Timing-Driven Maze Routing

Sung-Woo Hur, Ashok Jagannathan, and John Lillis

***Abstract*—This paper studies a natural formulation of the timing-driven maze routing problem. A multigraph model appropriate for global routing applications is adopted; the model naturally captures blockages, limited routing and wire-sizing resources, layer assignment, etc. Each edge in the multigraph is annotated with resistance and capacitance values associated with the particular wiring segment. The timing-driven maze routing problem is then to find paths which exhibit low resistance-capacitance (*RC*) delay or achieve a tradeoff between *RC* delay and total capacitance. An easy-to-implement labeling algorithm is presented to solve the problem along with effective speedup enhancements to the basic algorithm which yield up to 300 times speedup. It is suggested that such an algorithm will become a fundamental tool in an arsenal of interconnect optimization techniques. The tractability of the approach is supported via computational experiments.**

***Index Terms*—Design automation, dynamic programming, integrated circuit interconnections, routing.**

I. Introduction

WITH the growing influence of interconnect delay on overall system performance, a great deal of research has been done in recent years to ameliorate the problem via computer-aided design-based techniques. Much of the past work has concentrated on timing optimization through techniques such as automatic wire-sizing and tapering and repeater insertion. While such work is undeniably of fundamental interest, important issues relating to how such optimizations can be performed under a given context have often been overlooked. For instance, wire-sizing algorithms have assumed that wires can be sized anywhere on the routing region, without considering constraints imposed by congestion. Hence it is clear that such optimizations cannot be performed in isolation of such resource allocation issues.

As an illustration, consider routing a net from s to t under the routing conditions shown in Fig. 1. The grey boxes represent areas where optimal wire-sizing is not possible due to factors like congestion, etc.; the black boxes denote areas where routing is completely disallowed, for example, due to preplaced macros with strict design constraints. In such a case, the figure shows that the shortest path connecting the two pins is slower than the other path which detours. The corresponding cost and delay values are shown in the caption. Similarly, it is often the case that there exist multiple shortest paths each having varying resources for sizing and hence delays. This paper addresses these kind of issues during routing.

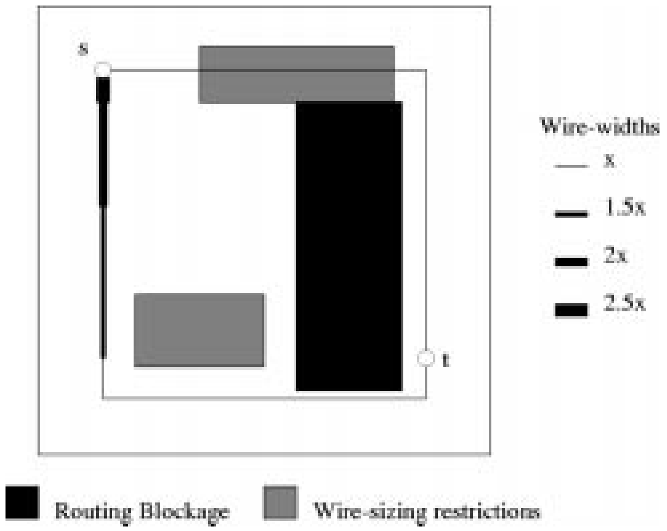

Fig. 1. Illustration of routing under restrictions where the shortest path is not the best path from a timing sense. For the shortest path, length $= 5940\mu$m, $c = 0.941$ pF and $d = 0.494\mu$s. For the path that detours, we have length $= 6000\mu$m, $c = 1.209$ pF, and $d = 0.466\mu$s. The driver resistance $R_s = 150.0\Omega$; sink capacitance $C_t = 50$ fF; sheet resistance $r_0 = 0.12\Omega/\mu$m; unit area capacitance $c_0 = 0.15$ fF/μm. Wire widths equivalent to $x,\ 1.5x,\ 2x$, and $2.5x$ of the minimum widths are used during routing.

In the timing-driven maze routing problem, we are given a routing graph (or multigraph) in which edges are annotated with resistance and capacitance values; the task is to find "good" paths connecting given source and destination vertices. There are several natural interpretations of "good"; for instance, a minimum delay solution whose cost (e.g., total capacitance[1]) does not exceed some given budget. Thus, in contrast to traditional maze routing [1], where *any* wiring connection is sufficient, our problem is fundamentally multidimensional since both cost and delay are considered. We present a straightforward labeling algorithm optimally solving the problem.

The key points of the paper are summarized as follows.

- The adopted multigraph model (i.e., having multiple edges between vertices) is quite general and naturally captures optimization techniques such as wire sizing (via alternative edges). Further, technology characteristics such as varying parasitics from one routing layer to the next are handled naturally as are the effects of vias. No assumptions are made about the parasitic values on wiring segments: resistive and capacitive values can be determined by any desired means.

[1]Total capacitance is a natural cost measure because of its correlation with both routing area and dynamic power consumption in CMOS technologies.

Manuscript received June 1, 1999; revised October 29, 1999. This work was supported in part by a Design Automation Conference Scholarship and in part by the National Science Foundation (NSF) under Career Award CCR9875945. This paper was recommended by Associate Editor D. Hill.

The authors are with the Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, College of Engineering, Chicago, IL 60607-7053 USA.

Publisher Item Identifier S 0278-0070(00)01791-7.

- While the algorithm is presented using the Elmore delay model, the framework is flexible enough to enable use of different delay estimators (e.g., estimators approximating the effect of wire-to-wire coupling or incorporating improved load modeling as in [3]).
- While the algorithms may find use in a number of different scenarios, an obvious application is in global routing. We expect the techniques to particularly be useful in modern global routing schemes which incorporate some notion of signal ordering or track assignment via a technique such as pseudopin assignment (see, e.g., [4]–[6]). We envision the application of the algorithms for critical global nets in a rip-up and reroute global routing scheme (e.g., [7]).
- The basic algorithm is a straightforward labeling algorithm which can be viewed as an generalization of Dijkstra's algorithm [8]. The resulting algorithm has pseudopolynomial running time which limits its practical application if implemented naively. An important contribution of the paper is a set of speed-up techniques based on the A^* algorithm [9] with careful lower-bounding of delay; the bounding techniques result speedups of up to 300 times versus the naive implementation. We believe the computational results demonstrate the praticality of the approach.

The remainder of the paper is organized as follows. In the Section II, we give problem formulation and explain some necessary terminologies. In Section III, we present a generic timing-driven maze routing algorithm and improvements over the generic algorithm. Section IV presents results on the test grids. We discuss some applications of our algorithm in Section V and this is followed by conclusion in Section VI.

II. Preliminaries

1) Multigraph Model: We assume that the routing graph is given as a multigraph (i.e., there may be multiple edges between vertices) and that each edge e in the multigraph has two labels: capacitance (c_e) and resistance (r_e). Some important features of this multigraph model are listed below.

- The resistive and capacitive values for the edges can be calculated by any arbitrary means—i.e., there is no restriction on the model used to calculate the parasitics of the edges. For instance, the effects of capacitive coupling with adjacent nets can be approximated in this model (different multiedges may be used to model different spacing choices).
- The model captures the properties of multilayer routing. Varying parasitics from one routing layer to the other and the effect of vias are inherently captured in this model. Also, multiple edges between a pair of vertices can be used to capture wire sizing choices in any routing region on a particular layer. This also addresses congestion constraints when routing multiple nets, as the number of edges and the sizes between any two nodes reflect the usage and availability of routing and sizing resources in the region.
- The model naturally reflects already used paths or routing blockages by the absence of edges between vertices.

2) Delay Estimators: We use the Elmore delay model [10] for interconnection delay. The propagation delay along a wire segment $e = (u, v)$ is approximated by

$$r_e(c_e/2 + C_v)$$

where c_e and r_e are, respectively, the capacitance and resistance of wire e and C_v is the total downstream capacitive load at vertex v. Driver delay is also taken into account in the algorithm.

3) Problem Formulation: We formulate the timing-driven maze routing problem as follows.

Given: A multigraph $G = (V, E)$ in which each edge $e \in E$ is annotated with a resistance r_e and a capacitance c_e, a source terminal s with driving impedance r_s, a sink terminal t with load capacitance c_t, and a capacitance constraint c_{spec}.

Objective: Find a path in G connecting s and t such that the Elmore delay of the path is minimized subject to the total wire capacitance not exceeding c_{spec}.

We note that there are several equally natural alternative formulations. For instance, we may wish to minimize total capacitance subject to a maximum delay specification. Our algorithms require only minor modifications for solving such variants.

4) Dominance Relation: Our algorithm for solving the problem is bottom-up and based on labeling schemes. Labels of a vertex u take the form of lists of candidate (sub-)solutions representing paths from the vertex u to the sink t. These paths are characterized by the following two parameters:

- c: total capacitance of the path $u \rightsquigarrow t$;
- d: resulting Elmore delay from u to t (including the delay of the driver at s if $u = s$).

At each vertex u, two paths are compared by a partial order. We use $(c, d) \prec (c', d')$ to indicate that the first path *dominates* the second, i.e., ($c < c'$ and $d \leq d'$) or ($c \leq c'$ and $d < d'$). Suppose u has two labels (c, d) and (c', d') each representing a different $u \rightsquigarrow t$ path and assume $(c, d) \prec (c', d')$. Then it is clear that path (c', d') is suboptimal and hence we need not consider any path extended from (c', d').

For notational convenience we use the dominance relation in the context of sets of vectors; let P be a set of (c, d)-pairs

$$P \prec (c', d') \leftrightarrow \exists\, (c, d) \in P \quad \text{such that} \quad (c, d) \prec (c', d').$$

III. Algorithm

In this section, we will first present a generic timing-driven routing algorithm and then the strategies to accelerate the generic algorithm. For any vertex u in the routing graph, let $P(u)$ denote a list of candidate paths from u to the sink t. Each such path is a candidate tail of some path $s \rightsquigarrow u \rightsquigarrow t$. If a candidate path $u \rightsquigarrow t$ is not dominated by any other candidate paths, we call it a *minimal* path.

The generic algorithm is shown in Fig. 2. We use two primary data structures: a list of candidate paths $P(u)$ for each vertex $u \in V - \{t\}$ and a priority queue, Q, of candidate paths ordered first by c and secondarily by d. $P(u)$ has important properties as described follows.

```
Algorithm Generic Timing-Driven Maze Routing
      input: $G = (V, E)$ and $c_{spec}$
      output: minimum delay path
              whose total wire capacitance is $\leq c_{spec}$
      Subroutine Candidates$(u, (c, d))$
c1    $L \leftarrow \emptyset$
c2    for each edge $e = (u, v) \in E$ incident to $u$
c3      if $(v = s)$ then // source
c4        $L \leftarrow L \cup \{s, (c + c_e, d + r_e(\frac{c_e}{2} + c) + r_s(c_e + c))\}$
c5      else
c6        $L \leftarrow L \cup \{v, (c + c_e, d + r_e(\frac{c_e}{2} + c))\}$
c7      endif
c8    endfor
c9    return $L$ ordered by $(c, d)$-pair: first by $c$ and secondarily by $d$
      Main Routine
m1    $P(t) \leftarrow \{(c_t, 0)\}$
m2    $P(u) \leftarrow \emptyset, \forall u \neq t$
m3    $Q \leftarrow$ Candidates$(t, (c_t, 0))$ // initialize queue
m4    while $(Q \neq \emptyset)$
m5        $(u, (c, d)) \leftarrow$ Dequeue$(Q)$
m6        if $(c > c_{spec})$ then
m7            return $P(s)$
m8        if $(P(u) \not\prec (c, d))$ then
m9            $P(u) \leftarrow P(u) \cup \{(c, d)\}$
m10           $L \leftarrow$ Candidates$(u, (c, d))$
m11           for each $(v, (c', d')) \in L$
m12               if $(P(v) \not\prec (c', d'))$ then
m13                   Enqueue$(Q, (v, (c', d')))$
m14           endfor
m15       endif
m16   endwhile
m17   return $P(s)$
```

Fig. 2. Generic timing-driven maze routing algorithm. The algorithm is bottom-up: labels are computed from the sink toward the source.

- $P(u)$ is always a list of *minimal* paths since, in step m8, the dequeued tuple is checked for dominance and it is discarded if it is dominated by $P(u)$.
- For the list $(c_1, d_1), (c_2, d_2), \cdots, (c_k, d_k)$ in $P(u)$, assuming (c_i, d_i) was inserted into $P(u)$ before (c_j, d_j) $(i < j)$, following property holds

$$c_1 < c_2 < \cdots < c_k \quad \text{and} \quad d_1 > d_2 > \cdots > d_k$$

  by the facts that $P(u)$ is a list of minimal paths and that the dequeued information (a candidate subpath from u to t) in step m5 is in nondecreasing order of c (since Q is a priority queue).
- Due to the above property, $P(u)$ can be maintained in a simple linked list, i.e., new paths are simply appended.

These properties enable us to check the dominance relation in constant time: in step m8, $P(u)$ dominates (c, d) if and only if $d_k \leq d$ with the assumption that $P(u)$ has k elements.

At the termination of the algorithm, a set of nondominated paths in $P(s)$ is left. Each $(c, d) \in P(s)$ represents a path p connecting s and t with total capacitance c and Elmore delay d. Although the last element in $P(s)$ is the minimum delay path under the given capacitance constraint (which is the solution to the problem), the members of $P(s)$ can be considered as a set of candidate solutions with different tradeoffs between capacitance and delay.

The algorithm has a pseudopolynomial bound on the running time as described in Section III-A.

A. Complexity Analysis

A worst case analysis of the algorithm can be derived based on the following assumptions:

Assumption 1: The capacitive values c_e are given as integers. This is not a limitation since discretization is always possible.

Assumption 2: The node degree in the target routing graph G is bounded by a constant (as is typical in VLSI applications).

The analysis gives a pseudopolynomial bound on the running time of the algorithm—i.e., the running time is not only a function of the size of the routing graph but also of the values with which the graph is annotated (namely c's).

Let $U = \sum_{e \in E} c_e$. The following observation follows from the minimality of $P(v)$.

Observation 1: In the worst case, $|P(v)|$ is bounded by U.

With this, we can then bound the size of the priority queue Q.

Observation 2: In the worst case, $|Q| = O(U \cdot |V|)$. Since each vertex v has a constant degree, each member of $P(v)$ can introduce only a constant number of candidates into Q thus giving the bound. Further, over the lifetime of the algorithm, the total number of enqueues into Q is $O(U \cdot |V|)$. [Since each member of Q is derived from some permanent label of some $P(v)$.]

From these observations, an overall bound on the running time is obtained as follows.

Theorem: The running time of the algorithm for the given problem formulation is $O(U \cdot |V| \cdot \log(U \cdot |V|))$.

Proof: The total running time is given by (number of dequeues) × (time to dequeue and test the dominance of the dequeued solution) + (number of enqueues) × (time to enqueue a candidate). The number of enqueues and dequeues is $O(U \cdot |V|)$ and the time to test the dominance of the dequeued solution at vertex v is $O(1)$. Further the time to perform an enqueue/dequeue is $O(\log(U \cdot |V|))$. The resulting complexity is $O(U \cdot |V| \cdot (\log(U \cdot |V|) + 1) + U \cdot |V| \cdot \log(U \cdot |V|)) = O(U \cdot |V| \cdot \log(U \cdot |V|)$. □

B. Examples

Fig. 3 shows a 4 × 4 routing grid in which each of the 22 edges are annotated with r, c pairs (no multiedges are included in this example for simplicity).

Two nondominated source-to-sink paths are discovered by the generic timing-driven maze routing algorithm (Fig. 4). Each solution gives a different capacitance versus delay tradeoff.

In Fig. 5 we show an example of all the final solutions of $P(s)$ using test grid "Grid-4" of which dimension is 100 ×100 (for detailed characteristics of the grid, refer to Section IV-B).

C. Speedup Strategy

In this section, we present techniques to accelerate the generic algorithm. The generic algorithm generates many subsolutions at intermediate nodes before finding solutions at the source node. By maintaining some useful information at each node and using a simple bounding technique, we are able to suppress the extension of suboptimal paths in the early

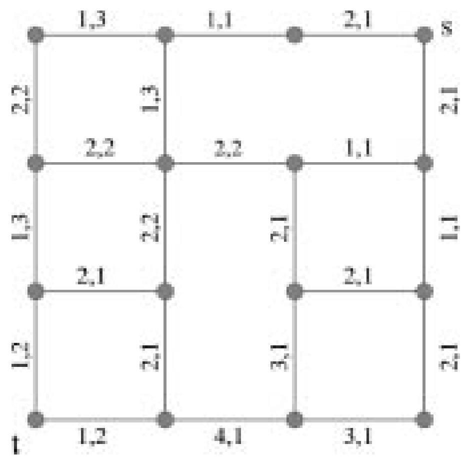

Fig. 3. An example of a grid. Each edge is labeled with resistance and capacitance pairs. We assume source resistance $r_s = 3$, sink capacitance $c_t = 3$ and $c_{\text{spec}} = \infty$.

stages of the algorithm. Our approach follows that of the A^* algorithm [9].

Suppose we have already found the first $k(>0)$ s-to-t paths in $P(s)$, i.e., $P(s) = (c_1, d_1), \cdots, (c_k, d_k)$ and we are considering a candidate label (c, d) at vertex v. Recall that the entries in $P(s)$ are discovered in nondecreasing order of c.

Let p: $s \rightsquigarrow v$ denote any path from the source s to vertex v. Let l_e denote the wire length of edge e. Let us define $l_{\min}(v)$ as

$$l_{\min}(v) = \min_{\text{paths } p:\ s \rightsquigarrow v} \left(\sum_{e \in p} l_e \right).$$

In other words, $l_{\min}(v)$ is a minimum distance of p: $s \rightsquigarrow v$ among all possible paths from s to v. Analogously, $c_{\min}(v)$ is defined as

$$c_{\min}(v) = \min_{\text{paths } p:\ s \rightsquigarrow v} \left(\sum_{e \in p} c_e \right)$$

i.e., $c_{\min}(v)$ is a lower-bound on the total capacitance of any path from s to v. Thus, $l_{\min}(v)$ and $c_{\min}(v)$ can be precomputed for each vertex v using two runs of Dijkstra's algorithm [8].

Given technology parameters, the minimum wire length of a path p: $s \rightsquigarrow v$, driving resistance r_s, and load capacitance at v, we can find a lower-bound on the delay of path p using methods such as those in [11] and [12] which give closed forms for unconstrained wire tapering.

While expanding any subpath at vertex u to v (v is upstream) using some edge e, we can use $l_{\min}(v)$, the downstream capacitance c of the subpath and the driving resistance r_s to determine a lower bound on the delay of the upstream path p: $s \rightsquigarrow v$ using the method in [11].[2] Let us define $d_{\min}(v, c)$ as

$$d_{\min}(v, c) = \text{lower-bound on delay of } p\text{: } s \rightsquigarrow v \text{ with load } c \text{ at } v \text{ obtained by the method in [11].}$$

[2]This bound must be computed for the most "optimistic" scenario with respect to unit resistance and capacitance in order to derive a legitimate lower-bound.

Given $c_{\min}(v)$ and $d_{\min}(v, c)$ at every node v, we use the following bounding technique to discard subsolutions from extending toward the source s. Let (c, d) represent a downstream path at node v. Then, $(d_{\min}(v, c) + d)$ gives a lower-bound on the delay of any path $s \rightsquigarrow v \rightsquigarrow t$ which is induced by the path (c, d). Hence, if $(d_{\min}(v, c) + d) \geq d_k$ of $P(s)$ [recall that (c_k, d_k) is the last solution in $P(s)$], this subpath need not be extended any further since all the source-to-sink paths induced by (c, d) will be dominated by (c_k, d_k).

Similarly, $(c_{\min} + c)$ is a lower-bound on the capacitance of any path $s \rightsquigarrow v \rightsquigarrow t$, which is induced by (c, d). Therefore, if $(c_{\min}(v) + c) \geq c_{\text{spec}}$, then (c, d) need not be considered for extension since all source-to-sink paths induced by (c, d) will violate the capacitance constraint.

Thus, by having the values of $c_{\min}(v)$ and $d_{\min}(v, c)$ at a node v, we can use both the above mentioned bounding schemes to discard subsolutions at early stages. However, if given c_{spec} is so large that no candidate is discarded by the capacitance bound, the effectiveness of our speedup strategy completely depends on the entries in $P(s)$ since the minimum delay solution in $P(s)$ would affect the number of discarded subsolutions—i.e., the lower the value of d_k in P_s, more solutions are likely to be discarded.

Further, the speedup strategy is of no use until we find at least one solution in $P(s)$. Note that finding the first solution in $P(s)$ by the generic algorithm may take a long time since it may generate many intermediate solutions before generating the final solution at s. This problem can be overcome by finding a source-to-sink path in a preprocessing step. While finding $c_{\min}(v)$ for each vertex v, we can also determine a minimum capacitance source-to-sink path and total delay d of the path. $(c_{\min}(t), d)$-pair of the path is clearly the first solution of $P(s)$ (ties in c are broken by d) and it can be used to discard subsolutions.

D. Faster Algorithm

In Fig. 6, we present an improved timing-driven algorithm based on the speed-up strategies explained above. Two steps are added as preprocessing steps to find $l_{\min}(v)$ and $c_{\min}(v)$ in the main routine and additional computations are added to discard subsolutions in the subroutine *Candidates*. Function *DelayLB* in step c5.4 in Fig. 6 gives the lower bound on the delay of the upstream path at v. To avoid the computational overhead in calculating the lower-bound on delay at v, the function *DelayLB* uses a table lookup technique. Given the wire length l of the upstream path p: $s \rightsquigarrow v$ and the loading capacitance c at v, we find an interpolated delay value using a precomputed table.

While in the generic algorithm in Fig. 2, the priority queue contains the (c, d)-pair associated with any downstream $v \rightsquigarrow t$ path, in the Faster algorithm, the priority queue is maintained with $(c + c_{\min}(v),\ d + d_{\min}(v, c))$-pair, where c and d are the respective capacitance and delay values of the downstream $v \rightsquigarrow t$ path, and $c_{\min}$ and $d_{\min}$ are the corresponding lower bounds on capacitance and delay of any upstream $s \rightsquigarrow v$ path. This idea is based on the A^* algorithm in [9], which makes our technique more goal-oriented. Further, by expanding paths according to $(c + c_{\min})$ and $(d + d_{\min})$ and pruning for irredundancy, the algorithm would become much more efficient.

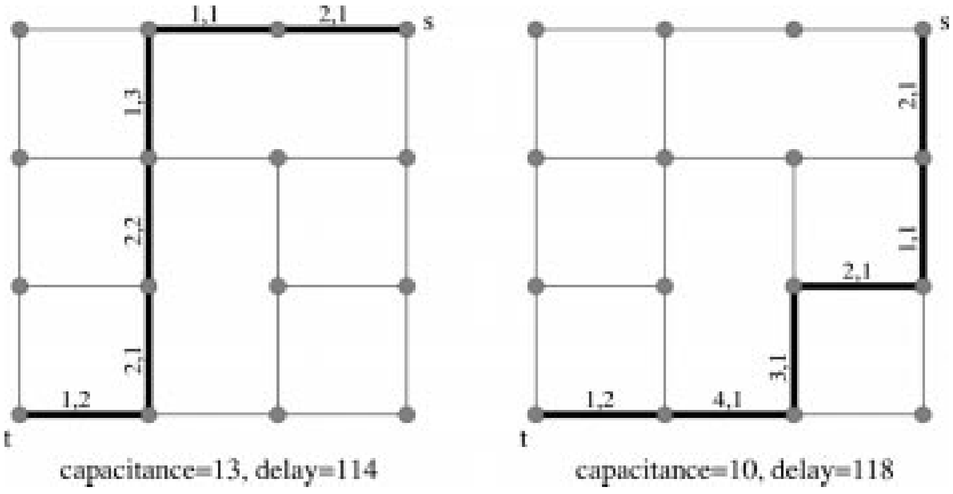

Fig. 4. Two different source-to-sink paths.

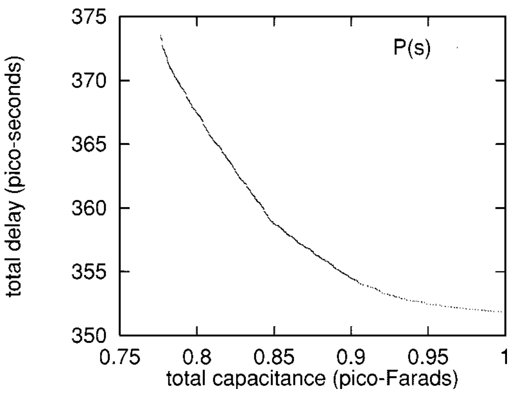

Fig. 5. All the solutions generated by test grid "Grid-4" whose dimension is 100 × 100. Source and sink are selected at random such that the distance between them is 5000 μm in L_1-metric. Solutions are obtained under the assumption that $r_s = 150\,\Omega$, and $c_t = 50$ fF.

Suppose two different subpaths, (c, d) and (c, d'), at node v have the same downstream capacitance c and $d < d'$. Since $d_{\min}(v, c)$ is same for any expanded paths from these subpaths, the priority queue will dequeue $(c + c_{\min}, d + d_{\min}(v, c))$-pair before $(c + c_{\min}, d' + d_{\min}(v, c))$-pair being dequeued if both are enqueued. When the latter path is dequeued and examined for dominance relation check in step m8 in Fig. 6, it will be simply discarded due to the former subpath. This observation supports that at a given node, the property

$$c_1 < c_2 < \cdots < c_k \quad \text{and} \quad d_1 > d_2 > \cdots > d_k$$

would still hold. Thus, the dominance relation check can still be done in constant time in the faster algorithm.

IV. Experiments

A. Test Grid Generation

We generated test grids in order to evaluate the effectiveness and tractability of the algorithms. Our objective was to approximate a typical scenario in practice and we believe that the following strategy does a reasonable job of achieving this goal.

Given the following input parameters characterizing the grid, the procedure for generating the test grids is outlined as follows.

- $n \times m$: The dimensions of the grid.
- M: Maximum number of multiple edges between adjacent grid points.
- I: Spacing of widths of the wires between adjacent grid points.
- r_s: The resistance at the driver.
- c_t: The sink capacitance.
- r_0, c_0: The resistance and capacitance per unit length of the wire as specified by the technology.

We generate a partial multigrid based on the assumption that the grid points are equally spaced along each dimension. Without loss of generality, we assume that the chip dimension is 1 × 1 cm.

A random spanning tree over the set of $n \times m$ points is generated. This ensures that there exists a path between any two points in the grid and hence the source and sink can be chosen arbitrarily. To this tree, we then add $n \times (m-1) + m \times (n-1)$ edges at random, without violating the constraint on the maximum number of multiple edges between adjacent points. By adding exactly this many additional edges, we ensure that we do not generate a complete grid.

The edges between any two grid points are assigned widths starting at w_0, and spaced at an interval I, i.e., w_0, $w_0 + I$, $w_0 + 2 \cdot I, \cdots, w_0 + (k-1) \cdot I$ where k is the number of edges between the adjacent grid points under consideration.

For each edge with length l_e and width w_e, the associated resistance and capacitance values are calculated as follows:

- $r_e = (1 + \epsilon) \cdot r_0 \cdot l_e / w_e$;
- $c_e = (1 + \epsilon) \cdot c_0 \cdot l_e \cdot w_e$;

where ϵ is chosen randomly in the interval $\{-0.1, \cdots, +0.1\}$ to approximate some variance due to factors like coupling.

B. Experimental Results

We have implemented the maze-routing algorithms with C and tested it on eight different grids on a 200-MHz Sun UltraSparc 1. The focus of the experiments was the computational tractability of the various approaches since they all guarantee solution optimality.

The characteristics of the grids are shown in Table I where M is the maximum number of multiple edges between two

```
      Algorithm Faster Timing-Driven Maze Routing
      input: $G = (V, E)$ and $c_{spec}$
      output: minimum delay path s.t. capacitance $c \le c_{spec}$
      Subroutine $Candidates(u, (c, d))$
c1    $L \leftarrow \emptyset$
c2    for each edge $e = (u, v) \in E$ incident to $u$
c3      if $(v = s)$ then // source
c4        $L \leftarrow L \cup \{s, (c + c_e, d + r_e(\frac{c_e}{2} + c) + r_s(c_e + c))\}$
c5      else
c5.1      if $(c_{min}(v) + c \ge c_{spec})$ then continue
c5.2      $d' \leftarrow d + r_e(\frac{c_e}{2} + c)$
c5.3      $c' \leftarrow c + c_e$
c5.4      $d_{min}(v, c') \leftarrow DelayLB\ (l_{min}(v), c')$
          // $d_k$ is minimum delay in $P(s)$
c5.5      if $(d' + d_{min}(v, c') \ge d_k)$ then continue
c6        $L \leftarrow L \cup \{v, (c', d')\}$
c7      endif
c8    endfor
c9    return $L$ ordered by $(c, d)$-pair: first by $c$ and secondarily by $d$
      Main Routine
m0    Compute $l_{min}(v)$ and $c_{min}(v)$, $\forall v$ using Dijkstra's algorithm
m0.1  $P(s) \leftarrow \{(c_0, d_0)\}$
      //$(c_0, d_0)$ is of the minimum capacitive $s$-to-$t$ path obtained by step m0
m1    $P(t) \leftarrow \{(c_t, 0)\}$
m2    $P(u) \leftarrow \emptyset$, $\forall u \ne t$
m3    $Q \leftarrow \emptyset$ // initialize queue
m3.1  $L \leftarrow Candidates(t, (c_t, 0))$
m3.2  for each $(v, (c', d')) \in L$
m3.3      $Enqueue(Q, (v,\ (c' + c_{min}(v), d' + d_{min}(v, c'))))$
m4    while $(Q \ne \emptyset)$
m5        $(u,\ (\hat{c}, \hat{d})) \leftarrow Dequeue(Q)$
m6        if $(\hat{c} > c_{spec})$ then
m7            return $P(s)$
m7.1      $c \leftarrow \hat{c} - c_{min}(u)$ // $c$ = cap. of downstream $u \rightsquigarrow t$ path
m7.2      $d \leftarrow \hat{d} - d_{min}(u, \hat{d})$ // $d$ = delay of downstream $u \rightsquigarrow t$ path
m8        if $(P(u) \not\prec (c, d))$ then
m9            $P(u) \leftarrow P(u) \cup \{(c, d)\}$
m10           $L \leftarrow Candidates(u, (c, d))$
m11           for each $(v, (c', d')) \in L$
m12               if $(P(v) \not\prec (c', d'))$ then
m13                   $Enqueue(Q, (v,\ (c' + c_{min}(v), d' + d_{min}(v, c'))))$
m14           endfor
m15       endif
m16   endwhile
m17   return $P(s)$
```

Fig. 6 Faster timing-driven maze routing.

grid-points and I is the spacing of widths of the wires between adjacent grid points. Annotated (r_e, c_e)-label for each edge e is calculated as explained in the previous section under the assumption that $r_0 = 0.12.\Omega/\mu$m and $c_0 = 0.15$ fF/μm. These values are typical of technology parameters for submicron technology appearing in the literature (e.g., [13]). We set r_s to 150 Ω and c_t to 50 fF for the experiment since we believe that these values are representative of unbuffered global wires.

We select source and sink at random such that $|s_x - t_x| + |s_y - t_y| \approx 5000$ μm where s_x and s_y (t_x and t_y) are x- and y-coordinate for source (sink) on the grid, respectively. This wire length was chosen to approximate a length of wire which is conceivably unbuffered and also benefit from wire sizing optimization. Given a test grid, source and sink, results have been collected over multiple runs of both the generic and faster versions of the algorithm.

For each data we made 10 runs with different source and sink and we show the minimum, average, maximum CPU-time on Table II.

The data clearly shows that a naive implementation of the algorithm is likely to be of little use in practice, but that by applying our speedup technique, runtimes become much more

TABLE I
CHARACTERISTICS OF GRIDS

	Grid-1	Grid-2	Grid-3	Grid-4	Grid-5	Grid-6	Grid-7	Grid-8
I	0.5	0.7	0.5	0.7	0.5	0.7	0.5	0.7
M	3	3	5	5	3	3	5	5
dimension	100×100				50×50			

TABLE II
EACH DATA IS OBTAINED BY TEN RUNS WITH DIFFERENT SOURCE AND SINK. AMONG TEN RUNS IT SHOWS THE MINIMUM, AVERAGE, AND MAXIMUM CPU TIME IN SECONDS. SOURCE AND SINK ARE SELECTED AT RANDOM, SUCH THAT THE DISTANCE BETWEEN THEM IS 5000 μm IN L_1-METRIC

		Grid-1	Grid-2	Grid-3	Grid-4	Grid-5	Grid-6	Grid-7	Grid-8
Generic	min	474.32	559.83	1012.83	1143.63	22.59	18.95	47.84	47.46
	avg	833.14	929.99	2023.33	1926.78	44.22	40.25	93.26	96.24
	max	1386.09	1714.72	3415.58	2983.40	92.48	76.82	172.73	186.37
Faster	min	1.76	1.72	2.40	2.39	0.42	0.41	0.63	0.59
	avg	4.23	3.87	6.58	6.11	0.53	0.49	0.77	0.74
	max	5.67	5.85	8.33	7.35	0.65	0.57	0.86	0.89

practical. The greatest speedup was achieved via delay bounding and is up to 300 times in some cases. Further, since it is expected that only the critical global nets in a design will need to be routed with such a technique, we suggest that the computational results demonstrate the practicality of the algorithm.

Required memory space for the generic algorithm is so huge that it is another factor that makes it impractical. For instance, the average memory space of 10 runs for Grid-1 is 330 MB for the generic algorithm while the improved algorithms consume only 28 MB for the same test instance.

V. DISCUSSION

While routing two-pins is a fundamental problem of interest, we also like to discuss some scenarios where the proposed algorithmic framework may be effectively used and mention some open problems in the area.

A natural extension of the algorithm is accommodating buffer insertion under a given context—i.e., inserting buffers where there are constraints on the buffer locations. The adopted multigraph model can naturally capture such restrictions on buffer locations by allowing buffers to be inserted only at specific nodes in the graph. In such a case, vertices may be labeled to indicate whether buffer insertion is possible or not. If we maintain the same problem formulation (minimization of delay subject to a bound on total capacitance or vice versa), subsolutions at vertex v would then be characterized by triples (c_1, c_2, d), where c_1 is the downstream capacitance seen at v (i.e., up to the first buffer) and c_2 is the total overall capacitance including all segments and the input capacitance of the buffers themselves. This follows the static topology buffer insertion approach presented in [14]. It should be noted that such an extension comes at the expense of higher computational overhead. Some recent work in this direction appears in [15].

Another domain in which the algorithm may find use is that of field programmable gate array routing where highly resistive pass transistors in the programmable routing paths can degrade performance drastically. This scenario is captured naturally in the multigraph model presented (i.e., a pass transistor will correspond to an edge in the multigraph).

In the situation of routing multiple-nets, the algorithm can be used in a rip-up and reroute framework [7]. The edges that are consumed by the route for the current net will not be visible during routing subsequent nets. Finding an appropriate ordering of the nets routing is an interesting open problem and a natural topic for further work (ordering based on some kind of criticality measure seems natural). Another important topic is how to capture and deal the effect of the net currently being routed on previously routed nets (e.g., the previously routed nets may be considered victims of capacitive coupling with the current net). A constraint-driven approach seems plausible, but additional work is needed in this area.

Another natural direction for future work is in addressing multipin nets. A blending of the techniques in this paper and those in [16] for timing-driven Steiner tree construction is a step in that direction.

We note also that there may be some potential practical speedup that can be gained by applying traditional techniques such as windowing. However, we note that it is possible such approaches may eliminate some desirable fast paths with small detours. The impact of this in practice is unclear.

VI. CONCLUSION

We have presented an algorithm and speedup techniques for the timing-driven maze routing problem using a multigraph model. The adopted multigraph model is quite general and naturally captures optimization techniques such as wire sizing via alternative edges.

The basic algorithm is a straightforward labeling algorithm which has pseudopolynomial running time and has been found impractical if implemented naively. Fortunately, the speedup techniques presented in this paper based on lower bounds make the algorithm more practical. These techniques result in speedups of up to 300 times versus the naive implementation, which prove the effectiveness of our speedup techniques.

The algorithms can be adapted with slight modification for other problems with different objectives such as minimization of total capacitance subject to total delay being less than a given $d_{\rm spec}$.

REFERENCES

[1] C. Y. Lee, "An algorithm for path connection and its applications," *IRE Trans. Electron Comput.*, vol. EC-10, pp. 346–365, Sept. 1961.

[2] E. F. Moore, "Shortest path through a maze," in *Annals of the Harvard Computing Laboratory*. Cambridge, MA: Harvard Univ. Press, 1959, pt. II, vol. 30—Proc. Int. Symp. Switching Circuits.

[3] J. Lillis and P. Buch, "Table-lookup methods for improved performance-driven routing," in *Proc. 35th DAC*, 1998, pp. 368–377.

[4] H.-P. Tseng, L. Scheffer, and C. Sechen, "Timing and crosstalk driven area routing," in *Proc. 35th DAC*, 1998, pp. 378–381.

[5] T. Xue, E. S. Kuh, and D. S. Wang, "Post global routing crosstalk risk estimation and reduction," in *Proc. ICCAD*, 1996, pp. 302–309.

[6] H. Zhou and D. F. Wong, "Global routing with crosstalk constraints," in *Proc. 35th DAC*, 1998, pp. 374–377.

[7] R. Nair, "A simple yet effective technique for global wiring," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 165–172, June 1987.

[8] E. W. Dijkstra, *A Discipline of Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1976.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst, Sci. Cybern.*, vol. SCC-4, pp. 100–107, July 1968.

[10] W. C. Elmore, "The transient response of damped linear network with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, pp. 55–63, 1948.

[11] J. P. Fishburn and C. A. Schevon, "Shaping a distributed-*RC* line to minimize Elmore delay," *IEEE Trans. Circuits Syst.—I*, vol. 42, pp. 1020–1022, Dec. 1995.

[12] C.-P. Chen, Y.-P. Chen, and D. F. Wong, "Optimal wire-sizing formula under the Elmore delay model," in *Proc. 33rd DAC*, 1996, pp. 487–490.

[13] M. Kang, W.-M. Dai, T. Dillinger, and D. LaPotin, "Delay bounded buffered tree construction for timing driven floorplanning," in *Proc. ICCAD*, 1997, pp. 702–712.

[14] 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, 1996.

[15] H. Zhou, D. F. Wong, I.-M. Liu, and A. Aziz, "Simultaneous routing and buffer insertion with restrictions on buffer locations," in *Proc. 36th DAC*, 1999, pp. 96–99.

[16] J. Lillis, C.-K. Cheng, T.-T. Y. Lin, and C.-Y. Ho, "New techniques for performance driven routing with explicit area/delay tradeoff and simultaneous wire sizing," in *Proc. 33rd ACM/IEEE DAC*, 1996, pp. 395–400.

Sung-Woo Hur is working toward the Ph.D. degree in electrical engineering and computer science at the University of Illinois at Chicago. His research interests are CAD algorithms and computational geometry. He received the B.S. degree in electronics from the Kyungpook National University in Korea and M.S. degree in computer science from Korea Advanced Institute of Science and Technology in Korea.

Ashok Jagannathan is working toward the M.S. degree in electrical engineering and computer science at the University of Illinois at Chicago. He received the Bachelors degree in Computer Science from the Regional Engineering College, Trichy, India in 1997. His research interests are in the area of physical design automation of integrated circuits.

John Lillis received the B.S. degree in computer science from the University of Washington in 1989 and the M.S. and Ph.D. degrees in computer science from the University of California at San Diego in 1992 and 1996, respectively. From 1996 to 1997, he was an NSF postdoctoral visitor at the University of California at Berkeley. In 1997, he joined the University of Illinois at Chicago where he is currently an Assistant Professor in the Department of Electrical Engineering and Computer Science. His research interests are in the areas of design automation for integrated circuits and combinatorial optimization.