Tabu Search Based Circuit Optimization

Sadiq M. Sait and Munir M. Zahra

Department of Computer Engineering
King Fahd University of Petroleum & Minerals
KFUPM Box 673, Dhahran-31261, Saudi Arabia
Tel: 966(3)-860-2110/2217 *e*-mail: sadiq@ccse.kfupm.edu.sa

Abstract

In this paper we address the problem of optimizing mixed CMOS/BiCMOS circuits. The problem, formulated as a constrained combinatorial optimization problem is addressed using a tabu search algorithm. Initially a random approach is adopted for selecting among available solutions. Further, as an alternative competing solution the concepts of simulated evolution are applied to classical tabu search. This allows for a stochastic criterion for selecting among available solutions as compared to the random approach of classical tabu search. Only gates on the critical sensitizable paths are considered for optimization. Such a strategy leads to sizeable circuit speed improvement with minimum increase in the overall circuit capacitance. Compared to earlier approaches, the presented techniques produce circuits with remarkable increase in speed (greater than 20%) for very small increase in overall circuit capacitance (less than 3%).

Keywords: Tabu Search, Simulated Evolution, Circuit Optimization, Search Algorithms, CMOS/BiCMOS, Mixed Technologies, Critical Path, False Path.

1 Introduction

Popularity of CMOS technology is due to its low DC power dissipation and high package density. The demand for superior performance motivated research and development that lead to the emergence of BiCMOS technology. BiCMOS is a combination of CMOS and Bipolar technologies, with advantages of both, high speed and high driving capabilities of Bipolar, as well as the low area and low power consumption of CMOS.

VLSI designs are evaluated with respect to three main performance criteria: speed, area, and power consumption. As these criteria are conflicting designers usually seek to optimize one criteria, namely speed, while satisfying specific constraints/requirements on area and power consumption.

One of the optimization techniques applied at the circuit level is the selection of logic blocks of the VLSI circuit, in terms of speed and area. For example, for standard cell designs, the optimization can be performed through a careful selection of different implementations of a block in the same technology. These alternative implementations vary in area, driving capabilities, intrinsic delay, and capacitive loading [15]. Another optimization strategy is to follow a mixed technology design approach. One possible choice is to mix CMOS/BiCMOS technologies. In terms of manufacturing process, this is feasible since the CMOS process is part of BiCMOS process. The CMOS-based BiCMOS process is a CMOS baseline process to which bipolar transistors are added. So, for a mixed design circuit, initially all cells are exposed to CMOS process. Then bipolar transistors are added to only those cells that are selected to be BiCMOS.

In this paper we discuss the problem of optimizing 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 basic idea is as follows. Given a circuit consisting of only CMOS cells, 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.

In [3], the above approach for optimizing standard cells circuits is used. The technique aims at improving circuit performance by making for each gate, a choice between CMOS or BiCMOS cells depending only on their load capacitance. The authors reported noticeable speed improvement on all the test circuits used. However, in their implementation, no constraints on power dissipation were placed, and the number of BiCMOS gates was high. The reported approach suffers from several problems, namely:

1. All the nodes are considered for optimization.
2. Output nodes are replaced whether they are on time critical paths or not.
3. The 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 get trapped at a local optimum solution.

The actual delay of a circuit is determined by the delay of its longest *sensitizable path*. A sensitizable path is a path which can be activated by at least one input vector. Those paths which cannot be activated by any input vector are called *false paths*. A path is *critical* if its total delay is greater than a threshold value. Thus, the problem of finding and estimating the delay of critical paths is called the *critical path problem* [7]. For static timing analysis techniques, the circuit is modeled as a directed acyclic graph in which three popular algorithms are used to trace the paths: Depth First Search (DFS) with/without pruning, Breadth First Search (BFS), and PERT-like trace.

In this work we enumerate all sensitizable critical paths according to the α-critical concept. First, all paths with average delay (including estimation of interconnect delays) exceeding an estimated threshold value are enumerated. From amongst these paths, only sensitizable paths are reported.

In the following section we discuss the α-critical approach. Discussion on false path problem is given in Section 3. The Circuit Optimization Problem (COP) is formulated as a combinatorial optimization problem in Section 4. Details of application using tabu search are given in Section 5. tabu search for the COP are presented in Section 6. Experimental results are provided in Section 7.

2 The 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 [1]. The $\alpha - critical$ algorithm aims at predicting 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 [1] some definitions and equations proposed to compute the path delay.

Let $\pi = \{v_1, v_2, \cdots, v_p\}$ 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,

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

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,

$$CD_{v_i} = BD_{v_i} + LF_{v_i} \times AcL_{v_i} \tag{2}$$

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,

$$ID_{v_i} = LF_{v_i} \times C_{v_i} \tag{3}$$

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, $\ldots$,m-pin) are collected from past designs of similar complexity[1]. 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 $\overline{ID_{v_i}}$ of net v_i and its corresponding variance $S^2_{v_i}$ are estimated as follows:

$$\overline{ID_{v_i}} = LF_{v_i} \times \overline{C_{v_i}} \;\; ; \;\; S^2_{v_i} = LF^2_{v_i} \times s^2_{v_i} \tag{4}$$

[1]this classification helps reduce the sample variance around the mean

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

$$T_\pi = \sum_{i=1}^{p-1}(CD_{v_i} + \overline{ID_{v_i}}) \quad ; \quad S_\pi^2 = \sum_{i=1}^{p-1} S_{v_i}^2 \tag{5}$$

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

$$T_{\max} = \max_{\pi \in \Pi}(T_\pi) \tag{6}$$

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

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:

$$T_\Pi + \alpha \times \sqrt{S_\Pi^2} \geq T_{max} \tag{7}$$

For a user specified α, the α-critical approach enumerates all paths which satisfy Equation 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 [1].

3 False Path Problem

The presence of false paths has many undesirable effects which include loss of accuracy and waste of optimization effort. False paths exist in a circuit because of several reasons, namely:

1. *Incompatible transitions:* A false path results from the combination of incompatible transitions.
2. *Incorrect signal flow:* Timing verifiers that operate at switch level encounter this problem. Due to the bidirectional nature of MOS transistors, the intended signal flow in structure such as barrel shifter is not always obvious [4].
3. *Logic dependency:* The most explicit source of false paths comes from some logic that depends on the output of other logic [17].

In recent years many techniques have been proposed to detect false paths. The reported techniques rely on various path sensitization criteria which fall into three types: *Static, Dynamic* and *Viable.*

Static Sensitization: These techniques are based on the D-Algorithm which is widely used in testing. One important assumption of the D-Algorithm is that, except for the signal to be propagated, all other signals in the circuit are assumed to have static values throughout the propagation process. This assumption, however, is not always true which may lead to incorrect results. First it may report some paths as false paths while in reality they are not. Second, it can underestimate the sensitizable path length [5, 4, 20].

Dynamic Sensitization: This approach is also based on D-algorithm, but it takes into consideration the stability requirement of the signals. The false paths reported in [8, 17, 12, 6] follow this approach.

Viability: The flaw in the dynamic sensitization condition is the absence of perfect knowledge on gate delays. 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 is, the above mentioned methods use criteria which are non-robust.

McGeer and Brayton [16] developed a technique that computes the longest viable path in combinational circuits. Their techniques are based on two conditions: correctness and robustness. These two requirements are derived from the idea of Boolean difference. The 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 found equal to logic 0, the path is claimed to be a false path.

4 Optimization of BiCMOS/CMOS VLSI Designs

The problem of optimizing a mixed technology design can be formulated as an optimization problem and solved using a variety of algorithms. For a mixed CMOS/BiCMOS design, there exist 2^m solutions for a circuit with m gates. Hence, full/brute force design space exploration is infeasible even for designs of moderate sizes.

4.1 Problem Definition

Given a netlist of CMOS gates, the objective is to find an optimal or near-optimal solution to the problem of replacing some of the CMOS gates with BiCMOS gates such that the overall delay is minimized with minimum increase in the power and area of the circuit. Since the delay of a circuit depends only on the longest sensitizable paths, only gates belonging to those paths are considered in the search space. The reader should note that changing the implementation of a gate instance from CMOS to BiCMOS does not always result in delay improvement. The BiCMOS gate has less delay than the corresponding CMOS gate if its fanout load CL is greater than a certain threshold CX [3]. Therefore only those nodes with $CL > CX$ should be considered in the search. 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= $(g_1, g_2, \ldots, g_n)$
- Vector $\mathcal{D}$ =$(\Delta D_1, \Delta D_2, \ldots, \Delta D_n)$

  $$\Delta D_i = D^c_{g_i} - D^b_{g_i}$$

  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.

  $D^c_{g_i}$ and $D^b_{g_i}$ is the circuit delay when gate g_i is implemented in CMOS and BiCMOS respectively.

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

  $$\Delta C_i = C^b_{g_i} - C^c_{g_i}$$

  where:

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

  $C^c_{g_i}$ and $C^b_{g_i}$ 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 [9].

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,

$$\begin{cases} \text{maximize} & \sum_{i \in S} \Delta D_i \\ \text{subject to} & \sum_{i \in S} \Delta C_i \leq C_T \end{cases} \tag{8}$$

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:

$$D_F = T_{max} - \Delta D \tag{9}$$

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

$$C_F = C_I - \Delta C \tag{10}$$

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

4.2 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 [11]. 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 [11].

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 1. As can be seen from the figure, there is no rule for guessing a solution that might lead to 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**
        $X_i \leftarrow$ **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
end COP

Figure 1: Nondeterministic COP algorithm.

4.3 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: [2]

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 [11]. 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:

$$\sum_{i\in I} p_i \tag{11}$$

such that

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

Here the solution is represented by the subset $I \subseteq \{1,\ldots,n\}$. 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:

$$T(C) = C_T \tag{12}$$

$$T(s_1, s_2, \ldots, s_n) = \Delta C_1, \Delta C_2, \ldots, \Delta C_n \tag{13}$$

$$T(p_1, p_2 \ldots p_n) = \Delta D_1, \Delta D_2, \ldots \Delta D_n \tag{14}$$

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.

4.4 Proposed Solution

To identify the gates of set S, we proceed in three steps:

1. Generation of the α-critical paths of the input circuit.
2. Eliminate the false paths.
3. Apply TS algorithm to select the gates of subset S among those covered by the sensitizable critical paths.

Phase I: This phase enumerates all critical paths according to the α-criticality described in Section 2. Path enumeration is achieved via a PERT-like trace of the circuit graph [23].

Phase II: After generating all critical paths for a given circuit, these paths are checked via false path checking procedure to extract the false paths. This step is necessary to speed up the optimization process by minimizing the input population as well as it gives accurate timing. As mentioned earlier, many techniques have been proposed for this purpose. We use the algorithm of [8, 22] with few modifications. The first modification is the use of a more accurate delay model [23]. Other modifications are related to the way of handling the generation of new events in the events propagation phase.

Phase III: This is the optimization phase. The input of this step is a set of the longest sensitizable paths of the given circuit. The process aims to replace some of the CMOS gates by BiCMOS gates in order to optimize the circuit for delay without exceeding a capacitance threshold constraint. The output is a mixed CMOS/BiCMOS circuit with optimum cost. This optimization step is achieved using Tabu Search heuristic.

Tabu Search is a *metaheuristic* which can be used as an independent search technique or as a higher level heuristic procedure for solving combinatorial optimization problems. It is 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 called *tabu list*. After a specified duration they are removed from the list and are free to be inserted again.

For a variety of problems, TS has found solutions superior to the best solution previously obtained by alternative methods. In other cases, it has demonstrated advantages such as ease of implementation, or the ability to handle additional considerations such as constraints not encompassed by an original problem formulation. [10]

5 Tabu Search (TS) Algorithm

Tabu search is an iterative procedure that works by making moves from one trial solution to another. An algorithmic description of a simple implementation of the TS is given in Figure 2.

The TS procedure starts from an initial feasible solution s (current solution) in the search space Ω. A neighborhood $\aleph(s)$ is defined for each s. A sample of neighbor solutions $\mathbf{V}^* \subset \aleph(s)$ is generated called *trial* solutions ($n = |\mathbf{V}^*| \ll |\aleph(s)|$), and comprises what is known as the candidate list. From this generated set of trial solutions, the best solution, say $s^* \in \mathbf{V}^*$ is chosen for consideration as the next solution. The move to s^* is considered even if s^* is worse than s, that is, even if $cost(s^*) > cost(s)$.

In order to prevent returning to previously visited solutions a memory or list $\mathbf{T}$, known as *tabu list*, is maintained. Whenever a move is accepted, its attributes are introduced into the tabu list. The purpose is to prevent the reversal of moves for the next $k = |\mathbf{T}|$ iterations because they *might* lead back to a previously visited solution.

In certain situations, it is necessary to overrule the tabu status. This is done with the help of the notion of *aspiration criterion*. Aspiration criterion overrides the tabu status of moves whenever appropriate. One aspiration criterion, also known as *best solution* aspiration criterion (AS_1) overrides the tabu restriction if the move produces a new best solution. In this work we used this and another aspiration criterion, called *aspiration by search direction* (AS_2) [18]. In aspiration by search direction, if an improving move e is made, then the reverse move $\overline{e}$ is accepted if it also is an improving move.

Initial solution: The initial solution can be any feasible solution. It is found that TS may take longer if given a poor initial solution. In our case, the initial solution is a set A of nodes of type CMOS only, which are covered by the sensitizable α-critical paths. As search proceeds, neighbor solutions are generated by swapping a CMOS gate with a BiCMOS gate or vice versa. The selection of the gate for swap can be done randomly. This approach is referred to as *Classical Tabu Search*. Another strategy is to select the gate based on its characteristics and attributes. Here we employed the *Simulated Evolution(SE)* heuristic to evaluate the goodness(fitness) of each gate in the candidate list. This approach is called *Evolutionary Tabu Search*. It is detailed in the next section.

Tabu list: Formulation of the tabu list is one of the main steps in TS. Since we have only one type of move we used one tabu list. Each entry in the tabu list contains the following information:

- gate number in the path,
- gate type, CMOS or BiCMOS,
- cost associated with this move,
- frequency of the move, and,
- a gain bit (0 or 1): this field is used if *aspiration by search direction* is used as discussed later.

Evaluation function: In order to select the best solution among several candidate solutions generated in each iteration we have to evaluate each solution. The evaluation function is formulated to incorporate all the parameters to be optimized. As mentioned earlier, our objective is to maximize $\sum_{i \in S} g_i$, that minimizes $T_{\max}$, subject to capacitance constraints. The evaluation function takes the following form,

$$E(s) = \sum_{i \in S} g_i - X \tag{15}$$

Where X is a penalty that is included if capacitance constraints are violated. The above evaluation function is suitable for short-term tabu search. TS based on long-term memory requires historical information. In the next section we will show how a record of history of the moves is used to diversify search to improve results [21].

5.1 Diversification and Long Term Memory

In many combinatorial optimization problems, application of short-term memory alone may not produce a good solution. In [10], it is shown that the long-term memory functions can be very important for obtaining best results. In our work, we apply frequency-based diversification. In this strategy, we keep track of the number of times a certain move has been made. At the point when diversification is to be made, we penalize those moves that have been most frequent, thereby taking the search to areas unvisited thus far [10, 13, 14]. Therefore, the following modifications to short-term TS are done.

1. When the short-term TS algorithm hits a local optima, the following actions are taken:
   (a) All BiCMOS nodes are swapped to CMOS. Denote the number of those nodes as NUM_OF_BiCMOS

(b) Search for least frequency nodes and replace them by BiCMOS. The search replacement process continues till the objective load threshold is reached or till the number of replaced nodes is equal to NUM_of_BiCMOS. Some of the BiCMOS nodes that were changed to CMOS may be changed again back to BiCMOS. Using the above, the search process is transferred to another region where the process might lead to a better solution.

2. Re-start the short-term memory component again and continue until another local optimum is reached; then repeat Step 1.

6 Evolutionary Tabu Search

As mentioned in the previous section moves can be generated by selecting gates based on their attributes. This approach is based on the evolutionary aspects of the *Simulated Evolution* heuristic. Before explaining how this can be done, an overview of SE is presented.

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 [19]. The main idea of SE is that 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 [21].

The algorithm starts with the initialization phase where various parameters are set to desired values. 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. Each step is explained briefly below [21].

Evaluation: In this step the goodness of each element e_i in the population P is evaluated. *Goodness* is defined as:

$$g_i = \frac{O_i}{C_i}$$

where O_i is the 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, O_i does not change from generation to generation and is therefore it is computed only once while C_i has to be recomputed at each call to the evaluation step. The goodness measure must be strongly related to the targer 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 of the element, the more likely that it will not be selected and hence 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 a problem-specific allocation function F_A. The allocation function may be a non-deterministic function which involves a choice among a number of alternative moves for each element.The order and type of of alteration of elements is 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.[21].

6.2 Evolutionary Tabu Search

In our approach to applying TS to COP, we used the two functions *Evaluation* and *Selection* of SE as an alternative stochastic method for generating moves. Each function is formulated for COP as follows:

Evaluation: Let $A = (g_1, g_2, \ldots, g_n)$ where each g_i satisfies the fanout load constraint $CL > CX$. For each g_i in A, we compute ΔD_i which is gain delay due to changing gate i implementation from CMOS to BiCMOS, that is

$$\Delta D_i = D^c_{g_i} - D^b_{g_i}$$

This computation is done only once. In this case

$$O_i = \Delta D_i \tag{16}$$

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

$$C_i = T'_i - T \tag{17}$$

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 relate 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:

$$G_i = 1 - 0.5(1 + \frac{C_i}{O_i}) \tag{18}$$

For example, if swapping gate i frm 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**
  $C_i = T'_i - T$
  $G_i = 1 - 0.5(1 + \frac{C_i}{O_i})$
END FOR EACH

Selection: After computing the goodness of all gains 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 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-selcted as a mechanism to escape from the trap of local optima, the bias B is used to maximize the gate's 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 iteration 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. Comparision of these two techniques is carried out in the next section.

7 Results & Discussion

The approach described in this paper has been tested on several ISCAS-85 benchmark circuits. For the nine ISCAS-85 circuits used, the percentage of false paths reported ranged from 0-24%. Note that these are false paths from among the critical paths. In all but one case the maximum delay of the circuit did not change due to removal of false paths, but the number of gates on the sensitizable paths was reduced. For Classical TS, experiments with short term and long term components were conducted to observe the behavior of TS in both cases. In addition, two aspiration criteria (AS_1 and AS_2) were experimented with. For SE based Evolutionary TS, experiments with long term components were conducted and the results compared with Classical TS.

For all experiments, we used capacitance threshold to be 10% of total capacitance, the required reduction in the delay to be 30%. The tabu list size is an important parameter in TS. If the size is too small, the search will start cycling, and if it is too large, the search will be too restrictive. We experimented with several list sizes, ranging from 4 to 12. List sizes between 5 and 7 achieved best results in nearly all the benchmarks used. Another parameter affecting the search process is the candidate list size. We experimented with several list sizes, ranging from 10 to 20 and found that the list sizes between 16 to 20 achieved best results.

All tools are developed in C and run on SUN SPARC workstations.

Table 1 memory component with customary aspiration criterion, AS_1. The number of BiCMOS gates required to speed up circuits using the presented approach is a very small percentage of the total number of gates. For medium to large sized circuits (greater than 200 gates) this number is less than 5%. For most circuits tested, a percentage delay reduction between 15% and 29.4% has been achieved with capacitance increase of less than 7%.

Table 2 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

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 1: Best results of long term memory of TS with AS_1.

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 2: Best results of long term memory of TS with AS_2.

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 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 $\overline{e}$ 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 TS with AS_1 because the TS algorithm is nondeterministic, hence several runs have to be conducted to get the best result.

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	30.0	43.405	2.675	6.16

Table 3: Best results of long term memory of Evolutionary TS.

Table 3 summarizes the best results obtained by applying long term ETS. Comparing these results with that of CTS (Table 1), it is seen 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, c449, c6288 and highway while both strategies produced the same results in case of c3540. This reflects the fact that ETS could generate better results as it selects and replaces only those gates which have a 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 for this kind of behaviour.

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

$$Random \leq Min(1, 1 - G_i + B) \tag{19}$$

If the goodness G_i is low, the probability of selecting the node i will be high. According to our goodness function, if a CMOS gate with low goodness is swapped, its goodness would increase. 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 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 use the bias B only in case a gate is BiCMOS so as to increase its selection probability even if its goodness is high. We experimented with different values of B from 0.1 to 0.7 with interval of 0.1. Although some good results are achieved with the values 0.5 - 0.7 as shown in the table, the use of bias B only is still not enough to avoid the trap of local optimality. Therefore, we have to modify our goodness function 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, Table 4 shows the average delay reduction for benchmark circuits. For each circuit, both CTS and ETS have been executed for 2000 iterations for different values of Tabu list size and candidate list size. Then the average value has been computed for each. It is obvious that ETS produces better solutions (i.e., more delay reduction) than CTS on the average. Hence, to achieve a certain objective solution, it is better to apply ETS for a few trials. However, ETS take 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 for each iteration in ETS is an expensive step. Second, each iteration in ETS takes much more time than the time spent in each iteration in CTS. Let us compare the move generation step in both approaches to clarify the point.

In CTS, in each iteration N moves are generated by selecting only N nodes randomly and then the necessary processing and evaluation is done. In ETS, in order to generate N moves, we randomly select a node and check if its goodness satisfies the inequality [refrand] or not. If the inequality is satisfied, then the node is accepted as one of the 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 CTS.

Tabu Search Behavior

	Classical TS		Evolutionary TS	
Circuit Name	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 4: Comparison between Classical TS and Evolutionary TS in terms of quality nd performance.

To show the behavior of short term memory and long term memory Classical 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 tabu list size = 5 and candidate list size = 14. The plot of these results is shown in Figure 3.

Some of the values of solution cost 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.

From Figure 3, 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 also see that when an illegal move is made, the solution cost is penalized which hardly improves again. The solution of this trap is to diverse the search. As it is clear from the figure 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 the 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 also got improved.

To compare the behavior of *CTS* versus *ETS*, both algorithms were applied on the same benchmark circuit c499 for identical tabu list size = 5 and candidate list size = 14. The data of current cost and best cost for 2000 iterations have been collected and plotted in figure 4 and figure 5 respectively. As seen in figure 4, *ETS* finds good solutions (here, good delay reduction is around 7 ns.) quickly in a few iterations (around 200) because it examines small sets 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 have high goodness, *ETS* will swap some of these gates producing worse solutions than before. Therefore it will get trapped at local optimum solution. On the other hand *CTS* requires more number of iterations to find good solutions because it examines all gates in the critical paths. However, after finding a good solution (delay reduction around 6ns in this case), it continues in generating good solutions and even better solutions than those found so far. This is clear from the figure where *ETS* produced good solutions with the following reduction in the delay: around 5ns after 500 iterations, around 6ns after 700 iterations, around 5 ns after 1200 iterations and around 7.5 ns after 1800 iterations. This reflects the fact that *CTS* is capable of escaping from the local optimum trap. Figure 5 gives more clearer idea about the quality of both *CTS ETS*. Obviously, *ETS* jumps quickly to a good solution and gets stuck there while *CTS* gradually produces better solutions as it runs for more time.

Generally the proposed Evolutionary Tabu Search produces local optimal solutions. However,it produces better average results as compared to Classical Tabu Search. Also, *ETS* takes much longer than *CTS* and hence to obtain a certain objective within a few executions regardless of the time, it may be applied.

In this work, the candidate list of neighboring solutions is built using both random generation strategy and the concept of Simulated Evolution. Future work may include power and area into the optimization process. This would require a power model for BiCMOS technology and accurate computation of CMOS and BiCMOS. Also the possibility of merging ECL with CMOS or ECL with BiCMOS may be considered.

Acknowledgment

The authors acknowledge the King Fahd University of Petroleum & Minerals, Dhahran, for support. Also, the assistance rendered by Mohammad Faheemuddin is appreciated.

References

[1] Khalid Al-Farrah. Timing driven floorplanning. *MS Thesis, KFUPM*, June 1995.

[2] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis.* Addison Wesley, 1991.

[3] 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.

[4] 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.

[5] D. Brand and V. Iyengar. Timing analysis using functional relationships. *Proceedings of ICCAD-86*, pages 126–129, 1986.

[6] 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.

[7] H. Chen, D. Du, and L. Liu. Critical path selection for performance optimization. *IEEE Transactions on Computer-Aided Design*, 12(2):185–195, February 1993.

[8] 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.

[9] S. Embabi, A. Bellaouar, and M. Elmasry. *Digital BiCMOS Integrated Circuit Design.* Kluwer Academic Publishers, 1993.

[10] Fred Glover. Tabu search: A tutorial. *Technical Report, University of Colorado*, November 1990.

[11] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press, 1990.

[12] 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.

[13] Roland Hubscher and Fred Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *Computers Ops Res*, 21(8):877–884, 1994.

[14] Manuel Laguna and Fred Glover. Bandwidth packing: A tabu search approach. *Management Science*, 39(4):492–400, April 1993.

[15] 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.

[16] 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.

[17] 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.

[18] C. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems.* Mc-Graw Hill Book Co., Europe, 1995.

[19] R.Kling and P.Banerjee. Esp:a new standard cell placement package using simulated evolution. *Proceedings of the international Design Automation Conference*, pages 60 – 66, 1987.

[20] J. P. Roth. Diagnosis of automata failures: A calculus and a new method. *IBM J. Res. Develo.*, pages 278–281, October 1966.

[21] Sadiq M. Sait and H. Youssef. Iterative Computer Algorithms and Their Applications in Engineering. *IEEE CS Press, to appear*, 1998.

[22] S. Yen et al. Efficient algorithms for extracting the K-most critical paths in timing analysis. *Proceedings of 26th Design Automation Conference*, pages 649–654, 1989.

[23] H. Youssef, Sadiq M. Sait, and Khalid Al-Farrah. Timing influenced force-directed floorplanning. *European Design Automation Conference with Euro-VHDL* **Euro-DAC'95**, *Brighton*, pages 156–161, September 1995.

Algorithm: Short-Term Tabu Search

Ω	:	Set of feasible solutions.
s	:	Current solution.
s^*	:	Best admissible solution.
c	:	Objective function.
$\aleph(s)$	:	Neighborhood of $s \in \Omega$.
$\mathbf{V}^*$	:	Sample of neighborhood solutions.
$\mathbf{T}$	:	Tabu list.
$\mathbf{AL}$	:	Aspiration Level.

Begin .
Start with an initial feasible solution $s \in \Omega$.
Initialize tabu lists and aspiration level.
For fixed number of iterations **Do**
    Generate neighbor solutions $\mathbf{V}^* \subset \aleph(s)$.
    Find best $s^* \in \mathbf{V}^*$.
    If move s to s^* is not in $\mathbf{T}$ **Then**
        Accept move and update best solution.
        Update tabu list and aspiration level.
        Increment iteration number.
    Else
        If $c(s^*) < \mathbf{AL}$ **Then**
            Accept move and update best solution.
            Update tabu list and aspiration level.
            Increment iteration number.
        EndIf
    EndIf
EndFor
End.

Figure 2: Algorithmic description of short-term TS.

Figure 3: Comparison between the behavior of short and long term memory TS.

Figure 4: Comparison between the behavior of Classical and Evolutionary TS in terms of current solution cost.

Figure 5: Comparison between the behavior of Classical and Evolutionary TS in terms of best solution cost.