Tradeoff Routing Resource, Runtime and Quality in Buffered Routing *

Xiaoping Tang
Cadence Design Systems
San Jose, CA 95134 USA
xtang@cadence.com

Martin D.F. Wong
University of Illinois
Urbana, IL 61801 USA
mdfwong@uiuc.edu

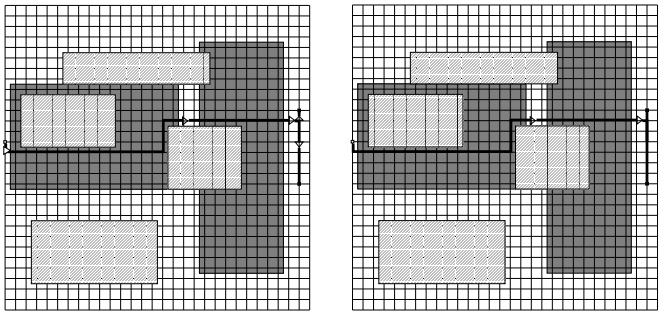
Abstract

With the wide use of hard macros and IP blocks in design, buffered routing (simultaneous routing and buffer insertion) becomes unavoidable. Routing resource allocation and distribution are serious concerns in buffered routing of deep submicron design. The capability of capturing the tradeoff between routing resource cost and signal delay is crucial in practice since the resource overuse of min-delay solution may cause congestion problem (congestion also means over-inserting buffers). However, many existing algorithms are mainly designed to minimize signal delay. In the paper, we first study the problem of minimizing the linear combination of delay and cost, and extend the graph-based algorithm in [10] to solve it. We then show that a variant of the algorithm can solve other problems such as maximizing delay reduction to cost ratio, minimizing routing cost subject to a delay constraint, and minimizing delay subject to the cost not exceeding a given budget. We also develop a hierarchical approach to buffered routing construction for problems with large number of sinks to tradeoff solution quality and runtime.

1. Introduction

The device density on chips grows quadratically with the rate of decrease in the feature size. Also, there is a trend that more functionalities are integrated into one chip (such as system-on-chip). Thus in very deep submicron era, chips become more congested, the number of used metal layers increases, and interconnect delay, especially global interconnect delay, dominates gate delay in determining the overall circuit performance. Meanwhile clock frequency keeps increasing. As a result, many techniques such as wire sizing and buffer insertion have to be employed to reduce interconnect delay. However, wire sizing and buffer insertion can not be used excessively. Otherwise, it may cause congestion problem that the design is unroutable or the inserted buffers are unplacable. As hard macros and IP blocks are widely used in design, buffered routing (simultaneous routing and buffer insertion) becomes unavoidable, since hard macros and IP blocks may allow route going over but forbid buffer insertion. As a result, many buffered routing algorithms are proposed to address the problem. However, many of the existing buffer insertion algorithms minimize the delay for a routing tree, or use min-delay as a major objective [7, 8, 9, 3, 13, 1, 10]. Besides delay minimization, we need to allocate and distribute routing resource carefully in buffered routing. One natural formulation of the problem is to capture a cost vs. performance tradeoff, e.g., minimizing congestion subject to a delay constraint. The capability to capture such tradeoff is crucial in buffered routing since the cost overhead of min-delay solution tends to be excessive. We illustrate the importance of the problem by using an example as shown in Figure 1. The min-delay solution shown in Figure 1(a) inserts 5 buffers, which achieves the delay of 1183ps. On the other hand, another solution achieves the delay of 1187ps by inserting 2 buffers as shown in Figure 1(b). For the few ps delay improvement (about 0.4%), three more buffers are used. This kind of solutions overuse resource in buffered routing.

In contrast to traditional routing optimization, the problem is fundamentally multidimensional since both cost and delay are considered. There are several variations of formulation. First, minimizing a linear combination of delay and cost, $D + \lambda C$ where D and C are the signal delay and routing cost respectively, and λ is the factor to balance the two items; second, minimizing routing cost subject to a delay constraint; third, minimizing delay while cost does not exceed some given budget; fourth, given a required delay constraint D_r, maximizing the delay reduction to cost ratio (**DRCR**), $\frac{D_r - D}{C}$.



(a) (b)

Figure 1: (a) A min-delay solution inserts 5 buffers, delay = 1183ps. (b) A more preferable solution uses 2 buffers, delay = 1187ps.

Jagannathan et al. [4] studied the buffer insertion problem to maximize the DRCR for 2-pin nets, which limits the application since in practice many nets have more than 2 pins. It is suggested in [4] that the DRCR is a natural composite objective function capturing the tradeoff between cost and delay. However, the primary problem to tradeoff cost and delay is to minimize the linear combination of delay and cost, $D + \lambda C$. By adjusting λ, we can sample a delay vs. cost tradeoff curve. Those solutions that lie above the curve are redundant (dominated). In the paper, we first extend the algorithm in [10] to solve the problem of minimizing the linear combination of delay and cost. We then show that a variant of the algorithm can efficiently identify points on the curve. Thus it can be used to solve other problems such as DRCR problem, minimizing routing cost subject to a delay constraint, and minimizing delay subject to the cost not exceeding a given budget.

Since thousands of nets need handling in buffered routing, the affordable runtime is limited. It is usual and preferable that several optimization modes are provided to users, for example, fast mode and slow mode. Thus users can tradeoff runtime and quality. Moreover, fast mode optimization can allow designer to check design feasibility and do trial-and-error analysis. Hierarchical approach is a natural way to improve runtime without sacrificing quality too much. Toward this purpose, we build a hierarchical router to achieve acceptable quality using much less runtime. The hierarchical router can also be used in the applications where the number of sinks is very large.

It should be noted that, although the approach presented in the paper is mainly based on the algorithm in [10], the idea can also be applied to other buffered routing algorithms to tradeoff routing resource, runtime and quality.

2. Review

Notice that a buffered routing tree is a set of buffer branches plus the connecting buffered paths. A buffered path is a set of connected wire paths (refer to Figure 2). Both wire path and buffer branch can be pre-computed, supposing the switch-level RC model is used for

*This work was partially supported by the National Science Foundation under grants CCR-0244236 and CCR-0306244.

buffer.

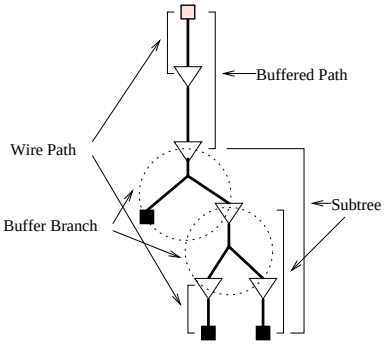


Figure 2: A buffered routing tree is decomposed into a set of buffer branches plus the connecting buffered paths. A buffered path is a set of connected wire paths.

The graph-based approach [10] first constructs a graph (called BP graph) to compute the optimal buffered path between any two nodes as in [6]. In the graph, each vertex represents a possible buffer choice at different location which allows to insert a buffer. The edge between two vertices (possible buffer insertion) represents an optimal wiresized path, and its weight representing the buffer-to-buffer delay can be obtained by looking up the pre-computed table. Therefore, a shortest path between any pair of vertices is an optimal buffered path connecting the two vertices of the pair. The selected vertices in the path represent buffer insertions. Thus, all-pair shortest path algorithm is applied to obtain the optimal buffered path between any pair of vertices.

For a net with multiple sinks, the remaining problem is how to determine buffer branch locations. The approach begins with considering 2 sinks (let us denote as t_1, t_2). The subtree connecting the two sinks contains one branch. All subtrees, which branch at v (i.e., v is the driver of the buffer branch), can be obtained by enumerating the possible branches at vertex v. Note that the optimal buffered path between any pair of nodes is already computed. Then the best subtree branching at v is picked up. The process proceeds in this way for every node. Then a graph (called G_Γ graph) is constructed as shown in Figure 3, where the source vertex is $\Gamma = \{t_1, t_2\}$, and other vertices are possible buffer nodes. The edge (Γ, w) represents the optimal subtree branching at w, and its weight is the maximum delay of the subtree. Then each path from Γ to each other vertex v corresponds to a subtree connecting v with t_1 and t_2 as shown in Figure 3(a), and therefore the shortest path represents the optimal subtree connecting to t_1 and t_2 as shown in Figure 3(b). Thereafter, a single-source shortest path algorithm (such as Dijkstra's algorithm) is applied to get all optimal subtrees for every node. When more than 2 sinks are concerned, first all optimal subtrees are constructed for every node to connect every 2 sinks, and then another graph is constructed to determine another buffer branch and to compute the subtree for every node, which connects to 3 or more sinks. The algorithm proceeds to create subtrees by increasingly considering more sinks. The final solution is obtained by using the back-tracing information to determine the branches and construct the whole tree.

3. Tradeoff Delay vs. Cost

For simplicity, we represent routing cost using the following formula $C = W + \alpha B$ where W is the total wire area, B is the total number of buffers and α is the factor for balancing between W and B. Note that the cost calculation is quite flexible. Our formulation is independent of the cost model. Other cost models such as computing capacitance can be used to represent cost.

3.1 Dominance

Each solution is characterized by two parameters D and C. They can be compared with a partial order. Given two solutions, S_1 and S_2, characterized by (D_1, C_1) and (D_2, C_2) respectively, if $D_2 > D_1$ and $C_2 > C_1$, then the solution S_2 is said to be dominated by S_1. The set of non-dominated solutions composes a delay vs. cost **tradeoff curve**. The dominated solutions lie above the curve.

In the following, we first study the problem of minimizing the linear combination of delay and cost. The graph-based algorithm is extended to solve the problem optimally. We then show that a variant of the algorithm can efficiently meet other optimization goals.

3.2 Graph-based Algorithm

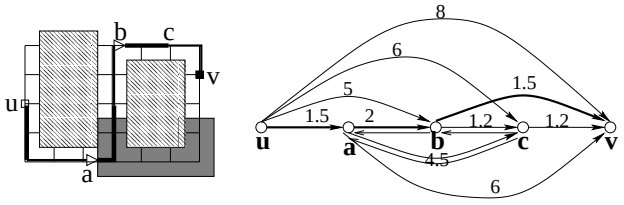


Figure 4: A shortest path between u and v represents an optimal buffered path with respect to the value $D + \lambda C = \sum d + \lambda \sum c$ over the edges in the path.

A routing tree is decomposed into a set of components, wire paths and buffer branches. Each wire path is associated with a cost: $w + \alpha$ where w is the wire area of the wire path and the number of buffers in wire path is one. Each buffer branch is associated with a cost: $w + \alpha r$ where w is the wire area of the buffer branch and r is the number of receivers in the buffer branch. Note that the upstream buffer of wire path or buffer branch should not be counted. A wire path or buffer branch may connect to sink. We regard a sink as a buffer. This does not affect the optimization result, since it just adds to the objective function a constant value, αk where k is the number of sinks.

Then we build graphs using the value of $d + \lambda c$ as the weight of an edge where d is the delay value and c is the routing cost of the edge. Thus in the BP graph, a shortest path between any pair of nodes is an optimal buffered path connecting the two nodes with respect to the value $D + \lambda C = \sum d + \lambda \sum c$ over the edges in the particular path, as shown in Figure 4. In the G_Γ graph, referring to Figure 3, a shortest path from source Γ to each other vertex v represents the optimal subtree connecting v to the set of sinks with respect to the value $D + \lambda C$. Therefore, the graph-based algorithm will minimize the objective function, $D + \lambda C$.

By adjusting the factor λ, we can sample the points of solutions and thus build a delay vs. cost curve.

3.3 Cost Minimization

Given a routing grid graph, a net with several sinks and a specified delay bound D_r, the goal is to find a routing tree solution with (D, C) for the net such that the total cost C is minimized subject to the delay constraint $D \leq D_r$.

Lemma 1. *The optimal solution of cost minimization problem lies on the delay vs. cost tradeoff curve.*

Due to space limitation, the proof, as well as the proofs in the rest of the paper, is omitted.

We use the algorithm of minimizing the linear combination of delay and cost to search for a solution. The idea is to start with an initial conjecture value for λ, and iteratively correct the value until we find a final value of λ (denoted as λ_o) which minimizes C subject to $D \leq D_r$.

Starting from the initial conjecture of λ, for each edge we assign edge weight $d + \lambda c$ where d is the delay value of the edge and c is the cost of the edge. Then we apply the graph-based algorithm to find a solution that minimizes $D + \lambda C = \sum d + \lambda \sum c$. Then we take one of the following actions with respect to D.

1. If D is equal to D_r or approximately equal to D_r, then we have the current $\lambda = \lambda_o$ and a solution that minimizes the routing cost subject to the delay constraint. The process is done.

2. If D is less than D_r, then we increase the value of λ, apply the graph-based algorithm with the new value λ and repeat the process until we find a satisfactory solution.

3. If D is greater than D_r, then we decrease the value of λ, apply the graph-based algorithm with the new value λ and repeat the process.

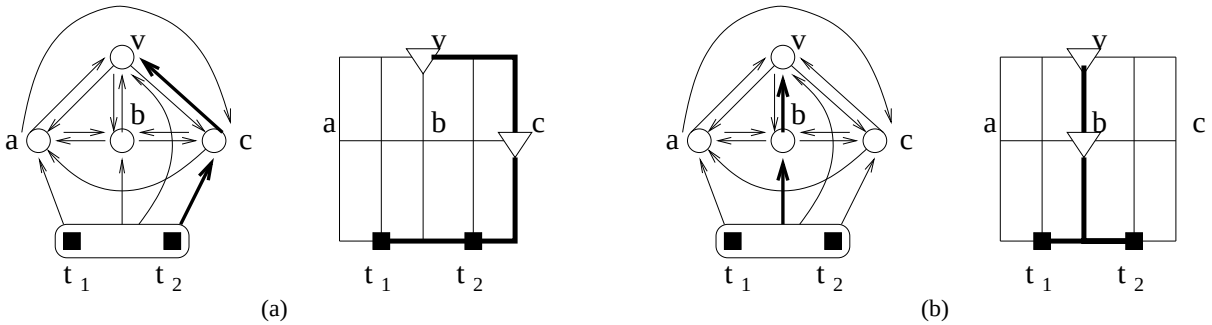


Figure 3: (a) A path from $\Gamma = \{t_1, t_2\}$ to each other vertex v corresponds to a subtree connecting v with t_1 and t_2. (b) The shortest path represents the optimal subtree connecting to t_1 and t_2. The edge $(\{t_1, t_2\}, b)$ represents the optimal subtree branching at b.

We use binary search to find the appropriate λ as in [4]. Beginning with an initial λ, if the resulting $D < D_r$, we keep doubling λ (i.e. using $\lambda' = 2\lambda$) until we get $D \geq D_r$. On the other hand, if the initial λ makes $D > D_r$, we keep halving λ (i.e. using $\lambda' = \lambda/2$) until we reach $D \leq D_r$. In this way, we can determine two values λ_{lower} and λ_{upper} such that the corresponding delay values are less than and greater than the reference delay D_r respectively. Then we try a new value of λ', $\frac{\lambda_{lower}+\lambda_{upper}}{2}$, to determine the resulting D. If $D < D_r$, then $\lambda_{lower} = \lambda'$. Otherwise, $\lambda_{upper} = \lambda'$. We keep trying until the final λ_o is obtained.

Obviously, the number of iterations is bounded by $O(\log \lambda_{max})$ where λ_{max} is the maximum value of λ.

3.4 Delay Minimization

Given a routing grid graph, a net with several sinks and a specified cost bound C_r, the goal is to find a routing tree solution with (D, C) for the net such that the delay D is minimized subject to the cost constraint $C \leq C_r$.

Similarly, we have the following result.

Lemma 2. *The optimal solution of delay minimization problem lies on the delay vs. cost tradeoff curve.*

Analogously, we can search for a solution for the problem by adjusting λ in minimizing the linear combination of delay and cost.

3.5 Delay Reduction to Cost Ratio Maximization

Given a routing grid graph, a net with several sinks and a specified delay bound D_r, the goal is to find a routing tree solution with (D, C) for the net such that the ratio $\frac{D_r - D}{C}$ is maximized and $D \leq D_r$.

Let λ_o denote the maximum ratio. Thus we have $\lambda_o = \frac{D_r - D}{C}$, i.e., $D + \lambda_o C = D_r$. It implies that $\lambda_o = \max\{\lambda | D + \lambda C \leq D_r\}$.

For the problem of maximizing delay reduction to cost ratio, we have the following lemma.

Lemma 3. *For any given D_r, the optimal solution of delay reduction to cost ratio maximization problem lies on the delay vs. cost tradeoff curve.*

Similarly, we can use binary search to find the value of λ_o for maximizing delay reduction to cost ratio.

It should be noted that the problem of maximizing the delay reduction to cost ratio is easier than the other two, minimizing routing cost subject to a delay constraint and minimizing delay subject to a cost budget. The algorithm is capable of finding the optimal solution for maximizing the delay reduction to cost ratio. However, it is just a heuristic to both minimizing routing cost subject to a delay constraint and minimizing delay subject to a cost budget, unless the curve is strictly convex. In the following, we prove if the curve is strictly convex, the algorithm is able to optimally solve the problem of minimizing cost subject to a delay constraint or minimizing signal delay subject to a cost budget.

Let us first define what is strictly convex. The delay vs. cost (DC) curve is strictly convex[1], if for any three points on the curve, (D_1, C_1), (D_2, C_2) and (D_3, C_3), $D_1 < D_2 < D_3$, it satisfies

$$\frac{C_2 - C_1}{D_2 - D_1} < \frac{C_3 - C_2}{D_3 - D_2}$$

A strictly convex DC curve is shown in Figure 5.

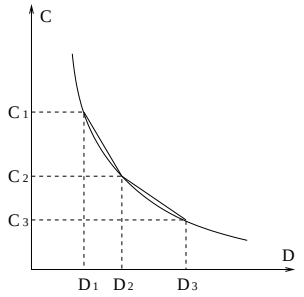


Figure 5: A strictly convex DC curve.

Lemma 4. *Given a solution (D, C) on a strictly convex DC curve, there must exist a value $\lambda_o \geq 0$ such that $D + \lambda_o C$ is an optimal solution in terms of minimizing the linear combination of delay and cost.*

Lemma 5. *λ_o is monotonically increasing as C decreases.*

Based on the above two lemmas (Lemma 4 and Lemma 5), we can easily conclude as follows.

Theorem 1. *If the DC curve is strictly convex, the algorithm, using binary search and minimizing the linear combination of delay and cost, is able to find the optimal solution that minimizes routing cost subject to a delay constraint or minimizes signal delay subject to a cost budget.*

Although many DC curves are not strictly convex, most of them are approximately convex in reality. Thus the algorithm can be applied to find an approximate solution.

4. Tradeoff Quality vs. Runtime

In practice, designers usually prefer several optimization modes to be provided, for example, fast mode and slow mode. Fast mode perspective allows designers, using affordable runtime, to quickly predict the overall performance, analyze the most critical nets, and get very realistic timing budget. Thereafter, designers can use slow-mode to refine the routing results and finalize the design with better quality. In the section, we apply a hierarchical approach for fast-mode optimization. Hierarchical approach starts with a global view of entire net, generates global routing plan, breaks into pieces to refine routing, and then builds up the entire buffered routing. Users can use different granularity of hierarchical approach to tradeoff quality and runtime. Actually, the effective use of hierarchical approach is also crucial in the routing of large fanout nets. The hierarchical approach has two aspects: global routing and clustering.

[1]For a continuous differentiable curve, $y = f(x)$, it is strictly convex if $f''(x) > 0$.

4.1 Global Routing

The technique of global routing is applied to fast capture the global view of entire net. First, we use a rough grid graph, where the sinks near a grid node are grouped into one new sink. In this way, we could reduce not only the number of sinks but also the number of buffer nodes considered for buffer insertion in the graph. Each node in the graph has a capacity. Then, the algorithm is applied to the rough grid graph to get a rough routing tree satisfying the capacity constraint. It should be noted that, by adjusting the granularity of the grid, we can get different runtime vs. quality. After that, to improve solution quality, we can use a fine grid on the rough routing tree obtained before, and compute a new fine routing tree.

4.2 Clustering

We first divide the sinks into a set of clusters. The center of a cluster is regarded as a new sink. Thus we build a buffered routing tree (denoted as t_G for reference) to connect the source to these new sinks. Then for each cluster, a buffered routing tree (denoted as t_c) is constructed to connect the sinks within the cluster. Note that the center of a cluster may not be a good position to locate a new source for the sinks in the cluster. Instead, we use the buffer, which is close to the bounding box of the cluster in the routing tree t_G, to be the new source for the cluster. Then a new routing tree is to be built to connect the new source to the sinks in the cluster. As an example, Figure 6 shows that the sinks are divided into 3 clusters. A routing tree is built to connect the source to the centers of all clusters. The buffer close to the bounding box of a cluster is regarded as the new source for the sinks in the clusters, which forms a subproblem.

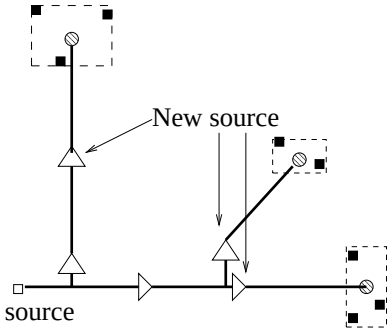


Figure 6: The sinks are divided to 3 clusters (in dashed boxes). The routing tree connects the source to the centers of all clusters. The buffer close to the bounding box of a cluster acts as a new source.

Clustering is a useful "divide and conquer" strategy. Organizing sinks into sensible groupings not only catches the idea of hierarchical design, but also reduces the complexity of our algorithm. Many clustering algorithms are available in [5]. We use Zahn's clustering algorithm, which constructs clusters by breaking minimum spanning tree. The clustering can be done recursively in multi-level for further improving runtime, although in reality two-level clustering is sufficient to handle large designs. At each level of clustering, users can specify a threshold for the number of clusters. Empirically, the more clusters, the better solution quality but consuming more runtime. In this way, users can also tradeoff runtime vs. quality.

5. Experimental Results

We have implemented the graph-based algorithm for minimizing the linear combination of delay and cost, $D + \lambda C$, and tested on different test cases. By adjusting the factor λ, we can build a delay vs. cost tradeoff curve and identify the solution that meets the optimization objective. We use the same technology parameters as in [10]. One type of buffers is used in the program with input capacitance of $23.4fF$ and output resistance of 180Ω. The intrinsic delay of buffer is set to be 36.4ps. There are three choices of wire width in our wire library. Figure 7 shows the delay vs. cost tradeoff curves for two test cases.

We have implemented a hierarchical router based on the approach mentioned above, and tested its performance on Sun Solaris (Ultra10). We compare the hierarchical approach with flat approach in terms of total cost ($D + \lambda C$) and runtime. A fixed λ is used in cost

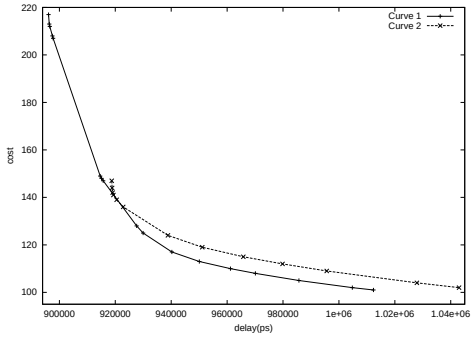


Figure 7: The delay vs. cost tradeoff curves for two test cases.

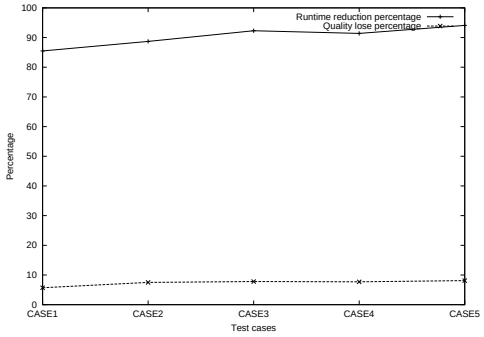


Figure 8: Comparison between hierarchical approach and flat implementation. The percentages of runtime reduction and quality (cost) lose are shown for different test cases.

function. The experiment is based on a set of testcases with various numbers of sinks. Figure 8 illustrates the ratios of runtime and quality (cost) between hierarchical approach and flat implementation. On average, the hierarchical router only loses about 8% quality (cost) but achieves about 90% runtime reduction (i.e., 10 times faster). Our main goal is to demonstrate how to tradeoff routing resource, runtime and quality in buffered routing, and the idea can also be applied to other buffered routing algorithms. Thus the actual runtime is omitted.

6. References

[1] C. Alpert et al., "Buffered Steiner trees for difficult instances", ISPD-01, pp. 4-9, 2001.

[2] M. Hrkic and J. Lillis, "S-tree: a technique for buffered routing tree synthesis", DAC-02, pp. 578-583, 2002.

[3] J. Cong and X. Yuan. "Routing tree construction under fixed buffer locations", DAC-00, pp. 379-384, 2000.

[4] A. Jagannathan, S.W. Hur, and J. Lillis. "A fast algorithm for context-aware buffer insertion", DAC-00, pp. 368-373, 2000.

[5] A.K. Jain and R.C. Dubes. "Algorithms for clustering data", Prentice hall, 1988.

[6] M. Lai and D.F. Wong, "Maze routing with buffer insertion and wire-sizing", DAC-00, pp. 374-378, 2000.

[7] J. Lillis, C.K. Cheng, and T.T.Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model", ICCAD-95, pp. 138-143, 1995.

[8] T. Okamoto and J. Cong. "Buffered Steiner tree construction with wire sizing for interconnect layout optimization", ICCAD-96, pp. 44-49, 1996.

[9] A.H. Salek, J. Lou and M. Pedram, "A simultaneous routing tree construction and fanout optimization algorithm", ICCAD-98, pp. 625-630, 1998.

[10] X. Tang, R. Tian, H. Xiang, and D.F. Wong. "A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints", ICCAD-2001, pp. 49-56, 2001.

[11] Semiconductor Industry Association, National Technology Roadmap for Semiconductors, 1997.

[12] L.P.P.P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay", ISCAS-90, pp. 865-868, 1990.

[13] H. Zhou, D.F. Wong, I.M. Liu, and A. Aziz, "Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations", DAC-99, pp. 96-99, 1999.