

though the proposed method is sound, it is not a complete method, i.e., there are pairs of equivalent circuits for which it cannot prove equivalence. The method can be used as an effective preprocessing step for a general method such as [10]. It is interesting to note that for some synthesis steps, the method is complete. This is, e.g., the case for circuits optimized with combinational synthesis techniques, and also for retimed circuits.

The proposed method assumes that an initial state is designated for both circuits. This initial state is used in two ways: It acts as a reference point to allow the detection of antivalent signals, and it is used to calculate the initial partition T_0 of the set F . The approach of [4] shows that the assumption of a designated initial state can be weakened. It should be possible to extend their work such that it also applies to the method presented in this paper.

ACKNOWLEDGMENT

The author would like to thank W. Kunz and D. Stoffel for providing the circuits of [25].

REFERENCES

- [1] P. Ashar, A. Gupta, and S. Malik, "Using complete-1-distinguishability for FSM equivalence checking," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 346–353.
- [2] D. Brand, "Verification of large synthesized designs," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 534–537.
- [3] J. R. Burch *et al.*, "Symbolic model checking for sequential circuit verification," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 401–424, Apr. 1994.
- [4] J. R. Burch and V. Singhal, "Robust latch mapping for combinational equivalence checking," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 563–569.
- [5] —, "Tight integration of combinational verification methods," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 570–576.
- [6] G. Cabodi *et al.*, "Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits," in *Proc. 34th Design Automation Conf.*, 1997, pp. 728–733.
- [7] H. Cho *et al.*, "Algorithms for approximate FSM traversal," in *Proc. 30th Design Automation Conf.*, 1993, pp. 25–30.
- [8] O. Coudert, C. Berthet, and J. C. Madre, "Verification of synchronous sequential machines based on symbolic execution," *Proc. Workshop Automatic Verification Methods for Finite State Machines*, vol. 407, pp. 365–373, 1989.
- [9] C. A. J. van Eijk and J. A. G. Jess, "Detection of equivalent state variables in finite state machine verification," *Workshop notes Int. Workshop Logic Synthesis*, pp. 3.35–3.44, 1995.
- [10] —, "Exploiting functional dependencies in finite state machine verification," in *Proc. European Design & Test Conf.*, 1996, pp. 9–14.
- [11] C. A. J. van Eijk, "Formal methods for the verification of digital circuits," Ph.D. dissertation, Eindhoven Univ. Technol., Eindhoven, The Netherlands, Sept. 1997.
- [12] T. Filkorn, "Symbolische methoden für die verifikation endlicher zustandssysteme," Ph.D. dissertation, Institut für Informatik der Technischen Universität München, Munich, Germany, 1992.
- [13] S. G. Govindaraju and D. L. Dill, "Verification by approximate forward and backward reachability," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 366–370.
- [14] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "On verifying the correctness of retimed circuits," in *Proc. Great Lakes Symp. VLSI*, 1996, pp. 277–281.
- [15] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "AQUILA: An equivalence verifier for large sequential circuits," in *Proc. Asian South Pacific Design Automation Conf.*, 1997, pp. 455–460.
- [16] J. Jain, R. Mukherjee, and M. Fujita, "Advanced verification techniques based on learning," in *Proc. 32nd Design Automation Conf.*, 1995, pp. 420–426.
- [17] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proc. 34th Design Automation Conf.*, 1997, pp. 263–268.
- [18] W. Kunz and D. Stoffel, *Reasoning in Boolean Networks—Logic Synthesis and Verification Using Testing Techniques*. Amsterdam, The Netherlands: Kluwer, 1997.
- [19] Y. Matsunaga, "An efficient equivalence checker for combinational circuits," in *Proc. 33th Design Automation Conf.*, 1996, pp. 629–634.
- [20] I.-H. Moon *et al.*, "Approximate reachability don't cares for CTL model checking," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 351–358.
- [21] D. K. Pradhan, D. Paul, and M. Chatterjee, "VERILAT: Verification using logic augmentation and transformations," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 88–95.
- [22] S. Quer *et al.*, "Incremental re-encoding for symbolic traversal of product machines," in *Proc. Eur. Design Automation Conf.*, 1996, pp. 158–163.
- [23] R. K. Ranjan *et al.*, "Using combinational verification for sequential circuits," in *Proc. Design, Automation Test Europe Conf.*, 1999, pp. 138–144.
- [24] L. Stok, I. Spillinger, and G. Even, "Improving initialization through reversed retiming," in *Proc. Eur. Design Test Conf.*, 1995, pp. 150–154.
- [25] D. Stoffel and W. Kunz, "Record & play: A structural fixed point iteration for sequential circuit verification," in *Proc. Int. Conf. Computer-Aided Design*, 1997, pp. 394–399.
- [26] H. J. Touati *et al.*, "Implicit state enumeration of finite state machines using BDD's," in *Proc. Int. Conf. Computer-Aided Design*, 1990, pp. 130–133.

Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations

Hai Zhou, D. F. Wong, I-Min Liu, and Adnan Aziz

Abstract—During the routing of global interconnects, macro blocks form useful routing regions which allow wires to go through but forbid buffers to be inserted. They give restrictions on buffer locations. In this paper, we take these buffer location restrictions into consideration and solve the simultaneous maze routing and buffer insertion problem. Given a block placement defining buffer location restrictions and a pair of pins (a source and a sink), we give a polynomial time exact algorithm to find a buffered route from the source to the sink with minimum Elmore delay.

Index Terms—Buffers, integrated circuit interconnections, layout, routing.

I. INTRODUCTION

With the evolution of very large scale integrated (VLSI) circuit fabrication technology, interconnect delay, especially global interconnect delay, has become the dominant factor in deep submicrometer design. Many techniques are employed to reduce interconnect delay; among them, buffer insertion has been shown to be an effective approach [1].

During routing process, especially that for global nets, there are macro blocks placed within the area. These blocks form useful routing regions because wires are allowed to run over them. But since buffers are implemented by transistors, a buffer "over" a macro block must be

Manuscript received March 4, 1999; revised January 4, 2000. This paper was recommended by Associate Editor C.-K. Cheng.

H. Zhou is with Advanced Technology Group, Synopsys, Inc., Mountain View, CA 94043 USA.

D. F. Wong, is with the Department of Computer Sciences, University of Texas, Austin, TX 78712 USA.

I.-M. Liu, and A. Aziz are with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712 USA.

Publisher Item Identifier S 0278-0070(00)05863-2.

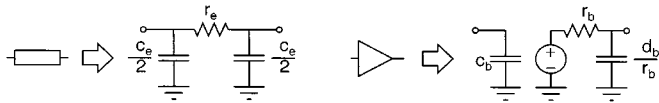


Fig. 1. Models of a wire and a buffer.

actually put into that block. However, it is impossible to insert a buffer into a macro block if it is an IP (Intellectual Property) where internal structure can not be changed, or it is a block such as memory which requires regular layout. Even in the case when that is theoretically possible, since a redesign of the influenced block is needed, it is usually not allowed due to the design flow. Therefore, macro blocks present themselves as routing resources for wires but obstacles for buffers.

Currently, these buffer location restrictions are taken care of only after routing stages. Work in post-layout buffer insertion [2]–[4] can be easily modified to avoid buffer insertion over macro blocks. But without considering macro blocks, a router may produce a route which runs so long over a block that later buffer insertion can not effectively reduce the delay. In maze routing, current approaches first find a shortest path and then insert buffers on it. If there is no macro block, it can be proved that this two-stage “routing and buffer insertion” approach gives an optimal solution. However, with macro blocks, a shortest path no longer guarantees minimum delay. Therefore, buffer location restrictions make it necessary to combine buffer insertion and routing. Although [5]–[7] did combine buffer insertion and routing tree construction, since they ignored macro blocks, they always used a shortest path to connect two points.

In this paper, we take the buffer location restrictions into consideration and solve the simultaneous maze routing and buffer insertion problem with these restrictions. Given two pins (a source and a sink) and a macro block placement which defines buffer location restrictions, we give a polynomial time exact algorithm to find a buffered route from source to sink such that the delay is minimized. Since the algorithm works on general graphs, we can extend it to global routing which also considers congestion minimization. Furthermore, it can also be extended to consider layer assignment in multilayer designs.

The rest of the paper is organized as follows. In Section II, we describe the delay model and problem formulation. In Section III, we give an algorithm to find a buffered path with minimum delay. In Section IV, we discuss how that algorithm can be used in global routing. Section V combines layer assignment with the buffered path construction. Experimental results and conclusions are given in Section VI.

II. DELAY MODEL FORMULATION

As in most previous work, we use the resistance-capacitance (RC) models shown in Fig. 1 for wires and buffers, and adopt the Elmore delay model [8] for their delays.

For a wire e , let l_e , c_e , and r_e denote its length, capacitance, and resistance, respectively. Using c_0 and r_0 to represent the unit length capacitance and resistance, we can write $c_e = c_0 l_e$ and $r_e = r_0 l_e$.

In traditional maze routing, we are given a routing area where there are regions occupied by other nets and need to find a path of minimum length to connect two pins. As fabrication gets into the deep submicrometer era, buffering becomes an inseparable part of wiring. Macro blocks introduce a new type of regions which does not exist in traditional maze routing: these regions are transparent to wires but not feasible for buffer insertion. Given such a routing area and the pin positions of a net, our objective is to find a buffered route with minimum delay.

We illustrate the importance of the problem by an example in Fig. 2, where dark boxes are used to represent the occupied regions where

wiring is not allowed, and gray boxes to represent macro blocks where buffering is not allowed. Fig. 2(a) gives the routing area where each unit has a length of 0.2 mm. We also set driver resistance at A and load capacitance at B to be 140 Ω and 2 fF, and unit length capacitance and resistance to be 0.21 fF/ μ m and 0.29 Ω / μ m, respectively. Furthermore, we assume there is only one type of buffer with $c_b = 2$ fF, $r_b = 140$ Ω , and $d_b = 40$ ps.

Traditional approach ignores macro blocks during routing. Thus, it finds a shortest path first and then inserts buffers outside macro blocks. An optimal solution by this approach is shown in Fig. 2(b) and its delay is 621.81 ps. As we can see, this shortest path has a very long part running over a macro block. Since the long over-block path can not be buffered and its delay is quadratic to the length, it produces a large delay. Taking this concern into consideration, the second approach treats macro blocks as obstacles and avoids wiring over them. It is also composed of two stages: find a shortest path without running over macro blocks, then insert buffers to minimize delay. An optimal solution by this approach is given in Fig. 2(c) and its delay is 680.62 ps. The drawback of this approach is the waste of routing resources. Furthermore, although buffers can now be inserted anywhere, a route may be too long to have short delay. On the contrary, wiring over macro blocks while wisely considering buffer insertion can give us a solution shown in Fig. 2(d). Its delay is 521.73 ps.

In maze routing, the entire routing area is represented as a grid graph. Each edge in the graph represents part of a routing track that can accommodate exactly one wire segment. Edges occupied by other nets and wiring obstacles are deleted from the graph. Given such a graph $G = (V, E)$, for each edge $(u, v) \in E$, we use $l(u, v)$, $c(u, v)$, and $r(u, v)$ to represent its length, capacitance, and resistance, respectively, and assume that $c(u, v) = c_0 l(u, v)$ and $r(u, v) = r_0 l(u, v)$. On each node $v \in V$, there is a label $p(v) \in \{0, 1\}$ indicating whether buffer insertion is allowed on that node. The problem we need to solve can be defined as follows.

Problem 1 (Buffered Maze Routing): Given a routing graph $G = (V, E)$, a buffer library B , and two nodes $s, t \in V$ with driver resistance R and load capacitance C , find a buffered path from s to t , that is, a sequence $(s = v_1, v_2, \dots, v_k = t)$ and a labeling $b(v_i) \in B \cup \{0\}$ for $1 \leq i \leq k$, such that the Elmore delay is minimized and the buffer insertion is feasible, that is, $b(v_i) \in B \Rightarrow p(v_i) = 1$, for $1 \leq i \leq k$.

III. FAST PATH ALGORITHM

In this section, we will give an exact algorithm to solve the buffered maze routing problem. It extends Dijkstra’s shortest path algorithm [9] to do a general labeling based on the Elmore delays. Since it finds a path with minimum delay instead of minimum length, we call it “fast path algorithm.”

A key property required by Dijkstra’s algorithm is that any subpath of a shortest path is also a shortest one. If we substitute path length by Elmore delay, the property no longer holds, because a path with minimum delay may have a subpath near the sink with a longer delay but a much smaller capacitance. Therefore, during the labeling process, besides delays, our algorithm needs also to remember capacitances if it starts from the sink or resistances if it starts from the source.

To simplify the presentation, we consider only starting from the sink. Each node is labeled with a set of capacitance-delay pairs, each represents a partial solution with that capacitance and delay. A capacitance-delay pair (c_1, d_1) is said to be *inferior* to another pair (c_2, d_2) if $c_1 \geq c_2$ and $d_1 \geq d_2$, since (c_1, d_1) in any solution can be substituted by (c_2, d_2) without increasing delay. A set of capacitance-delay pairs is said to be *nonredundant* if no one pair is inferior to any other. The pseudocode of the fast path algorithm is given in Fig. 3. Here, we use a quadruple (c, d, b, v) to represent each partial

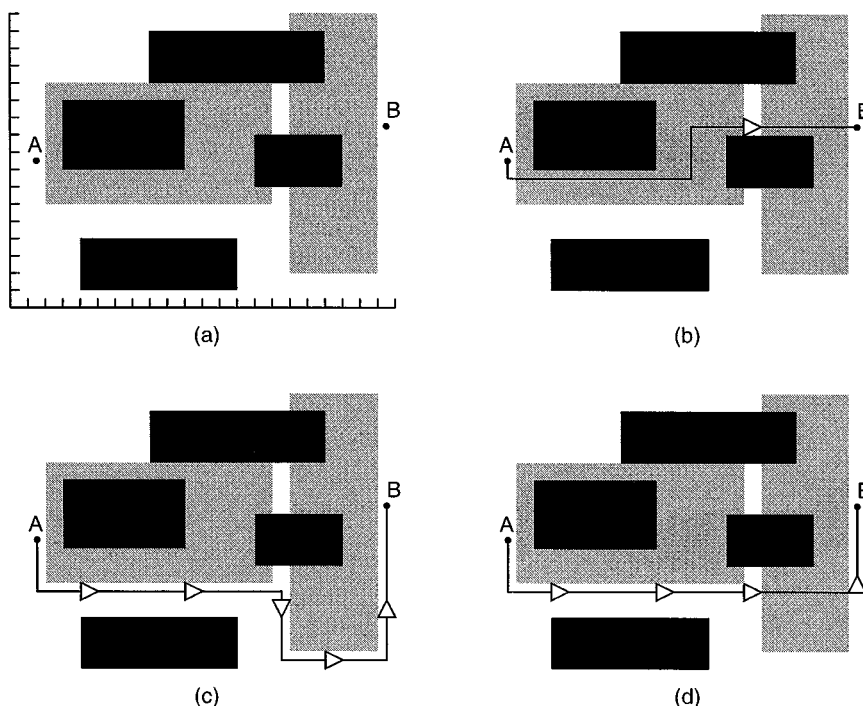


Fig. 2. (a) Routing area. (b) Shortest path + buffer insertion, delay = 621.81 ps. (c) Avoiding blocks + buffer insertion, delay = 680.62 ps. (d) Buffered maze routing, delay = 521.73 ps.

Algorithm Fast Path

```

Q ← {(C, 0, 0, t)};
while ( Q not empty) do
  (c, d, b, u) ← extract-min(Q);
  if c = 0 then
    return the solution;
  if u = s then
    S ← (0, d + Rc, b, u);
    Add S to Q and prune;
  else
    for each (u, v) ∈ E do
      S ← (c + c(u, v), d + r(u, v)(c + c(u, v)/2), 0, v);
      Add S to Q and prune;
  if p(u) = 1 and b = 0 then
    for each b' ∈ B do
      S ← (cb', d + rb'c + db', b', u);
      Add S to Q and prune;
    
```

Fig. 3. Pseudocode of the fast path algorithm.

solution. Besides the capacitance c and delay d , we also record the node v and the buffer insertion on it. The algorithm uses a priority queue [10] Q to sort the partial solutions and it is initialized with the first partial solution on sink t with capacitance C , delay 0, and no buffer. Each time, a partial solution with minimum delay is extracted from Q . If the solution includes the driver resistance, it is a complete solution with minimum delay and will be returned. Otherwise, it is used to update partial solutions of its neighbor nodes if it is not the source node. If the current node allows buffer but there is no buffer in the solution, partial solutions of the current node will be updated by inserting buffers from the library. When updating partial solutions for a node, we need to check the redundancy and drop the inferior ones and delete them from Q .

The following theorem shows the correctness and running time of the fast path algorithm.

Theorem 1: The fast path algorithm computes a buffered path of minimum delay in $O(|B||V|(|E| + |B||V|)\log |B||V|)$ time.

Proof: We expand the graph $G = (V, E)$ to a new graph $G' = (V', E')$ such that

$$V' = \{(v, c) | v \in V \text{ and } c \text{ is any capacitance seen from } v\}$$

$$E' = \{((u, c_1), (v, c_2)) | (u, v) \in E, c_2 = c_1 + c_0 l(u, v) \text{ or } \exists b \in B \text{ s.t. } c_2 = c_b\}.$$

For each $((u, c_1), (v, c_2)) \in E'$, we give a weight

$$w(((u, c_1), (v, c_2))) = \begin{cases} r(u, v)(\frac{1}{2}c(u, v) + c_1) \\ + r_b(c(u, v) + c_1) + d_b, & \text{if } c_2 = c_b \\ r(u, v)(\frac{1}{2}c(u, v) + c_1), & \text{otherwise.} \end{cases}$$

From the construction, it is easy to see that each buffered path in G corresponds to a path in G' and the Elmore delay of the buffered path is equal to the sum of weights on its corresponding path. Therefore, a shortest path in G' gives a buffered path in G with minimum delay. Since all the weights in G' are nonnegative, Dijkstra's algorithm finds a shortest path.

The fast path algorithm can be viewed as dynamically constructing G' in Dijkstra's algorithm. But it does not build all (v, c) 's for each node v . Node (v, c_1) with delay d_1 is pruned if there is another node (v, c_2) with delay d_2 such that $c_1 \geq c_2$ and $d_1 \geq d_2$. Since the subpath from (v, c_1) to t in any path can be substituted by another subpath from (v, c_2) to t without increasing the delay, the fast path algorithm correctly computes a buffered path with minimum delay.

To bound the running time of the algorithm, we show that, for each $v \in V$, the number of (v, c) 's not pruned is upper bounded by $|B||V|$. Consider all the paths from v to the sink t . If the first buffer is on v ,

the number of different capacitances seen from v is at most $|B|$. If the first buffer is on another node u , the number of different capacitances seen from v can become exponential since there can be exponential number of paths from u to v . But the number of capacitances not pruned is also bounded by $|B|$. This is because, without buffers on it, a path from u to v with minimum capacitance also has the minimum delay, thus makes all other paths pruned. Therefore, we can bound $|V'|$ by $|B||V|^2$. In the fast path algorithm, the number of EXTRACT-MIN operations on the priority queue is upper bounded by $|V'|$. But the dominant parts are the edge connection and the buffer insertion (the two for loops in the pseudocode). The number of edge connecting operations is upper bounded by $O(|B||V||E|)$. Since the number of partial solutions kept at any time is upper bounded by $|B||V|^2$, each of the DELETE and DECREASE-KEY operation in Q takes $O(\log |B||V|)$. Therefore, the amortized time for each edge connection (including pruning) can be bounded by $O(\log |B||V|)$. Thus, the total time for edge connections is $O(|B||V||E|\log |B||V|)$. On the other hand, the number of buffer insertion operations is upper bounded by $|B|^2|V|^2$, and the time for each operation can also be bounded by $O(\log |B||V|)$. Therefore, the total running time of the fast path algorithm is $O(|B||V|(|E| + |B||V|)\log |B||V|)$. ■

Since the running time is in terms of only actual input size (no capacitance or resistance sizes), the fast path algorithm is truly a polynomial time algorithm. Furthermore, the theorem only gives an upper bound. In reality, since many partial solutions will have the same capacitances and thus be pruned, the running time is much less. This is confirmed by the experimental results.

Note that in the fast path algorithm, it is restricted that at most one buffer can be inserted on a given node. Therefore, cascaded buffers on one node is not allowed. But the algorithm can be generalized to include cascaded buffers as follows. First, we can show that cascaded buffers in a minimum delay path will be monotonically connected.

Lemma 1: If two buffers b_1 and b_2 are inserted on one node in a minimum delay path and b_1 drives b_2 , then we must have $c_{b_1} < c_{b_2}$ and $r_{b_1} > r_{b_2}$.

Proof: If $c_{b_1} \geq c_{b_2}$, we can simply delete b_1 . Let R represent the driver resistance seen by b_1 , the original delay $Rc_{b_1} + d_{b_1} + r_{b_1}c_{b_2}$ is greater than Rc_{b_2} which is the later delay. This is a contradiction. On the other hand, if $c_{b_1} < c_{b_2}$ but $r_{b_1} \leq r_{b_2}$, we can delete b_2 . Let C represent the load capacitance see by b_2 , the original delay $r_{b_1}c_{b_2} + d_{b_2} + r_{b_2}C$ is greater than $r_{b_1}C$ which is the later delay. Another contradiction. ■

To allow cascaded buffers, there is no need to test $b = 0$ before the buffer insertion loop in the fast path algorithm. Based on the above property, there is also no need to consider all types of buffers if there are already buffers on the node. Instead, we only need to consider buffer types satisfying Lemma 1 thanks to the pruning, the maximum number of partial solutions on each node is still $|B||V|$. Cascading buffers only increases the number of buffer insertion operations on each node from $|B|^2|V|$ to $|B|^2(|V| + 1)$. Therefore, it has the same asymptotic worst case running time as the original algorithm.

IV. APPLICATION TO GLOBAL ROUTING

As shown by Theorem 1, the running time of the fast path algorithm does not depend on edge lengths or capacitance ranges. Therefore, besides maze routing, it can also be used on global routing, where edge lengths in the routing graph may be different.

However, since we only consider buffer insertions on vertices, edges of long lengths introduce limitations on the solutions. One way to deal with this is to first compute the minimum wire length between consecutive buffers and then use that length to direct the construction of the

```

Algorithm Congestion Minimization
 $Q \leftarrow (C, 0, 0, t, 0)$ ;
while ( $Q$  not empty) do
   $(c, d, b, u, g) \leftarrow \text{extract-min}(Q)$ ;
  if  $u = s$  then
    if  $d + Rc \leq D$  then
      return the solution;
    else
      for each  $(u, v) \in E$  do
        if  $d + r(u, v)(c + c(u, v)/2) < D$  then
           $S \leftarrow (c + c(u, v), d + r(u, v)(c + c(u, v)/2),$ 
             $0, v, g + g(u, v))$ ;
          Add  $S$  to  $Q$  and prune;
  if  $p(u) = 1$  and  $b = 0$  then
    for each  $b' \in B$  do
      if  $d + r_{b'}c + d_{b'} < D$  then
         $S \leftarrow (c_{b'}, d + r_{b'}c + d_{b'}, b', u, g)$ ;
        Add  $S$  to  $Q$  and prune;

```

Fig. 4. Pseudocode of the congestion minimization algorithm.

routing graph. Another way is to segment every long edge into a sequence of short segments [11].

In global routing, after a net is routed, the used edges are not deleted. Instead, the capacities of them are decreased and nets routed later are discouraged to use edges with smaller capacities. That is, for each $(u, v) \in E$ in $G = (V, E)$, another label $g(u, v) \in R$ is given to represent congestion cost. Given a delay constraint for a net, we want to find a buffered path with minimum congestion cost such that the delay constraint is satisfied. Formally, this can be defined as the following problem.

Problem 2 (Minimizing Congestion Under Delay Constraint): Given a routing graph $G = (V, E)$, a buffer library B , two nodes $s, t \in V$ with driver resistance R and load capacitance C , and a constant D , find a buffered path in G , that is, a sequence $(s = v_1, v_2, \dots, v_k = t)$ and a labeling $b(v_i) \in B \cup \{0\}$, such that the congestion cost $\sum_{i=1}^{k-1} g(v_i, v_{i+1})$ is minimized, the Elmore delay is bounded by D , and the buffer insertion is feasible, that is, $b(v_i) \in B \Rightarrow p(v_i) = 1$, for $1 \leq i \leq k$.

Similar to the problem of minimizing congestion under length constraint,¹ the above problem is intractable, as stated by the following theorem.

Theorem 2: The problem of minimizing congestion under delay constraint is NP-hard.

Proof Sketch: Given a graph $G = (V, E)$ with length $l(e)$ and congestion cost $g(e)$ for each $e \in E$ and a length bound L as a problem of minimizing congestion under length constraint, we can define a problem of minimizing congestion under delay constraint by restricting no buffer inserted anywhere, both driver resistance and load capacitance being zero and the unit length capacitance and resistance being one. Solving the problem with delay bound $L^2/2$ is then equal to solving the original problem. ■

However, the fast path algorithm can still be extended to solve the problem. Here, we need to extend a partial solution to include a congestion parameter g . A solution (c_1, d_1, g_1) is inferior to (c_2, d_2, g_2) if and only if $c_1 \geq c_2$, $d_1 \geq d_2$, and $g_1 \geq g_2$. The priority queue uses g as its key. The pseudocode of the algorithm is given in Fig. 4 where each partial solution is represented as (capacitance, delay, buffer, node, congestion). In practice, we can choose a minimum length such that,

¹The problem of minimizing congestion under length constraint is the same as the SHORTEST WEIGHT-CONSTRAINED PATH problem given in Garey and Johnson [12], thus is NP-hard.

with sufficient precision, all edge lengths can be represented as integral multiples of the minimum length. Similarly, congestion can also be approximated by integers. We use L to represent the summation of edge lengths in units of the minimum length, and K to represent the same thing for congestion. The running time of the congestion minimization algorithm can be bounded as follows.

Theorem 3: The running time of the congestion minimization algorithm is bounded by $O(KL|B|(|E| + |B||V|) \log KL|B||V|)$.

Proof: Similarly as the fast path algorithm, the algorithm expand a node $v \in V$ to a set of nodes (v, c, g) . The number of nodes not pruned is upper bounded by the number of noninferior triples (c, d, g) . For any capacitance-congestion pair (c, g) , there can be only one delay value d . Since capacitance is upper bounded by $L|B|$ and congestion is bounded by K , the number of nodes derived from a node $v \in V$ is then bounded by $KL|B|$. By a similar analysis as that on the fast path algorithm, the running time is $O(KL|B|(|E| + |B||V|) \log KL|B||V|)$. ■

V. MULTILAYER CONSIDERATION

In practice, routings are usually done on multiple layers and it is possible that unit length resistances and capacitances are different on different layers. In this case, the fast path algorithm given in Section III still works. But its running time becomes pseudopolynomial now.

If we can assume that each edge uses the same set of layers in the routing graph (which is usually true in global routing, since top layers are generally reserved for global nets), layer assignment can be done with buffered routing in polynomial time. We first consider the following layer assignment problem.

Problem 3 (Layer Assignment): Given a path $(s = v_1, v_2, \dots, v_k = t)$ in $G = (V, E)$, and a set of m layers each with a unit length capacitance c_i and a unit length resistance r_i for $1 \leq i \leq m$, find a layer assignment $y(v_i, v_{i+1})$ for each edge (v_i, v_{i+1}) with $1 \leq i \leq k - 1$ such that the Elmore delay from s to t is minimized.

First, we observe that for two layers i and j , if $c_i \leq c_j$ and $r_i \leq r_j$, then there is no need to consider layer j , since we can always reassign edges from layer j to i without increasing the delay. Here, without loss of generality, we can assume $c_i < c_j$ and $r_i > r_j$ for all $1 \leq i < j \leq m$. It can be shown that an optimal layer assignment has the following monotone property.

Lemma 2: For any two edges (v_i, v_{i+1}) and (v_j, v_{j+1}) such that $1 \leq i < j \leq k - 1$, if y is an optimal layer assignment, then

$$y(v_i, v_{i+1}) \geq y(v_j, v_{j+1}).$$

Proof: To simplify the presentation, we will denote (v_i, v_{i+1}) by e_i , for $1 \leq i \leq k - 1$. Suppose for the seek of contradiction that $y(e_i) = a$, $y(e_j) = b$ and $a < b$, which means $c_a < c_b$ and $r_a > r_b$. Because y is an optimal layer assignment, reassigning e_i from layer $y(e_i) = a$ to layer b could not decrease the delay. Hence, we have the following inequality:

$$\begin{aligned} \Delta D_1 &= \left(R + \sum_{h=1}^{i-1} l(e_h) r_{y(e_h)} \right) l(e_i) (c_a - c_b) \\ &\quad + \frac{1}{2} l(e_i)^2 (c_a r_a - c_b r_b) + l(e_i) (r_a - r_b) \\ &\quad \cdot \left(C + \sum_{h=i+1}^{k-1} l(e_h) c_{y(e_h)} \right) \\ &\leq 0 \end{aligned}$$

where R and C are the driver resistance and load capacitance, respectively.

Algorithm Buffered Path with Layer Assignment

```

Q ← (C, 0, 0, t, 1);
while ( Q not empty) do
  (c, d, b, u, y) ← extract-min(Q);
  if u = s and c = 0 then
    return the solution;
  if u = s then
    S ← (0, d + Rc, 0, u, y);
    Add S to Q and prune;
  else
    for each (u, v) ∈ E and each y ≤ i ≤ m do
      S ← (c + c_i l(u, v),
           d + r_i l(u, v) (c + c_i l(u, v) / 2), 0, v, i);
      Add S to Q and prune;
    if p(u) = 1 and b = 0 then
      for each b' ∈ B do
        S ← (c_{b'}, d + r_{b'} c + d_{b'}, u, 1);
        Add S to Q and prune;

```

Fig. 5. Pseudocode of the buffered path with layer assignment algorithm.

Similarly, reassigning (v_j, v_{j+1}) from layer $y(e_j) = b$ to layer a could not decrease the delay, either. Therefore, we have

$$\begin{aligned} \Delta D_2 &= \left(R + \sum_{h=1}^{j-1} l(e_h) r_{y(e_h)} \right) l(e_j) (c_b - c_a) \\ &\quad + \frac{1}{2} l(e_j)^2 (c_b r_b - c_a r_a) + l(e_j) (r_b - r_a) \\ &\quad \cdot \left(C + \sum_{h=j+1}^{k-1} l(e_h) c_{y(e_h)} \right) \\ &\leq 0. \end{aligned}$$

Therefore, we must have $(\Delta D_1 / l(e_i)) + (\Delta D_2 / l(e_j)) \leq 0$. But

$$\begin{aligned} \frac{\Delta D_1}{l(e_i)} + \frac{\Delta D_2}{l(e_j)} &= (c_b - c_a) \sum_{h=i}^{j-1} l(e_h) r_{y(e_h)} + \frac{1}{2} (l(e_i) - l(e_j)) \\ &\quad \cdot (c_a r_a - c_b r_b) + (r_a - r_b) \sum_{h=i+1}^j l(e_h) c_{y(e_h)} \\ &\geq (c_b - c_a) l(e_i) r_a + \frac{1}{2} l(e_i) (c_a r_a - c_b r_b) \\ &\quad + \frac{1}{2} l(e_j) (c_b r_b - c_a r_a) + (r_a - r_b) l(e_j) c_b \\ &> (c_b - c_a) l(e_i) r_a + \frac{1}{2} l(e_i) (c_a r_a - c_b r_a) \\ &\quad + \frac{1}{2} l(e_j) (c_b r_b - c_b r_a) + (r_a - r_b) l(e_j) c_b \\ &= \frac{1}{2} (c_b - c_a) l(e_i) r_a + \frac{1}{2} (r_a - r_b) l(e_j) c_b \\ &> 0. \end{aligned}$$

This is a contradiction. ■

Based on the above property, a candidate configuration can be generated by separating the $k - 1$ edges into m groups, and this can be done by selecting $m - 1$ positions for the separators. Therefore, the total number of candidate configurations are $\binom{m+k-2}{m-1}$. This also gives an algorithm to find the optimal layer assignment. In practice, we have only a small constant number of layers. Therefore, the number of configurations is a low degree polynomial in terms of the number of edges.

Now we can combine layer assignment with buffered path construction in the fast path algorithm to solve the following problem.

Problem 4 (Buffered Path with Layer Assignment): Given a routing graph $G = (V, E)$, a set of m layers with different parasitic characteristics, a buffer library B , and two nodes $s, t \in V$ with driver

TABLE I
EXPERIMENTAL RESULTS

Tech: $C = 1\text{fF}$, $R = 1\text{K}\Omega$, $c_0 = 0.21\text{fF}/\mu\text{m}$, $r_0 = 0.29\Omega/\mu\text{m}$, $c_b = 1\text{fF}$, $r_b = 1\text{K}\Omega$, $d_b = 57\text{ps}$								
Data			Shortest+Buffer		Fast Path		Improvements	
name	size(mm)	#blk	delay(ps)	len(mm)	delay(ps)	len(mm)	delay	len
ran1	15.0x15.5	6	7414.47	26.0	3946.88	27.2	87.86%	-4.0%
ran2	16.2x11.5	8	4632.86	24.6	3564.04	24.6	29.99%	0%
ran3	14.0x11.9	9	5221.35	23.2	3946.88	27.2	32.29%	-17.0%
ran4	11.7x13.8	6	8885.30	37.0	6057.30	39.8	46.69%	-7.0%
ran5	12.5x12.3	5	5820.45	33.4	4880.73	33.4	19.25%	0%
ran6	12.8x13.3	6	11662.15	36.6	5473.80	37.4	113.05%	-2.0%
ran7	11.9x11.9	3	6388.17	33.0	4800.30	33.0	33.08%	0%
ran8	10.3x12.5	8	9660.66	36.6	5356.80	36.6	80.34%	0%
ran9	13.9x12.2	7	14038.36	37.0	5435.40	37.0	158.28%	0%
ran10	10.8x10.6	10	9023.52	31.8	4908.38	32.6	83.84%	-2.0%

resistance R and load capacitance C , find a buffered path with layer assignment from s to t , that is, a sequence $(s = v_1, v_2, \dots, v_k = t)$, a labeling $b(v_i) \in B \cup \{0\}$ for $1 \leq i \leq k$, and another labeling $1 \leq y(v_i, v_{i+1}) \leq m$ for $1 \leq i \leq k-1$, such that the Elmore delay is minimized and the buffer insertion is feasible, that is, $b(v_i) \in B \Rightarrow p(v_i) = 1$, for $1 \leq i \leq k$.

In order to include layer assignment in the fast path algorithm, we add another parameter y to represent the layer assigned to the edge driven by the current node. When adding a new edge, we only assign it to the layers which observe the property in Lemma 2. The pseudocode is given as follows, where each partial solution is represented as (capacitance, delay, buffer, node, layer). The pseudocode is given in Fig. 5. The running time of the buffered path with layer assignment algorithm is given by the following theorem.

Theorem 4: The running time of the buffered path with layer assignment algorithm is upper bounded by $O(|B||V|^m(|E| + |B||V|) \log |B||V|^m)$.

Since m as the number of layers is usually a small constant, the algorithm actually runs in polynomial time.

VI. EXPERIMENTAL RESULTS AND CONCLUDING REMARKS

The technology parameters used in the experiments are based on the $0.18 \mu\text{m}$ technology predicted by the National Technology Roadmap of Semiconductors (NTRS) [13]. That is, we have unit length capacitance $c_0 = 0.21 \text{fF}/\mu\text{m}$ and unit length resistance $r_0 = 0.29\Omega/\mu\text{m}$. We also set the driver resistance and load capacitance to $1 \text{K}\Omega$ and 1fF , respectively. The buffer library has only one type of buffer with $r_b = 1 \text{K}\Omega$, $c_b = 1 \text{fF}$ and $d_b = 57 \text{ps}$.

Each problem has a rectangular routing area with height and width ranging from 10–17 mm. We randomly place some blocks in that area and also randomly generate two pins in it. The grid we use for maze routing has 0.1mm as its unit length. On a machine with a 180-MHz Pentium Pro processor and 64-M memory, the running time ranges from 14–51 s. In practical designs, not all nets need to be buffered. Only those critical and global nets need. Usually, the number of such nets is very small. Thus the running time is practical. As a comparison, we also implement the traditional method of first finding a shortest path and then inserting buffers. In Table I, we first report the dimension of each problem and the number of macro blocks in it. Then the delays from source to sink and the wire lengths by the two

methods are reported. To make the comparison clear, we also give the percentage improvements by fast path algorithm over the traditional approach.

From these experiments, we can see that, with only a minor increase on wire length, the fast path algorithm can dramatically reduce the delay.

Since connecting two points is a basic operation in tree construction, our algorithm can also be used in any routing tree construction for multiterminal nets, especially those using dynamic programming approach. For example, we can easily embed the fast path algorithm into [7] to construct a buffered routing tree with buffer location restrictions.

REFERENCES

- [1] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Reading, MA: Addison-Wesley, 1990.
- [2] L. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. Int. Symp. Circuits and Systems*, 1990, pp. 865–868.
- [3] L. N. Kannan, P. R. Suaris, and H.-G. Fang, "A methodology and algorithms for post-placement delay optimization," in *Proc. Design Automation Conf.*, 1994, pp. 327–332.
- [4] J. Lillis, C. K. Cheng, and T. T. Lin, "Optimal and efficient buffer insertion and wire sizing," in *Proc. Custom Integrated Circuits Conf.*, 1995, pp. 259–262.
- [5] —, "Simultaneous routing and buffer insertion for high performance interconnect," in *Proc. 6th Great Lakes Symp. VLSI*, 1996, pp. 148–153.
- [6] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 44–49.
- [7] A. H. Salek, J. Lou, and M. Pedram, "A simultaneous routing tree construction and fanout optimization algorithm," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 625–630.
- [8] W. C. Elmore, "The transient response of damped linear networks with particular regard to wide-band amplifiers," *J. Appl. Phys.*, vol. 19, no. 1, pp. 55–63, Jan. 1948.
- [9] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Math.*, vol. 1, pp. 269–271, 1959.
- [10] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1989.
- [11] C. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *Proc. Design Automation Conf.*, 1997, pp. 588–593.
- [12] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman, 1979.
- [13] "National technology roadmap for semiconductors," Semiconductor Industry Association, 1994.