A Fully Polynomial-Time Approximation Scheme for Timing-Constrained Minimum Cost Layer Assignment

Shiyan Hu, Zhuo Li, and Charles J. Alpert, *Fellow, IEEE*

***Abstract*—As VLSI technology enters the nanoscale regime, the interconnect delay becomes the bottleneck of circuit performance. Compared with gate delays, wires are becoming increasingly resistive, making it more difficult to propagate signals across the chip. However, more advanced technologies (65 and 45 nm) 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, the routability of the design may be hurt. The challenge is to assign a minimal amount of wires to thick metals to meet timing constraints. In this brief, the minimum cost layer assignment problem is proven to be NP-complete. As a theoretical solution for NP-complete problems, a fully 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, m is the number of routing layers, and M is the maximum cost ratio among 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 industrial test cases demonstrate that the new algorithm can run 2× faster than the optimal dynamic programming algorithm, with only 2% additional wire.**

***Index Terms*—Fully polynomial-time approximation scheme (FPTAS), interconnect synthesis, layer assignment, NP-complete.**

I. Introduction

As VLSI technology enters the nanoscale regime, the 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 another. In the 1990s [2]–[6], the interconnect synthesis literature focused on simultaneous buffering and wire sizing. Wire sizing assumes that the resistance and capacitance per square micrometer are constants; thus, doubling the width of a wire halves the resistance but doubles the capacitance. Furthermore, if one doubles the width of the wire, there is one less routing track available, which means 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, the foregoing discussion is no longer true. There are many more routing layers available (certain 65-nm technologies can have eight layers of metal, and some 45-nm technologies can have ten layers), and if the timing closure tool does not make layer assignment, then the router will do it. To close on timing for critical nets that need to go long distances, layer assignment needs to be controlled by optimization before routing. Furthermore, the optimal buffering solution of course depends on the parasitics of the chosen metal layers. Because the resources for thick metals already exist, bumping a wire up to thick metals 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 metals. However, if the router pushes thick metals down to thin metals, 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 a few works on layer assignment regardless of its remarkable effect on timing improvement. Recently, [1] has presented efficient algorithms for simultaneous buffering and layer assignment. 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 the layer assignment problem from a theoretical point of view.

In this brief, the layer assignment problem is formulated as using a minimum amount of wire resources to meet a timing target for a buffered routing tree. In the problem, only the wire layer but neither the buffer size nor the location can be changed. An important assumption is made, which states that, between any consecutive buffers, all wires need to be assigned to the same layer [5]. A layer here does not refer to the traditional wire layer; rather, it is defined based on RC parasitics since what really matters for layer assignment is the ability to switch parasitics. While nonuniform wire sizing or cross-layer assignment is useful in the postrouting stage, the assumption is reasonable for the early stage of the design flow. It is shown in [5] that when buffering is considered, uniform wire sizing can almost achieve the same quality compared with buffering with nonuniform wire sizing.

Manuscript received November 18, 2008; revised February 7, 2009. First published May 27, 2009; current version published July 17, 2009. This paper was recommended by Associate Editor L. Lavagno.

S. Hu is with the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton, MI 49931 USA (e-mail: shiyan@mtu.edu).

Z. Li and C. J. Alpert are with the IBM Austin Research Laboratory, Austin, TX 78758 USA (e-mail: lizhuo@us.ibm.com; alpert@us.ibm.com).

Digital Object Identifier 10.1109/TCSII.2009.2022203

In this brief, we prove that the timing-constrained minimum cost layer assignment problem is NP-complete. Recall that a fully polynomial-time approximation scheme (FPTAS) refers to an algorithm that is able to compute a solution at most $1+\epsilon$ times worse than the optimal solution with a runtime polynomial in the input size of the problem instance and $1/\epsilon$. We propose an FPTAS for the layer assignment 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 FPTAS algorithms that are complicated and impractical, our FPTAS algorithm works well in practice. Note that there are some previous theoretical studies on the layer assignment problem, such as [9]. However, their problem formulations and proofs are focused on via minimizations and are thus quite different from ours. In addition, there is no FPTAS, which is our main contribution.

II. Preliminaries

Given a buffered routing tree $T=(V,E)$, where V consists of a driver, sinks, Steiner nodes, and buffer locations, and $E \subseteq V \times V$. Denote by $V_r(T)$, $V_t(T)$, and $V_b(T)$ the driver (root), the set of sinks (terminals), and the set of buffers in a tree T, respectively. In most cases, only a 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 [6], [10], and [11]. For simplicity, the routing tree T is assumed to be binary in this brief. A general routing tree can be converted to a binary tree using the techniques in, e.g., [11].

A set of m routing layers is 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. The proposed techniques will mainly be applied in the physical synthesis context, where excessive timing evaluations are needed. Thus, fast Elmore delay-based timing estimation is used. That is, $d(e,l)=R_e\cdot(C_e/2+C_l)$, where R_e, C_e, and C_l refer to the edge resistance, edge capacitance, and load capacitance, respectively. The wire cost used in this brief is generic. Given an edge e on a layer l, denote by $w(e,l)$ the cost of the edge. For example, the wire cost could be defined using wire area, wire congestion estimation, or a combination of the two. In this brief, wire costs are assumed to be bounded.

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 RAT for every sink, or equivalently, if the RAT of the driver is no earlier than its arrival time.

Define *the subtree under layer assignment* of T, denoted by T_a, as a subtree that 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 a buffer) of T_a as *root*, denoted by $V_r(T_a)$, and all other buffers/sinks as *terminals*, denoted by $V_t(T_a)$. Each subtree is associated with a set of candidate layers, which can be formed by considering various constraints such as the maximum capacitance constraint of the driving buffer and available wire resources. For clarity, the assumptions made in this brief are summarized as follows: 1) The technique is applied at the signal net level; 2) no wire shaping is allowed, i.e., wires between consecutive driver/buffers/sinks are assigned to a pair of horizontal and vertical layers with similar RC characteristics; 3) the signal net is binary and has a single driver; and 4) the cost of a wire is bounded.

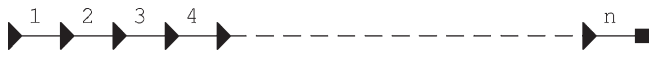

Fig. 1. Tree with one driver, one sink, $n-1$ buffers, and n wires. The input capacitance and the driving resistance for the driver, the sink, and all buffers are zero.

TABLE I
DELAY AND COST VALUES FOR WIRES

	First Available Layer		Second Available Layer	
Wire i	Wire Delay	Wire Cost	Wire Delay	Wire Cost
$wire_1$	x_1	x_2	x_2	x_1
$wire_2$	x_3	x_4	x_4	x_3
...	...	...	...	...
$wire_n$	x_{2n-1}	x_{2n}	x_{2n}	x_{2n-1}

The cost of layer assignment for the tree T, called *tree cost*, is defined as the sum of the costs for all edges in T. Our problem can be formulated as follows.

Timing-constrained minimum cost layer assignment: Given a buffered binary routing tree T, a set of routing layers L, and the costs of each wire on each layer, to compute a layer assignment solution such that the RAT at the driver is no earlier than its arrival time and the tree cost is minimized.

III. NP-Complete Proof

Motivated by [11], the aforementioned problem is shown to be NP-complete by reducing from the bipartition problem. The decision problem can be formulated as checking whether there is a solution with the RAT at the driver no smaller than a RAT constraint and the tree cost no greater than a cost constraint. Given any layer assignment solution, it is easy to verify the aforementioned two conditions in polynomial time. Thus, the problem is in NP. To prove that the problem is NP-hard, we reduce from the NP-complete bipartition problem [12], as follows: Given a set of $2n$ positive integers $X=\{x_1,x_2,\ldots,x_{2n}\}$ and a positive integer N, where $\sum_{i=1}^{2n}x_i=2N$, to decide whether there is an index set I such that either i or $i+1$ is in I, for $i=1,3,5,\ldots,2n-1$, and $\sum_{i\in I}x_i=\sum_{i\notin I}x_i=N$.

Given an instance of the bipartition problem, an instance of the timing-constrained minimum cost layer assignment problem is constructed as follows: In the layer assignment instance, there are one driver, one sink, $n-1$ buffers in between, and n wires. Refer to Fig. 1 for the instance. Assume that the input capacitance and the driving resistance for the driver, the sink, and all buffers are zero. There are two possible routing layers for each wire. Refer to Table I for the characteristics for candidate routing layers for each wire. For example, for the first wire, the delay and the cost of assigning the wire to the first layer are x_1 and x_2, respectively. The delay and the cost for assigning the wire to the second layer are x_2 and x_1, respectively. Note that the constructed instance is also reasonable in practice since a delay–cost tradeoff is often observed between layers. For example, if $x_1>x_2$, this means that the delay is higher and the cost is lower at one layer, whereas the delay is lower and the cost is higher at another layer. The RAT at the sink is set to N, and the arrival time at the driver is set to 0.

We claim that there is a bipartition for the given instance if and only if there is a layer assignment solution with the RAT at least 0 and the tree cost at most N for the constructed instance.

We begin with the "only if" direction. Given a solution for the constructed layer assignment instance, a layer is selected for each wire. Denote by X' the set of delays on wires. Due to the layer characteristics, one of x_i and x_{i+1} is in X' for $i = 1, 3, 5, \ldots, 2n-1$. The path delay is $\sum_{x_i \in X'} x_i$, and the path cost is $\sum_{x_i \in X - X'} x_i$. Subsequently, $N - \sum_{x_i \in X'} x_i \geq 0$ and $\sum_{x_i \in X - X'} x_i \leq N$. Thus, we have $\sum_{x_i \in X} x_i = \sum_{x \in X - X'} x_i = N$, which is a solution for the bipartition problem instance.

We next prove the "if" direction. Given a solution for the bipartition problem instance, one just needs to accordingly set layers in the layer assignment instance, and then, the RAT will be 0 and the cost will be N. We arrive at the following theorem.

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

IV. Algorithmic Flow

Our new FPTAS is motivated by [8] for a minimum cost single-source–single-destination problem. In our layer assignment problem, 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.

Denote by W^* the tree cost for the optimal solution of the timing-constrained minimum cost layer assignment problem. An FPTAS 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 $P = \text{NP}$. 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 oracle, the minimum cost layer assignment solution can be computed by performing a binary search in the range formed by the lower and upper bounds of W^*. The main task is to construct an oracle, and then, the total runtime of our FPTAS will be the number of iterations in the binary search multiplied by the time for a single oracle run.

V. 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 algorithm in [6]; 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 that link to sinks, where different layer assignments are computed subject to the lowest possible wire cost budget. These partial solutions will be updated by processing the subtrees next to the subtrees in the last step. The process is iterated until the driver is reached. During this process, the RAT will also be propagated from sinks to the driver. A layer assignment solution satisfies the timing constraint if and only if the RAT at the driver is no earlier than the arrival time at the driver. If the timing constraint is met, the cost will be returned as the optimal cost. Otherwise, the whole procedure will again be performed with an incremented wire cost budget.

For any vertex v in T, a function $q(v, w)$ is defined as the largest possible RAT at v with the cost budget w (precisely, with the 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.

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 $q(v, w)$ is propagated layer by layer, and there is no cross-layer propagation since it is not desirable, as indicated in [5]. 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 also to be processed. $q_l(v_i, w)$ can be computed as

$$q_l(v_i, w) = \max \{ q_l(v_j, w - w(e(v_i, v_j), l)) - d(e(v_i, v_j), l), q_l(v_i, w-1) \} \quad (1)$$

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 RAT of v_i when v_i is a sink. Note that all edge delays $d(e, l)$ can be precomputed 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 2-D array and an access takes constant time. $\max$ is taken to find the best solution subject to the cost budget. Note that our technique can easily be extended to handle vias. This can be accomplished by computing $d(e(v_i, v_j), l)$ as the delay on both wires and vias.

"Branch Merge": Suppose that v_j and v_k have been processed and that they share the common immediate upstream vertex v_i, which is a branching point. For the ease of illustration, as in [6], 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 iterations 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

$$q_l(v_i, w) = \max \Big\{ q_l(v_i, w-1), \max_{w_l + w_r = w} \min \{ q_l(v_{i,l}, w_l), q_l(v_{i,r}, w_r) \} \Big\} \quad (2)$$

where w_l and w_r are nonnegative integer costs. Thus, a "Branch Merge" operation takes $O(w)$ time (note that two "Add Wire"

w	q(v_{i,l}, w)	q(v_{i,r}, w)
4	10	9
3	8	7
2	7	4
1	5	2
0	0	1

Fig. 2. Branch merge with the current $w = 4$.

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 the 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 $O(wm|V_t(T_a)|)$.

It is helpful to look at a simple example to illustrate our "Branch Merge" operation. In Fig. 2, the cost value w is vertically shown, whereas the q value for each w is horizontally shown. For example, $q(v_{i,l}, 2) = 7$, which refers to the largest RAT at $v_{i,l}$ subject to a cost budget of 2. Branch merge is performed as follows: Currently, $w = 4$. When $w_l = 0$ and $w_r = 4$, q at v_i is $\min\{0, 9\} = 0$. When $w_l = 1$ and $w_r = 3$, q at v_i is $\min\{5, 7\} = 5$. We compute all these q, and their maximum is $\{0, 5, 4, 2, 1\} = 5$. Suppose that it is also larger than $q(v_i, 3)$. We then know that the maximum RAT subject to cost 4 is from merging the solutions with a cost budget of 1 at the left branch and with a cost budget of 3 at the right branch.

Note that our algorithm is quite different from [6]. In [6], 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 the driver, where Q refers to the slack, C refers to the downstream capacitance, and W refers to the cost. Since it will explore all nonredundant solutions, the number of maximum cost values (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 aforementioned 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 processing of the driver finds that the RAT at the driver $q(V_r(T), w)$ is smaller than the arrival time at the 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 aforementioned dynamic programming, which can find the optimal solution. The oracle construction is motivated by [8], 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 the wire delay, in T by $((x\epsilon)/(n))$. Precisely, for each edge e in T, the wire cost $w(e, l)$ is scaled to $\lfloor((w(e, l)n)/(x\epsilon)\rfloor$.

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

1) A layer assignment solution that 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 $(n/\epsilon) \cdot (x\epsilon/n) + x\epsilon = (1 + \epsilon)x$ by noting that the rounding error is at most $x\epsilon/n$ 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 a cost smaller than $(1 + \epsilon)x$ that can satisfy the timing constraint. It means that $W^* < (1 + \epsilon)x$.
2) The algorithm proceeds to $w = \lceil n/\epsilon \rceil$, and the timing constraint cannot be met for any computed layer assignment solution. In the original graph, this means that any layer assignment with the cost at most $(n/\epsilon) \cdot (x\epsilon/n) = x$ cannot satisfy the timing constraint. Thus, $W^* \geq x$.

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

C. Fast Logarithmic Scale Binary Search

Given the oracle, the layer assignment solution can be computed by performing a binary search within the lower bound W_l^* and the upper bound W_u^* of W^*. It can be improved by the following logarithmic scale binary search technique proposed in [8] and [13]: Let $M = W_u^*/W_l^*$, which is the ratio between the initial upper and lower bounds. As in [8] and [13], one can set $x = \sqrt{((W_l^* \cdot W_u^*)/(1 + \epsilon))}$. There are two cases after an oracle query.

1) The upper bound is set to $(1 + \epsilon)x = \sqrt{W_l^* \cdot W_u^* \cdot (1 + \epsilon)}$, whereas the lower bound remains as W_l^*.
2) The lower bound is set to $x = \sqrt{((W_l^* \cdot W_u^*)/(1 + \epsilon))}$, whereas the upper bound remains as W_u^*.

In either case, the ratio of the new upper and lower bounds is $\sqrt{(W_u^*/W_l^*) \cdot (1 + \epsilon)} = \sqrt{M \cdot (1 + \epsilon)}$. x can then be set as aforementioned 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 the upper and lower bounds is $M^{(1/2^i)} \cdot (1 + \epsilon)^{(\sum_i (1/2^i))}$. Since $\sum_i (1/2^i) < 1$ and $1 + \epsilon > 1$, the second term is smaller than $(1 + \epsilon)$. We will show how many queries are needed such that $M^{(1/2^i)} \leq 2$. This happens after $i = \log \log M$ queries to the oracle. The ratio of the 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 since we always wish to compute a solution well approximating the optimal solution. When $\epsilon \leq 1$, $W_u^* \leq 4W_l^*$. At this moment, one just needs to scale the edge cost by $W_l^*\epsilon/n$ (precisely, scale and round each $w(e, l)$

TABLE II
COMPARISON OF THE OPTIMAL DYNAMIC PROGRAMMING ALGORITHM AND THE NEW FPTAS ALGORITHM. FOR THE DYNAMIC PROGRAMMING SOLUTION, THE OPTIMAL TOTAL WIRE COST IS 102 513, AND THE CPU TIME IS 295.2 s. THE ACTUAL APPROXIMATION RATIO AND THE SPEEDUP ARE COMPUTED BY COMPARING WITH DYNAMIC PROGRAMMING

New FPTAS				
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×

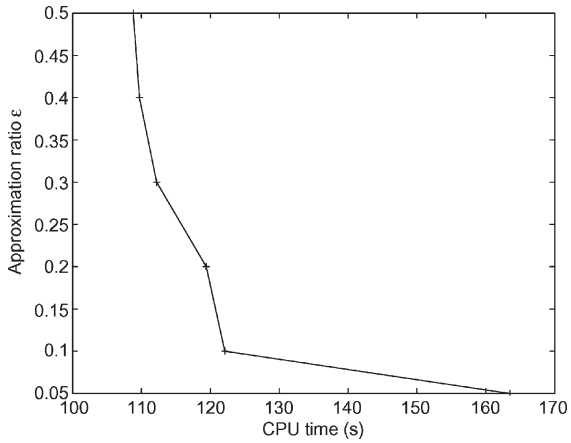

Fig. 3. CPU time (in seconds) versus approximation ratio ϵ.

to $\lfloor((w(e,l)n)/(W_l^*\epsilon))\rfloor$ and use the dynamic programming algorithm in Section V-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 are 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 differ by at most $((W_l^*\epsilon)/(n)) \cdot (n-1) < W_l^*\epsilon \leq W^*\epsilon$. Thus, the solution on the scaled graph gives a layer assignment solution with the cost at most $1+\epsilon$ times the cost of the optimal solution in the original graph. It is clear that the total runtime of FPTAS is bounded by $O(\log \log M)$ oracle queries, where each query needs $O(mn^3/\epsilon^2)$ time. We arrive at 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, m is the number of routing layers, and M is the maximum cost ratio among layers.

VI. EXPERIMENTAL RESULTS

The new FPTAS algorithm for the timing-constrained minimum cost layer assignment problem is implemented in C++. We compare the new algorithm with the dynamic programming algorithm, which computes the optimal layer assignment solution. The experiments are performed on a set of 500 buffered nets extracted from an industrial ASIC chip, and there are eight routing layers. The wire cost is measured by the scaled wire area in this brief. However, other metrics can easily be incorporated in our FPTAS algorithm.

The comparison of our FPTAS algorithm and the dynamic programming algorithm is summarized in Table II. FPTAS algorithms are often very complicated, and only a few (e.g., [13]) of them are practical. As shown in Table II, the new FPTAS works well in practice. It well approximates the optimal solution and often obtains $> 2\times$ speedup compared with dynamic programming. For example, when the 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 the fast logarithmic scale binary search. The most important feature of our algorithm is that the new FPTAS has theoretical guarantees on the approximation ratio. The actual approximation ratio computed by comparing the obtained solution with the optimal solution indicates that the new FPTAS actually performs better than its theoretical guarantee. It is also clear in Fig. 3 that the runtime is inversely proportional to ϵ.

VII. CONCLUSION

This brief has presented the first theoretical advance in the timing-driven minimum cost layer assignment, which is a critical component in interconnect synthesis [1]. We proved that the problem is NP-complete and proposed an FPTAS for the problem. Our experiments demonstrate that the new algorithm runs $2\times$ faster than dynamic programming, with only 2% additional wire.

REFERENCES

[1] Z. Li, C. Alpert, S. Hu, T. Muhmud, S. Quay, and P. Villarrubia, "Fast interconnect synthesis with layer assignment," in *ISPD*, 2008, pp. 71–77.
[2] J. Fishburn and C. Schevon, "Shaping a distributed-RC line to minimize Elmore delay," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 42, no. 12, pp. 1020–1022, Dec. 1995.
[3] C.-P. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 7, pp. 1014–1025, Jul. 1999.
[4] J. Cong and K.-S. Leung, "Optimal wire sizing under Elmore delay model," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 3, pp. 321–336, Mar. 1995.
[5] C. Alpert, A. Devgan, J. P. Fishburn, and S. T. Quay, "Interconnect synthesis without wire tapering," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 1, pp. 90–104, Jan. 2001.
[6] J. Lillis, C.-K. Cheng, and T.-T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE J. Solid-State Circuits*, vol. 31, no. 3, pp. 437–447, Mar. 1996.
[7] 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," *Proc. IEEE*, vol. 95, no. 3, pp. 573–599, Mar. 2007.
[8] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Math. Oper. Res.*, vol. 17, no. 1, pp. 36–42, Feb. 1992.
[9] M. Marek-Sadowska, "An unconstrained topological via minimization problem for two-layer routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-3, no. 3, pp. 184–190, Jul. 1984.
[10] 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. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 11, pp. 2009–2022, Nov. 2007.
[11] W. Shi, Z. Li, and C. J. Alpert, "Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost," in *ASPDAC*, 2004, pp. 609–614.
[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[13] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, "Polynomial time approximation algorithms for multi-constrained QoS routing," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 656–669, Jun. 2008.