Obstacle-Avoiding Rectilinear Minimum-Delay Steiner Tree Construction towards IP-Block-Based SOC Design

Jingyu Xu, Xianlong Hong, Tong Jing, Yang Yang

Dept. of Computer Science & Technology, Tsinghua Univ., Beijing, 100084, P. R. China

Abstract
With System-on-Chip design, IP blocks form routing obstacles that deteriorate global interconnect delay. In this paper, we present a new approach for obstacle-avoiding rectilinear minimal delay Steiner tree (OARMDST) construction. We formalize the solving of minimum delay tree through the concept of an extended minimization function, and trade the objective into a top-down recursion, which wisely produces delay minimization from source to critical sinks. We analyze the topology generation with treatment of obstacles and exploit the connection flexibilities. To our knowledge, this is the first in-depth study of OARMDST problem based on topological construction. Experimental results are given to demonstrate the efficiency of the algorithm.

1. Introduction
With dramatic increase in the complexity of VLSI circuits, interconnect delay has become the dominant factor in deep sub-micron design in determining the overall circuit performance. Therefore, the problem of constructing a rectilinear Steiner tree (RST) with minimum delay has been studied extensively[1-5]. Typically, the routing process is conducted in the presence of obstacles. These obstacles are occupied by either logic blocks or wires in the previously routed nets. With the trend toward IP-block-based System-on-Chip (SOC) design, it is more likely that the presence of macro cells as obstacles (e.g., memories or noise-sensitive mixed-signal/analog/RF blocks which do not allow over-the-cell routing) may significantly lengthen wires and cause delay deterioration. The rectilinear Steiner minimum tree (RSMT) construction with the treatment of obstacles were studied and several heuristics have been developed [6-10].

However, a very limited number of works[11-13] have considered the delay minimization and the obstacle problem simultaneously, a problem that we refer to as the obstacle-avoiding rectilinear minimal delay Steiner tree (OARMDST) problem. An automatic clock tree design system was developed in [11], with the scheme of changing tapping point to avoid obstacles. [12] proposed a routing tree construction algorithm with simultaneous buffer insertion and wire sizing under obstacle constraints. The routing problem was formulated as a series of graph problems in [12] and the solution was obtained by finding shortest paths in a series of graphs. [13] described a two-step routing approach that optimized timing by generating solutions in the class of Minimal Rectilinear Steiner Arborescence topologies on the substrate with obstructions. The router first built the Minimal Rectilinear Shortest Path Steiner Tree on the coarse Hanan grid with obstructions and then found a detailed solution by a rules-based gridless procedure.

In this paper, we propose a new approach to the OARMDST problem based on topological construction. The important features of our work are as follows.
1) We analyze the topology generation with treatment of obstacles and propose strategy to handle various connection cases during the routing tree construction.
2) We develop a dynamic searching and computation procedure that enables us to solve the minimal delay tree from the source to critical sinks recursively. The similar idea can be applied to general performance-driven Steiner tree construction as well.
3) In exploiting the connection flexibilities, some useful properties are also described.

The remainder of this paper is organized as follows. In Section 2, we give some preliminary definitions and describe the routing problem formulation in presence of obstacles. The OARMDST algorithm is given in detail in Section 3. Section 4 presents heuristics for speeding up the procedure. Section 5 shows the experimental results, followed by an overall conclusion in Section 6.

2. Preliminaries

2.1. Delay model
We adopt the distributed RC delay model developed by Sakurai [14] for delays. Sakurai delay explicitly captures the cascade delay function of a given node with its predecessor nodes, which benefits the solving of the...
routing tree. In the case of multi-terminal nets, the delay is formulated as
\[ T_D(s) = \beta R_s C_s \] (1)
\[ T_D(w) = T_D(v) + \alpha \tilde{c} L_{vw} + \beta \tilde{r} L_{vw} C_w \] (2)

Where node \( s \) is the source, node \( v \) is the predecessor node of node \( w \), \( T_D(w) \) is the Sakurai delay from source \( s \) to \( w \). \( L_{vw} \) is the wire length from \( v \) to \( w \). \( C_w \) is the total capacitance of the downstream Steiner tree of \( w \). \( \alpha \) and \( \beta \) are the wire capacitance and resistance per unit length, respectively. \( R_s \) is the on-resistance of the transistor of the voltage source. \( \alpha \) equals 1.02 and \( \beta \) equals 2.21, which yields 90\% of the signal’s final delay time.

2.2. Problem formulation

The escape segments are a generalization of the line search escape segments used by Hightower[15]. Following the description of an analogy to interstate highway travel in [6], we depict the escape segments for a layout instance in Figure 1(a). An obstacle corresponds to a city. On every side of an obstacle, we generate an escape segment that forms a portion of the obstacle’s beltway. This is for the space requirements between the boundaries of obstacles and nets. To enable connections between obstacles, each beltway escape segment is extended to also form a highway escape segment. It is a maximal segment that ends with its abutment to either an obstacle border or the perimeter of the routing region. We finally extend the segments from the terminals in a manner similar to the highway escape segments.

Motivated by the observation that the escape segments are blocked by obstacles to form further intersections, we combine the escape segments with the conventional underlying grid graph (defined by the intersections of the horizontal and vertical lines drawn through the demand points) through some transformation. Consequently, the routing problems can be solved similarly by approaches for those on the conventional grid graphs. We extend the escape segments of Figure 1(a) to let each of them cross the obstacles and end with the perimeter of the routing region. Thus, the region is divided into a rectangular array of sub regions (see Figure 1(b)). The intersections falling inside the boundaries of obstacles are treated as obstacle points. If any sub-block by partition of an obstacle contains horizontal (or vertical) lines only, we can add a vertical (or horizontal) line to obtain intersections. Hence, an extended routing graph can be formed as shown in Figure 1(c).

**Problem 2.1. OARMDST** Given a set of terminals \( N \), a set of obstacle points \( O \), a source node \( s \in N \) and a critical sink \( t \in N \) on the extended routing graph \( G = (V, E) \), find a routing tree \( T \) for \( N \) such that delay from \( s \) to \( t \) is minimized.

3. The OARMDST algorithm

3.1. General idea

Dynamic programming is an efficient method to solve every sub-problem as the partial solution and compute an optimal solution in a bottom-up fashion. As an application of dynamic programming technique, the Dreyfus-Wagner [16] recursion obtains minimized wire length through searching of all possible topologies for \( N \). Use \( S_t(K \cup \{v\}) \) to denote the Steiner tree of \( K \cup \{v\} \), where \( K \subseteq N \) and \( v \in V - K \), while the definition of \( P_t(K \cup \{v\}) \) is similar, with one extra constraint that \( \deg(v) \geq 2 \). Let \( p(v, w) \) be the shortest path from node \( v \) to \( w \) in \( G \). The following two recursions are used:

\[ P_t(K \cup \{v\}) = \min \{ S_t(K' \cup \{v\}) + S_t(K - K' \cup \{v\}) \} \]
\[ S_t(K \cup \{v\}) = \min \{ \min \{ p(v, w) + S_t(K) \mid w \in K \}, \min \{ p(v, w) + P_t(K \cup \{w\}) \mid w \not\in K \} \} \]

In wire-length minimization, the optimality of a partial solution is source-independent. However, for delay minimization, a partial solution depends on how the sub terminal set yielding the solution is connected to the source. We modify the traditional bottom-up dynamic programming to be a top-down procedure conducted from source to the critical sink. During the procedure, every internal node is treated as a pseudo-source in terms of its downstream sub tree. We introduce three forms of delay, where \( T_i \) and \( T_t \) describe the delay from an internal node to the critical sink in a given sub terminal set and a sub tree, respectively, and \( T_c \) is the delay between two nodes.

**Definition 3.1. Cascaded delays.** CD5(Cascaded Delay in Set) \( T_c(w, K, t) \) is the delay from internal node \( w \) to critical sink \( t \) in sub set \( K \cup \{w\} \) \( T_c(w, K, t) = 0 \) if \( K \cup \{w\} \).
\{w\} does not contain \( t \). For source node \( s \), \( T_c(s, K, t) = T_d(s, t) + \beta R_c C_v \), where \( T_d(s, t) \) is the Sakurai delay from \( s \) to \( t \) and \( K = N - \{s\} \). Similarly, CD{T(cascaded delay in tree)} \( T_v(S_m, t) \) is the delay from internal node \( w \) to \( t \) in sub Steiner tree \( S_m \).

Hence, we have the following two equations:

\[
T_v(s, K, t) = T_d(s, t) + \beta R_c C_v + \sigma \alpha \sum_{w \in P} L_w \quad \text{(5)}
\]

\[
T_v(s, K, t) = T_d(s, t) + \beta R_c C_v + (\alpha \sigma \sum_{w \in P} L_w)^{\frac{1}{2}} + T_v(w, K, t) \quad \text{(6)}
\]

where \( i_k = \begin{cases} 1, \text{when } t \in K \\ 0, \text{when } t \notin K \end{cases} \), \( K' \subset K \land K^c \neq \emptyset \), \( \nu \notin O \land \omega \notin O \).

We rewrite (5) and (6) in the form of Dreyfus-Wagner recursion. Let \( S_v(K \cup \{\nu\}, R_w) \) be the minimum delay Steiner tree of \( K \cup \{\nu\} \) corresponding to \( R_w \), where \( R_w = R_v + \tilde{r} \sum_{x \in \text{path}(\nu, v)} L_w \). A similar definitions holds for \( P_v(K \cup \{\nu\}, R_w) \).

\[
\bigcup \{\nu\}, R_w), \text{ with one extra constraint that } \deg(\nu) \geq 2. \text{ Let } T_v(S_v(K \cup \{\nu\}, R_w), t) \text{ be the delay from } \nu \text{ to critical sink } \nu t \text{ in Steiner tree } S_v. \text{ We derive the recursive functions for minimum delay:}
\]

\[
T_v(P_v(K \cup \{\nu\}, \{\nu\}, t)) = \min\{T_v(S_v(K \cup \{\nu\}, \{\nu\}, t)) + T_v(P_v(K \cup \{\nu\}, \{\nu\}, t)) \}
\]

\[
T_v(S_v(K \cup \{\nu\}, \{\nu\}, t)) = \min\{\min\{T_v(S_v(K \cup \{\nu\}, \{\nu\}, t)) + T_v(P_v(K \cup \{\nu\}, \{\nu\}, t)) \}
\]

\[
\quad + (\beta R_c C_v + \sigma \alpha \sum_{w \in P} L_w) \}
\]

\[
\quad \min\{T_v(S_v(K \cup \{\nu\}, \{\nu\}, t)) + T_v(P_v(K \cup \{\nu\}, \{\nu\}, t)) \}
\]

\[
\quad + (\beta R_c C_v + \sigma \alpha \sum_{w \in P} L_w) \}
\]

\[
\end{align}
\]

We extend the minimization objective with a new function \( \min \) in (7) and (8). Using \( T_v(s, w) \) to denote the delay between node \( s \) and \( w \), \( \min \) is given by the following definition.

\[
\min T_v(s, w, t) = T_d(s, w, t) + T_v(w, K, t) \quad \text{(9)}
\]

where \( T_d(s, w, t) = \beta R_c C_v + \sigma \alpha \sum_{w \in P} L_w \)

and \( w = \{w|\min T_d(s, w, t) = T_d(s, w, t) + T_v(w, K, t)\} \)

For a fixed internal node \( v \) that contains critical sink \( t \) in its downstream Steiner tree, the delay from source \( s \) to \( t \) (denoted as \( T_d(s, t) \)) is affected by tapping point \( w \). According to (1) and (2), the variation part of \( T_d(s, t) \) by changing \( w \) can be extracted as below:

\[
T_d(s, t) = \beta R_c C_v + \sigma \alpha \sum_{w \in P} L_w \}
\]

\[
\quad + \beta \sum_{x \in y \in P} L_x C_y \}
\]

\[
\end{align}
\]

\( w \) yielding the minimum \( T_d(s, t) \) gives the optimal \( T_d(s, t) \) for a fixed \( v \). Therefore, the solving of \( w \) in (9) can be translated into calculating (10) and finding \( w \) yielding minimum \( T_d(s, t) \).

Equation (7) and (8) imply a dynamic searching procedure. For a given set of terminals \( N \), the optimal topology is either a \( S \) tree or \( P \) tree according to (7) and (8), both of which determine a tapping point (say, \( w \)) for connecting the source \( s \) and the set of rest sinks \( K \) \( K = \{N\} - s \). For subset \( K \), we recursively find its \( S \) tree and \( P \) tree as well. Here, we call terminals such as source \( s \) the higher-level terminals and sets such as \( K \) the lower-level subsets. The final routing tree is created by increasingly considering lower-level subsets in the procedure. The coefficient of \( i_k \) in (8) accounts for the delay from the higher-level terminal to the lower-level subset.

3.2. Routing tree construction with the treatment of obstacles

For a subset \( K \), the Steiner tree \( T_k \) is connected to the higher-level terminal \( v \) by tapping point \( w (w \in V) \) through shortest path routing between \( v \) and \( w \). Previous shortest connections (in the following, the terms “previously routed wires” or “previous connections” are used interchangeably to refer to connections in the same net) may affect later connections by occupying available grids. Figure 2 gives an example, where Figure 2(a) shows the case of \( S \) tree for node \( s1 \) with \( \deg(s_1) = 1 \). The lower-level shortest connection of \( v_1v_2 \) is blocked by \( s_1v_1 \). The case of \( P \) tree is shown in Figure 2(b), where the \( \deg(s_1) \) is at least 2 and the later connection is \( s_1v_2 \). For cases like Figure 2(a), if the blockage causes overlapping, there are essentially two ways to obtain a new routing tree through changing previous connections: we can either alter the shortest path of previous connection (i.e., \( s_1 \) to \( v_1 \), and assume the delay is computed from \( s_1 \) to \( v_l \)) or merge the overlapped part. Suppose the resulting trees are \( T_n \) and \( T_m \), respectively and the nearest common root of branches containing \( s_1v_1 \) and \( v_1v_2 \) is \( s_0 \) (in Figure 2(a) and Figure 2(b), \( s_1 \) is \( s_0 \)). As will be proven later, if merging will not lead to the change of \( \deg(s_0) \), therefore \( s_0 \) has the same degree in \( T_n \) and \( T_m \). Similarly, \( T_m \) turns out to have better delay performance than \( T_n \) (e.g., in Figure 2(a), the rightmost topology \( T_m \) is derived by transformation of the leftmost, which has better delay of \( s_1v_1 \) and \( s_1v_2 \) than the middle tree \( T_n \). However, the same conclusion does not hold for Figure 2(b), where \( s_1 \) has different degree in \( T_n \) and \( T_m \). For cases like Figure 2(b), we observe that the right tree \( T_m \) can be obtained by changing the searching order to be \( s_1, v_2, v_1 \) and taking \( s_1v_2 \) as fixed obstacles. However, it does not guarantee the minimized delay. As shown in Figure 3, connections \( AB \) and \( CD \) are contained in two branches that incident on a higher-level node \( s_0 \). The dashed line in Figure 3(a) represents the overlapping caused by shortest paths routing. Routing trees obtained by altering searching order are shown in Figure 3(b) and Figure 3(c), respectively. However, the minimized delay may be yielded by Figure 3(d), where neither of the two connections is a shortest path connection. As will be seen later, we apply a \( shove \) operation to exploit the connection flexibilities.
Figure 2. Illustration of the connected wires as obstacles for later connections. (a) \(v_1v_2\) is blocked by routed connection \(s_1v_1\) in \(S\) tree of node \(s_1\). (b) \(s_1v_2\) is blocked by \(s_1v_1\) in \(P\) tree of \(s_1\).

Figure 3. A blockage instance. (a) \(AB\) have blockage with \(CD\). (b) Search \(CD\) first. (c) Search \(AB\) first. (d) The optimal sub tree.

3.3. Generalization

We generalize the case of blockage into sub cases.

**Definition 3.2. Blockage.** Given a pair of nodes \(v_1\) and \(v_2\) and a set of obstacle points \(O_o\), there exists an obstacle-\(O_o\)-avoiding shortest path between \(v_1\) and \(v_2\), denoted as \(p_o\). During the construction procedure using (7) and (8), obstacle points \(O' = O_o + \{\text{grids occupied by previous connections}\}\). Suppose the obstacle-\(O'\)-avoiding shortest path between \(v_1\) and \(v_2\) is \(p_o\). If length\((p_o)\) > length\((p_o)\), then this is called a blockage between wire \(v_1v_2\) and previous connections. Blockage causes either cross point or overlapping.

**Definition 3.3. \(T_m\).** Suppose that the blockage happens between wire \(v_1v_2\) and a previous connection \(s_1v\). The new trees obtained by altering \(s_1v\) through other shortest paths without causing blockage are called \(T_m\).

**Definition 3.4. \(T_n\).** Suppose that the blockage happens between wire \(v_1v_2\) and a previous connection \(s_1v\). If blockage causes overlapping, the new topology obtained by merging overlapped part is called \(T_n\).

The blockage can be one of the following cases only:

**Case A.** The shortest path of \(v_1v_2\) is blocked by a connection \(s_1v\) only.

**Case A1.** \(v_1v_2\) is a lower-level connection contained in the downstream Steiner tree of \(v\). The searching order is \(s_1, v, v_1\) and \(v_2\). Therefore \(s_1\) acts as nearest common root \(s_0\).

**Case A10.** Cross point blockage.

**Case A11.** Overlapping blockage. Merging will not lead to the change of \(\text{deg}(s_0)\). Thus \(T_n\) has same \(\text{deg}(s_0)\) as the merging tree \(T_m\).

**Case A12.** Overlapping blockage. Merging leads to the change of \(\text{deg}(s_0)\). Thus \(T_n\) has different \(\text{deg}(s_0)\) as the merging tree \(T_m\).

**Case A2.** \(s_1v\) and \(v_1v_2\) are contained in two branches that incident on \(s_0\) \(\big\{\text{a pair of complementary subsets} K' \text{ and } K-K' \big\}\) defined in equation (5), where \(K\) is the lower-level set of \(s_0\).

**Case B.** The shortest path of \(v_1v_2\) is blocked by at least two previous connections.

Essentially, case A1 and A2 generalize blockage cases in construction of the downstream \(S\) tree and \(P\) tree for node \(s_0\) respectively. Case B involves both the two cases. We state the following theorem for case A1.

**Theorem 3.1.** Suppose that \(v_1v_2\) is a lower-level connection contained in the downstream Steiner tree of \(v\) and the searching order is \(s_1, v, v_1\) and \(v_2\). If the shortest path between \(v_1\) and \(v_2\) is blocked by \(s_1v\), then for \(T_n\) that has same \(\text{deg}(s_1)\) as the merging tree \(T_m\), \(T_m\) has better delay performance than \(T_n\).

**Proof.** We use the symbols labeled in Figure 4 to conduct the proof. Figure 4 gives a general example for case A1, where \(s_1\) acts as \(s_0\) of the two connections. The possible shortest paths between \(s_1\) and \(v\) are shown as dashed curves in Figure 4(a), where the actual connection (the lower dashed curve) blocks the connection between nodes \(v_1\) and \(v_2\) with overlapped part \(w_1w_2\). Figure 4(b) and Figure 4(c) illustrate \(T_m\) and \(T_n\), respectively, where \(T_n\) has same \(\text{deg}(s_1)\) as the merging tree \(T_m\). The delay from \(s_1\) to \(v\) in \(T_n\) is given by \(T'_d(s_1, v)\). Note that if there are higher-level connections connected to \(s_j\), \(R_j\) can be looked as the on-resistance of source plus the wire resistance from source to \(s_j\).

\[
T_d(s_1, v) = \beta R_s C_s + \alpha cL^2_{v_1v_2} + \beta R_{s_2}C_{s_2} + \alpha \tilde{c}L^2_{w_1w_2} + \beta R_{w_1}C_{w_1} + \alpha \tilde{c}L^2_{w_1w_2} + \beta R_{w_2}C_{w_2}
\]

where \(C_s\) is the total downstream capacitance of \(v\).

For \(T_n\), the delay from \(s_1\) to \(v\) is given by \(T'_d(s_1, v)\).

\[
T'_d(s_1, v) = \beta R_s C_s + \alpha cL^2_{v_1v_2} + \beta R_{v_1}C_{v_1}
\]

Figure 4. Illustration of blockage. (a) \(s_1v\) blocks later connection \(v_1v_2\). (b) \(T_m\) (c) Altering the shortest connection of \(s_1v\) to get \(T_m\). (d) The case that \(T_m\) has different \(\text{deg}(s_0)\) from \(T_n\). (e) \(w_1\) occupies the another degree of \(v\) for shortest connection. (f) A possible topology generated from (e).
Since $T_n$ has a greater wire length than $T_m$, $C_{T_n} > C_{T_m}$. 

$$T'_d(s_1, v) - T'_d(s_1, v) > \alpha \lambda w_{s_1} + \beta L_{s_1} C_v - (\alpha \lambda w_{s_1}) + \beta L_{s_1} C_v = - (\alpha \lambda w_{s_1}) + \beta L_{s_1} C_v = 0$$

Note that $L_{s_1} - L_{s_2} + L_{w_2} - L_{w_1} + L_{w_1} - C_v = C_{w_2} - C_{w_1} > C_v$. 

Substituting it into (11),

$$T'_d(s_1, v) - T'_d(s_1, v) > \beta L_{s_1} C_v - (\alpha \lambda w_{s_1}) + \beta L_{s_1} C_v = \beta L_{s_1} C_v$$

This is because all the possible shortest paths between a pair of nodes can only occupy at most two degrees of one node. Therefore, we can slide the overlapping points $w_1$ and $w_2$ along one of the path like shown in Figure 4(a). If $w_1$ crosses $v$, there can be two cases only: (i) $w_2$ occupies the another degree of $v$ for shortest path connection (the upper curve between $s_1$ and $v$). Then it becomes the case shown in Figure 4(e), which can be transformed into constructing the $P$ tree of $w_1$ or $w_2$, as the example shown in Figure 4(f). (ii) $w_1/w_2$ generates a downstream tree of $v$ (In Figure 4(b), $w_1$ is slid towards $v$ to form a sub tree incident on $v$). This case belongs to case A11 and has been included in above proof by considering $C_v$. Similarly, if $w_2$ crosses $s_1$ as shown in Figure 4(d), it becomes case A12, which causes a loop: $s_1-v_1-s_1-v_2$. If $v_2$ is slid to $w_1$ (or $v_1$ is slid to $w_2$), it is case A10.

For case A11, we prove that $T_m$ has better delay performance than $T_n$ while $T_m$ can be obtained by taking $w_2$ as a tapping point. Hence, we discard non-optimal $T_n$ and skip current searching thread.

As shown in Figure 4(d), case A12 implies a loop that cannot be eliminated by altering connection $s_1/v$ using $T_n$. Therefore, we skip current searching thread for case A12.

If blockage causes overlapping, case A2 also implies a loop: $s_{p/r}v_{p/r}$ overlapped part and $s_{p/r}v_{p/r}$ overlapped part. However, it can be avoided by using available $T_m$. We apply two strategies to handle case A2: i) Treat $v_{p/r}$ as obstacles and find obstacle-avoiding shortest path for $s_{p/r}$ to obtain available $T_m$. ii) If $T_m$ does not exist, invoke shove to generate possible detoured connections as candidate solutions. The basic idea of shove (see Figure 3) is to generate new paths based on transformation of blocked wire segments. For overlapping blockage, we increasingly consider more segments on the overlapped wires and shove each segment to left(up) or right(down) direction to form a detoured sub path. The shove description is given in Figure 5, where the initial input are two blocked connections and original obstacle points $O$.

Output routing trees are compared for delay at the highest-level common root among the paths altered.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>shove(AB, CD, O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB, CD: the connections having blockage</td>
<td></td>
</tr>
<tr>
<td>$O$: the obstacles points</td>
<td></td>
</tr>
<tr>
<td>1. if cross point blockage</td>
<td></td>
</tr>
<tr>
<td>2. then $O' = O' + [AB]$</td>
<td></td>
</tr>
<tr>
<td>3. find obstacle-$O'$-avoiding shortest path for CD</td>
<td></td>
</tr>
<tr>
<td>4. if path not found, return</td>
<td></td>
</tr>
<tr>
<td>5. if CD is blocked by path EF, shove(CD, EF, O')</td>
<td></td>
</tr>
<tr>
<td>6. else store tree into temporary Setresult</td>
<td></td>
</tr>
<tr>
<td>7. then $O' = O' + [CD]$</td>
<td></td>
</tr>
<tr>
<td>8. find obstacle-$O'$-avoiding shortest path for AB</td>
<td></td>
</tr>
<tr>
<td>9. if path not found, return</td>
<td></td>
</tr>
<tr>
<td>10. if AB is blocked by path GH, shove(AB, GH, O')</td>
<td></td>
</tr>
<tr>
<td>11. else store tree into temporary Setresult</td>
<td></td>
</tr>
<tr>
<td>12. else /<em>overlapping blockage</em>/</td>
<td></td>
</tr>
<tr>
<td>13. construct Setresult to store overlapped segments</td>
<td></td>
</tr>
<tr>
<td>14. determine the relative position of AB and CD</td>
<td></td>
</tr>
<tr>
<td>15. if $O$ is left/up wire, $C:= right/down wire$</td>
<td></td>
</tr>
<tr>
<td>16. if Setresult $\notin \Phi$</td>
<td></td>
</tr>
<tr>
<td>17. then $C_2$ shoves $C_1$ to left(up) direction/*</td>
<td></td>
</tr>
<tr>
<td>18. shove-leftup($C_1, C_2, Setresult, O$)</td>
<td></td>
</tr>
<tr>
<td>19. else $C_1$ shoves $C_2$ to right(down) direction/*</td>
<td></td>
</tr>
<tr>
<td>20. shove-rightdown($C_1, C_2, Setresult, O$)</td>
<td></td>
</tr>
<tr>
<td>/* shove-rightdown is similar */</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. The shave algorithm.

Similarly, case A10 is searching-order-dependent and is applied with the same strategy as case A2. For case B, blockage sections are handled one by one.

In realization of OARMDST algorithm, we adopt the Rubin algorithm [17] for single-pair shortest path finding and extend it to handle the non-uniform grid graphs. The treatment with respect to different case of blockage is referred to the discussion above.

4. Experimental results

We have implemented both OARMDST algorithm and Obstacle-Avoiding RSMT algorithm in C language, and tested them on a Sun Enterprise 450 workstation. The Obstacle-Avoiding RSMT algorithm is based on traditional dynamic programming that gives the optimal wire-length routing trees.

Microelectronics Center of North Carolina (MCNC) benchmark circuits are used as the test data. Obstacles are generated by a rule-based random procedure, with the number ranging from 1 to 8, to cover the possible routing
area of the nets. According to (1)-(2), we choose $\alpha=1.02$ and $\beta=2.21$ in delay calculation. The parameters for $2\mu m$ technology are transformed to $0.2\mu m$ with unit length wire resistance $0.198\Omega/\mu m$, capacitance $0.2fF/\mu m$.

Table 1 gives delay and wire length comparison of the two algorithms (the Obstacle-Avoiding RSMT is denoted as OA-RSMT in the table). Row index is the circuit name and serial number of nets in MCNC benchmark. Compared with the Obstacle-Avoiding RSMT, the OARMDST algorithm is capable of delivering an up to 17% delay reduction in the final solution.

The run times of the two algorithms depend on both net configuration and distribution of obstacles. For nets with less than 6 pins, the run times of the two algorithms are comparable. For nets with 6 or more pins, the Obstacle-Avoiding RSMT algorithm is much faster. This is because in minimum wire-length tree construction, the overlapped shortest connections can be directly merged to form a new topology with smaller wire length. Therefore, in implementation of the Obstacle-Avoiding RSMT algorithm, we do not inspect the blockage case, which saves a lot of computations.

The wire length comparison of our approach with the Obstacle-Avoiding RSMT is given in the last three columns of Table 1. Since the Obstacle-Avoiding RSMT algorithm represents the optimal wire length performance in presence of obstacles, we give the wire length increase of our algorithm in the table.

<table>
<thead>
<tr>
<th>Net No.</th>
<th>Number of Pins</th>
<th>OA-RSMT Delay (ns)</th>
<th>OARMDST Delay (ns)</th>
<th>% Delay Improvement</th>
<th>OA-RSMT CPU Time (s)</th>
<th>OARMDST CPU Time (s)</th>
<th>OA-RSMT Wire Length (\mu m)</th>
<th>OARMDST Wire Length (\mu m)</th>
<th>% Wire Length Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>C5:9</td>
<td>3</td>
<td>0.255</td>
<td>0.229</td>
<td>10.2%</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>1578</td>
<td>1596</td>
<td>1.1%</td>
</tr>
<tr>
<td>C5:596</td>
<td>3</td>
<td>0.489</td>
<td>0.406</td>
<td>17.0%</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>2454</td>
<td>2523</td>
<td>3.0%</td>
</tr>
<tr>
<td>C5:684</td>
<td>4</td>
<td>0.302</td>
<td>0.279</td>
<td>7.6%</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>1482</td>
<td>1482</td>
<td>0.0%</td>
</tr>
<tr>
<td>C5:303</td>
<td>4</td>
<td>0.481</td>
<td>0.404</td>
<td>16.0%</td>
<td>&lt;0.001</td>
<td>0.01</td>
<td>2144</td>
<td>2326</td>
<td>8.5%</td>
</tr>
<tr>
<td>C5:400</td>
<td>4</td>
<td>0.526</td>
<td>0.477</td>
<td>9.3%</td>
<td>&lt;0.001</td>
<td>&lt;0.001</td>
<td>2120</td>
<td>2414</td>
<td>13.9%</td>
</tr>
<tr>
<td>C5:494</td>
<td>5</td>
<td>0.284</td>
<td>0.246</td>
<td>13.4%</td>
<td>&lt;0.001</td>
<td>0.02</td>
<td>1410</td>
<td>1452</td>
<td>3.0%</td>
</tr>
<tr>
<td>C5:469</td>
<td>5</td>
<td>0.298</td>
<td>0.277</td>
<td>7.0%</td>
<td>0.02</td>
<td>0.05</td>
<td>1392</td>
<td>1394</td>
<td>0.0%</td>
</tr>
<tr>
<td>C5:633</td>
<td>6</td>
<td>0.571</td>
<td>0.524</td>
<td>8.2%</td>
<td>0.21</td>
<td>19.49</td>
<td>2816</td>
<td>3156</td>
<td>12.1%</td>
</tr>
<tr>
<td>C5:695</td>
<td>6</td>
<td>0.393</td>
<td>0.366</td>
<td>6.8%</td>
<td>0.29</td>
<td>43.87</td>
<td>2254</td>
<td>2416</td>
<td>7.2%</td>
</tr>
<tr>
<td>C5:80</td>
<td>7</td>
<td>0.248</td>
<td>0.231</td>
<td>6.9%</td>
<td>0.08</td>
<td>3.30</td>
<td>1560</td>
<td>1666</td>
<td>6.8%</td>
</tr>
<tr>
<td>C5:226</td>
<td>7</td>
<td>0.441</td>
<td>0.409</td>
<td>7.3%</td>
<td>0.30</td>
<td>103.45</td>
<td>3072</td>
<td>3138</td>
<td>2.1%</td>
</tr>
<tr>
<td>C5:273</td>
<td>8</td>
<td>0.334</td>
<td>0.288</td>
<td>13.8%</td>
<td>1.27</td>
<td>498.35</td>
<td>2048</td>
<td>2276</td>
<td>11.1%</td>
</tr>
</tbody>
</table>

References