595
41060	6.3	2.157	0.280
40060	6.110	1.533	0.329
40090	6.110	1.533	0.329
42010	2.66	1.328	0.532
43010	3.6	1.467	0.476
44010	3.7	1.527	0.532
42030	2.7	1.577	0.490
43030	3.59	1.275	0.567
44030	4.90	1.595	0.553
40180	0.0	0.333	0.007
40190	0.0	0.333	0.007
40200	0.0	0.333	0.007
40500	1.76	1.033	0.495
40040	1.69	1.650	0.165
44220	2.6	1.375	0.470
45220	4.0	1.492	0.490
44310	3.2	1.650	0.530
45320	3.1	1.598	0.535
46330	3.4	1.605	0.555
43210	1.97	1.377	0.470
44210	2.99	2.503	0.565
55210	4.1	2.187	0.500
52210	2.3	1.343	0.480
54220	4.7	1.753	0.500
56210	4.7	2.050	0.075
53210	1.94	1.287	0.520
55320	3.6	1.485	0.500
56330	3.5	1.430	0.535
54210	2.9	1.673	0.510

Table A.4: BiCMOS library.

Bibliography

- [AF95] Khalid Al-Farrah. Timing driven floorplanning. *MS Thesis, KFUPM*, June 1995.
- [Ali94] Shahid Ali. Scheduling and allocation in high-level synthesis using genetic algorithm. *MS Thesis, KFUPM*, June 1994.
- [Baa91] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis*. Addison Wesley, 1991.
- [BAB94] A. R. Baba-Ali and A. Bellaouar. An optimization tool for mixed CMOS/BiCMOS standard cells circuits. *Arabian Journal of Science and Engineering*, 19:4B:883–888, October 1994.
- [BI86] D. Brand and V. Iyengar. Timing analysis using functional relationships. *Proceedings of ICCAD-86*, pages 126–129, 1986.
- [BMCM90] J. Benkoski, E. Meersch, L. Claesen, and H. De Man. Timing verification using statically sensitizable paths. *IEEE Transaction on Computer-*

Aided Design, 9(10):1073–1083, October 1990.

- [BT94] R. Battiti and G. Tecchiolli. Simulated annealing and tabu search in the long run: a comparison on QAP tasks. *Computers Math. Applic.*, 28(6):1–8, 1994.
- [CD93] H. Chen and D. Du. Path sensitization in critical path problem. *IEEE Trans. on Computer-Aided Design of Integrated Cir. and Sys.*, 12(2):196–207, February 1993.
- [CDL93] H. Chen, D. Du, and L. Liu. Critical path selection for performance optimization. *IEEE Transaction on Computer-Aided Design*, 12(2):185–195, February 1993.
- [DYG89] D. Du, H. Yen, and S. Ghanta. On the general false path problem in timing analysis. *Proceedings of 26th Design Automation Conference*, pages 555–560, 1989.
- [EBE93] S. Embabi, A. Bellaouar, and M. Elmasry. *Digital BiCMOS Integrated Circuit Design*. Kluwer Academic Publishers, 1993.
- [Glo90] Fred Glover. Tabu search: A tutorial. *Technical Report, University of Colorado*, November 1990.
- [Hd87] A. Hertz and D. deWerra. Using tabu search techniques for graph coloring. *Computing*, 29:345–351, 1987.

- [HG94] Roland Hubscher and Fred Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *Computers Ops Res*, 21(8):877–884, 1994.
- [HPS93] S. Huang, T. Parng, and J. Shyu. A polynomial-time heuristic approach to approximate a solution to the false path problem. *30th ACM/IEEE Design Automation Conference*, pages 118–122, 1993.
- [HS90] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1990.
- [KB87] R. Kling and P. Banerjee. ESP: A new standard cell placement package using simulated evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.
- [KLG94] James P. Kelly, Manuel Laguna, and Fred Glover. A study of diversification strategies for the quadratic assignment problem. *Computer Ops Res*, 21(8):885–893, 1994.
- [LC91] Andrew Lim and Yeow-Meng Chee. Graph partitioning using tabu search. *1991 IEEE International Symposium on Circuits and Systems*, pages 1164–1167, 1991.
- [LG93a] Manuel Laguna and Fred Glover. Bandwidth packing: A tabu search approach. *Management Science*, 39(4):492–400, April 1993.

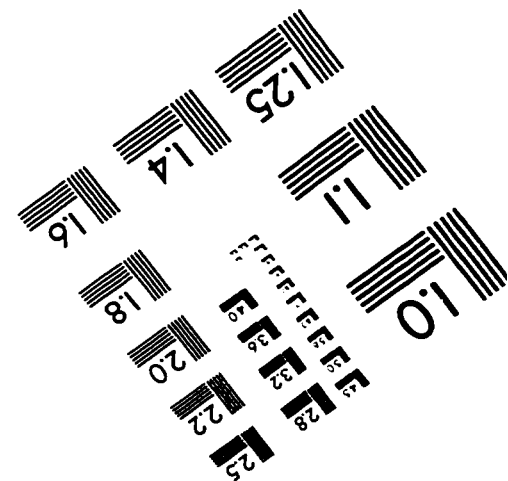
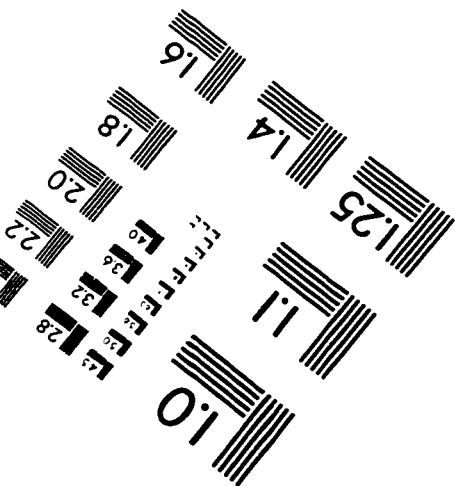
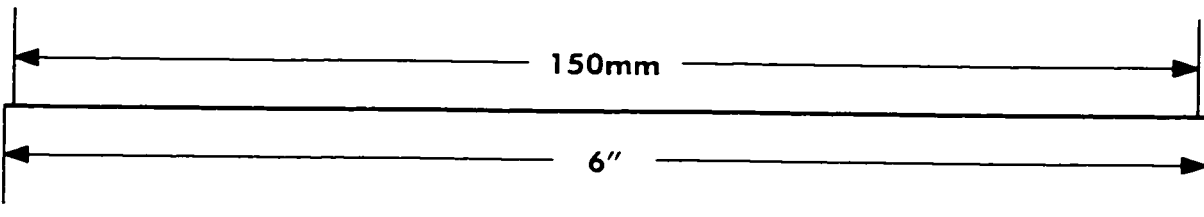
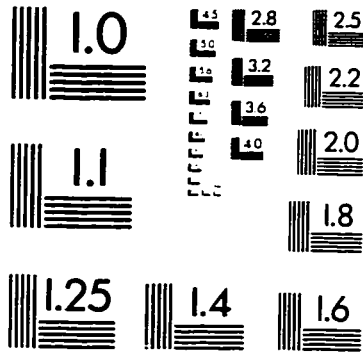
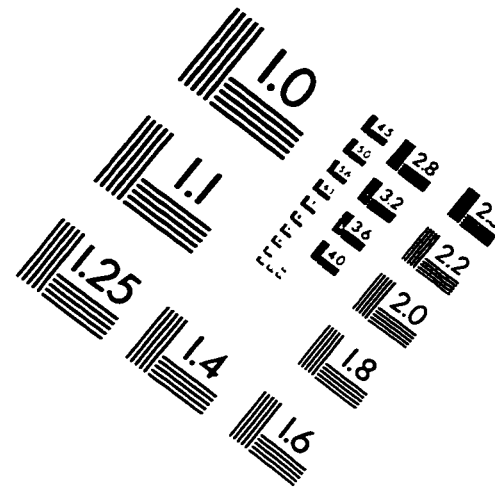
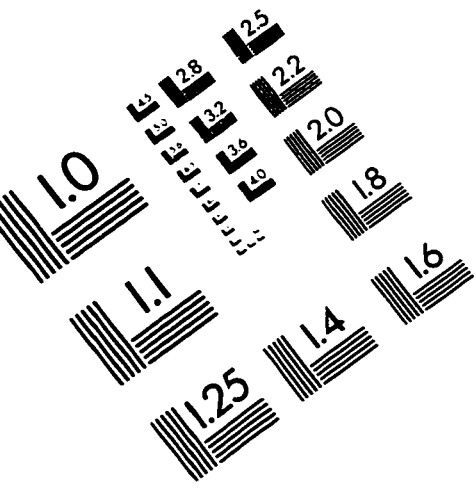
- [LG93b] Manuel Laguna and Fred Glover. Integrating target analysis and tabu search for improved scheduling systems. *Expert Systems with Application*, 6:287–297, 1993.
- [LMSK90] S. Lin, M. Marek-Sadowska, and E. Kuh. Delay and area optimization in standard-cell design. *Proc. 27th Design Automation Conference*, pages 349–352, 1990.
- [MB89] P. McGeer and R. Brayton. Efficient algorithm for computing the longest viable path in a combinational network. *26th ACM/IEEE Design Automation Conference*, pages 561–573, 1989.
- [PCM89] S. Perremans, L. Claesen, and H. De ManV. Static timing analysis of dynamically sensitizable paths. *26th ACM/IEEE Design Automation Conference*, pages 568–573, 1989.
- [Ree95] C. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill Book Co., Europe, 1995.
- [Rot66] J. P. Roth. Diagnosis of automata failures: A calculus and a new method. *IBM J. Res. Develo.*, pages 278–281, October 1966.
- [SV92] L. Song and A. Vannelli. VLSI placement using tabu search. *Microelectronics Journal*, 17(5):437–445, 1992.

- [SY95] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice (in press)*. McGraw-Hill Book Co., Europe, 1995.
- [SY98] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms and Their Applications in Engineering*. IEEE Computer Society Press, 1998.
- [YDG89] S. Yen, D. Du, and S. Ghanta. Efficient algorithms for extracting the k most critical paths in timing analysis. *Proceedings of 26th Design Automation Conference*, pages 649–654, 1989.
- [You90] Habib Youssef. Timing analysis of cell based VLSI design. *PH.D. Thesis, Minnesota State University*, January 1990.

Vita

- Born in 1969 in Syria.
- Graduated in the field of Computer Engineering from KFUPM in 1992 with high honour.
- Enrolled for MS program in 1992 at KFUPM.
- Student member of IEEE Saudi Arabia from 1992 - 1993.
- Worked as a customer engineer at International Turnkey Systems from 1993 - 1996
- Working as a sales support engineer in GE Information Services since 1996.

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