Simultaneous Routing and Buffer Insertion with Restrictions on Buffer Locations*

Hai Zhou[1], D.F. Wong[1], I-Min Liu[2], and Adnan Aziz[2]
[1] Department of Computer Sciences, University of Texas, Austin, TX 78712
[2] Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712

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.

1 Introduction

With the evolution of VLSI fabrication technology, interconnect delay, especially global interconnect delay, has become the dominant factor in deep sub-micron design. Many techniques are employed to reduce interconnect delay; among them, buffer insertion has been shown to be an effective approach [2].

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 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 re-design 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 [11, 7, 8] 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 [9, 10] did combine buffer insertion and routing tree construction, since they ignored macro blocks, they always used a shortest path to connect two points.

*This work was partially supported by a grant from the Intel Corporation.

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 multi-layer designs.

2 Delay model and problem formulation

As in most previous work, we use the RC models shown in Figure 1 for wires and buffers, and adopt the Elmore delay model [5] for their delays.

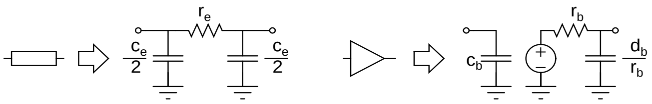

Figure 1: Models of a wire and a buffer

For a wire e, let l_e, c_e, 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 sub-micron 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 Figure 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. Figure 2(a) gives the routing area where each unit has a length of $0.2mm$. We also set driver resistance at A and load capacitance at B to be 140Ω and $2fF$, and unit length capacitance and resistance to be $0.21fF/\mu m$ and $0.29\Omega/\mu m$, respectively. Furthermore, we assume there is only one type of buffer with $c_b = 2fF, r_b = 140\Omega$, and $d_b = 40ps$.

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 Figure 2(b) and its delay is $621.81ps$. 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

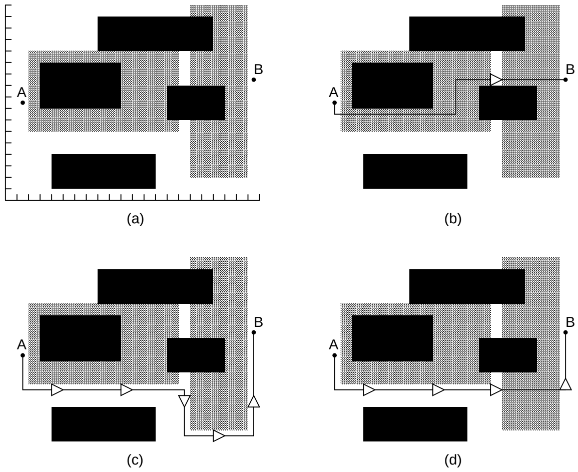

Figure 2: (a) Routing area (b) Shortest path + buffer insertion, delay = $621.81ps$ (c) Avoiding blocks + buffer insertion, delay = $680.62ps$ (d) Buffered maze routing, delay = $521.73ps$

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 Figure 2(c) and its delay is $680.62ps$. 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 Figure 2(d). Its delay is $521.73ps$.

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 $l(u, v), c(u, v), r(u, v)$ to represent its length, capacitance, and resistance, respectively. 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 $(s = v_1, v_2 \ldots,$ $v_k = t)$ and a labeling $b(v_i) \in B \cup \{0\}$ for $1 \le i \le k$, 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 $1 \le i \le k$.*

3 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 [4] 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 sub-path 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 sub-path 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 $c_1 \ge c_2$ and $d_1 \ge d_2$, since (c_1, d_1) can be substituted by (c_2, d_2) in any solution without increasing delay. A set of capacitance-delay pairs is said to be *non-redundant* if no pair is inferior to any other. The pseudo-code of the fast path algorithm is given as follows.

```
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 ← (c_b', d + r_b' c + d_b', b', u);
      Add S to Q and prune;
```

Here, we use a quadruple (c, d, b, v) to represent each partial 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 [3] 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 $O(|B|^3|V|^2|E| \log |B||V|)$ time.*

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. That is,

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 $r_{b_1} > r_{b_2}$.*

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. Since this modification increases the number of buffer insertion operations on each node from $|B|^2|V||E|$ to $|B||V||E|2^{|B|}$, but does not change the number of partial solutions on any node, the running time becomes $|B|^2|V|^2|E|2^{|B|} \log |B||V|$.

4 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 can be different.

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 $1 \le i \le k$.*

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.*

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 $c_1 \ge c_2, d_1 \ge d_2$, and $g_1 \ge g_2$. The priority queue uses g as its key. The pseudocode of the algorithm is given as follows, where each partial solution is represented as (capacitance, delay, buffer, node, congestion).

```
Algorithm Congestion Minimization
Q ← (C, 0, 0, t, 0);
while ( Q not empty) do
  (c, d, b, u, g) ← extract-min(Q);
  if u = s then
    if d + Rc ≤ D then
      return the solution;
  else
    for each (u, v) ∈ E do
      if d + r(u, v)(c + c(u, v)/2) < D then
        S ← (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' ∈ B do
        if d + r_b' c + d_b' < D then
          S ← (c_b', d + r_b' c + d_b', b', u, g)
          Add S to Q and prune;
```

[1] The problem of minimizing congestion under length constraint is the same as the SHORTEST WEIGHT-CONSTRAINED PATH problem given in Garey and Johnson [6], thus is NP-hard.

In practice, we can choose a minimum length such that, with sufficient precision, all edge lengths can be represented as integral multiples of the minimum length. Similarly, congestions 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 $O(K^2L^2|B|^3|E| \log KL|B||V|)$.*

5 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 3 still works. But its running time becomes pseudo-polynomial now.

If we can assume that all layers are available for each edge in the routing graph (which is usually true in global routing), 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 $1 \le i \le m$, find a layer assignment $y(v_i, v_{i+1})$ for each edge (v_i, v_{i+1}) with $1 \le i \le k - 1$ such that the Elmore delay from s to t is minimized.*

First, we observe that for two layers i and j, if $c_i \le c_j$ and $r_i \le r_j$, then there is no need to consider layer j, since we can always re-assign 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 $1 \le i < j \le m$. 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 $1 \le i < j \le k - 1$, if y is an optimal layer assignment, then*

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

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 resistance R and load capacitance C, find a buffered path with layer assignment from s to t, that*

Tech: $C = 1fF, R = 1K\Omega, c_0 = 0.21fF/\mu m, r_0 = 0.29\Omega/\mu m, c_b = 1fF, r_b = 1K\Omega, d_b = 57ps$								
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%

Table 1: Experimental Results

is, a sequence $(s = v_1, v_2 \ldots, v_k = t)$*, a labeling* $b(v_i) \in B \cup \{0\}$ *for* $1 \le i \le k$*, and another labeling* $1 \le y(v_i, v_{i+1}) \le m$ *for* $1 \le i \le k-1$*, 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* $1 \le i \le k$.

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 pseudo-code is given as follows, where each partial solution is represented as (capacitance, delay, buffer, node, layer).

```
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', b', u, 1)
      Add S to Q and prune;
```

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* $O(m|B|^3|V|^{2(m-1)} \log |B||V|)$.

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

6 Experimental results and concluding remarks

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

Each problem has a rectangular routing area with height and width ranging from $10mm$ to $17mm$. We randomly place some blocks in that area and also randomly generate two pins in it. The grid used for maze routing has $0.1mm$ as its unit length. On a machine with a 180MHz Pentiun Pro processor and 64M memory, the running time ranges from 14 seconds to 51 seconds. As a comparison, we also implement the traditional method of first finding a shortest path and then inserting buffers. In Table 1, 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 multi-terminal nets. For example, we can easily embed the fast path algorithm into [10] to construct a buffered routing tree with buffer location restrictions.

References

[1] Semiconductor Industry Association. National technology roadmap for semiconductors, 1994.

[2] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, 1990.

[3] T. H. Cormen, C. E. Leiserson, and R. H. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[4] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math.*, 1:269–271, 1959.

[5] W. C. Elmore. The transient response of dampled linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19(1):55–63, January 1948.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., 1979.

[7] L. N. Kannan, P. R. Suaris, and H.-G. Fang. A methodology and algorithms for post-placement delay optimization. In *DAC*, pages 327–332, 1994.

[8] J. Lillis, C. K. Cheng, and T. T. Lin. Optimal and efficient buffer insertion and wire sizing. In *CICC*, pages 259–262, 1995.

[9] T. Okamoto and J. Cong. Buffered Steiner tree construction with wire sizing for interconnect layout optimization. In *ICCAD*, pages 44–49, 1996.

[10] A. H. Salek, J. Lou, and M. Pedram. A simultaneous routing tree construction and fanout optimization algorithm. In *ICCAD*, 1998.

[11] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *ISCAS*, pages 865–868, 1990.