EFFICIENT HEURISTICS FOR THE MINIMUM SHORTEST PATH STEINER ARBORESCENCE PROBLEM WITH APPLICATIONS TO VLSI PHYSICAL DESIGN

Jason Cong[‡], *Andrew B. Kahng*[†‡], *and Kwok-Shing Leung*[†‡]

[†] Cadence Design Systems, San Jose, CA 95134
[‡] UCLA Computer Science Department, Los Angeles, CA 90095-1596

ABSTRACT

Given an undirected graph $G = (V, E)$ with positive edge weights (lengths) $w : E \rightarrow \Re^+$, a set of terminals (sinks) $N \subseteq V$, and a unique root node $r \in N$, a *shortest-path Steiner arborescence* (simply called an arborescence in the following) is a Steiner tree rooted at r spanning all terminals in N such that every source-to-sink path is a shortest path in G. Given a triple (G, N, r), the Minimum Shortest-Path Steiner Arborescence (MSPSA) problem seeks an arborescence with minimum weight. The MSPSA problem has various applications in the areas of VLSI physical design, multicast network communication, and supercomputer message routing; various cases have been studied in the literature. In this paper, we propose three efficient heuristics for the MSPSA problem and present applications to VLSI physical design. Experiments indicate that our algorithms generate near-optimal results and achieve speedups of several orders of magnitude over existing algorithms.

1. INTRODUCTION

Given an undirected graph $G = (V, E)$ with positive edge weights (lengths) $w : E \rightarrow \Re^+$, a set of terminals (sinks) $N \subseteq V$, and a unique root node $r \in N$, a *shortest-path Steiner arborescence* (simply called an arborescence in the following) is a Steiner tree rooted at r spanning all terminals in N such that every source-to-sink path is a shortest path in G. Given a triple (G, N, r), the Minimum Shortest-Path Steiner Arborescence (MSPSA) problem seeks an arborescence with minimum weight.

The MSPSA problem is a special case of the Minimum Steiner Arborescence (MSA) problem, which has been well studied in the literature (for example, [13, 10]). Given a triple (G, N, r) wherein G is a directed graph, the MSA problem seeks a minimum-weight Steiner tree spanning all nodes in N with all edges directed away from r. If G' is the shortest-path directed acyclic graph of G (defined in the next section), it is easy to see that an MSA of G' is an MSPSA of G. Both the MSA and the MSPSA problems are NP-hard [13, 2].

The rectilinear version of the MSPSA problem is called the Minimum Rectilinear Steiner Arborescence (MRSA) problem. Given a set of terminals N (including the root r located at the origin), let $G_{H(N)} = (V_{H(N)}, E_{H(N)})$ be the induced Hanan grid graph [11] of N. It can be shown that an MSPSA of $(G_{H(N)}, N, r)$ is an MRSA of N. Exact methods for the MRSA problem can be classified into (1) dynamic programming, (2) integer programming, and (3) branch-and-bound/enumeration techniques. The DP-based approach was first used in the work of Ladeira de Matos [16], and more recently in the RSA/DP algorithm by Leung and Cong [18]. Nastansky *et al.* [20] formulated the MRSA problem (and its D-dimensional generalization) as an integer program, and solved it with implicit enumeration techniques. Cong and Leung presented the Atree/BnB [5] and RSA/BnB [18] algorithms, both of which employ branch-and-bound techniques to effectively prune the search space. Finally, Ho *et al.* [12] gave two exhaustive enumeration algorithms with $O(|N|^{3k})$ (k is the number of "dominating" layers) and $O(|N|^2 3^{|N|})$ runtime complexities, respectively.

Rao *et al.* [21] presented the RSA algorithm, which was the first known heuristic for the MRSA problem; the RSA output has length no more than twice optimal, with runtime being $O(|N| \log |N|)$ if all terminals are located in the first quadrant, and $O(|N|^3 \log |N|)$ in the general case. Runtime for the general case was improved to $O(|N| \log |N|)$ by Córdova and Lee [8]. In [6], Cong *et al.* presented the Atree algorithm, based on making "safe moves". Téllez and Sarrafzadeh [23] gave the pRDPT algorithm, which is based on optimally solving a restricted version of the MRSA problem. More recently, Kahng and Robins gave a simple adaptation of their Iterated 1-Steiner algorithm to the MRSA problem [15], and Leung and Cong presented the k-IDeA algorithm whose performance is very close to optimal in practice [19].

The hypercube version of the MSPSA problem, also called the Optimal Communication Tree or Optimal Multicast Tree problem in the literature, has been studied by Choi *et al.* [3, 2], Lan *et al.* [17], and Sheu and Su [22]. The problem is NP-hard [2], and heuristics include the LEN heuristic [17], the COVER heuristic [2], and the more recent ShSu heuristic [22].

There has been relatively little research on the general MSPSA problem. In [1], Alexander and Robins presented the Path Folding (PFA) algorithm, an adaptation of the RSA heuristic, and the IDOM algorithm, which iteratively adds the best Steiner node as a terminal (analogous to the Iterated 1-Steiner algorithm). They further showed that the MSPSA problem cannot be approximated within a factor of $\Theta(\log |N|)$ times optimal unless deterministic polylog space coincides with non-deterministic polylog space.

The MSPSA and the MRSA problems have applications to performance-driven VLSI physical design. Cong *et al.* showed that rectilinear Steiner arborescences outperform traditional heuristic Steiner minimum trees for delay optimization in submicron process technology [6]. Alexander and Robins applied the PFA and IDOM algorithms to route timing-critical nets in FPGAs [1]. Cong and Madden [7] proposed a multi-source routing algorithm based on constructing minimum-cost minimum-diameter arborescences.

In this paper we propose three efficient heuristics for the MSPSA problem and present applications to VLSI physical design. The first algorithm, called the RSA/G algorithm, is based on the RSA heuristic for the MRSA problem. We then describe the k-IDeA/G algorithm which generalizes the k-IDeA algorithm [19]. The last algorithm, k-IA/G, improves upon the complexity of the IDOM algorithm of Alexander and Robins. Finally, we present experimental results which show that our algorithms generate near-optimal results and achieve speedups of several *orders* of magnitude over existing arborescence algorithms. Due to space limitations, all theorems are presented without proof. Readers may refer to [4] for details.

2. PRELIMINARIES

Given $G = (V, E)$, we define the *distance label* of $v \in V$, denoted $\Delta(v)$, to be the shortest-path distance of v from r in G. The shortest-path directed acyclic subgraph (SPDAG) of G is denoted $G' = (V', E')$, with $V' = V$ and the directed edge $(v, v') \in E'$ if and only if $(v, v') \in E$ and $\Delta(v') - \Delta(v) = w(v, v')$. Clearly, any arborescence of G is a subgraph of G', hence we focus on solving the MSPSA problem on SPDAGs (with proper orientation of the edges). Given a general graph G, its SPDAG G' can be constructed in $O(|E| + |V| \log |V|)$ time using Dijkstra's algorithm with a Fibonacci heap [9]. We rank the nodes of V in order of increasing distance labels, and we use v_i, $1 \leq i \leq |V|$, to denote the i^{th}-ranked node, where v_1 is the root, and $v_{|V|}$ is the farthest node from the root (Dijkstra's algorithm can be modified to output this ranking without increasing runtime or space complexity). The following discussion assumes that the input graph G is already an SPDAG, and we do not distinguish between G and G' unless otherwise noted. For simplicity, we further assume that $v_{|V|}$ is a terminal (otherwise, we can find the maximum i such that $v_i \in N$ and remove nodes $v_{i+1}, \cdots, v_{|V|}$ and their incident edges from G, since none of them are in any source-to-sink shortest path).

Given $(v, v') \in E$, v is called a *parent* of v', and v' a *child* of v. We use C_i to denote the set of children of v_i in G. That is, $C_i = \{v \mid (v_i, v) \in E\}$. Given two nodes $v, v' \in V$, we say v' is *reachable* from v, denoted $v \preceq v'$, if and only if there exists a (directed) path in G from v to v', and $v \prec v'$ if and only if $v \preceq v'$ and $v \neq v'$. If $v \preceq v'$, then $v \rightsquigarrow v'$ denotes a shortest path from v to v' in G. Unless otherwise noted, in the following we assume $v, v', v'' \in V$ and $1 \leq i, j, k \leq |V|$.

3. THE RSA/G ALGORITHM

We begin by reviewing the Minimum Rectilinear Steiner Arborescence (MRSA) problem. Recall that a rectilinear Steiner arborescence is a Steiner tree in the Manhattan plane spanning all terminals in N, such that each source-to-sink path is a rectilinear shortest path. In [21], Rao *et al.* presented the RSA heuristic which constructs an arborescence in a bottom-up fashion, starting with $|N|$ subtrees each consisting of a terminal in N. RSA iteratively *merges* a pair of subtree roots v and v' such that $\langle v, v' \rangle$ is as far from the source as possible, where $\langle v, v' \rangle$ is the point on the bounding box of v and v' that is closest to r. The algorithm terminates when only one subtree remains.

A straightforward generalization of RSA to the MSPSA problem is as follows. Let P be the set of active root nodes (initially $P = N$). Then, iteratively find a node $v \in V$ such that (1) there exist $v', v'' \in P$ ($v' \neq v''$) with $v \preceq v'$ and $v \preceq v''$, and (2) $\Delta(v)$ is maximized among all such nodes satisfying (1). Then, for each $v' \in P$ with $v \preceq v'$, construct a shortest path $v \rightsquigarrow v'$ and remove v' from P. Finally, insert v into P. This process is repeated until $P = \{r\}$. Alexander and Robins gave a straightforward implementation of this approach, called the Path Folding Algorithm (PFA) [1]. Because the PFA algorithm requires frequent computation of the least common ancestor of pairs of nodes in the SPDAG (up to $O(|V||N|^2)$ times), its overall time complexity is $O(|N||E| + |V||N|^2 \log |V|)$.

We adopt a slightly different approach, visiting the nodes in V in decreasing rank order (i.e., starting from $v_{|V|}$), and maintaining a *peer set* consisting of all the subtree roots whose ranks are higher than the rank of the current node. We use P_i and A_i to respectively indicate the peer set and the partially constructed arborescence after visiting v_i (and before visiting v_{i-1}). Let $X_i = \{v \mid v_i \prec v \text{ and } v \in P_{i+1}\}$ be the subset of P reachable from v_i, just before v_i is visited). There are three possible scenarios:

- Terminal Merger Opportunity (TMO): $v_i \in N$
- Steiner Merger Opportunity (SMO): $v_i \notin N$ and $|X_i| > 1$
- Otherwise: $v_i \notin N$ and $|X_i| \leq 1$

If either TMO or SMO applies, we *merge* all the nodes in X_i (if any) into v_i, and update the peer set and the arborescence respectively, i.e. $P_i = P_{i+1} - \{X_i\} + \{v_i\}$ and $A_i = A_{i+1} + \{v_i \rightsquigarrow v \mid v \in X_i\}$. Otherwise, $P_i = P_{i+1}$ and $A_i = A_{i+1}$ (neither the peer set nor the arborescence is changed). The algorithm starts with $A_{|V|+1} = \emptyset$ and $P_{|V|+1} = \emptyset$, and terminates once P_1 and A_1 are computed; A_1 is returned. The time complexity depends on how fast X_i is computed, and the following two theorems show that this can be done efficiently.

Theorem 1 Let $Y_i = \{v \mid v_i \preceq v \text{ and } v \in P_i\}$ be the subset of P reachable from v_i immediately after v_i is visited. Then, $X_i = (\cup_{v_j \in C_i} Y_j) \cap P_{i+1}$. □

Theorem 2 $|Y_i| \leq 1$ for $1 \leq i \leq |V|$. □

These theorems lead to a very efficient scheme to determine X_i. First, Theorem 2 indicates that Y_i has either zero or one element. Therefore, we can use constant per-node memory to store the set Y_i at the end of visiting v_i. Second, Theorem 1 implies that X_i can be computed by first taking the union of Y_j for each child v_j of v_i, and then intersecting with P_{i+1}. We can perform the union and intersection operations in time linear in the number of children[1], and so the complexity of one iteration of computation is $O(|C_i|)$. The overall time complexity is $\sum_{i=1}^{i=|V|} O(|C_i|) = O(|E|)$, or $O(|E| + |V| \log |V|)$ including Dijkstra's algorithm, which is significantly better than the $O(|N||E| + |V||N|^2 \log |V|)$ complexity of the PFA algorithm [1]. Our algorithm, called RSA/G, is summarized in Table 1. Note that Table 1 also describes a more general version of RSA/G, called RSA/G/ext, which allows some Steiner nodes to be marked as permanently *deleted* (discussed in the next section). The algorithm will not perform any Steiner merger at such locations. In RSA/G, we simply set deleted[i] = false for all i.

[1]This is achieved by properly indexing the sinks and Steiner nodes. Note that this is possible despite the fact that there can be many more nodes in the peer set than $|C_i|$.

```
Function RSA/G/aux($i$,$P$)
Globals (set by RSA/G/ext() or RSA/G()): $G, N, Y,$ deleted
  $X \leftarrow (\cup_{v_j \in C_i} Y_j) \cap P$;
  if $i = 1$ then
      return $\{v_i \rightsquigarrow v \mid v \in X\}$;
  else if $v_i \in N$ or ($|X| > 1$ and deleted[i] = false) then
      $Y_i \leftarrow \{v_i\}$;
      return $\{v_i \rightsquigarrow v \mid v \in X\}$ +
             RSA/G/aux($i - 1, P - X + \{v_i\}$);
  else
      $Y_i \leftarrow X$;
      return RSA/G/aux($i - 1, P$);
```

```
Function RSA/G/ext($G$,$N$,deleted)
Given an SPDAG $G = (V,E)$ with ranked nodes, a set of
terminals $N \in V$, and the array marking permanently deleted
nodes, return the arborescence according to the RSA/G algo-
rithm with deleted array.
  return RSA/G/aux($|V|, \emptyset$);
```

```
Function RSA/G($G$,$N$)
Given an SPDAG $G = (V,E)$ with ranked nodes and a set
of terminals $N \in V$, return the arborescence according to the
RSA/G algorithm.
  foreach $1 \leq i \leq |V|$ do deleted[$i$] $\leftarrow$ false;
  return RSA/G/aux($|V|, \emptyset$);
```

Table 1. The RSA/G/ext and RSA/G Algorithms.

4. THE K-IDEA/G ALGORITHM

Recently, Leung and Cong presented an exponential-time branch-and-bound algorithm called RSA/BnB that solves the MRSA problem optimally [18]. They observed that the RSA heuristic is suboptimal precisely because Steiner mergers are greedy and (sometimes) suboptimal. To obtain an optimal solution, they proposed a variant of RSA, called RSA/BnB, which enumerates all sequence of choices between *merging* and *skipping* at each SMO (when $v_i \notin N$ and $|X_i| > 1$). RSA/BnB is optimal since either merging or skipping at an SMO is optimal, and merging at a TMO is *always* optimal [18]. The RSA/BnB algorithm was subsequently generalized to the RSA/BnB/G algorithm for the MSPSA problem [4]. A heuristic variant of RSA/BnB, called the k-IDeA algorithm, was presented in [19]. In k-IDeA, up to k SMOs are skipped along any path in the branch-and-bound diagram. The best set of skipped nodes are then marked as permanently *deleted* and the algorithm is repeated until there is no further improvement. In practice, even 1-IDeA or 2-IDeA is essentially optimal [19].

Unfortunately, the k-IDeA algorithm cannot be trivially generalized as in the RSA/G generalization. This is because Theorem 2 no longer holds: after an SMO is skipped at v_i (meaning there exist $v, v' \in P_{i+1}$ with $v_i \prec v$ and $v_i \prec v'$, but they are not merged at v_i), when some parent (say v_j) of v_i is visited, both v, v' can be in the peer set. In such event, $|Y_j| > 1$.

Given two nodes $v, v' \in V$, $v'' \in V$ is called a *merging point* (MP) of v and v' if and only if $v'' \preceq v$ and $v'' \preceq v'$. Furthermore, v'' is called a *maximal merging point* (MMP) of v and v' if and only if v'' is an MP of v and v', and no descendant of v'' is an MP of v and v'. Note that any pair of nodes in G has at least one MP. In [4], Leung and Cong showed the following.

Theorem 3 [4] In an optimal arborescence A_{opt}, every Steiner node v (of degree three or more) in A_{opt} is an MMP of every pair of children of v in A_{opt} (ignoring all degree-two Steiner nodes). □

```
Function IDeA/G/aux($i$,$P$,$k$)
Globals (set by IDeA/G()): $G, N, Z,$ deleted
  $X \leftarrow (\cup_{v_j \in C_i} Z_j) \cap P$;
  if $i = 1$ then
      return $(\emptyset, \sum_{v \in X} |v_i \rightsquigarrow v|)$;
  else if $v_i \in N$ then
      $Z_i \leftarrow \{v_i\}$;
      $(D, C) \leftarrow$ IDeA/G/aux($i - 1, P - X + \{v_i\}, k$);
      return $(D, C + \sum_{v \in X} |v_i \rightsquigarrow v|)$;
  else if $|X| > 1$ then
      if deleted[$i$] = true then
          $Z_i \leftarrow \emptyset$;
          return IDeA/G/aux($i - 1, P, k$);
      else if $k > 0$ then
          $Z_i \leftarrow \{v_i\}$;
          $(D_m, C_m) \leftarrow$ IDeA/G/aux($i - 1, P - X + \{v_i\}, k$);
          $C_m \leftarrow C_m + \sum_{v \in X} |v_i \rightsquigarrow v|$;
          $Z_i \leftarrow \emptyset$;
          $(D_s, C_s) \leftarrow$ IDeA/G/aux($i - 1, P, k - 1$);
          $D_s \leftarrow D_s \cup \{v_i\}$;
          return $|C_m| < |C_s|$ ? $(D_m, C_m) : (D_s, C_s)$;
      else
          $Z_i \leftarrow \{v_i\}$;
          $(D, C) \leftarrow$ IDeA/G/aux($i - 1, P - X + \{v_i\}, k$);
          return $(D, C + \sum_{v \in X} |v_i \rightsquigarrow v|)$;
  else
      $Z_i \leftarrow X$;
      return IDeA/G/aux($i - 1, P, k$);
```

```
Function IDeA/G($G$,$N$,$k$)
Given an SPDAG $G = (V,E)$ with ranked nodes and a set of
terminals $N \in V$, and the maximum number of nodes $k$ to be
deleted, return the arborescence according to the $k$-IDeA/G
algorithm.
  $C_{best} \leftarrow \infty$;
  foreach $1 \leq i \leq |V|$ do deleted[$i$] $\leftarrow$ false;
  while true do
      $(D, C) \leftarrow$ IDeA/G/aux($|V|, \emptyset, k$);
      if $C < C_{best}$ then
          $C_{best} \leftarrow C$;
          foreach $v_i \in D$ do deleted[$i$] $\leftarrow$ true;
      else
          return RSA/G/ext($P$,$N$,deleted);
```

Table 2. The k-IDeA/G Algorithm.

A generalization of Theorem 3 states that if a Steiner node v_i (of degree three or more) in a (possibly suboptimal) arborescence A is not an MMP of some pair of children of v_i in A, say v and v' (ignoring all degree-two Steiner nodes), then there exists a node v_j such that $v_i \prec v_j$ and $v_j \preceq v$ and $v_j \preceq v'$. The new arborescence $A' = (A - \{v_i \rightsquigarrow v, v_i \rightsquigarrow v'\}) + \{v_i \rightsquigarrow v_j, v_j \rightsquigarrow v, v_j \rightsquigarrow v'\}$, with $|A'| = |A| - |v_i \rightsquigarrow v_j| < |A|$, yields a reduction in total tree length. Therefore, it suffices to consider Steiner mergers at MMPs only. Let Z_i be the subset of Y_i such that v_i is an MP of some pair of nodes $v, v' \in Z_i$ if and only if v_i is also an MMP of v and v'. Then, we have the following theorem:

Theorem 4 [4] $|Z_i| \leq 1$ for $1 \leq i \leq |V|$. □

In other words, we can simply use Z_i instead of Y_i in the algorithm, and the computation of X_i will still take $O(|C_i|)$ time. Our algorithm, called k-IDeA/G, is described in Table 2. At the end of each iteration, the set of $\leq k$ skipped nodes resulting in the lowest tree length is marked as permanently *deleted* (a deleted node remains deleted throughout, and the $\leq k$ SMOs skipped in the current iteration

```
Function IA/G/0($i$,$P$)
Globals (set by IA/G()): $G, N, d_{min}$
  $C \leftarrow 0$;
  foreach $v \in P$ do $C \leftarrow C + d_{min}(v)$;
  foreach $v_j \in N, 1 < j \leq i$ do $C \leftarrow C + d_{min}(v_j)$;
  return $(\emptyset, C)$;
```

```
Function IA/G/aux($i$,$P$,$k$)
Globals (set by IA/G()): $G, N, R$
  $X \leftarrow R_i \cap P$;
  if $k = 0$ then
      return IA/G/0($i$,$P$);
  else if $i = 1$ then
      return $(\emptyset, \sum_{v \in X} |v_i \rightsquigarrow v|)$;
  else if $v_i \in N$ then
      $(S, C) \leftarrow$ IA/G/aux($i-1, P - X + \{v_i\}, k$);
      return $(S, C + \sum_{v \in X} |v_i \rightsquigarrow v|)$;
  else if $|k > 0|$ then
      $(S_m, C_m) \leftarrow$ IA/G/aux($i-1, P - X + \{v_i\}, k-1$);
      $(S_m, C_m) \leftarrow (S_m \cup \{v_i\}, C_m + \sum_{v \in X} |v_i \rightsquigarrow v|)$;
      $(S_s, C_s) \leftarrow$ IA/G/aux($i-1, P, k$);
      return $|C_m| < |C_s|$ ? $(S_m, C_m) : (S_s, C_s)$;
  else
      return IA/G/aux($i-1, P, k$);
```

```
Function IA/G($G$,$N$,$k$)
Given an SPDAG $G = (V, E)$ with ranked nodes, a set of
terminals $N \in V$, a root $r \in N$, and the maximum number
of Steiner merger $k$, return the arborescence according to the
$k$-IA/G algorithm (with the side effect that $N$ also contains
the Steiner nodes at the end).
  compute $R, d_{min}$;
  $C_{\text{best}} \leftarrow \infty$;
  while true do
      $(S, C) \leftarrow$ IA/G/aux($|V|, \{v_{|N|}\}, k$);
      if $C < C_{\text{best}}$ then
          $C_{\text{best}} \leftarrow C$;
          foreach $v \in S$ do $N \leftarrow N \cup \{v\}$;
          update $R, d_{min}$;
      else
          return RSA/G($G$,$N$);
```

Table 3. The k-IA/G Algorithm.

do not include previously deleted nodes). The process is repeated until no further improvement is obtained. Finally, RSA/G/ext() is called (with the set of permanently deleted nodes) to return the arborescence.

We showed in [4] that the function IDeA/G/aux() (one iteration of the k-IDeA algorithm) has $O(|E||N|^k)$ runtime complexity, where k is the number of allowed deletions. In practice, k-IDeA/G almost always terminates after only a few iterations, hence its practical time complexity is also $O(|E||N|^k)$.

5. THE K-IA/G ALGORITHM

The IDOM heuristic of Alexander and Robins [1] iteratively finds a node $v \in V - N$ maximizing $|\text{MSpA}(G, N, r)| - |\text{MSpA}(G, N \cup \{v\}, r)|$, where $|\text{MSpA}(G, N, r)|$ is the length of the minimum spanning arborescence of N in G rooted at r. N is then replaced by $N \cup \{v\}$ in the next iteration, until no further improvement is possible. The algorithm is a straightforward adaptation of the Iterated 1-Steiner approach [14] to the graph arborescence problem, and the runtime complexity is $O(|N||E| + |V||N|^3)$.

We now propose a heuristic inspired by the above approach, with a strategy similar to k-IDeA/G. Recall that k-IDeA/G can be viewed as a restricted version of the RSA/BnB/G algorithm in which at most k Steiner mergers are *skipped.* Our proposed algorithm, called k-IA/G, is a symmetrical restricted version of RSA/BnB/G in which at most k Steiner mergers are *allowed.* More precisely, k-IA/G also visits the nodes in decreasing rank order. If the current node v is a TMO, the terminal merger is always performed. If v is an SMO, both merging and skipping are tried unless k Steiner mergers have already been performed along the path, in which case the SMO is skipped.

Like the RSA/G and the k-IDeA/G algorithms, k-IA/G first computes X_i when node v_i is visited, then takes appropriate action based on X_i. With RSA/G and k-IDeA/G, the bounds on $|Y_i|$ and $|Z_i|$ respectively shown in Theorems 2 and 4 allow X_i to be computed efficiently in $O(|C_i|)$ time using $O(|V|)$ space. Unfortunately, for k-IA/G no similar theorem applies. Instead, for each $v_i \in V$, let us use R_i to denote the subset of sinks in N *interested in* merging into v_i if v_i is a Steiner node. Given a node $v \in N$, let $d_{min}(v) = \min_{v' \in N, v' \prec v} |v' \rightsquigarrow v|$. Then, $R_i = \{v \mid v \in N, v_i \prec v, \text{ and } |v_i \rightsquigarrow v| < d_{min}(v)\}$. In other words, a terminal v is interested in merging into a "downstream" potential Steiner node v_i (such that $v_i \prec v$) if v_i is closer to v than any of v's potential parents in N. The d_{min}'s and R_i's can be computed and maintained in $O(|E||N|)$ time and $O(|V||N|)$ space; a given X_i can then be computed by taking the intersection of R_i and P_{i+1}, which requires $O(|P|)$ time.

The complete k-IA/G algorithm is described in Table 3. k-IA/G calls the function IA/G/aux() to find the best (maximum reduction in tree length) set of $\leq k$ Steiner nodes, and adds them to the terminal set N. When $k = 0$, the function IA/G/0() is called instead of IA/G/aux(); this function simply computes the sum of the distances between each remaining terminal or Steiner node and the closest "downstream" terminal. Note that IA/G/0(i,P) can be implemented to run in $O(d)$ time (we consider all the remaining sinks to be *deleted* after returning from IA/G/0()). IA/G() calls IA/G/aux() repeatedly until no further reduction in tree length is obtained. Finally, RSA/G() is called to return the arborescence; at this point, N includes all the Steiner nodes.

We showed in [4] that the IA/G/aux() function (one iteration of the k-IA/G algorithm) has $O(|V|^k|N|)$ runtime complexity, where k is the number of allowed Steiner mergers. As a result, the overall complexity of the k-IA/G algorithm is $O(|E||N| + i|V|^k|N|)$, where i is the number of iterations. The extra $O(|E||N|)$ complexity is due to the one-time computation of d_{min} and R and the subsequent updates. Since $i = O(|N|)$ in the worst case (the average case is also $O(|N|)$ as shown in the next section), we have an overall runtime complexity of $O(|E||N| + |V|^k|N|^2)$. This compares favorably with the IDOM algorithm of Alexander and Robins, which has a complexity of $O(|E||V| + |V||N|^3)$.

6. EXPERIMENTAL RESULTS

6.1. Comparison I

We implemented the RSA/G, k-IDeA/G, and k-IA/G algorithms using GNU C++ in the SUN Unix environment, and compared against the PFA and IDOM algorithms of Alexander and Robins [1]. All experiments were performed on a SPARC-5 and all CPU times are for this machine. We performed experiments in the style of [1], whose goal was to compare the runtime and solution quality of different Steiner and arborescence algorithms on a typical FPGA routing instance with various levels of congestions. Rout-

# sinks	IKMB	IZEL	PFA	IDOM	RSA/G	1-IA/G	1-IDeA/G	RSA/BnB/G
No congestion (no pre-routed nets)								
6	(11.38) 0.00	(12.07) -0.05	1.75	1.75	1.75	1.75	1.75	1.75
8	(25.10) 0.00	(24.69) -0.19	4.21	4.30	4.16	4.49	4.00	3.91
10	(31.00) 0.00	(30.36) -0.49	5.76	5.61	5.71	5.61	5.37	5.37
12	(27.79) 0.00	—	7.05	6.85	6.84	6.89	6.30	6.30
Low congestion (10 pre-routed nets)								
6	(26.11) 0.00	(21.95) -0.42	14.72	14.57	14.57	14.57	14.57	14.57
8	(35.12) 0.00	(28.20) -0.56	20.17	20.10	20.09	20.05	19.91	19.91
10	(32.34) 0.00	—	21.92	21.89	21.92	21.89	21.85	21.85
12	(37.68) 0.00	—	23.67	23.63	23.77	23.67	23.57	23.53
Medium congestion (20 pre-routed nets)								
6	(29.17) 0.00	(28.10) -0.20	25.27	25.27	25.52	25.27	25.27	25.27
8	(33.99) 0.00	(29.99) -0.32	26.78	26.94	26.78	26.94	26.74	26.74
10	(52.07) 0.00	—	28.72	28.72	28.72	28.72	28.72	28.72
12	(52.23) 0.00	—	35.57	35.46	35.54	35.48	35.43	35.43

Table 4. Average tree length (as % above that of IKMB) for three different congestion levels. The numbers in parentheses (IKMB and IZEL only) are the average maximum source-to-sink pathlengths (as % above the optimal). The data for IZEL is incomplete due to runtime exceeding the maximum allowance of 100 seconds per net.

ing was done on a 20×20 grid graph, wherein edge weights model the congestion induced by previously routed nets. Three different levels of congestion were modeled: (a) no congestion (no pre-routed nets), (b) low congestion (10 pre-routed nets), and (c) medium congestion (20 pre-routed nets); see [1] for more details. For each net size (6, 8, 10, and 12), 50 random nets were generated and routed on the weighted graph that modeled the given congestion (congestions were newly generated for each net). We compared the IKMB and IZEL Steiner algorithms (the two best-performing graph Steiner algorithms in the literature) and the PFA and IDOM arborescence algorithms from [1], the optimal RSA/BnB/G algorithm from [4], and our algorithms RSA/G, 1-IDeA/G, and 1-IA/G. For each net, we normalized the tree length produced by each heuristic to that of IKMB, and the maximum source-to-sink pathlength of each heuristic was normalized to optimal. Table 4 gives the average tree length (as % above that of IKMB) and the average maximum source-to-sink pathlength (as % above optimal). When the congestion level is low, arborescences and Steiner trees have very similar total tree length. However, as the congestion level increases, arborescences tend to have longer tree length but shorter maximum source-to-pathlength when compared to Steiner topologies. All of the six arborescence algorithms we tested (PFA, IDOM, RSA/BnB/G, RSA/G, 1-IDeA/G, and 1-IA/G) gave similar routing quality. In fact, the comparison with the optimal solutions generated by RSA/BnB/G (which is exponential-time but fast enough to generate data for this experiment) shows that all of the heuristics are near-optimal for this application.

6.2. Comparison II

We further compare the several arborescence algorithms on larger examples. Specifically, we used the same 20×20 grid graph setup as in the previous comparison, with the congestion level set to low (10 pre-routed nets). For each net size from 3 to 34, 50 random nets were generated and routed using each arborescence heuristic (PFA, IDOM, RSA/G, 1-IDeA/G, 2-IDeA/G, and 1-IA/G), with the optimal RSA/BnB/G algorithm used for comparison.

Figures 1(a) and 1(b) show the average and maximum deviation of tree length from the optimal (RSA/BnB/G) solution for each algorithm, and Figure 1(c) shows the percentage of trials when each algorithm is optimal. As before, all arborescence algorithms tested are very close to optimal. However, 1-IDeA/G and 2-IDeA/G clearly stand out according to all three measures – average deviation, maximum deviation, and % optimality. Note that PFA and RSA/G gave slightly different results since they have different tie-breaking schemes (similarly for IDOM and 1-IA/G). Figure 1(d) shows the runtimes in CPU seconds; we see that RSA/G, 1-IDeA/G, and 1-IA/G are significantly faster than PFA and IDOM while giving the same or better solution quality. This is also depicted in Figure 1(e) and Figure 1(f), which show the average speedup of RSA/G, 1-IDeA/G, 2-IDeA/G, and IA/G over PFA and IDOM, respectively. Our algorithms are orders of magnitude faster than PFA and IDOM even on instances of modest size (for example, 1-IDeA/G averages 46X and 217X faster than PFA and 691X and 2830X faster than IDOM, on 15-sink and 30-sink instances, respectively). Thus, substantial runtime improvement over existing PFA- and IDOM-based FPGA routing algorithms is expected with our new heuristics.

6.3. Comparison III

We also "stress tested" our algorithms by running them on a grid that is four times larger (40×40), with a medium congestion level (20 pre-routed nets). The size of the nets tested ranges from 40 to 150, and for each net size, 50 random nets were generated and routed by PFA (only up to 34 nets), RSA/G, 1-IA/G, 1-IDeA/G, and 2-IDeA/G. This comparison highlights runtime and solution quality when the problem size is large.

For a given routing instance, an algorithm is called a *winner* if it generates a solution with the lowest tree length among all algorithms tested. Figure 2(a) shows the percentage of trials when each algorithm is a winner; 1-IDeA/G and 2-IDeA/G are consistently as good as or better than the other algorithms. Runtimes are shown in Figure 2(b). Our algorithms are extremely fast when compared to IDOM and PFA; average CPU times for both 1-IDeA/G and 1-IA/G were less than one second, and for 2-IDeA/G were less than four seconds, even for the largest test cases.

We also observe that 1-IDeA/G is superior to 1-IA/G in *both* quality and runtime. Figure 3 shows that on average 1-IDeA/G requires significantly fewer iterations than 1-IA/G; 1-IDeA/G finished in six iterations or less (practically con-

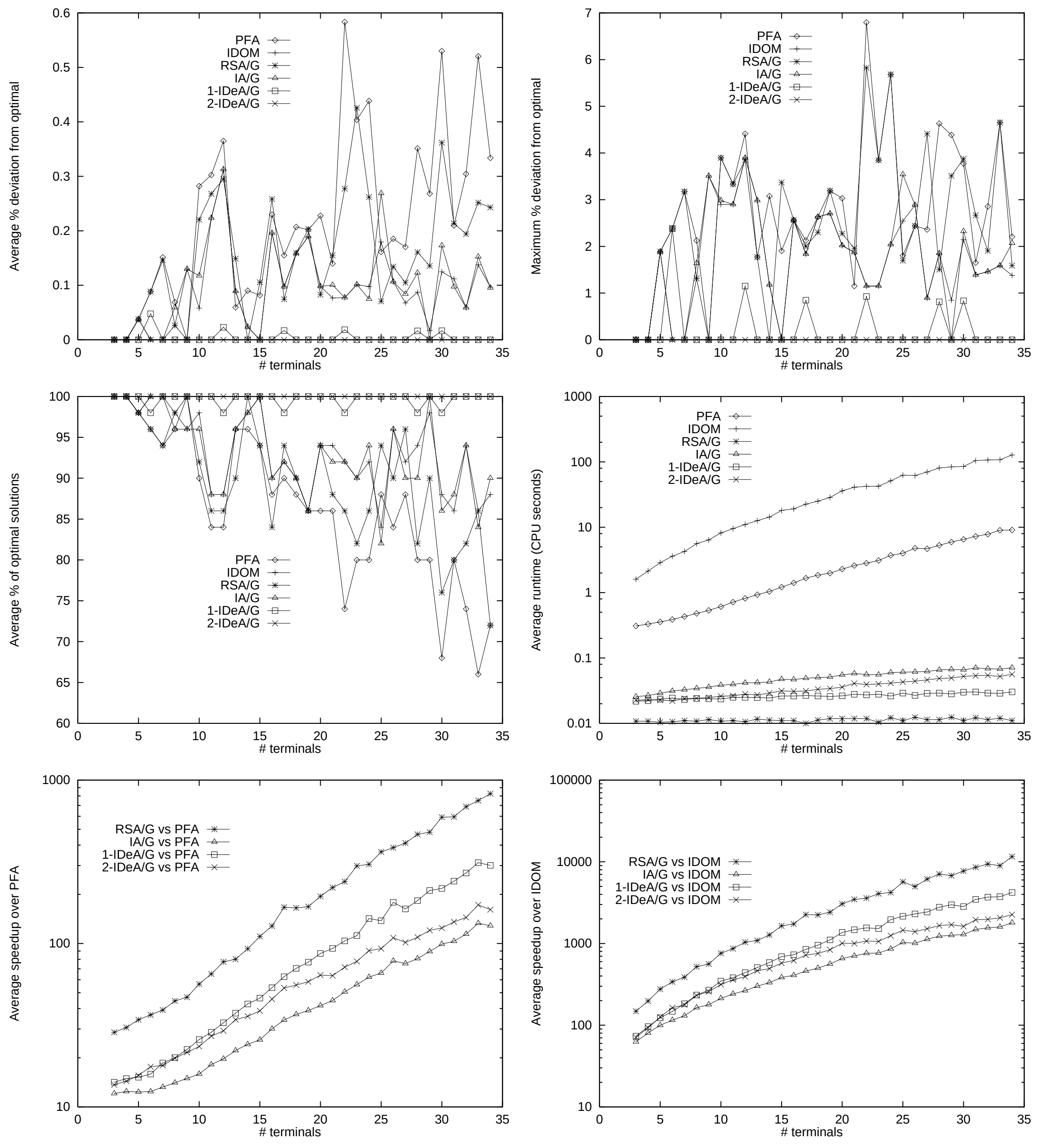


Figure 1. Comparison II – (a) The average % deviation of the solution from optimal. (b) The maximum % deviation. (c) The % of trials when the routing solution is optimal. (d) The runtime in CPU seconds. (e) The speedup over PFA. (f) The speedup over IDOM.

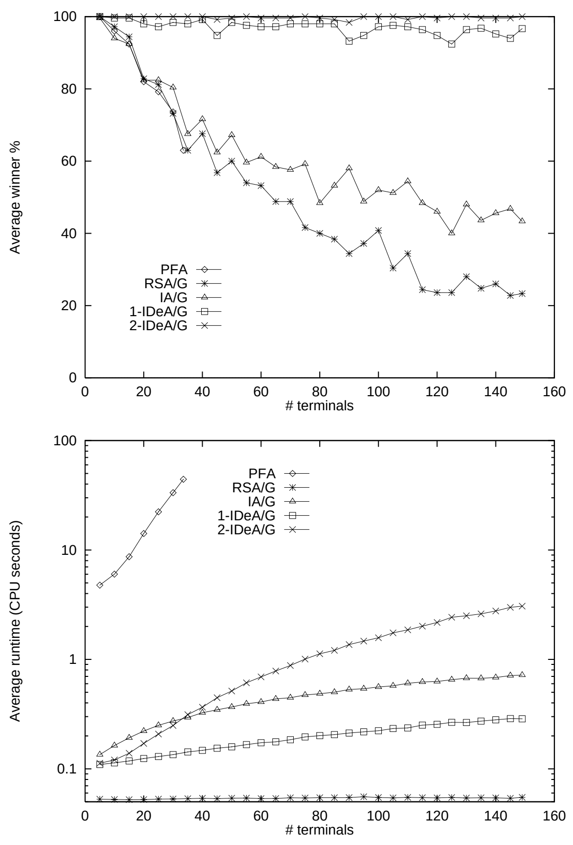


Figure 2. Comparison III – (a) The % of trials when the routing solution is optimal. (b) The average runtime in CPU seconds.

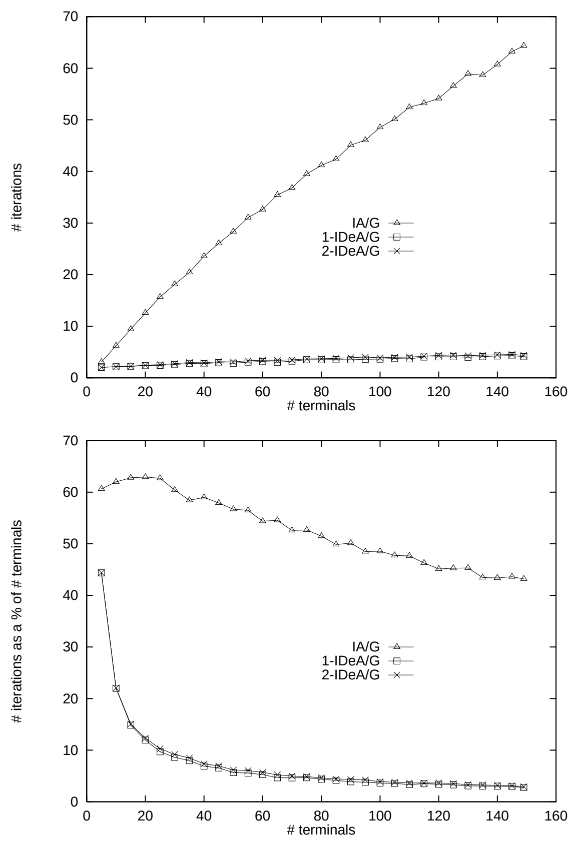


Figure 3. Comparison III – (a) The average number of iterations for 1-IDeA/G, 2-IDeA/G, and 1-IA/G. (b) The average as % of the number of terminals.

stant) on all our test cases, while 1-IA/G requires a nearly linear number of iterations. This is not surprising since the number of iterations of 1-IA/G is one plus the number of (degree three or higher) Steiner nodes in the arborescence, which is a linear or near-linear function of $|N|$. Hence, the effective runtime complexities of 1-IDeA/G and 1-IA/G are $O(|E||N|)$ and $O(|E||N| + |V||N|^2)$, respectively.

From these experiments we conclude that RSA/G and 1-IDeA/G are the two best arborescence algorithms to use in terms of runtime and solution quality.

6.4. Comparison IV

Finally, we also studied graph-based routing in a regime that models the presence of obstacles. In a layout region of size 4000×4000, we randomly generate a set of n terminals N, and a set of $2n$ rectangles R (length and width are both within $[400, 600]$). We then construct the Hanan grid graph $G_{N,R}$ induced by the points and the corners of the rectangles, then construct a new graph G by deleting any edges of $G_{H,R}$ that lie within rectangles in R. For each n, $3 \leq n \leq 10$, 10 random examples with all terminals reachable from each other were generated and routed using IKMB, RSA/G, 1-IDeA/G, and 1-IA/G. Figure 4 shows the average tree length as a percentage above that of IKMB, along with the runtime (in CPU seconds) for each of the four algorithms. The arborescences are on average 6% to 17% longer than the Steiner trees constructed by IKMB, but have much smaller maximum source-to-sink pathlengths; runtimes are orders of magnitude smaller. Finally, Figure 5 shows a 30-terminal, 30-rectangle example and the solution generated by 1-IDeA/G in 0.49 CPU seconds.

7. CONCLUSION

We have presented several efficient heuristics for the MSPSA problem, improving upon previous work in both runtime and solution quality. We have also presented detailed complexity analyses as well as extensive experimental results that suggest our algorithms will be more effective in practice than other arborescence algorithms. We believe that applications to performance-driven global routing, FPGA routing and non-VLSI domains such as multicast routing are all promising.

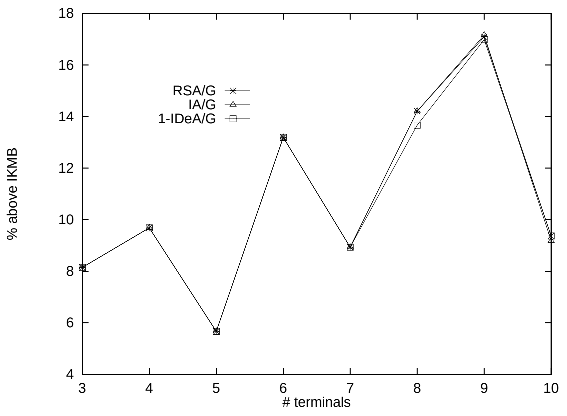
ACKNOWLEDGMENTS

The authors are grateful to Professors Michael J. Alexander and Gabriel Robins for providing the IKMB, IZEL, PFA, and IDOM source codes. Jason Cong was partially supported by NSF Young Investigator Award MIP-9357582.

REFERENCES

[1] M. J. Alexander and G. Robins, "New Performance-Driven FPGA Routing Algorithms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12 (1996), pp. 1505 – 1517.

[2] H. A. Choi and A. H. Esfahanian, "A Message-Routing Strategy for Multicomputer Systems", *Networks*, 22 (1992), 627-646.



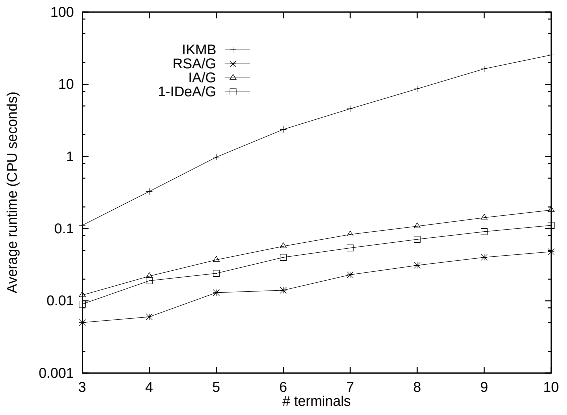


Figure 4. Comparison IV – (a) Average tree length (as % above that of IKMB). (b) Average runtime in CPU seconds.

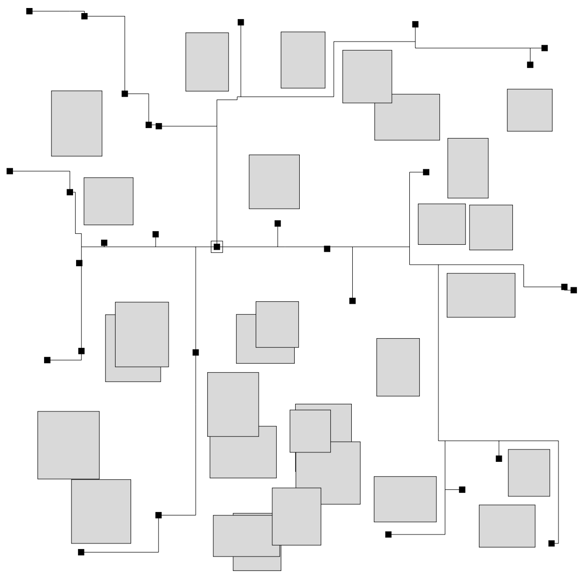


Figure 5. An 30-terminal 30-rectangle example and the 1-IDeA/G solution.

[3] H. A. Choi, A. H. Esfahanian, and B. C. Houck, "Optimal Communication Trees with Application to Hypercube Multicomputers", *Proc. Sixth Int'l Conf. on the Theory and Application of Graph Theory*, 1988, pp. 245 – 264.

[4] J. Cong, A. B. Kahng, and K. S. Leung, "Efficient Algorithms for the Minimum Shortest Path Steiner Arborescence Problem with Applications to VLSI Physical Design", *Manuscript*, 1997. (http://ballade.cs.ucla.edu/~ksleung/manuscript/ispd97.ps)

[5] J. Cong and K. S. Leung, "On the Construction of Optimal or Near-Optimal Steiner Arborescence", *UCLA Computer Science Tech. Report CSD-960033*, 1996.

[6] J. Cong, K. S. Leung, and D. Zhou, "Performance Driven Interconnect Design Based on Distributed RC Delay Model", *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 606 – 611.

[7] J. Cong and P. H. Madden, "Performance Driven Routing with Multiple Sources", *Proc. Int'l Symp. on Circuits and Systems*, 1995, pp. 1157 – 1169.

[8] J. Córdova and Y. H. Lee, "A Heuristic Algorithm for the Rectilinear Steiner Arborescence Problem", *University of Florida CIS Department Tech. Report TR-94-025*, 1994.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[10] M. X. Goemans and Y. S. Myung, "A Catalog of Steiner Tree Formulations", *Networks*, 23 (1993), pp. 19 – 28.

[11] M. Hanan, "On Steiner's Problem with Rectilinear Distance", *SIAM Journal of Applied Mathematics*, 14 (1966), pp. 255 – 265.

[12] J. M. Ho, M. T. Ko, T. H. Ma, and T. Y. Sung, "Algorithms for Rectilinear Optimal Multicast Tree Problem", *Proc. Int'l. Symposium on Algorithms and Computation*, 1994, pp. 106 – 115.

[13] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, 1992.

[14] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11 (1992), pp. 893 – 902.

[15] A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI*, Kluwer Academic Publishers, 1995.

[16] R. R. Ladeira de Matos, "A Rectilinear Arborescence Problem", *Dissertation*, University of Alabama, 1979.

[17] Y. Lan, A. H. Esfahanian, and L. M. Ni, "Multicast in Hypercube Multiprocessors", *Journal of Parallel and Distributed Computing*, 8 (1990) 30-41.

[18] K. S. Leung and J. Cong, "Fast Optimal Algorithms for the Minimum Rectilinear Steiner Arborescence Problem", *UCLA Computer Science Tech. Report CSD-960037*, 1996 (extended abstract to appear in *Proc. IEEE Symp. on Circuits and Systems*, 1997).

[19] K. S. Leung and J. Cong, "IDeA: An Efficient Near-Optimal Algorithm for the Minimum Rectilinear Steiner Arborescence Problem", *Manuscript in Preparation.*

[20] L. Nastansky, S. M. Selkow, and N. F. Stewart, "Cost Minimal Trees in Directed Acyclic Graphs", *Zeitschrift für Operations Research*, 18 (1974), pp. 59 – 67.

[21] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The Rectilinear Steiner Arborescence Problem", *Algorithmica*, 7 (1992), pp. 277 – 288.

[22] J. P. Sheu and M. Y. Su, "A Multicast Algorithm for Hypercube Multiprocessors", *International Conference on Parallel Processing*, 3 (1992), pp. 18 – 22.

[23] G. E. Téllez and M. Sarrafzadeh, "On Rectilinear Distance-Preserving Trees" *Proc. IEEE Symp. on Circuits and Systems*, 1 (1995), pp. 163 – 166.