A Polynomial Time Approximation Scheme For Timing Constrained Minimum Cost Layer Assignment

Shiyan Hu
Dept. of Electrical and Computer Engineering
Michigan Technological University,
Houghton, Michigan 49931

Zhuo Li
IBM Austin Research Laboratory
11501 Burnet Road,
Austin, Texas 78758

Charles J. Alpert
IBM Austin Research Laboratory
11501 Burnet Road,
Austin, Texas 78758

***Abstract*— As VLSI technology enters the nanoscale regime, interconnect delay becomes the bottleneck of circuit performance. Compared to gate delays, wires are becoming increasingly resistive which makes it more difficult to propagate signals across the chip. However, more advanced technologies (65nm and 45nm) provide relief as the number of metal layers continues to increase. The wires on the upper metal layers are much less resistive and can be used to drive further and faster than on thin metals. This provides an entirely new dimension to the traditional wire sizing problem, namely, layer assignment for efficient timing closure. Assigning all wires to thick metals improves timing, however, routability of the design may be hurt. The challenge is to assign minimal amount of wires to thick metals to meet timing constraints.**

In this paper, the minimum cost layer assignment problem is proven to be NP-Complete. As a theoretical solution for NP-complete problems, a polynomial time approximation scheme is proposed. The new algorithm can approximate the optimal layer assignment solution by a factor of $1+\epsilon$ in $O(m \log\log m \cdot n^3/\epsilon^2)$ time for $0 < \epsilon < 1$, where n is the number of nodes in the tree and m is the number of routing layers. This work presents the first theoretical advance for the timing-driven minimum cost layer assignment problem. In addition to its theoretical guarantee, the new algorithm is highly practical. Our experiments on 500 testcases demonstrate that the new algorithm can run $2\times$ faster than the optimal dynamic programming algorithm with only 2% additional wire.

I. Introduction

As VLSI technology enters the nanoscale regime, interconnect delay becomes the bottleneck of circuit performance since device scaling outpaced interconnect scaling. As a result, interconnect synthesis [1], which consists of optimizations on interconnect including buffer insertion, layer assignment and buffer/wire sizing, becomes prevalent in physical design.

Interconnect synthesis is the construction of wire routes combined with buffering to get signals from one place to the other. In the 1990s ([2], [3], [4], [5], [6], [7]) the interconnect synthesis literature focused on simultaneous buffering and wire sizing. Wire sizing assumes that the resistance and capacitance per square micron is a constant, so doubling the width of a wire halves the resistance but doubles the capacitance. Further, if one doubles the width of the wire, there is one less routing track available, meaning that wire sizing almost certainly hurts routability. Thus, traditionally wire sizing is only sparingly used during practical physical synthesis for timing closure [1].

However, with layer assignment, it is no longer true. There are many more routing layers available (certain 65nm technologies can have eight layers of metal and some 45nm technologies can have ten layers) and if the timing closure tool does not make the layer assignment then the router will. To close on timing for critical nets that need to go long distances, layer assignment needs to be controlled by optimization before routing. Further, the optimal buffering solution of course depends on the parasitics of the metal layers chosen. Because the resources for thick metal already exist, bumping a wire up to thick metal will generally not hurt routability as long as there is enough wiring resource on those planes to handle the assignment. From a timing closure perspective, it is reasonable for the router to decide to bump up wires to thick metal. However, if the router pushes thick metal down to thin metal, it could severely hurt timing. Thus, layer assignment optimization needs to be conservative in thick metal assignment to minimize such "timing surprise".

There are quite few works on layer assignment regardless of its remarkable effect on timing improvement. Recently, the work [1] presents efficient algorithms for simultaneous buffering and layer assignment algorithms for timing closure. However, they are mostly focused on heuristics without theoretical proofs which makes the problem less understood in theory. This work aims to advance the understanding of layer assignment problem from a theoretical point of view.

In this paper, layer assignment problem is formulated as using minimum amount of wire resources to meet a timing target for a buffered routing tree. Similar to uniform wire sizing [6], layer assignment in this paper will assign the entire wire to the same layer. Note that only wire layer but neither buffer size nor location can be changed. This formulation is particularly interesting when design proceeds to the late stage of a physical synthesis flow such as Placement Driven Synthesis (PDS) flow [8]. At such stage, tuning existing layout (i.e., repowering) or adding significant ECO change (i.e., rebuffering) are generally not desired or even prohibited due to physical space constraints, or simply the fact that all placement based optimizations may have been applied to the maximum degree in previous stages. On the other hand, though, one may still possibly improve the timing by only tuning layer assignment on each wire without affecting the placement. It could also be used to minimize the thick metal resources by properly demoting wires from higher layers to lower layers (while still meeting timing budget) to make design more routing-friendly. Further, in the formulation, one has the freedom to model wire resources for addressing the needs in late design stage. For example, the cost metric could be wire area, wire congestion estimation or a combination of them.

In this paper, we prove that the timing constrained minimum cost layer assignment problem is NP-complete. We then propose a polynomial time approximation (PTAS) scheme[1] for the problem, which is based on constructing an oracle such that the comparison of any number with the optimal layer assignment cost can be answered without knowing the optimal layer assignment. Unlike many PTAS algorithms which are complicated and not practical, our PTAS algorithm performs very well in practice. The main contribution of this paper is summarized as follows.

[1]A polynomial time approximation scheme (PTAS) refers to an algorithm which is able to compute a solution at most $1+\epsilon$ times worse than the optimal solution with runtime polynomial in the input size of the problem instance and inversely proportional to ϵ. Given an NP-complete problem, a PTAS is generally considered as an ultimate solution in theory.

- The timing constrained minimum cost layer assignment problem is proven to be NP-complete.
- A new PTAS algorithm is proposed to approximate the optimal solution by a factor of $1+\epsilon$ in $O(m \log\log m \cdot n^3/\epsilon^2)$ time for $0 < \epsilon < 1$.
- This work presents the first theoretical advance for the timing-driven minimum cost layer assignment problem.
- In contrast to most PTAS algorithms, our PTAS algorithm is highly practical. Experimental results on 500 testcases demonstrate that the new PTAS algorithm approximates the optimal solution by only 2% additional wire in about half of the runtime.

Note that there are some previous theoretical studies on layer assignment problem such as [10]. However, their problem formulations and proofs are focused on Via minimizations and are thus quite different from ours. In addition, there is no PTAS, which is the main contribution of this paper, has been proposed in layer assignment literature.

II. Preliminaries

Given a buffered routing tree $T = (V, E)$ where V consists of driver, sinks, Steiner nodes and buffer locations, and $E \subseteq V \times V$. Denote by $V_r(T), V_t(T), V_b(T)$ the driver (root), the set of sinks (terminals), and the set of buffers in a tree T, respectively. In most cases, only single driver signal net is of interest in VLSI design, thus, $|V_r(T)| = 1$. Let $n = |V|$. $V_b(T)$ can be computed by various buffering techniques such as [7], [11], [12], [13]. For simplicity, the routing tree T is assumed to be binary in this paper.

A set of m routing layers are given as $L = \{l_1, l_2, \ldots, l_m\}$. Given an edge e on a layer l, denote by $d(e,l)$ the delay of the edge. In this paper, Elmore delay is adopted which is widely used during interconnect synthesis in physical design. That is, $d(e,l) = R_e \cdot (C_e/2 + C_l)$ where R_e, C_e, C_l refer to the edge resistance, edge capacitance and load capacitance, respectively. The wire cost used in this paper is generic. Given an edge e on a layer l, denote by $w(e,l)$ the cost of the edge. For example, wire cost could be defined using wire area or wire congestion estimation. Note that in practice, wire costs are bounded. Given an edge e, the ratio of the maximum possible wire cost to the minimum possible wire cost is a constant, i.e., $\frac{\max_i w(e,l_i)}{\min_j w(e,l_j)} = O(1)$. In this paper, this assumption is relaxed to handle more general cases where the maximum cost ratio is assumed as $\frac{\max_i w(e,l_i)}{\min_j w(e,l_j)} = O(m)$. For example, if wire cost is equal to wire area, for the same wire length, the ratio of the largest wire area over the smallest wire area is m if the set of layers is as $\{1\times, 2\times, 3\times, 4\times, \ldots, m\times\}$. In fact, our PTAS algorithm still works even if we assume that $\frac{\max_i w(e,l_i)}{\min_j w(e,l_j)} = O(2^m)$ or even $O(2^{2^m})$[1].

Each sink in $V_t(T)$ is associated with a required arrival time (RAT), which specifies the latest time a signal can arrive at the sink. The driver $V_r(T)$ has an arrival time. A net is said to satisfy the timing constraint if the arrival time is no greater than the required arrival time for every sink, or equivalently, if RAT of driver is no earlier than its arrival time.

Define *the subtree under layer assignment* of T, denoted by T_a, as a subtree which starts and ends with driver/buffers/sinks and has no other buffer inside. Given any T_a, we call the root (which is a driver or buffer) of T_a as *root*, denoted by $V_r(T_a)$, and all other buffers/sinks as *terminals*, denoted by $V_t(T_a)$.

The cost of a layer assignment for the tree T, called *tree cost*, is defined as the sum of the costs for all edges in T. Our problem is to meet a timing target by layer assignment with minimum tree cost. It can be formulated as follows.

Timing Constrained Minimum Cost Layer Assignment: Given a buffered binary routing tree $T = (V, E)$, a set of routing layers L, and costs of each wire on each layer, to compute a layer assignment solution such that the required arrival time at driver is no earlier than its arrival time and the tree cost is minimized.

Motivated by [13], we prove that the above is NP-complete by reducing from bipartition problem. The proof is omitted due to space limitation.

Theorem 1: The timing constrained minimum cost layer assignment problem is NP-complete.

III. Algorithmic Flow

Our new PTAS is motivated by [9]. In [9], a PTAS is proposed for path based optimization, precisely, for a delay constrained minimum cost single source single destination problem. It does not consider a full routing tree which heavily involves path merging, cannot handle layer assignment and the constraint is set for delay but not slack. To tackle these challenges, a new PTAS algorithm is proposed based on an oracle constructed from a dynamic programming algorithm for slack based layer assignment problem considering path merging in trees.

We first assume that all cost values are positive integers. This is reasonable since they are bounded positive real numbers in practice and we can always scale them up by a single large number to make them all integers. The optimal layer assignment solution will not change due to uniform scaling.

Denote by W^* the tree cost for the optimal solution of the timing constrained minimum cost layer assignment problem. A polynomial time approximation scheme is to compute a solution satisfying the timing constraint with the cost at most $(1+\epsilon)W^*$ in a time polynomial to the tree size and the number of layers. In addition, the time bound needs to be inversely proportional to ϵ, for any positive number ϵ, due to the NP-completeness nature of the problem unless P=NP. For this, an *oracle* will be first constructed such that a query on whether $W^* \geq x$ for any x can be answered in polynomial time *without knowing the value of* W^*. After obtaining such an orale, the minimum cost layer assignment solution can be computed by performing a binary search in the range formed by lower and upper bound of W^*.

The main task is to construct an oracle and then the total runtime of our PTAS will be the number of binary search multiplied by the time for a single oracle run.

IV. Algorithm

A. Dynamic Programming

Before constructing the oracle, we first see how to compute the optimal solution W^* for the layer assignment problem by dynamic programming. Our dynamic programming algorithm looks similar to the one in [7], however, there are underlying critical differences.

In our approach, at a high level, the tree will be processed in a bottom-up fashion. Roughly speaking, layer assignment is first performed to the subtrees T_a linking to sinks where different layer assignments are computed subject to a lowest possible wire cost budget. These partial solutions will be updated by processing the subtrees next to the subtrees in last step. The process is iterated until the driver is reached. During this process, required arrival time will be also propagated from sinks to driver. A layer assignment satisfies the timing constraint if and only if the required arrival time at driver is no earlier than the arrival time at driver. If the timing constraint

[1]Our PTAS runs in $O(m \log\log m \cdot n^3/\epsilon^2)$ time where the first m refers to the number of layers and the second m refers to the maximum cost ratio.

is met, the cost will be returned as the optimal cost. Otherwise, the whole procedure will be performed again with an incremented wire cost budget.

For any vertex v in T, a function $q(v, w)$ is defined as the largest possible required arrival time at v with cost budget w (precisely, with total cumulative cost for the subtree rooted at v no greater than w). In dynamic programming, since every cost value is an integer, the best timing-driven solution is computed with increasing integer cost budget (i.e., $w = 1, 2, \ldots$). The algorithm stops when we find the first solution satisfying the timing constraint and that cost budget will be the optimal cost W^*.

In the algorithm, suppose that there is a subtree under layer assignment T_a where all terminals of T_a have been processed and the root is not yet processed. That is, $q(v, w)$ has been computed for each $v \in V_t(T_a)$. $q(V_r(T_a), w)$ is to be computed. For this, we start with each terminal and propagate $q(v, w)$ using two operations, namely, "Add Wire" and "Merge". Note that $q(v, w)$ are propagated layer by layer and there is no cross-layer propagation since wire tapering is not desired [6]. For clarity, denote by $q_l(v, w)$ the $q(v, w)$ corresponding to layer l.

"Add Wire": Suppose v_j has been processed, its immediate upstream vertex v_i, which is not a branching point, is to be processed. $q_l(v_i, w)$ can be computed as

$$\begin{aligned} q_l(v_i, w) \quad &= \quad \max\{q_l(v_j, w - w(e(v_i, v_j), l)) \\ &\quad - d(e(v_i, v_j), l), q_l(v_i, w-1)\}, \end{aligned} \tag{1}$$

where $d(e, l)$ refers to the delay for wire e at layer l, $w(e, l)$ refers to the cost for wire e at layer l, and $q_l(v_i, w)$ is set to the required arrival time of v_i when v_i is a sink. Note that all edge delays $d(e, l)$ can be pre-computed before the dynamic programming algorithm since no driver/buffers/sinks can be changed and only same-layer wires can be connected. This takes totally $O(mn)$ time and it is a one-time cost. An "Add Wire" operation takes constant time since all entries of previously computed $q(v, w)$ can be stored in a two-dimensional array and an access takes constant time. $\max$ is taken to find the best solution subject to the cost budget.

"Branch Merge": Suppose that v_j and v_k have been processed and they share the common immediate upstream vertex v_i which is a branching point. For the ease of illustration, as in [7], two vertices $v_{i,l}$ and $v_{i,r}$ are created at the same location of v_i. They need to be created only once at $w = 1$ and later iteration will use/update q values. $q_l(v_{i,l}, w)$ and $q_l(v_{i,r}, w)$ are computed using "Add Wire" operations. $q(v_i, w)$ is updated as

$$\begin{aligned} q_l(v_i, w) \quad &= \quad \max\{q_l(v_i, w-1), \max\nolimits_{w_l + w_r = w} \\ &\quad \min\{q_l(v_{i,l}, w_l), q_l(v_{i,r}, w_r)\}\}, \end{aligned} \tag{2}$$

where w_l and w_r are nonnegative integer costs. Thus, a "Branch Merge" takes $O(w)$ time (note that two "Add Wire" operations are counted when bounding the complexity for all "Add Wire" operations) since one needs to compute the cases for $w_l = 0, 1, \ldots, w$. It has to be performed in this way since solutions may be the same for current T_a but still very different in history. Note that $\min$ is taken since one needs to guarantee the worst-case performance of two branches. It is clear that processing a subtree T_a needs $O(w|V_t(T_a)|)$ time per layer and the total time is $O(wm|V_t(T_a)|)$.

Note that our algorithm is quite different from [7]. In [7], all (Q, C, W) are computed in a single run of dynamic programming and the optimal cost solution satisfying the timing constraint is picked at driver, where Q refers to slack, C refers to downstream capacitance and W refers to cost. Since it will explore all non-redundant solutions, the number of maximum cost value (and the number of solutions) could be much larger than W^*. In our case, (q, W) are built for each w and w is increased until a solution satisfying the timing constraint is obtained.

The algorithm is now clear. We start with $w = 1$. Process T_a immediate upstream to sinks using the above operations, and then process T_a next to them. After processing the driver $V_r(T)$, a second iteration will be performed with $w = 2$. This process continues until a processing of driver finds that RAT at diver $q(V_r(T), w)$ is smaller than the arrival time at driver and then w is returned as the optimal cost W^*. For each w, $O(mnw)$ time is needed since $\sum_{T_a} |V_t(T_a)| = O(n)$ and note that all "Add Wire" operations need $O(mn)$ time. Thus, the total runtime is bounded by $O(mnW^{*2})$.

B. Oracle Construction

Recall that given any integer x, the oracle can decide whether $x \geq W^*$. Such an oracle will be constructed based on the above dynamic programming which can find the optimal solution. The oracle is motivated by [9] which solves a different problem.

To construct the oracle, for any positive number ϵ (generally we assume that $\epsilon < n$), we first scale each wire cost (for each layer), but not wire delay, in T by $\frac{x\epsilon}{n}$. Precisely, for each edge e in T, wire cost $w(e, l)$ is scaled to $\lfloor \frac{w(e,l)n}{x\epsilon} \rfloor$.

The dynamic programming algorithm is performed to the scaled graph with the same timing constraint as in original graph but with a different cost budget bound of n/ϵ. That is, the above dynamic programming is performed for each w until w reaches n/ϵ. There are two cases.

- A layer assignment which satisfies the timing constraint is found for $w \leq n/\epsilon$. The total cost is no more than n/ϵ in the scaled graph. In the original graph, its cost will be smaller than $\frac{n}{\epsilon} \cdot \frac{x\epsilon}{n} + x\epsilon = (1+\epsilon)x$ by noting that the rounding error is at most $\frac{x\epsilon}{n}$ for each edge cost and is smaller than $x\epsilon$ for the whole tree cost over $n-1$ edges. Thus, there is a solution with cost smaller than $(1+\epsilon)x$ which can satisfy the timing constraint. It means that $W^* < (1+\epsilon)x$.
- The algorithm proceeds to $w = \lceil n/\epsilon \rceil$ and timing constraint cannot be met for any computed layer assignment solution. In the original graph, this means that any layer assignment with cost at most $\frac{n}{\epsilon} \cdot \frac{x\epsilon}{n} = x$ cannot satisfy timing constraint. Thus, $W^* \geq x$.

For the time complexity of the oracle, since the algorithm stops when w reaches n/ϵ, the runtime is bounded by $O(mn^3/\epsilon^2)$.

C. Fast Logarithmic Scale Binary Search

We are to describe a fast logarithmic scale binary search technique used in [15], [9]. One can use the oracle to make the query for $x \in [W_l^*, W_u^*]$. When $W^* < (1+\epsilon)x$, the upper bound will be reset to $(1+\epsilon)x$. When $W^* \geq x$, the lower bound will be reset to x. Recall that initially $\frac{W_u^*}{W_l^*} = O(m)$ as indicated in Section II. As in [15], [9], one can set $x = \sqrt{\frac{W_l^* \cdot W_u^*}{1+\epsilon}}$. There are two cases after an oracle query with x.

- The upper bound is $(1+\epsilon)x = \sqrt{W_l^* \cdot W_u^* \cdot (1+\epsilon)}$ while the lower remains as W_l^*.
- The lower bound is $x = \sqrt{\frac{W_l^* \cdot W_u^*}{1+\epsilon}}$ while the upper bound remains as W_u^*.

In either case, the ratio of new upper and lower bounds is $\sqrt{\frac{W_u^*}{W_l^*} \cdot (1+\epsilon)} = \sqrt{m \cdot (1+\epsilon)}$. x can be then set as above using the new bounds and the ratio becomes $m^{1/4}(1+\epsilon)^{(1/2+1/4)}$. In general, one can prove that after i oracle queries, the ratio between upper and lower bounds is

$$m^{\frac{1}{2^i}} \cdot (1+\epsilon)^{(\sum_i \frac{1}{2^i})}. \tag{3}$$

Since $\sum_i \frac{1}{2^i} < 1$ and $1 + \epsilon > 1$, the second term is smaller than $(1 + \epsilon)$. We are to show how many queries are needed such that $m^{\frac{1}{2^i}} \leq 2$. This happens after $i = \log \log m$ queries to oracle. The ratio of upper and lower bounds is then at most $2(1+\epsilon)$, i.e., $W_u^* \leq 2(1+\epsilon)W_l^*$.

We focus on the runtime complexity when $0 < \epsilon < 1$, which is of particular interest in PTAS since we always wish to compute a solution approximating the optimal solution by an arbitrarily small ϵ. When $\epsilon \leq 1$, $W^*u \leq 4W_l^*$. At this moment, one just needs to scale edge cost by $\frac{W_l^* \epsilon}{n}$, precisely, scale each $w(e,l)$ to $\lfloor \frac{w(e,l)n}{W_l^* \epsilon} \rfloor$ and use the dynamic programming algorithm in Section IV-A to compute the optimal solution in the scaled graph. It will be able to find the optimal solution before the cost reaches $\lceil 4n/\epsilon \rceil$, which needs $O(mn^3/\epsilon^2)$ time. On one hand, if there is no rounding errors when scaling the edge costs, the optimal paths will be same in both graphs. On the other hand, if there are rounding errors, the costs of two optimal paths will be differed by at most $\frac{W_l^* \epsilon}{n} \cdot (n-1) < W_l^* \epsilon \leq W^* \epsilon$. Thus, the layer assignment solution on the scaled graph gives a layer assignment with cost at most $1 + \epsilon$ times the cost of the optimal solution in the original graph. It is clear that the total runtime of our PTAS is bounded by performing $\log \log m$ oracle queries where each query needs $O(mn^3/\epsilon^2)$ time. We reach the following theorem.

Theorem 2: A $(1 + \epsilon)$ approximation to the timing constrained minimum cost layer assignment problem can be computed in $O(m \log \log m \cdot n^3/\epsilon^2)$ time for any $0 < \epsilon < 1$, where n is the number of nodes in the tree and m is the number of routing layers.

V. Experimental Results

The new PTAS algorithm for the timing constrained minimum cost layer assignment problem is implemented in C++. We compare the new algorithm to the dynamic programming algorithm which computes the optimal layer assignment solution. The experiments are performed to a set of 500 buffered nets. The wire cost is measured by scaled wire area in this paper. However, other metric can be easily incorporated in our PTAS algorithm.

TABLE I

Comparison of the optimal dynamic programming algorithm and the new PTAS algorithm. For the dynamic programming solution, the optimal total wire cost is 102513 and CPU is 295.2 seconds. Approximation Ratio and Speedup are computed by comparing to the dynamic programming algorithm.

New PTAS				
Approx. Ratio ϵ	Total Cost	CPU(s)	Actual Approx. Ratio	Speedup
0.05	104734	163.5	2.2%	1.8×
0.10	107895	122.1	5.3%	2.4×
0.20	115574	119.4	12.7%	2.5×
0.30	126941	112.2	23.8%	2.6×
0.40	133449	109.7	30.2%	2.7×
0.50	151112	108.8	47.4%	2.7×

The comparison of our PTAS algorithm and the dynamic programming algorithm is summarized in Table I. PTAS algorithms are often very complicated and only few (e.g., [15]) of them are practical. As one can see from Table I, the new PTAS works well in practice. It well approximates the optimal solution and often obtains $> 2\times$ speedup compared to dynamic programming. For example, when approximation ratio $\epsilon = 0.05$, the new algorithm runs about $2\times$ faster than dynamic programming with only 2% additional wire cost. The speedup comes from fast logarithmic scale binary search. The most important feature of our algorithm is that the new PTAS has theoretical guarantees on approximation ratio. The actual approximation ratio computed by comparing the obtained solution to the optimal solution indicates that the new PTAS performs actually better than its theoretical guarantee. It is also clear from Figure 1 that the runtime is inversely proportional to ϵ.

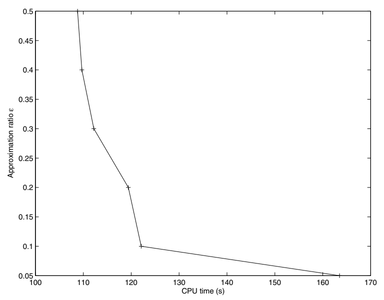

Fig. 1. CPU time (s) v.s. Approximation ratio ϵ.

References

[1] Z. Li, C. Alpert, S. Hu, T. Muhmud, S. Quay, and P. Villarrubia, "Fast interconnect synthesis with layer assignment," *ISPD*, 2008.

[2] J. Fishburn and C. Schevon, "Shaping a distributed-rc line to minimize elmore delay," *IEEE Trans. on Circuits and Systems-I*, vol. 42, no. 12, pp. 1020–1022, 1995.

[3] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 7, pp. 1014–1025, 1999.

[4] J. Cong, K.-S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed rc delay model," *DAC*, pp. 606–611, 1993.

[5] J. Cong and K.-S. Leung, "Optimal wire sizing under elmore delay model," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 3, pp. 321–336, 1995.

[6] C. Alpert, A. Devgan, J. P. Fishburn, and S. T. Quay, "Interconnect synthesis without wire tapering," *IEEE Transactions on Computer-Aided Design*, vol. 20, no. 1, pp. 90–104, 2001.

[7] J. Lillis and C.-K. Cheng and T.-T.Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE Journal of Solid State Circuits*, vol. 31, no. 3, pp. 437–447, 1996.

[8] C. Alpert, S. Karandikar, Z. Li, G.-J. Nam, S. Quay, H.Ren, C. Sze, P. Villarrubia, and M. Yildiz, "Techniques for fast physical synthesis," *Proceedings of IEEE*, vol. 95, no. 3, pp. 573–599, 2007.

[9] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36–42, 1992.

[10] M. Marek-Sadowska, "An unconstrained topological via minimization problem for two-layer routing," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 3, pp. 184–190, 1984.

[11] S. Hu, C. J. Alpert, J. Hu, S. Karandikar, Z. Li, W. Shi, and C. N. Sze, "Fast algorithms for slew constrained minimum cost buffering," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 2009–2022, 2007.

[12] Z. Li and W. Shi, "An $O(bn^2)$ time algorithm for optimal buffer insertion with b buffer types," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 484–489, 2006.

[13] W. Shi, Z. Li and C. J. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," *ASPDAC*, pp. 609–614, 2004.

[14] M. Garey and D. Johnson, "Computers and intractability: A guide to the theory of np-completeness," 1979.

[15] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial time approximation algorithms for multi-constrained qos routing," *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 656-669, 2008.