Timing-Driven Floorplanning with Intermediate Buffer Insertion

Maggie Z.-W. Kang
Wayne W.-M. Dai
Tom Dillinger
David LaPotin

UCSC-CRL-97-03
February 14, 1997

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

As the devices and lines shrink into the deep submicron range, it is more effective to insert the intermediate buffers rather than to widen the wires. Almost all existing timing driven floorplanning and placement algorithms and tools don't consider the option of buffer insertion, so many good solutions with smaller area and better routability may be unnecessarily excluded. In this paper, we propose a new methodology of floorplanning and placement where buffered trees are used to estimate the wiring delay. Instead of treating the delay as one of the objectives as done by most of the previous works, we formulate the Delay Bounded Minimum Buffered Tree (DBMB-tree) as follows: given a net and delay bounds associated with critical terminals, construct a tree with intermediate buffers inserted to minimize the total wiring length while satisfying the given delay bounds. Based on the Elmore delay model, we propose an efficient algorithm to construct DBMB-tree for floorplanning and placement. Experiment results show that using buffer insertion at the floorplanning and placement stage yields significantly better solutions in terms of both chip area and total wire length.

Existing multi-objective floorplanning tools use weighted cost summation subject to user-defined constraints to evaluate the solution. It is difficult, if not impossible, to derive a set of weight values from the vaguely defined multiple objectives and the optimization results are very sensitive to the choice of the weight values. Multi-dimension cost vector is introduced to represent the cost value for each objective explicitly [10]. We define the acceptance function in multiple dimensions based on the votes of invited experts, and order the objectives by their sensitivities. Multiple objectives are introduced gradually during the optimization process. The experiment results demonstrate the efficiency of the approach.

Keywords: Floorplanning, Timing Constraints, Buffer Insertion, Total Wiring Length, Delay Bounded Minimum Buffered Tree, Multi-Objectives, Cost Vectors.

1 Introduction

In high speed design, long lines should be treated as distributed RC delay lines and the delay of the lines can be reduced by wire widening or intermediate buffer insertion. Widening the wire makes it a more capacitive line with smaller line resistance. By increasing the sizes of buffers at the source, the wire delay will be reduced. On the other hand, the intermediate buffers decouple a large load that is off the critical path or divide a long wire into short segments each of which has small line resistance, making the delay of the line more linear with the length. As the devices and lines shrink into the deep submicron range, it is more effective, both in terms of power and area, to insert the intermediate buffer rather than to widen the wires.

Because floorplanning and placement have a significant impact on achievable signal delay in timing driven layout design, many research centered on timing driven floorplanning and placement. Almost all existing timing driven floorplanning and placement algorithms don't consider the option of buffer insertion. Only wire lengths or Elmore delay of the lines are used to evaluate the delay. However, it is common practice to insert buffers late at the routing stage. A large industry design may contain as many as tens of thousands intermediate buffers. Therefore it is too conservative to ignore the option of buffer insertion during floorplanning and many good solutions with smaller area and better routability may be unnecessarily excluded.

To make the feasible solution space more realistic, in this paper, we propose a new methodology of floorplanning and placement where buffered trees are used to estimate the wiring delay. We formulate the Delay Bounded Minimum Buffered Tree (DBMB-tree) as follows: given a net and delay bounds associated with critical terminals, construct a tree with intermediate buffers inserted to minimize the total wiring length while satisfying the given delay bounds. Based on Elmore delay model, we propose an efficient algorithm to construct DBMB-trees for floorplanning and placement. Experiment results show that using buffer insertion at the floorplanning and placement stage yields significantly better solutions in terms of both chip area and total wiring length.

The byproduct of this work is the explicit exploration of multiple objectives. Buffer insertion added another competing criteria into the already complicated design space. While a good tradeoff is searched for the competing objectives, such as chip area, total wire lengths, and power consumption, some other constraints have to be met such as delay bound, chip aspect ratio, total number of buffers, and legal buffer locations.

Existing multi-objective floorplanning tools use weighted cost summation subject to user-defined constraints to evaluate the solution. It is difficult, if not impossible, to derive a set of weight values from the vaguely defined multiple objectives. The optimization results are very sensitive to the choice of the weight values. Furthermore, the constant weights may not be sufficient to keep the terms of the cost function properly balanced throughout the optimization process. Finally, the reported single solution with minimized aggregated cost value may not represent the best tradeoff among multiple objectives. The inherent weight and constraint specification problems

can be eliminated by explicit design space exploration [10]. Explicit design space exploration is performed by using a multi-dimensional cost vector and searching for a set of non-redundant solutions representing the best tradeoffs of the cost dimensions. A desirable solution may be chosen from the set and only at that time the trade-offs get made. This methodology fits in the Genetic Algorithm (GA) very well since GA maintains a population of solutions [10]. However, GA has no explicit way to make continuous local search but causes large jumps in the solution space. On the other hand, Simulated Annealing (SA) generates a single sequence of solutions and searches for an optimum solution along this search path.

We propose a new stochastic optimization method, named genetic simulated annealing (GSA) [15]. GSA successfully combines the local stochastic hill climbing features from simulated annealing and the global crossover operation from genetic algorithm.

Multi-dimension cost vector is introduced to represent the cost value for each objective explicitly. SA-based local search generates the single search path and accept each solution along the path by the acceptance function based on its cost and current temperature. The traditional acceptance function is one dimensional and limits the explicit cost value representation in multiple dimensions. We define the multi-dimension acceptance function based on the votes of invited experts. The objectives are ordered by the sensitivity defined for each of them. Multiple objectives are introduced gradually during the optimization process but not simultaneously. The experiment results demonstrate the efficiency of the approach.

The remainder of the paper is as follows. Section 2 reviews the related works on interconnect optimization and intermediate buffer insertion, and overviews the underlying idea of our DBMB-tree algorithm. Section 3 describes the DBMB algorithm in detail and presents the experiment results on general floorplanning using DBMB. The Multiple objective optimization is discussed in Section 4, followed by the conclusion in Section 5.

2 Related Works and Overview of DBMB

2.1 Related Works

Until recently, the minimum steiner tree has been used to construct the routing since it has the smallest total wiring length. However, the source-to-sink path delay in a minimum steiner tree may be too large to satisfy the required delay bound. A number of algorithms have been proposed to make the trade-offs between the total wiring length and the radius (the longest source-to-sink path length) of the tree [1, 5, 7, 8, 11, 17]. The "Bounded-Radius Minimum Routing Tree (BRMRT)" [6] uses parameter ϵ to specify the tradeoff between the minimum radius and the minimum cost. Fig. 2.1 shows three interconnection trees for the same net with different ϵ value.

For deep submicron design, the path length is no longer accurate for the estimation of the path delay. To directly optimize Elmore delay taking into account different

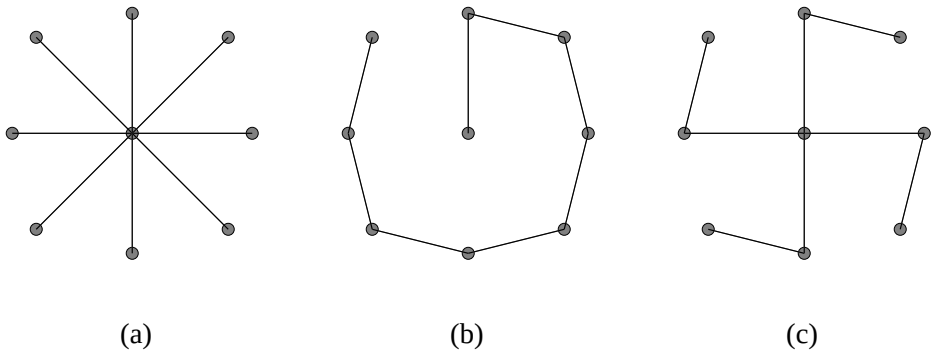

Figure 2.1: (a) the shortest path tree (SPT) with $\epsilon = 0$; (b) the minimum cost spanning tree (MST) with $\epsilon = \infty$; (c) a "tradeoff" between the two constructions with $0 < \epsilon < \infty$

load capacitances of the sinks, several attempts have been made. With exponential timing complexity, the branch and the bound algorithms [4, 3] provide the optimal and near-optimal solutions that minimize the delay from the source to an identified critical sink or a set of critical sinks. For a set of critical sinks, it minimizes the linear combination of sink delays. It is very difficult to choose the proper weights, or the criticality, for the linear combination. While [12] proposed a modified Dreyfus-Wagner Steiner tree algorithm for minimizing the maximal source-to-sink delay, [20] proposed an algorithm for maximizing the delay slack, differences between the real delays and the given delay bounds, at sinks. The maximal source-to-sink delay is not necessarily interesting when the corresponding sink is off the critical path. On the other hand, there maybe more than one critical sinks in the same net, which associate with multiple critical paths. In typical deep submicron designs, more than 60% of the paths in a timing critical design may be critical [24].

Intermediate buffer insertion creates another degree of freedom for interconnect optimization. Early works on fanout optimization problem focused on the construction of buffered trees in logic synthesis [2, 22, 23] without taking into account the wiring effect. Recently, layout driven fanout optimization have been proposed [14, 25]. For a given steiner tree, a polynomial time dynamic programming algorithm was proposed in [26] for the delay-optimal buffer insertion problem. Based on the dynamic programming algorithm, [16] integrated wire sizing and power minimization with the tree construction under a more accurate delay model taking signal slew into account. Inspired by the same dynamic programming algorithm, simultaneous steiner tree construction and buffer insertion algorithm was proposed by [19], and late the work extended to include wire sizing [18]. In the formulation of the problem [19, 18], the main objective is to maximize the required arrival time at the root of the tree, which is defined as the minimum among the differences between the arrival time of the sinks and the delay from the root to the sinks.

Because timing driven floorplanning and placement are usually iterated with the static timing analysis tools, the critical path information is often available and the timing requirement for critical terminals becomes more and more clear when the

iteration progresses. Therefore, it is sufficient to have delay bounded rather than optimizing the delay as in [19, 18].

[27] proposed a Delay Bounded Minimum Steiner tree algorithm (DBMST) to construct a low cost Steiner tree with bounded delay constraints. DBMST algorithm consists of two phases: initialization of tree satisfying the given delay bounds and iterative refinement of the topology to reduce the wiring cost while satisfying the delay bounds associated with critical sinks. The Elmore delay at sinks are very sensitive to the topology change and they have to be recomputed every time the topology is changed. DBMST algorithm searches all possible topological update exhaustively at each iteration and it is very time consuming.

Our DBMB-tree algorithm makes following two major contributions:

- Treating the delay bounds provided by static timing analysis tools as constraints rather than formulating the delay into the objectives, as in [16, 26].
- Efficient buffer insertion and tree refinement algorithm which try to minimize total wire length and the number of buffers.

2.2 Overview of DBMB-tree Algorithm

We formulate the Delay Bounded Minimum Buffered tree (DBMB-tree) as follows: given a signal net and the delay bounds associated with the critical terminals, construct a tree with intermediate buffers inserted to minimize the total wiring length while satisfying the given delay bounds. Based on Elmore delay model, we develop an efficient algorithm for DBMB spanning tree construction.

Contrast to DBMST [27] which starts with max-delay-slack tree, DBMB-tree algorithm begins with a routing tree which trades off the minimum total wiring length and the minimum source-sink path length based on the criticality defined for critical sinks.

For those sinks not meeting the delay bounds, the algorithm first applies buffer insertion to reduce the Elmore delay of the sinks. Then for the sinks whose delay bounds still can not be satisfied with intermediate buffer insertion, the algorithm refines the topology of the tree by the cut-and-link operation to reduce the source-to-sink path delay. The cut-and-link operation will not increase the Elmore delay from the source to other sinks and minimize the increase in total wire length. If necessary, the buffer insertion operation will be invoked for the second time. The algorithm guarantees to find a feasible, hopefully optimal or near optimal, solution, if one exists. The overall time complexity of the DBMB spanning tree algorithm is $O(n^2)$, where n is the total number of terminals of the net.

3 Description of DBMB-tree Algorithm

3.1 Delay Model

Before discussing the DBMB-tree algorithm, we briefly review the Elmore delay model. Elmore delay model is based on the first moment of the impulse response for

a distributed RC representation of the routing tree [13, 21] and it is widely used in interconnect optimization [13].

Given a signal net $S = \{s_0, s_1, \cdots, s_n\}$ with s_0 the *source* and s_1, $\cdots$, s_n *sinks*, routing tree T rooted at the source s_0, let e_{wv} denote the edge from node v to its parent node w in T. The resistance and capacitance of edge e_{wv} are denoted by r_{wv} and c_{wv}, respectively. Let T_v denote the subtree of T rooted at v, and let c_i denote the load capacitance of sink s_i. We use C_v to denote the *tree capacitance* of T_v, defined to be the sum of the load capacitances of the sinks and the capacitances of the edges in T_v. The Elmore delay along edge e_{wv} equals to $r_{wv}(\frac{c_{wv}}{2} + C_v)$. Let r_d denote the on-resistance of the output driver at the source. Then $t_{0,i}$, the Elmore delay from the source to the sink s_i, is:

$$t_{0,i} = r_d C_0 + \sum_{e_{wv} \in path(s_0, s_i)} r_{wv}(\frac{c_{wv}}{2} + C_v) \tag{3.1}$$

Assume r_{wv} and c_{wv} proportional to the length of e_{wv}, scaled by the unit resistance r_0 and unit capacitance c_0, the first term $r_d C_0$ in Eq. 3.1 is linear with the total wire length of the tree T, while the second term has quadratic dependence on the path length from s_0 to s_i and also depends on the capacitive load of the sinks in T.

3.2 DBMB-tree Algorithm

We formulate the DBMB spanning tree problem as follows:

Input: A signal net $S = \{s_0, s_1, \cdots, s_n\}$, where s_0 is the source and s_1, $\cdots$, s_n the sinks. The geometric locations for each terminal of S and delay bound $del(s_i)$ associated with each critical sink s_i.

Output: Spanning tree T rooted at s_0 which spans S and has intermediate buffers inserted.

Objective: Minimize the total wiring length of T.

Constraint: For each critical sink s_i: $t_{0,i} \leq del(s_i)$, where $t_{0,i}$ is the Elmore delay from the source to s_i.

In the DBMB-tree algorithm presented here, we only consider non-inverting buffers. The algorithm can be easily extended to handle the inverting buffers. Let t_b, r_b and c_b denote the internal delay, resistance and capacitance of the non-inverting buffer respectively.

DBMB-tree construction consists of three phases:

1. Estimate the achievable optimal delay $t^*_{0,i}$ for each critical sink $s_i \in S$. If $\exists s_i \in S : t^*_{0,i} > del(s_i)$, no buffered tree satisfying the delay bounds exists.
2. Construct an initial spanning tree T using *Prim-Dijkstra tradeoff* algorithm which obtains the good routing cost and good source-sink path length at critical sinks simultaneously.
3. Refine T by placing buffers or applying the cut-and-link operations. For each critical sink s_i, whose Elmore delay doesn't meet the specified bound:
   - Insert intermediate buffers if the reduced Elmore delay $t'_{0,i} \leq del(s_i)$.

- Otherwise, disconnect s_i and the subtree T_i from T. Select a sink $s_j \in T$ and connect s_j to s_i. Insert a buffer in edge e_{ji} eliminating the effect of topology changes to the Elmore delay from the source to the sinks other than those in T_i. Insert buffers at other edges if necessary. The Elmore delay $t_{0,i}$ in the modified tree is guaranteed to be within the given bound $del(s_i)$ while the increase of total wiring length is minimized.

Given a tree topology T, assume uniform wiring lines and only one buffer can be placed at each edge e_{jk}. Let $d_{j,k}$ denote the length of edge e_{jk}. The Elmore delay along edge e_{jk} without buffer insertion is :

$$t_{j,k} = r_0 d_{j,k}(\frac{c_0 d_{j,k}}{2} + C_k) \tag{3.2}$$

When a buffer is placed x away from s_j on edge e_{jk}, the edge delay is:

$$t_{j,k} = r_0 x(\frac{c_0 x}{2} + c_b) + r_0(d_{j,k} - x)(\frac{c_0(d_{j,k} - x)}{2} + C_k) + r_b(c_0(d_{j,k} - x) + C_k) + t_b \tag{3.3}$$

where $r_b << r_0 d_{j,k}$, $c_b << c_0 d_{j,k}$ and $t_b << t_{j,k}$. The optimal x value can be found for the single buffer accordingly [9]. For the purpose of floorplanning, we assume the buffer is placed immediately after s_j in the following.

3.3 Estimation of Achievable Optimal Delay

Given the terminal locations determined by the floorplanning or placement, the algorithm first estimates, for each critical sink, the shortest path delay that can be obtained with buffer insertion. Fig. 3.1 shows the buffered shortest path tree T_{BSP} for the given net $S = \{s_0, s_1, \cdots, s_{10}\}$. Let $l^*_{0,i}$ denote the length of the shortest path from s_0 to s_i. According to Eq. 3.1, the Elmore delay of s_i in T_{BSP}, denoted by $t^*_{0,i}$, gives the smallest delay achievable with buffer insertion:

$$t^*_{0,i} = r_d n c_b + t_b + r_b(c_0 l^*_{0,i} + c_i) + r_0 l^*_{0,i}(\frac{c_0 l^*_{0,i}}{2} + c_i) \tag{3.4}$$

We introduce *criticality* α_i for each sink s_i to measure how difficult the delay bound can be achieved: $\alpha_i = \frac{t^*_{0,i}}{del(s_i)}$, i.e. the ratio of the achievable smallest delay $t^*_{0,i}$ to specified delay bound of s_i. If $t^*_{0,i} << del(s_i)$, then $\alpha_i \to 0$. If $t^*_{0,i} > del(s_i)$, $\alpha_i > 1$ then the bound is too tight to be achievable for the given floorplan or placement. If $\forall i \in [1, n]$, $t^*_{0,i} \leq del(s_i)$, then the algorithm continues phase 2 and 3, otherwise the floorplanning or placement is timing infeasible. $O(n)$ is sufficient for the calculation of $t^*_{0,i}$ and α_i for all the sinks.

3.4 Construction of Initial Spanning Tree

The second term of Elmore delay $t_{0,i}$ in Eq. 3.1 has quadratical dependence on the path length from s_0 to s_i. For sinks with higher criticality $\alpha_i \sim 1$, $l_{0,i}$ should be closer to the shortest path length $l^*_{0,i}$.

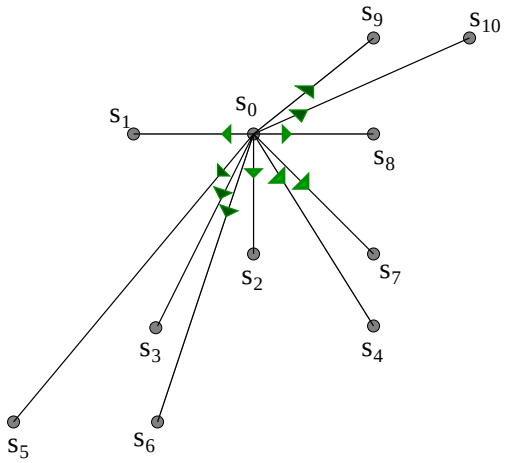

Figure 3.1: Shortest path tree with buffer insertion, which connects each sink s_i with source s_0 directly and places a buffer on each source-sink connection.

Instead of starting with extreme solutions with minimum total wiring lengths or minimum path lengths, we combine Prim's MST algorithm with Dijkstra's SPT algorithm and use criticality to make the tradeoff between both objectives. Initially tree T consists of only s_0 and iteratively edge e_{ij} and sink s_j are added to T, where $s_i \in T$ and $s_j \in S - T$. s_i and s_j are chosen in such a way to minimize:

$$\alpha_j l_{0,i} + d_{i,j} \tag{3.5}$$

When for each $s_j \in S : \alpha_j = 0$, the algorithm is identical to Prim's MST and constructs T with minimum wiring length as in Fig. 3.2 (a). As α_j increases, T has larger wiring length but smaller path length as in Fig. 3.2 (b). The algorithm is identical to Dijkstra's SPT when $\alpha_j = 1$ for each $s_j \in S$ as in Fig. 3.2 (c).

We define *slack* for each sink s_i as follows:

$$\delta_i = del(s_i) - t_{0,i} \tag{3.6}$$

After the initial construction, if $\forall s_i \in S$: $\delta_i \geq 0$, the algorithm has already found the buffered spanning tree subject to the given constraints. Otherwise, continues phase 3 to reduce the Elmore delay at s_i with no increase of Elmore delay at other sinks and with minimal increase of the total wiring length.

The capacitances for all subtrees can be calculated by traversing the routing tree in depth first order. The delays along all edges and the delay from the source to sinks, which equals the added delay of edges along the source-sink path, can be computed by traversing the routing tree in breadth first order. Therefore, the Elmore delays and slacks of all the sinks can be calculated in $O(n)$ time. The timing complexity of this initialization is $O(n^2)$, due to the Prim-Dijstra's tradeoff algorithm.

3.5 Intermediate Buffer Insertion

Given a routing topology T, for a critical sink s_i whose Elmore delay doesn't meet the bound, we define the driving path $p^d_{i,j}$ for each sink $s_j \in T$:

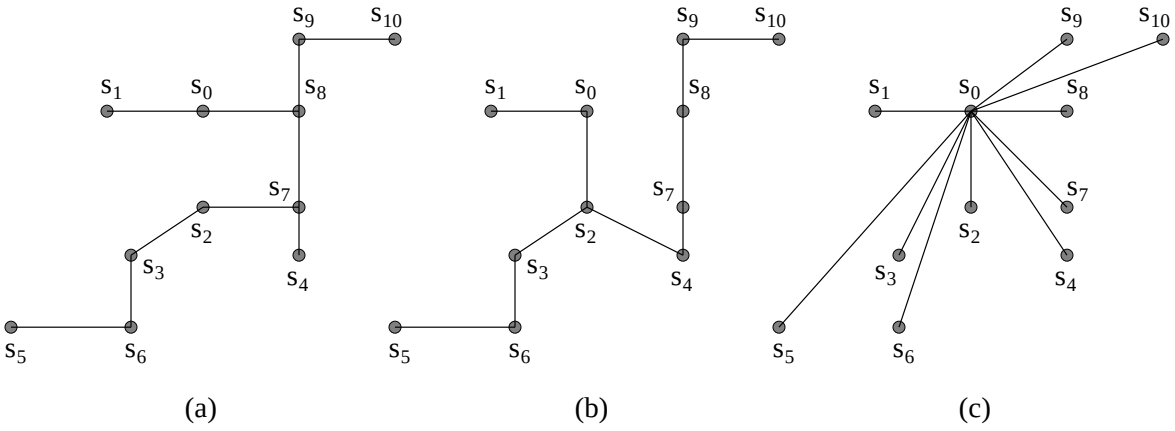

Figure 3.2: *Prim-Dijkstra* tradeoff: when $\forall s_j \in S \ : \ \alpha_j = 0$, the algorithm is identical to Prim's MST and constructs T with minimum wiring length shown in (a). As α_j increases, T has larger wiring length but smaller path length shown in (b), and the algorithm is identical to Dijkstra's SPT when $\alpha_j = 1$ for each $s_j \in S$ shown in (c).

$$p^d_{i,j} = \{e_{wv} \mid e_{wv} \in path(s_0, s_i) \ \wedge \ e_{wv} \in path(s_0, s_j) \ \wedge \ no\ buffer\ on\ path(s_v, s_j)\} \tag{3.7}$$

Let $l^d_{i,j}$ denote the length of $p^d_{i,j}$. To calculate the driving path length for all other sinks with respect to s_i, first trace the path $path(s_0, s_i)$ from s_i to s_0 and mark each edge on this path. Then traverse T from s_0; for each visited sink s_v, $l^d_{i,v}$ can be calculated based on $l^d_{i,w}$ and $d_{w,v}$ where s_w is the parent of s_v and $d_{w,v}$ the length of e_{wv}. The driving path length for s_v is given by:

$$l^d_{i,v} = \begin{cases} l^d_{i,w} + d_{w,v} & if\ e_{wv} \in path(s_0, s_i)\ and\ no\ buffer\ on\ e_{wv} \\ d_{w,v} & if\ e_{wv} \in path(s_0, s_i)\ and\ one\ buffer\ on\ e_{wv} \\ l^d_{i,w} & if\ e_{wv} \notin path(s_0, s_i)\ and\ no\ buffer\ on\ e_{wv} \\ 0 & if\ e_{wv} \notin path(s_0, s_i)\ and\ one\ buffer\ on\ e_{wv} \end{cases} \tag{3.8}$$

Therefore the driving path length of other sinks with respect to one particular sink s_i can be computed in linear time.

After inserting one buffer at edge e_{jk} in topology T, the reduction of Elmore delay at s_i is :

$$\Delta t^{jk}_{0,i} = r_d \Delta C^{jk}_0 + l^d_{i,j}(C_k + c_{jk} - c_b) \tag{3.9}$$

where ΔC^{jk}_0 equals to $C_k + c_{jk} - c_b$ if there is no buffer along the path $path(s_0, s_j)$ as illustrated in Fig. 3.3 (a); it equals to 0, otherwise as illustrated in Fig. 3.3 (b). For a sink in T_k, the buffer placed on edge e_{jk} introduces a little delay penalty: $t_b + r_b(C_k + c_{jk})$, where t_b and r_b, the internal delay and the resistance of the buffer respectively, are usually very small. Traversing T from s_0, ΔC^{jk}_0, thus $\Delta t^{jk}_{0,i}$ can be calculated in linear time.

To reduce the Elmore delay from the source to the critical sink s_i, whose Elmore delay doesn't meet the bound, select an edge e_{jk} with maximal $t^{jk}_{0,i}$ and insert one

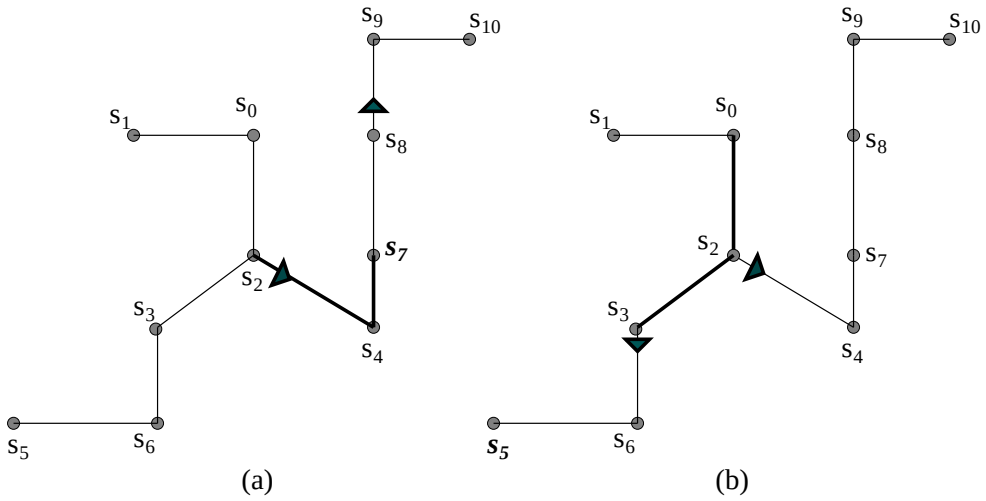

Figure 3.3: Given the tree topology with a buffered edge e_{24}, when a buffer is placed on edge e_{89} as shown in (a), the driving path for s_8 with respect to sink s_7 is: $p^d_{7,8} = \{e_{24}, e_{47}\}$. There is one buffer on path $path(s_0, s_8)$, $\Delta C_0^{89} = 0$, thus the reduction of Elmore delay at s_7: $\Delta t^{89}_{0,7} = (r_{24} + r_{47})(C_9 + c_{89} - c_b)$. On the other hand, when a buffer is placed on edge e_{36} as shown in (b), the driving path for s_3 with respect to sink s_5 is: $p^d_{5,3} = \{e_{02}, e_{23}\}$. There is no buffer on path $path(s_0, s_3)$, $\Delta C_0^{36} = C_6 + c_{36} - c_b$, so the reduction of Elmore delay at s_5: $\Delta t^{36}_{0,5} = (r_d + r_{02} + r_{23})(C_6 + c_{36} - c_b)$.

buffer on it. After the buffer insertion, the capacitances of subtrees rooted at the sinks along the path $path(s_0, s_j)$ can be updated in linear time. This process continues until the Elmore delay from the source to s_i satisfies the given bound or buffers have been inserted to all unbuffered edges and the delay bound $del(s_i)$ still can't be satisfied.

For each critical sink s_i, $l^d_{i,v}$ and $\Delta t^{jk}_{0,i}$ can be computed in $O(n)$ time. There are at most $n-1$ buffers may be inserted and the update after each insertion take $O(n)$ time. So the overall complexity of total buffer insertion is $O(n^2)$.

3.6 Cut-and-link Operation

The Elmore delay $t_{0,i}$ in Eq. 3.1 depends on two factors: the subtree capacitance driven by the source-sink path $path(s_0, s_i)$ and the path length $l_{0,i}$. Accordingly, two approaches of reducing $t_{0,i}$ can be adopted: placing buffers in T to decouple the large load from the source-sink path and modifying the tree topology to shorten $l_{0,i}$. When the delay bound $del(s_i)$ can not be satisfied with buffer insertion in T, the algorithm refines the topology to decrease the source-sink path length $l_{0,i}$ to further reduce $t_{0,i}$.

To reduce the Elmore delay at s_i, subtree rooted at s_i, T_i, is disconnected from T. A sink $s_j \in T$ is selected and T_i is re-connected by adding edge e_{ji}. Such cut-and-link operation is illustrated in Fig. 3.4. One buffer is placed at the new edge e_{ji} and other buffers may be placed if necessary as described in the last section. As the result, the

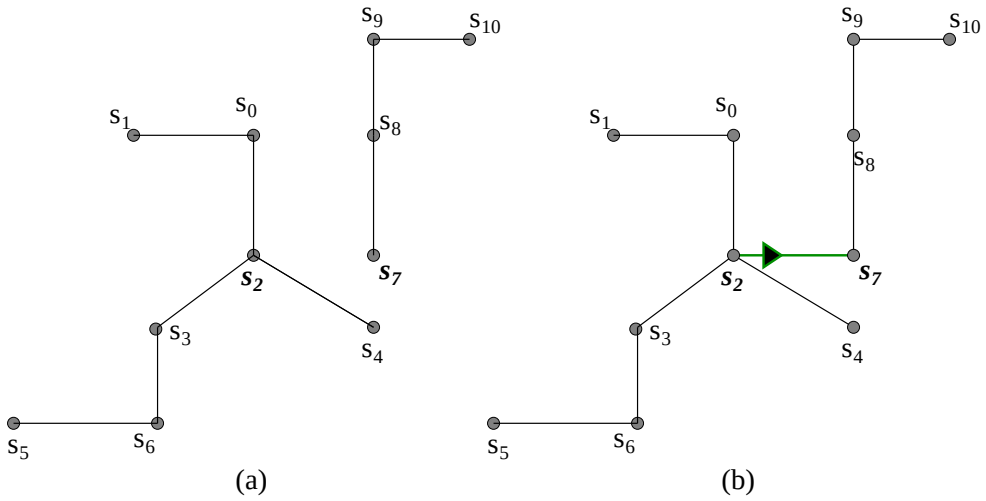

Figure 3.4: Cut-and-link operation: disconnect subtree T_7 from T and delete edge e_{47} in (a); select another sink s_2 and connect s_7 to s_2. One buffer is placed on the new connection in (b).

Elmore delay at s_i satisfies the delay bound. To show the cut-and-link operation is correct, we need to prove:

1. A sink $s_j \in T$ exists such that after adding edge e_{ji} and inserting buffers, $t'_{0,i} \leq del(s_i)$ in the new tree.
2. The Elmore delay from the source to other sinks will not be increased after the topology change.

Recall the achievable smallest path delay estimated in Eq. 3.4, the $t^*_{0,i}$ can be obtained by directly connecting s_0 to s_i and placing buffers on edge e_{0j} and e_{ij}. According to phase 1, the Elmore delay at s_i is less or equal to the given bound $del(s_i)$, as illustrated by the same example in Fig. 3.5.

For each sink $s_k \notin T_i$, the Elmore delay $t'_{0,k}$ will not be increased because the source-sink path remains the same and the capacitance of subtree T_k will be reduced or remains the same due to the buffer insertion on edge e_{ji}. On the other hand, for each sink $s_k \in T_i$, $t'_{0,k} = t'_{0,i} + t'_{i,k}$, $t'_{0,k}$ will not be increased because $t'_{0,i}$ is reduced and the topology of T_i remains the same. Therefore the cut-and-link operation will not increase the Elmore delay at other sinks.

Let $t^j_{0,i}$ denote the Elmore delay from the source to s_i achieved by connecting s_i to s_j and placing buffers on each unbuffered edge shown in Fig. 3.6 (b). The algorithm chooses s_j such that $t^j_{0,i} \leq del_i$ and $d_{j,i}$ is minimum.

To calculate $t^j_{0,i}$, we define the *smallest delay* $\overline{t_{0,j}}$ as the Elmore delay from s_0 to the sink s_j in a given T with one buffer inserted at each edge :

$$\overline{t_{0,j}} = \overline{t_{0,h}} + d_b + r_b(c_{hj} + n_j c_b) + r_{hj}(\frac{c_{hj}}{2} + n_j c_b) \tag{3.10}$$

where s_h is the parent of s_j in T and n_j the number of children at s_j. $t^j_{0,i}$ can be calculated accordingly:

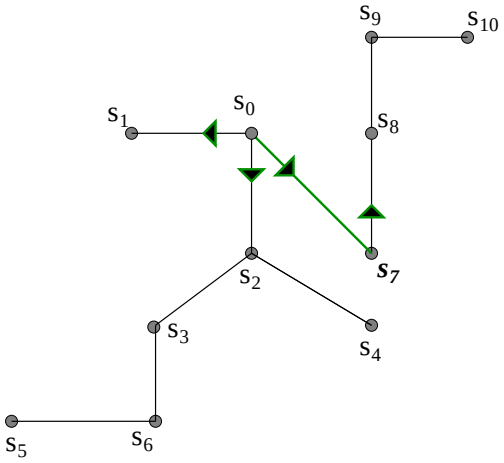

Figure 3.5: s_0 is selected and connected to s_7. The buffers are inserted on edge e_{07}, e_{01}, e_{02} and e_{78}. The Elmore delay at s_7: $t^*_{0,7} \leq del(s_7)$.

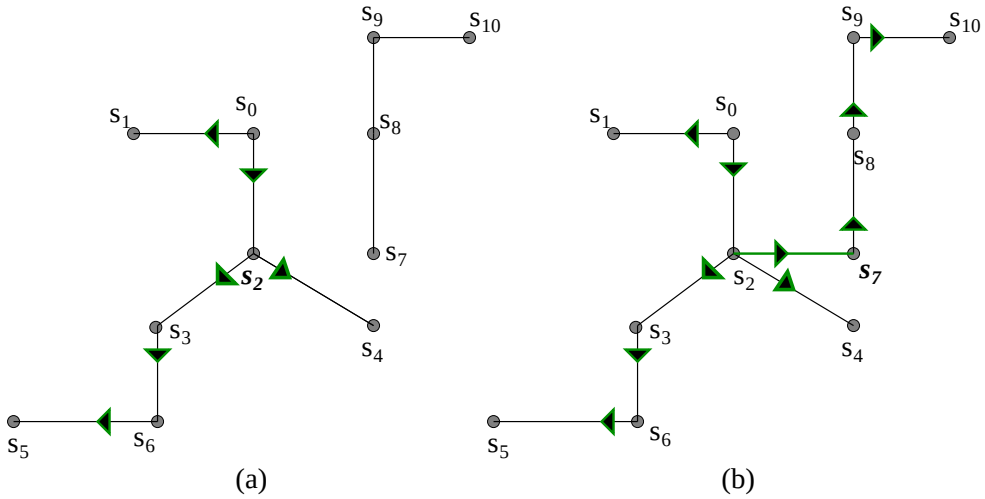

Figure 3.6: (a): Given the tree topology, assume inserting one buffer on each edge, according to Eq. 3.10, $\overline{t_{0,2}} = \overline{t_{0,0}} + d_b + r_b(c_{02} + 2c_b) + r_{02}(\frac{c_{02}}{2} + 2c_b)$ where 2 is the number of children of s_2. (b): $t^2_{0,7}$ is the Elmore delay at s_7 by connecting s_2 to s_7 and inserting one buffer on each edge of T. According to Eq. 3.11, $t^2_{0,7} = \overline{t_{0,2}} + (r_b + r_{27})c_b + d_b + r_b(c_{27} + C_7) + r_{27}(\frac{c_{27}}{2} + C_7)$.

$$t^j_{0,i} = \overline{t_{0,j}} + (r_b + r_{hj})c_b + d_b + r_b(c_{ji} + C_i) + r_{ji}(\frac{c_{ji}}{2} + C_i) \tag{3.11}$$

All $\overline{t_{0,i}}$, thus $t^j_{0,i}$ can be computed in linear time. For all critical sinks, the buffer insertion and the cut-and-link operation in phase 3 can be done in $O(n^2)$ time. Combining with the complexity of phase 1 and 2, overall DBMB-tree algorithm runs in $O(n^2)$, which n is the number of terminals in the signal net.

Table 3.1: Test Examples

Circuit	Blocks	Nets	Critical Pins	Min. Area	Min. Wire
ami33	33	153	33	1,156,449	31,098
ami49	49	412	50	35,445,424	335,496
sim66	66	306	66	2,312,898	62,196
sim98	98	824	100	70,890,848	670,992

Table 3.2: The Optimal Solution with and w/o DBMB-tree Estimation

	without Buffer Insertion				with Buffer Insertion				Improvement	
Circuit Name	Area (times)	Wire (times)	Aspect Ratio	CPU (sec.)	Area (times)	Wire (times)	Aspect Ratio	CPU (sec.)	Area (%)	Wire (%)
ami33	1.50	3.48	1.01	2278	1.11	3.06	0.61	3279	26.0	12.0
ami49	1.26	5.60	0.60	2720	1.13	5.25	0.68	3961	10.3	6.25
sim66	1.33	4.93	0.95	5927	1.16	4.13	1.27	6998	12.8	16.2
sim98	1.31	8.73	0.91	6967	1.16	8.24	0.78	9060	11.5	5.6

3.7 Experiment Results

We apply DBMB spanning tree algorithm in general floorplanning based on GSA stochastic optimization. The system is implemented in C language and tested on SUN SPARC 20 workstation. Table 3.1 describes the characteristics of four test circuits used in the experiment. sim66 and sim98 are generated from benchmark ami33 and ami49 by doubling the blocks and netlist. The minimum area and wire length for each circuit represent the lower bounds on the chip area and total wire length. When we say area = 1.50, that means the area equals to one and half times of minimum area achievable for that circuit. For each example, the same parameters are used for the GSA stochastic optimization. The results shown in table 3.2 demonstrate that, with buffer insertion, the floorplanning can get significantly better solution, in which the chip area and total wiring length can be improved up to 26% and 16.2% respectively. Fig. 3.7 and Fig. 3.8 show outputs of the placement with and without buffer insertion for circuit sim98 respectively.

At each step of SA-based local search, the wiring cost for the new solution is updated incrementally, therefore, the running time for the optimization with buffer insertion is comparable with that without buffer insertion.

4 Multiple Objective Optimization in GSA

For deep submicron technology, the optimization problem of floorplanning and placement becomes more and more complicated, in which a good tradeoff among multiple conflicting objectives is hard to find.

Traditional methods, based on weighted cost summation and user-defined constraints, can not achieve the best tradeoff because of the difficulty to derive a set of weight values or the satisfiable bounds. [10] proposed a multi-dimension cost vector to evaluate a solution by the cost value in each dimension explicitly, rather than by single aggregated cost value. A set of non-redundant solutions are searched without

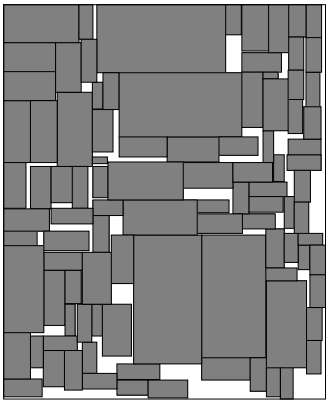

Figure 3.7: Placement of circuit sim98 with buffer insertion. Achieve total area of 82,233,384 and total wire length of 5,528,974.1 in 9060 sec. on Sun SPARC 20 workstation.

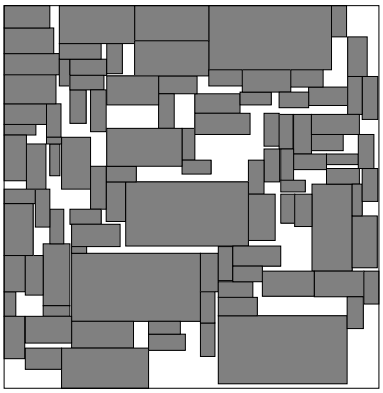

Figure 3.8: Placement of circuit sim98 without buffer insertion. Achieve total area of 92,867,011 and total wire length of 5,857,760.2 in 6967 sec. on Sun SPARC 20 workstation.

making any tradeoff. A desirable solution can be chosen from the output set of best solutions and the tradeoff is made only at that time. This methodology fits in the Genetic Algorithm (GA) very well since GA maintains a set of non-redundant solutions naturally [10].

However, the performance of GA highly depends on the quality of crossover operation which mimics the propagation and causes large jump in the search space. Most of crossover operations for building block placement studied so far are based on the random process. Therefore the genetic algorithm jumps randomly for long time to obtain good solutions.

Contrast to GA, Simulated Annealing (SA) starts with an initial solution and searches the local region exhaustively based on continuous small moves. But it takes long time to cover a large solution space. In addition, the single best-so-far solution preserved during SA cannot represent the solution space with good tradeoffs among multiple objectives.

A new stochastic optimization method, namd genetic simulated annealing (GSA) [15] combines the advantages of both SA and GA. It maintains the population of solutions, searches the local region continuously by SA-based small moves, and jumps in the search space by crossover after the local search. Therefore GSA can search the solution space both locally and globally within limited cpu time.

We apply GSA to general floorplanning problem, in which multiple conflicting objectives are optimized explicitly. Using multi-dimension cost vector, solutions are evaluated by the cost value in each dimension directly and a set of solutions with best tradeoff among multiple objectives are preserved.

SA-based local moves generate the single search path and search for optimum solution along this path. The candidate solution is accepted by the function based on the cost value and current annealing temperature. To support explicit multi-dimension cost representation, we define the multi-dimension acceptance function, in which each dimension accepts the up-hill moves in that dimension with the adaptive probability and the solution is accepted based on the votes of experts from each dimension. Therefore no tradeoff need to be made to determine the acceptance of a new solution.

Furthermore the stochastic random search can fall into local minima easily because it is difficult to make better tradeoffs among many objectives. We introduce the sensitivity for each objective and optimize the multiple objectives in order based on their sensitivities, but not simultaneously. Experiment results demonstrate the properly ordered optimization procedure for multiple objectives can improve the results and speed up the process significantly.

4.1 Acceptance Function

Without resorting to a single-valued cost measure, the multi-dimension cost vector is used to specify the "good" tradeoff and to compare the solutions. Given a set of feasible floorplanning realizations Π, for solution $x \in \Pi$, $c(x) = (x_1, x_2, \cdots, x_n)$ is the cost vector in which x_i is the cost value of i^{th} objective and n the number of objectives. Similar with [10], a *goal* vector $g = (g_1, g_2, \cdots, g_n)$, in which $\forall i : 0 \leq g_i \leq \infty$, is specified by users. There are two kinds of goals in g: $g_i = 0$ means that the minimum cost value of i^{th} objective is wanted; and $g_i > 0$ gives the maximum value which can be accepted. The vector g defines a set of satisfactory solutions:

$$S_g = \{x \in \Pi \mid \forall i \;\; g_i > 0 : x_i \leq g_i\} \tag{4.1}$$

Contrast to traditional user-defined bounds, the values specified by $g_i > 0$ are only used to guide the search process and need not be obtainable. Therefore they are significantly easier to be specified than the traditional bounds.

To compare the relative quality between two solutions x and y, we define that solution x is *preferable* to y, written $x \prec y$ as in [10].

At each step of the SA-based local search, the new solution x' is produced by changing the small fraction of current solution x and is accepted based on the decision

of *committee*, which consists of the invited *experts*. There is an expert for each objective, who votes on accepting the solution $x^{'}$ according to the cost reduction of the corresponding objective. For i^{th} objective, the expert votes to accept the solution $x^{'}$ if the following condition holds:

$$(x_i^{'} \leq g_i) \vee (x_i^{'} \leq x_i) \vee (\eta \leq e^{-\Delta x_i/T}) \tag{4.2}$$

where η is the random-generated float number falling in $[0, 1]$, and T is the current *temperature*. For objective $g_i = 0$, Δx_i denotes the cost reduction of i^{th} objective: $\Delta x_i = x_i - x_i^{'}$. For $g_i > 0$, Δx_i is the slack of i^{th} objective: $\Delta x_i = g_i - x_i^{'}$.

The experts accept the hill-climbing moves ($g_i = 0$ and $\Delta x_i < 0$) or the infeasible moves ($g_i > 0$ and $\Delta x_i < 0$) with the probability defined by $min\ \{1, e^{-\Delta x_i/T}\}$ in Eq. 4.2. Therefore the ability of escaping local minima of SA is conserved and the given bound value $g_i > 0$ need not be the strict bound but the guidance for the stochastic optimization. Consequently each expert makes decision based only on the cost value of single objective and no tradeoff need to be made among different competing objectives. GSA accepts the new solution $x^{'}$ if and only if each invited expert in the committee votes *yes*.

4.2 Ordered Optimization of Multiple Objectives

To explain when the experts are invited to the committee, we define the *sensitivity* S_i, as the average percentage of changed cost value for i^{th} objective over the $m + 1$ times SA-based local search:

$$S_i = \frac{\sum_{i=1}^{m} \frac{|x_{i+1} - x_i|}{x_i}}{m} \tag{4.3}$$

Given m is large enough, we can discover which objective is most sensitive to the local operations in average. The more sensitive, the easilier the cost value of the objective alternates with the local operations and thus the later the objective should be considered.

When more than two objectives are considered simultaneously, the random process can easily fall into the local minima and it takes long time to reach better tradeoffs. To guide the random search and speed up the process, we optimize multiple objectives in order.

Initially the objective with lowest sensitivity is brought in and the corresponding expert joins the committee. The new solutions accepted have better performance in the objective under consideration and relative worse performance in others. When the solution satisfies the user-defined goal for the objectives under consideration or the cost surfaces of objectives under consideration become flat, a next new objective is brought in and at the same time the corresponding expert joins the committee. Once objectives are brought in and the corresponding experts join the committee, they stay in the committee until the end. The algorithm terminates when the satisfied solutions are found or all objectives are brought in and their cost surfaces become flat.

4.3 GSA with Multiple Objectives

Following is the outline of the new GSA algorithm with multiple objectives :

```
01:  initialize-population(P),
02:  initialize-committee(E),
03:  initialize-solution(x),
04:  repeat
05:      SA-loop begins
06:          x' = SA-search(x),
07:          if (accept-solution(x')) then
08:              update-population(P, x),
09:              x ← x',
10:              if (invite()) then update-committee(E),
11:          else update-population(P, x'),
12:      SA-loop ends
13:      select x_{p1}, x_{p2} ∈ P,
14:      x_{c1} = crossover(x_{p1}, x_{p2}),
15:      x_{c2} = crossover(x_{p2}, x_{p1}),
16:      update-population(P, x_{c1}),
17:      update-population(P, x_{c2}),
18:      if (x_{c1} ≺ x_{c2}) then x ← x_{c1},
19:      else x ← x_{c2},
20:  until goal() or converge() or cpu-limit(),
21:  reports P.
```

Figure 4.1: Outline of GSA Algorithm with Multiple Objectives.

At each iteration, the SA-based local search starts with the current solution x, and generates the new candidate $x^{'}$ by applying moves, exchanges or rotations as in [15]. The new solution is accepted based on the votes of current committee E in *accept-solution()*. If $x^{'}$ is accepted, insert x into population P and continue the search from $x^{'}$. Otherwise, insert $x^{'}$ into P and restart the search from x.

When a solution y is inserted into P in *update-population()*, each individual solution $z \in P$ is visited. If $\exists z \in P$ and $z \prec y$, then y won't be added. For each $z \in P$ and $y \prec z$, delete z from P. If no solution preferable to y found in P, y is added into P.

invite() decides when a new objective is brought in and at the same time the corresponding expert joins the committee. At the end of SA search, two parent solutions are chosen randomly from P and the crossover in [15] is applied to generate the two child solutions x_{c_1} and x_{c_2}. The next iteration starts with the new generation. When *goal*() detects that the satisfied solutions have been found, or *converge*() detects the cost surfaces become flat, or the CPU time exceeds the given limit *cpu-limit*(), the process terminates and the set of best solutions found so far in P is reported.

4.4 Experiment Results

We apply multi-dimension cost vector and multi-dimension acceptance function described above in the general floorplanning optimization based on GSA stochastic algorithm. Given the goal vector $g = (g_{area}, g_{wire}, g_{aspect})$, the chip area and total wiring length are minimized while aspect ratio close to g_{aspect} is desirable.

For comparison, we also use SA with the cost function $cost = area + \lambda\ wire$, subject to the given aspect ratio constraint g_{aspect}. λ is the weight value for total wiring length, and the weight for area is normalized to 1.0 for simplicity.

Both GSA and SA runs under SUN SPARC 20 workstation, using the same set of annealing schedule within same cpu time. GSA reports the set of optimal solutions, in one pass, covering the large solution space, shown by the curve marked by "cost-vector" in Fig. 4.2. Three separate SA runs using different weight value for λ report three best-so-far solutions shown by the curve marked "cost-ratio" in Fig. 4.2. We choose λ in large range and the three solutions reported by SA are very close to the solutions found by GSA. Therefore without knowing the weight values for multiple objectives, one pass GSA can achieve the set of optimal solutions including a wide range of tradeoff among the multiple objectives. Even with knowing the proper weight values, one pass SA can only find single solution for that particular weight.

To demonstrate the optimal solution space is very sensitive to the optimization order for multiple objectives, three GSA processes running in the same environment are compared using the same test circuit: ami98. Each process uses same set of stochastic parameters and runs in same cpu time, and optimizes the multiple objectives in different order. The optimal solution spaces explored by the experiment are shown in Fig. 4.3, in terms of chip area and total wiring length. Clearly, properly ordered optimization for multiple objectives can speed up the stochastic process significantly and explore the optimal solution space more effectively.

5 Conclusion

In this paper, we propose a new methodology of floorplanning and placement where the intermediate buffer insertion is used as another degree of freedom in delay calculation. The timing constraints of a floorplan are evaluated many times during our stochastic optimization process. The introduction of buffer insertion increases the complexity of the evaluation significantly due to the complexity of buffered tree construction and the complication of multi-objective optimization.

An efficient algorithm to construct a Delay Bounded Minimum Buffered (DBMB) spanning tree has been developed. One of the key reasons to make this tree construction efficient is that we treat the delay bounds as constraints rather than formulating the delay into the objectives as did in most of the previous works. In fact, our problem formulation is more realistic for the path based timing driven layout design. Buffer insertion adds another competing criteria into the already complicated design space, making the traditional approaches using weighted cost summation of multiple competing objectives even worse. However, it is not trivial to apply the explicit design

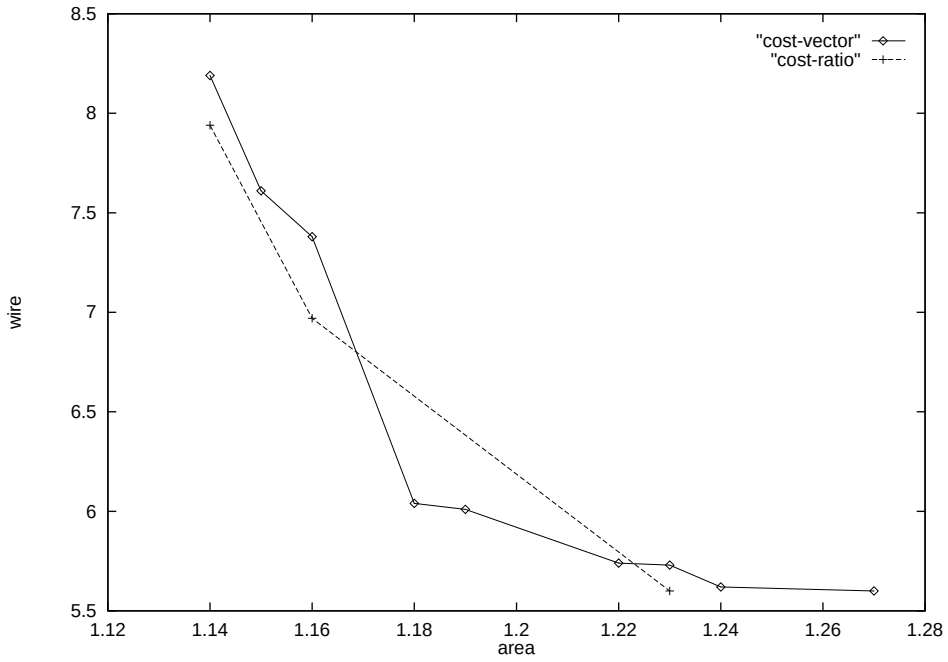

Figure 4.2: Multiple objective optimization for ami98: the set of solutions marked by "cost-vector" is reported by one run of GSA using vector-valued cost function to represent and evaluate solutions in each dimension explicitly. Comparing to this, three runs of SA use the weighted cost summation $area + \lambda wire$ and report three best-so-far solutions marked by "cost-ratio", in which the λ is 1.0, 10.0 and 100.0 from left to right respectively. Each process uses the same cpu time and the same parameters for stochastic search.

space exploration methodology to our Genetic Simulated Annealing (GSA) approach. To support the multi-dimension cost representation, the acceptance function in multiple dimensions has to be provided to accept the candidate solutions along the single search path generated by SA-based local moves. In the second part of the paper, we define the acceptance function in multiple dimensions based on the votes of invited experts and propose a method to introduce different objectives in order based on their sensitivities. The efficient DBMB spanning tree algorithm together with the new multiple objective optimization method made our buffered tree based floorplanning and placement more effective.

References

[1] C. J. Alpert, T. C. Hu, J. H. Huang, and A. B. Kahng. A direct combination of the prim and dijkstra constructions for improved performance-driven global routing. In *Proc. of IEEE Intl. Symp. on Circuits and Systems*, pages 1869–1872, 1993.

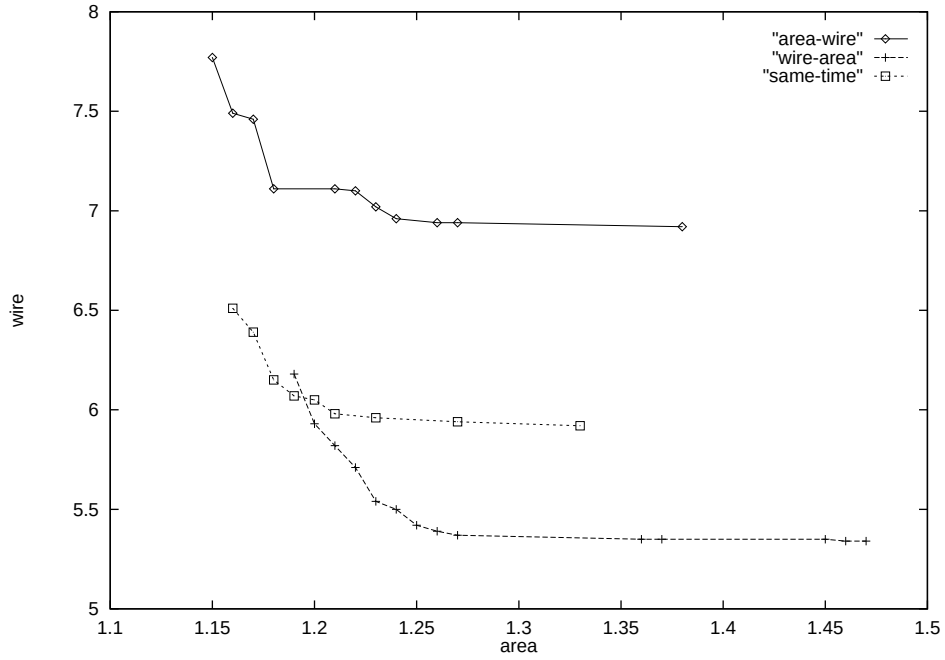

Figure 4.3: Ordered Optimization of Multiple Objectives for ami98: three GSA processes use the same parameters for stochastic search and run on SUN SPARC 20 using the same cpu limit 740 sec. Given the goal vector: $g = (g_{area}, g_{wire}, g_{aspect}) = (0, 0, 0.9)$, the chip area and wire length are optimized subject to the aspect ratio constraint. The objectives are optimized in different orders: the "area-wire" curve is obtained by the optimization in *aspect ratio* $\rightarrow$ *area* $\rightarrow$ *wire length* order; "wire-area" curve in *aspect ratio* $\rightarrow$ *wire length* $\rightarrow$ *area* order; and "same-time" curve by simultaneous optimization for each objective.

[2] C. L. Berman, J. L. Carter, and K. F. Day. The fanout problem: From theory to practice. In *Proc. 1989 Decennial Caltech Conf.*, pages 69–99, 1989.

[3] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins. Rectilinear steiner trees with minimum elmore delay. In *Proc. 31st ACM/IEEE Design Automation Conf.*, pages 381–387, June 1994.

[4] K. D. Boese, A. B. Kahng, and G. Robins. High-performance routing trees with identified critical sinks. In *Proc. 30th ACM/IEEE Design Automation Conf.*, pages 182–187, June 1993.

[5] J. P. Cohoon and L. J. Randall. Critical net routing. In *Proc. IEEE Intl. Conf. on Computer Design*, pages 174–177, 1991.

[6] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Performance-driven global routing for cell based ic's. In *Proc. IEEE Intl. Conf. Computer Design*, pages 170–173, Cambridege, MA, October 1991.

[7] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Provably good performance-driven global routing. *IEEE Trans. Computer Aided Design*,

pages 739–752, 1992.

[8] Jason Cong, Kwok-Shing Leung, and Dian Zhou. Performance-driven interconnect design based on distributed rc delay model. In *Proc. 30th ACM/IEEE Design Automation Conf.*, pages 606–611, June 1993.

[9] Sanjay Dhar and Mark A. Franklin. Optimum buffer circuits for driving long uniform lines. *IEEE Journal of Solid-State Circuits*, 26(1):32–40, January 1991.

[10] H. Esbensen and E. S. Kuh. An mcm/ic timing-driven placement algorithm featuring explicit design space exploration. In *Proc. 1996 IEEE Multi-Chip Module Conf.*, pages 170–175, Santa Cruz, CA, February 1996.

[11] J. M. Ho, D. J. Lee, C. H. Chang, and C. K. Wong. Bounded-diameter spanning tree and related problems. In *Proc. ACM Symp. on Computational Geometry*, pages 276–282, 1989.

[12] X. Hong, T. Xue, E. S. Kuh, C. K. Cheng, and J. Huang. Performance-driven steiner tree algorithms for global routing. In *Proc. 30th ACM/IEEE Design Automation Conf.*, pages 177–181, Baltimore, MD, June 1993.

[13] Andrew B. Kahng and Gabriel Robins. *On Optimal Interconnections for VLSI.* Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, MA 02061, 1995.

[14] L. N. Kannan, P. R. Suaris, and H. G. Fang. A methodology and algorithms for post-placement delay optimization. In *Proc. ACM/IEEE Design Automation Conf.*, pages 327–332, 1994.

[15] Seiichi Koakutsu, Maggie Kang, and Wayne W.-M. Dai. Genetic simulated annealing and application to non-slicing floorplan design. In *Proc. 5th ACM/SIGDA Physical Design Workshop*, pages 134–141, Virginia, USA, April 1996.

[16] John Lillis, Chung-Kuan Cheng, and Ting-Ting Y. Lin. Optimal and efficient buffer insertion and wire sizing. In *Proc. IEEE 1995 Custom Integrated Circuits Conf.*, pages 259–262, 1995.

[17] Andrew Lim and Siu-Wing Cheng. Performance oriented rectilinear steiner trees. In *Proc. of 30th Design Automation Conf.*, pages 171–176, June 1992.

[18] Takumi Okamoto and Jason Cong. Buffered steiner tree construction with wire sizing for interconnect layout optimization. In *Proc. 1996 IEEE/ACM International Conf. on Computer Aided Design*, pages 44–49, San Jose, CA, Nov. 1996.

[19] Takumi Okamoto and Jason Cong. Interconnect layout optimization by simultaneous steiner tree construction and buffer insertion. In *Proc. 5th ACM/SIGDA Physical Design Workshop*, pages 1–6, Reston, Virginia, April 1996.

[20] S. Prasitjutrakul and William J. Kubitz. A timing-driven global router for custom chip design. In *IEEE Intl. Conf. on Computer Aided Design*, pages 48–51, 1990.

[21] Jorge Rubinstein, Paul Penfield, and Mark A. Horowitz. Signal delay in RC tree networks. *IEEE Trans. on Computer-Aided Design*, CAD-2(3):202–211, July 1983.

[22] K. J. Singh and A. Sangiovanni-Vincentelli. A heuristic algorithm for the fanout problem. In *Proc. ACM/IEEE Design Automation Conf.*, pages 357–360, 1990.

[23] H. J. Touati, C. W. Moon, R. K. Brayton, and A. Wang. Performance oriented technology mapping. In *Proc. 6th MIT VLSI Conf.*, pages 79–97, 1990.

[24] M. Y. Mike Tsai and R. S. Tsay. Ic layout shift at deep-submicron level. *Electronic Engineering Times*, pages 820–866, October 1994.

[25] H. Vaishnav and M. Pedram. Routability-driven fanout optimization. In *Proc. ACM/IEEE Design Automation Conf.*, pages 230–235, 1993.

[26] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal elmore delay. In *Proc. International Symposium on Circuits and Systems*, pages 865–868, 1990.

[27] Qing Zhu. *Chip and Package Co-Synthesis of Clock Networks.* PhD thesis, Univ. of California, Santa Cruz, Santa Cruz, CA, June 1995.