Efficient Algorithms 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

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 several heuristics and exact algorithms for the MSPSA problem with applications to VLSI physical design. Experiments indicate that our heuristics generate near-optimal results and achieve speedups of 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, [12, 9]). 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 [12, 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 [10] 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 [15], and more recently in the RSA/DP algorithm by Leung and Cong [17]. Nastansky *et al.* [19] 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 [4] and RSA/BnB [17] algorithms, both of which employ branch-and-bound techniques to effectively prune the search space. Finally, Ho *et al.* [11] 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.* [20] 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 [7]. In [5], Cong *et al.* presented the Atree algorithm, based on making "safe moves". Téllez and Sarrafzadeh [22] 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 [14], and Leung and Cong presented the k-IDeA algorithm whose performance is very close to optimal in practice [18].

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.* [16], and Sheu and Su [21]. The problem is NP-hard [2], and heuristics include the LEN heuristic [16], the COVER heuristic [2], and the more recent ShSu heuristic [21].

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 [5]. Alexander and Robins applied the PFA and IDOM algorithms to route timing-critical nets in FPGAs [1]. Cong and Madden [6] proposed a multi-source routing algorithm based on constructing minimum-cost minimum-diameter arborescences.

In this paper we propose three heuristics and two exact algorithms for the MSPSA problem in the following order ($\Box$ = exponential-time exact algorithm, $\triangle$ = polynomial-time heuristic):

- ($\triangle$) RSA/G (Section 3) – an efficient adaptation of the greedy RSA heuristic in [20].
- ($\Box$) RSA/BnB/G (Section 4) – an optimal exponential-time branch-and-bound variant of RSA/G (analogous to the RSA/BnB algorithm in [17]).
- ($\Box$) RSA/DP/G (Section 5) – a fast implementation of RSA/BnB/G based on dynamic programming (analogous to the RSA/DP algorithm [17]).
- ($\triangle$) k-IDeA/G (Section 6) – a "scaled-down" near-optimal version of RSA/BnB/G.
- ($\triangle$) k-IA/G (Section 7) – a natural dual of k-IDeA/G that implements the IDOM heuristic [1] efficiently.

Experiments indicate that our heuristics generate near-optimal results and achieve speedups of *orders* of magnitude over existing arborescence algorithms.

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 $\Delta(v') \Leftrightarrow \Delta(v) = w(v, v')$. 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)[1]. 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 [8]. 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

[1] Actually, any arborescence is a subgraph of $G'' = (V'', E'')$, where $v \in V'' \subseteq V'$ if and only if v is on a shortest path from r to some $t \in N$, and $(v, v') \in E'' \subseteq E'$ if and only if $v, v' \in V''$, and $(v, v') \in E'$. In other words, G'' is the union of all the shortest paths from the root to the sinks. Although G'' can be substantially smaller than G', to simplify the discussion we focus on G and G'.

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 (v' is called a child of v in the arborescence). 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 [20], 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. $P_i = P_{i+1} \Leftrightarrow \{X_i\} + \{v_i\}$ 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. An example is shown in Figure 3. The time complexity depends on how fast X_i is computed, and the following three theorems show that this can be done efficiently.

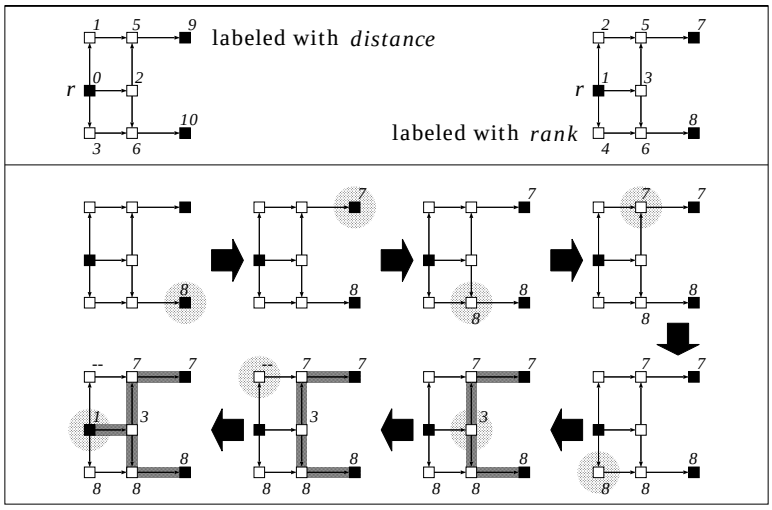

Figure 1: An example run of the RSA/G algorithm. The upper left figure shows a SPDAG with 9 nodes and 3 terminals (one of which is the root r), wherein each node is labeled with its distance from r. The upper right figure shows the ranking of the nodes (based on the distances), and RSA/G will visit the nodes in decreasing rank order. We label the node v_i with the unique element (if any) of the set Y_i after v_i is visited. The execution is: v_8, v_7 (TMO), v_6, v_5, v_4 (no action), v_3 (SMO, $v_3 \rightsquigarrow v_7$ and $v_3 \rightsquigarrow v_8$), v_2 (no action), v_1 (TMO, $v_1 \rightsquigarrow v_3$). The P_i's are as follows: $P_9 = \emptyset; P_8 = \{v_8\}; P_7 = P6 = P_5 = P_4 = \{v_7, v_8\}; P_3 = P_2 = \{v_3\}; P_1 = \{v_1\}$. The A_i's are as follows: $A_9 = A_8 = A_7 = A_6 = A_5 = A_4 = \emptyset; A_3 = A_2 = \{v_3 \rightsquigarrow v_7, v_3 \rightsquigarrow v_8\}; A_1 = \{v_1 \rightsquigarrow v_3, v_3 \rightsquigarrow v_7, v_3 \rightsquigarrow v_8\}$.

Theorem 1 Given i, j with $j \leq i$, $v_i \in P_j \Rightarrow v_i \in P_k$ for any $j \leq k \leq i$.

Proof: For any node v_i, $1 < i \leq |V|$, either (1) there does not exist any k such that $v_i \in P_k$, or (2) there exists $i' < i$ such that $v_i \in P_k$ for all $i' \leq k \leq i$, and $v_i \notin P_k$ for all $k < i'$ or $k > i$. Therefore, if there exists $j \leq i$ such that $v_i \in P_j$, it is necessary that case (2) applies, and so $v_i \in P_k$ for any k such that $i' \leq j \leq k \leq i$, where $i' \leq j$. □

Theorem 2 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, $X_i = (\cup_{v_j \in C_i} Y_j) \cap P_{i+1}$.

Proof : $v \in X_i \Leftrightarrow v_i \prec v$ and $v \in P_{i+1}$

$\Leftrightarrow \exists\, v_j \in C_i$ s.t. $v_j \preceq v$ and $v \in P_{i+1}$

$\Leftrightarrow \exists\, v_j \in C_i$ s.t. $v_j \preceq v, v \in P_j$ and $v \in P_{i+1}$ $\quad (\star)$

$$\Leftrightarrow \quad \exists\, v_j \in C_i \text{ s.t. } v \in Y_j \text{ and } v \in P_{i+1}$$

$$\Leftrightarrow \quad v \in \cup_{v_j \in C_i} Y_j \text{ and } v \in P_{i+1}$$

$$\Leftrightarrow \quad v \in (\cup_{v_j \in C_i} Y_j) \cap P_{i+1}$$

where $(\star)$ denotes the application of Theorem 1 (note that $i+1 \leq j$) in the "$\Rightarrow$" direction. □

Theorem 3 $|Y_i| \leq 1$ for $1 \leq i \leq |V|$.

Proof: We prove this inductively. The theorem holds for $i = |V|$ since $Y_{|V|} = \{v_{|V|}\}$. Assume that $|Y_{i+1}| \leq 1$ for some i; we will show that this implies $|Y_i| \leq 1$. According to the algorithm, in the case of TMO or SMO (where $v_i \in N$ or $|X_i| > 1$), we have $P_i = P_{i+1} \Leftrightarrow X_i + \{v_i\}$ and therefore $|Y_i| = |\{v_i\}| = 1$. Otherwise, $v_i \notin N$ and $|X_i| \leq 1$, and so we have $P_i = P_{i+1}$ and $|Y_i| = |Y_{i+1}| \leq 1$. □

These theorems lead to a very efficient scheme to determine X_i. First, Theorem 3 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 node v_i after visiting v_i. Second, Theorem 2 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[2], and so the time complexity of visiting v_i 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 we describe a more general version of RSA/G 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. The default version of RSA/G simply sets deleted$[i]$ = false for all i.

4 The RSA/BnB/G Algorithm

Recently, Leung and Cong presented an exponential-time branch-and-bound algorithm called RSA/BnB that solves the MRSA problem optimally [17]. They observed that the RSA heuristic is suboptimal precisely because Steiner mergers are greedy and (sometimes) suboptimal. To obtain an optimal solution, they suggested trying out *both* merging and skipping at SMOs. The algorithm, called RSA/BnB, essentially enumerates all sequence of choices between *merging* and *skipping* at each SMO, and [17] showed that this method enumerates at least one optimal arborescence.

Like RSA, RSA/BnB can also be generalized to the MSPSA problem. Like RSA/G, RSA/BnB/G visits nodes in decreasing rank order, and performs merger if the current node v is an TMO. If v is an SMO, however, RSA/BnB/G tries out *both* merging and skipping, and return the better solution.

[2] 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(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 algorithm.
	Globals: Y
	$i \leftarrow \|V\| + 1$;
	$P \leftarrow \emptyset$;
	$A \leftarrow \emptyset$;
	while $i > 1$ **do**
	$i \leftarrow i \Leftrightarrow 1$;
	$X \leftarrow (\cup_{v_j \in C_i} Y_j) \cap P$;
	if $v_i \in N$ **or** ($\|X\| > 1$ **and** deleted[i] = false) **then**
(a)	$Y_i \leftarrow \{v_i\}$;
	$P \leftarrow P \Leftrightarrow X + \{v_i\}$;
	$A \leftarrow A + \{v_i \rightsquigarrow v \mid v \in X\}$;
	else
(b)	$Y_i \leftarrow X$;
	return A;

Table 1: The RSA/G Algorithms. (a) TMO or SMO; (b) Otherwise.

Unfortunately, the generalization is not as trivial as before. This is because Theorem 3 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), $|Y_i| \geq |\{v, v'\}| = 2$.

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.

Theorem 4 Given an arborescence A, if there exists some Steiner node v with two or more children and v is not an MMP of every pair of children of v, there exists A' which has a lower cost than A.

Proof: Assume the contrary that A is optimal but a Steiner node v with two or more children is not an MMP of some children v', v''. Then there exists a node $\tilde{v}$ such that $v \prec \tilde{v}$ and $\tilde{v} \preceq v'$ and $\tilde{v} \preceq v''$. The new arborescence $A' = A \Leftrightarrow \{v \rightsquigarrow v', v \rightsquigarrow v''\} + \{v \rightsquigarrow \tilde{v}, \tilde{v} \rightsquigarrow v', \tilde{v} \rightsquigarrow v''\}$, with $|A'| = |A| \Leftrightarrow |v \rightsquigarrow \tilde{v}| < |A|$, yields a reduction in total tree length. A contradiction.

Therefore, it suffices to consider Steiner mergers at MMPs only. Let Z_i be Y_i if $|Y_i| = 1$, and $\emptyset$ otherwise, then we have the following theorem:

Theorem 5 $|Z_i| \leq 1$ for $1 \leq i \leq |V|$. □

We can simply use $X'_i = (\cup_{v_j \in C_i} Z_j) \cap P_{i+1}$ (which also takes $O(|C_i|)$ time to compute), instead of X_i in the algorithm. The RSA/BnB/G algorithm is summarized in Table 2, and its optimality is proven in Appendix I. An example is shown in Figure 4. The set of skipped nodes resulting in the lowest tree length is marked as permanently *deleted*, and finally RSA/G() is called (with the set of deleted nodes) to return the arborescence.

	Function RSA/BnB/G/aux(i, P)
	Globals (set by RSA/BnB/G()): G, N, Z, deleted
	$X \leftarrow (\cup_{v_j \in C_i} Z_j) \cap P$;
	if $i = 1$ **then**
	return $(\emptyset, \sum_{v \in X} \lvert v_i \rightsquigarrow v \rvert)$;
	else if $v_i \in N$ **then**
(a)	$Z_i \leftarrow \{v_i\}$;
	$(D, C) \leftarrow$ RSA/BnB/G/aux$(i \Leftrightarrow 1, P \Leftrightarrow X + \{v_i\})$;
	return $(D, C + \sum_{v \in X} \lvert v_i \rightsquigarrow v \rvert)$;
	else if $\lvert X \rvert > 1$ **then**
(b)	$Z_i \leftarrow \{v_i\}$;
	$(D_m, C_m) \leftarrow$ RSA/BnB/G/aux$(i \Leftrightarrow 1, P \Leftrightarrow X + \{v_i\})$;
	$C_m \leftarrow C_m + \sum_{v \in X} \lvert v_i \rightsquigarrow v \rvert$;
	$Z_i \leftarrow \emptyset$;
	$(D_s, C_s) \leftarrow$ RSA/BnB/G/aux$(i \Leftrightarrow 1, P)$;
	$D_s \leftarrow D_s \cup \{v_i\}$;
	return $\lvert C_m \rvert < \lvert C_s \rvert$? $(D_m, C_m) : (D_s, C_s)$;
	else
(c)	$Z_i \leftarrow X$;
	return RSA/BnB/G/aux$(i \Leftrightarrow 1, P)$;

Function RSA/BnB/G(G, N, k)
Given an SPDAG $G = (V, E)$ with ranked nodes and a set of terminals $N \in V$ return the arborescence according to the RSA/BnB/G algorithm.
$(D, C) \leftarrow$ RSA/BnB/G/aux$(\lvert V \rvert, \emptyset)$;
foreach $v_i \in V$ **do** deleted$[i] \leftarrow (v_i \in D)$? true : false;
return RSA/G$(P, N,$ deleted$)$;

Table 2: The RSA/BnB/G Algorithm. (a) TMO; (b) SMO; (c) Otherwise

5 The RSA/DP/G Algorithm

Leung and Cong [18] showed that each subproblem in the recursive call to the RSA/BnB algorithm can be *completely* characterized by a triple (P, K, C), which can be defined and solved recursively. As a result, hashing-based dynamic programming technique can be applied to avoid computing any given subproblem more than once. The algorithm, called RSA/DP, is significantly faster than RSA/BnB and capable of solving a 250-terminal MRSA problems optimally in one hour. The reader may refer to [18] for more details.

This result can be generalized to the MSPSA problem, as each subproblem in the recursive call to RSA/BnB/G/aux can also be completely characterized by a triple (P, v_i, C), where v_i is the node to be visited, P is the current peer set (i.e. $P = P_{i+1}$), and C is the cost of the partially-constructed arborescence so far. We also call i the *rank* of the triple. In addition to the characterization, we need to be able to (1) determine the set X_i (the subset of P reachable from v_i) given an arbitrary triple (P, v_i, C), and (2) determine whether v_i is an MMP. To solve (1), we first completely characterize the "$\prec$" relation, which requires $|V|^2$ bits and $O(|V|^2 + |V||E|)$ preprocessing time (quite reasonable for exact algorithm); Let $W_{i,j} = \{v \mid v_i \prec v \text{ and } v \in P_j\}$, then X_i is simply $W_{i,i+1}$. To solve (2), it suffices to look at each $W_{j,i+1}$ such that $v_j \in C_i$. Then, v_i is an MMP if and only if none of the $W_{j,i+1}$'s have more than one element.

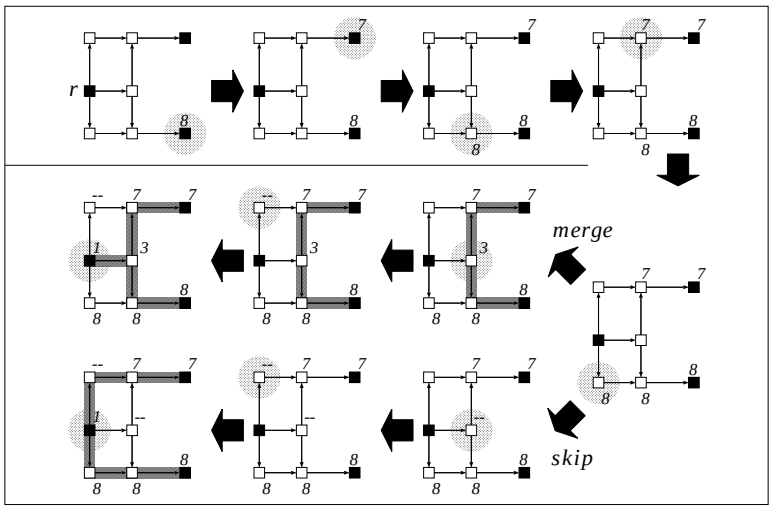

Figure 2: An example run of the RSA/BnB/G algorithm, based on the previous example. Like RSA/G, RSA/BnB/G visits the nodes in decreasing rank order. We label the node v_i with the unique element (if any) of the set Z_i after v_i is visited. Two branches are created at v_3, since it is an SMO, one corresponding to *skipping* and one to *merging*. Both branches are executed, and the branch with the lowest total weight is the optimal arborescence (the merging branch in this example). For the merging branch, the P_i's and A_i's are as follows: $P_3 = P_2 = \{v_3\}; P_1 = \{v_1\}. A_3 = A_2 = \{v_3 \rightsquigarrow v_7, v_3 \rightsquigarrow v_8\}; A_1 = \{v_1 \rightsquigarrow v_3, v_3 \rightsquigarrow v_7, v_3 \rightsquigarrow v_8\}$. For the skipping branch: $P_3 = P_2 = \{v_7, v_8\}; P_1 = \{v_1\}$. $A_3 = A_2 = \emptyset; A_1 = \{v_1 \rightsquigarrow v_7, v_1 \rightsquigarrow v_8\}$.

The algorithm, called RSA/DP/G, is summarized in Table 5. Our implementation of RSA/DP/G characterizes each subproblem with a tuple (P, v_i, C, S), where S is the set of nodes at which Steiner mergers occur[3]. RSA/DP/G visits nodes in decreasing rank order in the fashion of RSA/G. If the current node is an SMO, the branching and merging subproblems are characterized as two tuples, which are then *conditionally* inserted into the hash table (denoted "$\oplus$" in the algorithm) to be solved later. To be more precise, given a new tuple $T = (P, v_i, C, S)$, it is inserted into the hash table if either (1) a tuple of the form $T' = (P, v_i, C', S')$ already exists in the hash table, and $C' > C$ (in such case, the tuple T' will be replaced by T), or (2) no such tuple T' exists. Note that conditional insertion guarantees that at most one tuple of the form $(P, v_i, \Leftrightarrow, \Leftrightarrow)$ exists in the hash table at any given time.

If the expansion of a tuple T leads to the creation of a tuple T', it is necessary that T' has a lower rank than T since nodes are visited in decreasing rank order. By expanding tuples in decreasing rank order, we guarantee that RSA/DP/G will expand at most one tuple of the form $(P, v_i, \Leftrightarrow, \Leftrightarrow)$ for each pair (P, v_i)[4]. In other words, no subproblem is solved more than once.

[3] Strictly speaking, only P and i are needed to characterize the subproblem to be solved. C is used for pruning purpose, and S for re-generating the optimal solution once determined.

[4] This is because no other tuple of the same form exists in the hash table (1) at the time of expansion of the tuple (a property of conditional insertion), and (2) in subsequent execution (all tuples generated later will have a lower rank).

Function RSA/DP/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/DP/G algorithm.

```
    compute the "≺" relation;
    H ← {({v_|V|}, |V|, 0, ∅)};
    i ← |V|;
    while i ≠ 0 do
(⋆)     find T = (P, i, C, S) ∈ H such that i is maximized;
        H ← H ⇔{T};
        X ← ∅;
        foreach v_j ∈ C_i do
            W ← {v | v_j ≺ v and v ∈ P};
            if |W| > 1 then goto (⋆);
            X ← X + W;
        repeat
            if v_i ∈ N then
(a)             P ← P ⇔X + {v_i};
                C ← C + Σ_{v∈X} |v_i ⇝ v|;
                i ← i ⇔1;
            else if |X| > 1 then
(b)             H ← H ⊕ {(P, i ⇔1, C, S};
                H ← H ⊕ {(P ⇔X + {v_i}, i ⇔1, C + Σ_{v∈X} |v_i ⇝ v|, S ∪ {v_i})};
                goto (⋆);
            else
(c)             i ← i ⇔1;
    (P, i, C, S) ← the only element left in H;
    foreach v_i ∈ V do deleted[i] ← (v_i ∈ S) ? false : true;
    return RSA/G(P, N, deleted);
```

Table 3: The RSA/DP/G Algorithm. (a) TMO; (b) SMO; (c) Otherwise.

6 The k-IDeA/G Algorithm

Since RSA/BnB/G tries out both skipping and merging at every SMO, it requires exponential runtime in the worst case. We now describe a simple heuristic variant of RSA/BnB/G which allows up to k SMOs to be 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. The heuristic, called k-IDeA/G (which stands for *I*terated k-*De*letion for *A*rborescence) is described in Table 4. 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 do not include previously deleted nodes). The process is repeated until no further improvement is obtained. Finally, RSA/G() is called (with the set of permanently deleted nodes) to return the arborescence.

We can show that the function IDeA/G/aux() (one iteration of the k-IDeA/G algorithm) has $O(|E||N|^k)$ time complexity, where k is the number of allowed deletions (the complexity analysis is detailed in Appendix II). Experimental result (see Section 8) shows that in practice, k-IDeA/G almost always terminates after only a few iterations. Hence, the average case time complexity of k-IDeA/G is also $O(|E||N|^k)$.

```
Function IDeA/G/aux(i, P, k)
Globals (set by IDeA/G()): G, N, Z, deleted
    X ← (∪_{v_j ∈ C_i} Z_j) ∩ P;
    if i = 1 then
        return (∅, Σ_{v ∈ X} |v_i ⇝ v|);
    else if v_i ∈ N then
(a)     Z_i ← {v_i};
        (D, C) ← IDeA/G/aux(i ⇔1, P ⇔X + {v_i}, k);
        return (D, C + Σ_{v ∈ X} |v_i ⇝ v|);
    else if |X| > 1 then
        if deleted[i] = true then
(b)         Z_i ← ∅;
            return IDeA/G/aux(i ⇔1, P, k);
        else if k > 0 then
(c)         Z_i ← {v_i};
            (D_m, C_m) ← IDeA/G/aux(i ⇔1, P ⇔X + {v_i}, k);
            C_m ← C_m + Σ_{v ∈ X} |v_i ⇝ v|;
            Z_i ← ∅;
            (D_s, C_s) ← IDeA/G/aux(i ⇔1, P, k ⇔1);
            D_s ← D_s ∪ {v_i};
            return |C_m| < |C_s| ? (D_m, C_m) : (D_s, C_s);
        else
(d)         Z_i ← {v_i};
            (D, C) ← IDeA/G/aux(i ⇔1, P ⇔X + {v_i}, k);
            return (D, C + Σ_{v ∈ X} |v_i ⇝ v|);
    else
(e)     Z_i ← X;
        return IDeA/G/aux(i ⇔1, P, k);
```

```
Function IDeA/G(G, N, k)
Given an SPDAG G = (V, E) with ranked nodes, a set of terminals N ∈ V,
and the maximum number of nodes k to be deleted, return the arborescence
according to the k-IDeA/G algorithm.
    C_best ← ∞;
    foreach 1 ≤ i ≤ |V| do deleted[i] ← false;
    repeat
        (D, C) ← IDeA/G/aux(|V|, ∅, k);
        if C < C_best then
            C_best ← C;
            foreach v_i ∈ D do deleted[i] ← true;
        else
            return RSA/G(P, N, deleted);
```

Table 4: The k-IDeA/G Algorithm. (a) TMO; (b) SMO, node deleted; (c) SMO, skipping allowed; (d) SMO, skipping exhausted; (e) Otherwise.

7 The k-IA/G Algorithm

The IDOM heuristic of Alexander and Robins [1] iteratively finds a node $v \in V \Leftminus N$ maximizing $|\mathsf{MSpA}(G, N, r)| \Leftminus |\mathsf{MSpA}(G, N \cup \{v\}, r)|$, where $|\mathsf{MSpA}(G, N, r)|$ 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 [13] to the graph arborescence

problem, and the time complexity is $O(|N||E| + |V||N|^3)$.

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**
(a) $(S, C) \leftarrow$ IA/G/aux$(i \Leftrightarrow 1, P \Leftrightarrow X + \{v_i\}, k)$;
  return $(S, C + \sum_{v \in X} |v_i \rightsquigarrow v|)$;
else if $|k > 0|$ **then**
(b) $(S_m, C_m) \leftarrow$ IA/G/aux$(i \Leftrightarrow 1, P \Leftrightarrow X + \{v_i\}, k \Leftrightarrow 1)$;
  $(S_m, C_m) \leftarrow (S_m \cup \{v_i\}, C_m + \sum_{v \in X} |v_i \rightsquigarrow v|)$;
  $(S_s, C_s) \leftarrow$ IA/G/aux$(i \Leftrightarrow 1, P, k)$;
  return $|C_m| < |C_s|$? $(S_m, C_m) : (S_s, C_s)$;
else
(c,d) **return** IA/G/aux$(i \Leftrightarrow 1, P, k)$;

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};
$C_{\text{best}} \leftarrow \infty$;
repeat
  $(S, C) \leftarrow$ 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
    foreach $v_i \in V$ **do** deleted$[i] \leftarrow (v_i \in N)$? false : true;
    return RSA/G$(G, N,$ deleted$)$;

Table 5: The k-IA/G Algorithm. (a) TMO; (b) SMO, merging allowed; (c) SMO, merging exhausted; (e) Otherwise.

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, (which stands for *I*terated k-*A*rborescence) 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 heuristics, 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 3 and 5 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 5. 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 can show that the `IA/G/aux`() function (one iteration of the k-IA/G algorithm) has $O(|V|^k|N|)$ time complexity, where k is the number of allowed Steiner mergers (the complexity analysis is detailed in Appendix III). 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 time 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 $O(|E||V| + |V||N|^3)$.

8 Experimental Results

8.1 Comparison I

We implemented all algorithms and heuristics proposed in this paper 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. Routing 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 to 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 Steiner algorithm (one of the best-performing graph Steiner algorithm in the literature) with several arborescence algorithms including PFA and IDOM from [1], our optimal algorithms RSA/BnB/G and RSA/DP/G, and our heuristics 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.

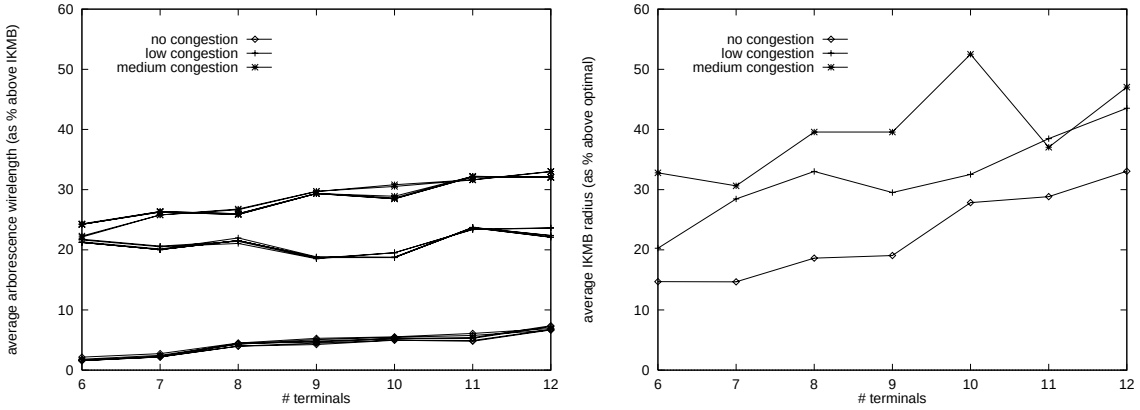

Figure 3: Comparison I – (a) The average wirelength of various arborescence algorithms (PFA, IDOM, RSA/G, 1-IDeA/G, 1-IA/G, and RSA/BnB/G) as a % above the IKMB wirelength under the conditions of (1) no congestion, (2) low congestion, and (3) medium congestion. (b) The average radius (maximum source-to-sink pathlength) of IKMB-constructed Steiner as a % above optimal.

Figure 3(a) gives the average tree length of various arborescence algorithms (as % above that of IKMB) under various congestion levels, and Figure 3(b) gives the average radius (maximum source-to-sink pathlength) of IKMB-constructed Steiner as a % above the maxiumum Manhattan source-to-sink distance. When there is little or no congestion, 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 radius when compared to Steiner topologies. All of the six arborescence heuristics we tested (PFA, IDOM, RSA/G, 1-IDeA/G, and 1-IA/G) gave similar solution quality, and this is the reason why we did not distinguish the tree lengths among different algorithms in Figure 3(a). Moreover, the comparison with the optimal solutions shows that all of the heuristics are near-optimal for this application.

8.2 Comparison II

We also "stress tested" our algorithms and heuristics 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 3 to 150, and for each net size, 50 random nets were generated and routed by PFA, IDOM, RSA/BnB/G and RSA/DP/G, 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 (note that the data for PFA, IDOM, RSA/BnB/G, and RSA/DP/G were incomplete

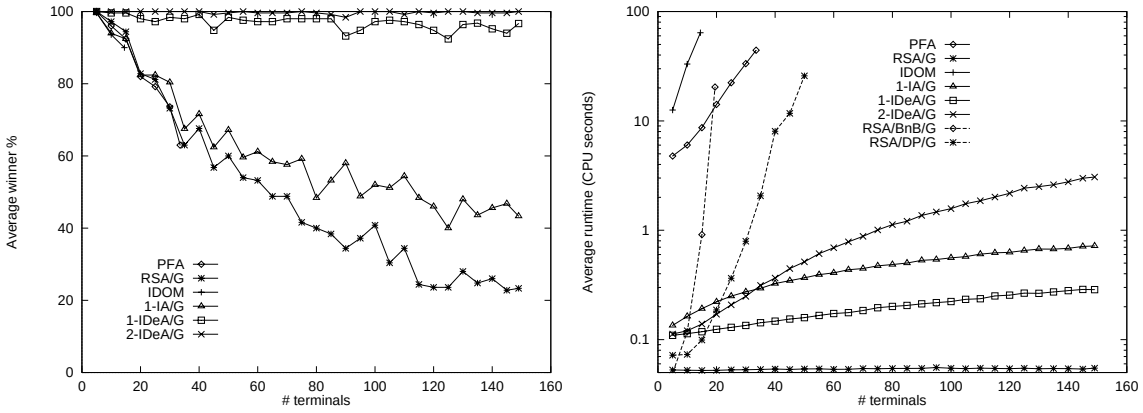

Figure 4: Comparison II – (a) The % of trials when each heuristic is a winner. (b) The average runtime in CPU seconds.

because they exceeded the runtime limit).

For a given routing instance, a heuristic is called a *winner* if it generates a solution with the lowest tree length among all heuristics tested (note that RSA/BnB/G and RSA/DP/G are not included). Figure 4(a) shows the percentage of trials when each heuristic is a winner; 1-IDeA/G and 2-IDeA/G are consistently as good as or better than the other heuristics. Runtimes are shown in Figure 4(b). 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. Moreover, our heuristics are orders of magnitude faster than PFA and IDOM even on instances of modest size (for example, on 15-sink instances 1-IDeA/G averages 53X faster than PFA and 291X faster than IDOM, respectively). Thus, substantial runtime improvement over existing PFA- and IDOM-based FPGA routing algorithms is expected with our new heuristics.

We also observe that 1-IDeA/G is superior to 1-IA/G in *both* quality and runtime. Figure 5 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 constant) 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.

8.3 Comparison III

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

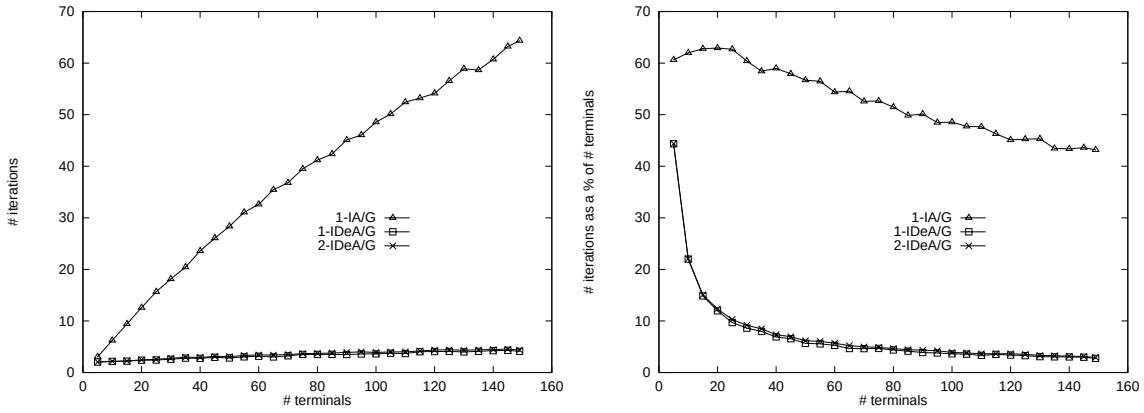

Figure 5: Comparison II – (a) The average number of iterations for 1-IDeA/G, 2-IDeA/G, and 1-IA/G. (b) The average as % of the number of terminals.

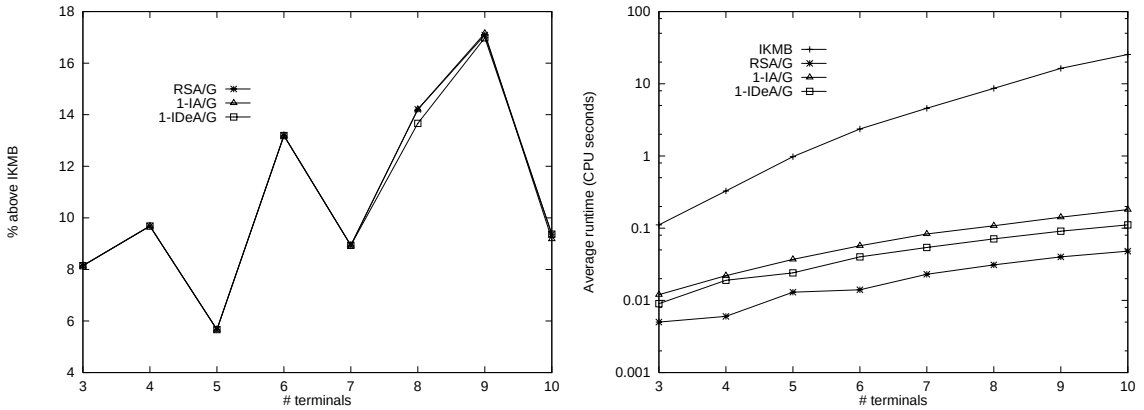

Figure 6: Comparison III – (a) Average tree length (as % above that of IKMB). (b) Average runtime in CPU seconds.

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 6 shows the average tree length as a percentage above that of IKMB, along with the runtime (in CPU seconds) for each of the four heuristics. 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 7 shows a 30-terminal, 30-rectangle example and the solution generated by 1-IDeA/G in 0.49 CPU seconds.

9 Conclusion

We have presented several efficient heuristics and exact algorithms 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 are more effective in practice than other arborescence algorithms. We believe that applications to performance-driven global routing, FPGA routing

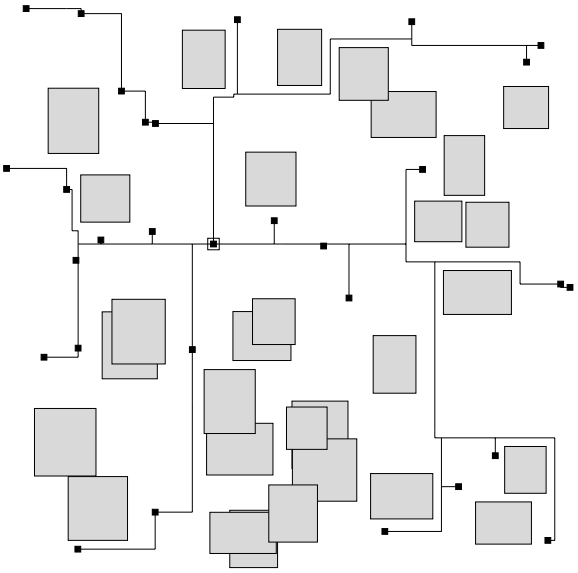

Figure 7: An 30-terminal 30-rectangle example and the 1-IDeA/G solution.

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, 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.

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

[5] 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.

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

[7] 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.

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

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

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

[11] 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.

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

[13] 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.

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

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

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

[17] 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).

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

[19] 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.

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

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

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

Appendix I – Optimality of RSA/BnB/G

We shall prove the optimality of RSA/BnB/G in the fashion of [17]. Let B denote a generic node in the branch-and-bound (BNB) diagram, and v_B, i_B, $\mathcal{A}_B$ be the node to be visited, the rank of the node, and the partial arborescence constructed immediately *after* visiting v_B, respectively (note that $v_B = v_{i_B}$ and $\mathcal{A}_B = A_{i_B-1}$). If v_B is a TMO, B_T denotes the child node of B in the BNB diagram (note that $i_{B_T} = i_B \Leftrightarrow 1$). If v_B is an SMO, B_M and B_S denote the two child nodes of B corresponding to merging and skipping respectively (note that $i_{B_S} = i_{B_T} = i_B \Leftrightarrow 1$). We say B is *optimal* if and only if there exists an optimal arborescence $\mathcal{A}^*_B$ which satisfies the following two optimality criteria:

- [OC1(B)] $\mathcal{A}_B \subseteq \mathcal{A}^*_B$, and
- [OC2(B)] $\nexists\ i$ such that $i \geq i_B$ and $v_i \rightsquigarrow v_j \in \mathcal{A}^*_B \Leftrightarrow \mathcal{A}_B$.

These two criteria guarantee that (1) the current partial arborescence is optimal, and (2) it is possible to generate an optimal arborescence even though only nodes with a lower rank than i_B will be visited in the remaining execution.

Lemma 1 If B is optimal and v_B is a TMO, B_T is optimal and A^*_B satisfies both OC1(B_T) and OC2(B_T).

Proof: First of all, we shall show that $\mathcal{A}_{B_T} = \mathcal{A}_B + \{v_{B_T} \rightsquigarrow v \mid v \in X_{i_{B_T}}\} \subseteq \mathcal{A}^*_B$. Assume the contrary that this is not the case. Then there must exist $v_i \rightsquigarrow v \in \mathcal{A}^*_B$ such that $v \in X_{i_{B_T}}$ and $i < i_{B_T}$. We can construct an arborescence $\mathcal{A}'_B = \mathcal{A}^*_B \Leftrightarrow \{v_i \rightsquigarrow v\} + \{v_B \rightsquigarrow v\}$, and we have $|\mathcal{A}'_B| = |\mathcal{A}^*_B| + \Delta(v_i) \Leftrightarrow \Delta(v_B) < |\mathcal{A}^*_B|$, contradicting the assumption that $\mathcal{A}^*_B$ is optimal. Therefore, $\mathcal{A}_{B_T} \subseteq \mathcal{A}^*_B$, and $\mathcal{A}^*_B$ satisfies OC1(B_T).

Second, we shall show that $\nexists\ v_i \rightsquigarrow v_j \in \mathcal{A}^*_B \Leftrightarrow \mathcal{A}_{B_T}$ such that $i \geq i_{B_T}$. Assume the contrary that such $v_i \rightsquigarrow v_j$ exists. Note that the optimality of $\mathcal{A}^*_B$ implies that $i < i_B$. Therefore, $i_{B_T} \leq i < i_B \Rightarrow i_{B_T} \leq i < i_{B_T} + 1 \Rightarrow i = i_{B_T} \Rightarrow v_i = v_{B_T}$. However, this is impossible since no nodes in the peer set dominates v_{B_T} after the terminal merger at v_{B_T} As a result, no such path $v_i \rightsquigarrow v_j$ exists, and $\mathcal{A}^*_B$ satisfies OC2(B_T). □

Lemma 2 If B is optimal and v_B is a Steiner merging point, either (1) B_M is optimal and $\mathcal{A}^*_B$ satisfies both OC1(B_M) and OC2(B_M), or (2) B_S is optimal and $\mathcal{A}^*_B$ satisfies both OC1(B_S) and OC2(B_S).

Proof: If $v_B \in \mathcal{A}^*_B$, then using an argument similar to the previous proof, we can prove that $\mathcal{A}^*_B$ satisfies both OC1(B_T) and OC2(B_T), and so B_T is optimal.

If $v_B \notin \mathcal{A}^*_B$, $\mathcal{A}^*_B$ satisfies OC1(B_S) since $\mathcal{A}_{B_S} = \mathcal{A}_B \subseteq \mathcal{A}^*_B$. To show that $\mathcal{A}^*_B$ satisfies OC2(B_S), we assume the contrary that $\exists\ v_i \rightsquigarrow v_j \in \mathcal{A}^*_B \Leftrightarrow \mathcal{A}_{B_S}$ such that $i \geq i_{B_S}$. Using the same argument as before, we have $i_{B_S} \leq i < i_B \Rightarrow v_i = v_{B_S}$. However, this is contradictory since $v_B \notin \mathcal{A}^*_B$. □

Theorem 6 There exists at least one leaf node in the BNB diagram in the RSA/BnB/G algorithm that is optimal in tree length.

Proof: It follows from the facts that the root node of the BNB diagram is optimal ($P_{|V|+1} = \emptyset$ and $\mathcal{A}_{|V|+1} = \emptyset$), and that each optimal node has at least one child node that is optimal. □

Appendix II – Complexity of k-IDeA/G

Consider the recursive function `IDeA/G/aux`() which does the majority of the work, and whose time complexity is a function of the triple (i, d, k), where i is the rank of the current node, d is the number of nodes removed from the peer set in the subsequent execution (recursion), and k is the number of SMOs that can be skipped in the subsequent execution. Let $f_{G,N}(i, d, k)$ denote its time complexity. The execution of `IDeA/G/aux`(), excluding the recursion, is dominated by the computation of X_i, which takes $O(|C_i|)$ time. In the following, we simply say that it takes C_i time without loss of precision. Table 6 gives the recursive definition of $f_{G,N}(i, d, k)$ according to the different possible scenarios, for $i \geq 1$, $d \geq 0$, and $k \geq 0$. The base case of the algorithm is $f_{G,N}(1, d, k) = |C_i|$. Note that all of these complexities are directly inferred from the algorithm. The following theorem bounds the complexity of $f_{G,N}(i, d, k)$.

	Description	$v_i \in N$?	deleted$[i]$?	$\|X_i\| > 1$?	$k > 0$?
(a)	TMO	true	—	—	—
(b)	SMO, NODE DELETED	false	true	—	—
(c)	SMO, SKIPPING ALLOWED	false	false	true	true
(d)	SMO, SKIPPING EXHAUSTED	false	false	true	false
(e)	OTHERWISE	false	false	false	—

	Description	$f_{G,N}(i, d, k) =$
(a)	TMO	$\|C_i\| + f_{G,N}(i \Leftrightarrow 1, d \Leftrightarrow \|X\|, k)$
(b)	SMO, NODE DELETED	$\|C_i\| + f_{G,N}(i \Leftrightarrow 1, d, k)$
(c)	SMO, SKIPPING ALLOWED	$\|C_i\| + f_{G,N}(i \Leftrightarrow 1, d \Leftrightarrow \|X\|, k) + f_{G,N}(i \Leftrightarrow 1, d, k \Leftrightarrow 1)$
(d)	SMO, SKIPPING EXHAUSTED	$\|C_i\| + f_{G,N}(i \Leftrightarrow 1, d \Leftrightarrow \|X\|, k)$
(e)	OTHERWISE	$\|C_i\| + f_{G,N}(i \Leftrightarrow 1, d, k)$

Table 6: The complexity of the IDeA/G/aux function for any $i \geq 1$, $d \geq 0$, and $k \geq 0$ ("—" denotes DON'T CARE).

Theorem 7 For any $i \geq 1$, $d \geq 0$, and $k \geq 0$, $f_{G,N}(i, d, k) \leq \sigma_i(\frac{d+2}{2})^k$, where $\sigma_i = \sum_{1 \leq j \leq i} |C_j|$.

Proof: We use induction on i. For $i = 1$, $f_{G,N}(i, d, k) = |C_i| \leq |C_i|(\frac{d+2}{2})^k$ for any $d \geq 0$, $k \geq 0$. Assume that $f_{G,N}(i, d, k) \leq \sigma_i(\frac{d+2}{2})^k$ for $i = m \Leftrightarrow 1 \geq 1$; we shall prove that $f_{G,N}(m, d, k) \leq \sigma_m(\frac{d+2}{2})^k$ holds for all of the scenarios described in Table 6. Note that it suffices to prove the result for scenarios (b) and (c), since the proofs for (a), (d), and (e) are subsumed by that for (b).

In case (b), we have

$$\begin{aligned} f_{G_T}(m, d, k) &= |C_m| + f_{G,N}(m \Leftrightarrow 1, d, k) \\ &\leq |C_m| + \sigma_{m-1}(\frac{d+2}{2})^k \leq (|C_m| + \sigma_{m-1})(\frac{d+2}{2})^k = \sigma_m(\frac{d+2}{2})^k \end{aligned}$$

In case (c), we have

$$\begin{aligned}
f_{G,N}(m,d,k) &= |C_m| + f_{G,N}(m \Leftrightarrow 1, d \Leftrightarrow |X|, k) + f_{G,N}(m \Leftrightarrow 1, d, k \Leftrightarrow 1) \\
&\leq |C_m| + \sigma_{m-1}(\frac{d \Leftrightarrow |X|+2}{2})^k + \sigma_{m-1}(\frac{d+2}{2})^{k-1} \\
&\leq |C_m| + \sigma_{m-1}(\frac{d \Leftrightarrow 2+2}{2})^k + \sigma_{m-1}(\frac{d+2}{2})^{k-1} \\
&\leq |C_m| + \sigma_{m-1}(\frac{d}{2})(\frac{d+2}{2})^{k-1} + \sigma_{m-1}(\frac{d+2}{2})^{k-1} \\
&= |C_m| + \sigma_{m-1}(\frac{d}{2}+1)(\frac{d+2}{2})^{k-1} \\
&= |C_m| + \sigma_{m-1}(\frac{d+2}{2})^k \ \leq \ (|C_m| + \sigma_{m-1})(\frac{d+2}{2})^k \ = \ \sigma_m(\frac{d+2}{2})^k
\end{aligned}$$

Therefore, the induction step holds for $i = m$ for each of the scenarios (a) to (e). □

Theorem 8 The `IDeA/G/aux()` function (one iteration of the k-IDeA/G algorithm) has complexity $O(|E||N|^k)$, where k is the number of allowed deletions.

Proof: Note that any arborescence has at most $|N| \Leftrightarrow 1$ Steiner nodes, and therefore at most $|N| + |N| \Leftrightarrow 1 \Leftrightarrow 1 = 2|N| \Leftrightarrow 2$ nodes are removed from the peer set (only r remains in P at the end). Therefore, the complexity of the function is at most $f_{G,N}(|V|, 2|N| \Leftrightarrow 2, k) \leq \sigma_{|V|}(\frac{(2|N|-2)+2}{2})^k = O(|E||N|^k)$. □

Appendix III – Complexity of k-IA/G

We use an analysis scheme similar to that presented for k-IDeA/G. The complexity of the function `IA/G/aux()` can be expressed as a function of the triple (i, d, k), where i is the rank of the current node, d is the number of nodes removed from the peer set in the subsequent execution (recursion), and k is the number of Steiner mergers allowed in the subsequent execution. In what follows, let $g_{G,N}(i, d, k)$ be the complexity of calling `IA/G/aux()`. The time complexity of executing `IA/G/aux()` at node v_i, excluding the recursive calls, is $O(|P_{i+1}|)$. However, we know that $|P_{i+1}| \leq d+1$ since all nodes in the peer set (except r, which is added to the peer set at the end) will have been deleted when the recursion terminates. As before, we simply say that `IDeA/G/aux()`, excluding the recursive calls, takes $d+1$ time. Table 7 gives the (recursive) definitions of $g_{G,N}(i, d, k)$ according to the different possible scenarios, for $i \geq 1$, $d \geq 1$, and $k \geq 0$. The base cases of the algorithm is $g_{G,N}(1, d, k) = d+1$ and $g_{G,N}(i, d, 0) = d+1$. The following theorem bounds the complexity of $g_{G,N}(i, d, k)$.

Theorem 9 For any $i \geq 1$, $d \geq 0$, and $k \geq 0$, $g_{G,N}(i, d, k) \leq i^k(2d+1)$.

Proof: We use induction on i. Note that it suffices to prove the result for scenarios (a), (b), and (c), since (d) is subsumed by (c). The theorem holds for $i = 1$ since $g_{G,N}(1, d, k) = d+1 \leq i^k(2d+1)$ for any $d \geq 0, k \geq 0$, It also holds for $k = 0$ since $g_{G,N}(i, d, 0) = d+1 \leq i^k(2d+1)$ for any $i \geq 1, d \geq 0$. Assume that $g_{G,N}(i, d, k) \leq i^k(2d+1)$ for any i, d, and k with $1 \leq i < m$ $(d \geq 0, k \geq 1)$.

	Description	$v_i \in N$?	$\lvert X_i\rvert > 1$?	$k > 0$?
(a)	TMO	true	—	—
(b)	SMO, MERGING ALLOWED	false	true	true
(c)	SMO, MERGING EXHAUSTED	false	true	false
(d)	OTHERWISE	false	false	—

	Description	$g_{G,N}(i,d,k) =$
(a)	TMO	$(d+1) + g_{G,N}(i \Leftrightarrow 1, d \Leftrightarrow \lvert X\rvert, k)$
(b)	SMO, MERGING ALLOWED	$(d+1) + g_{G,N}(i \Leftrightarrow 1, d \Leftrightarrow \lvert X\rvert, k \Leftrightarrow 1) + g_{G,N}(i \Leftrightarrow 1, d, k)$
(c)	SMO, MERGING EXHAUSTED	$(d+1) + g_{G,N}(i \Leftrightarrow 1, d, k)$
(d)	OTHERWISE	$(d+1) + g_{G,N}(i \Leftrightarrow 1, d, k)$

Table 7: Time complexity of the IDeA/G/aux function for any $i \geq 1$, $d \geq 1$, and $k \geq 0$.

In case (a) we have

$$\begin{aligned} g_{G,N}(m,d,k) &= (d+1) + g_{G,N}(m \Leftrightarrow 1, d \Leftrightarrow |X|, k) \\ &\leq (d+1) + (m \Leftrightarrow 1)^k (2(d \Leftrightarrow |X|) + 1) \\ &\leq (2d+1) + (m \Leftrightarrow 1)^k (2d+1) \;=\; (1 + (m \Leftrightarrow 1)^k)(2d+1) \;\leq\; m^k(2d+1) \end{aligned}$$

In case (b) we have, if $k > 1$,

$$\begin{aligned} g_{G,N}(m,d,k) &= (d+1) + g_{G,N}(m \Leftrightarrow 1, d \Leftrightarrow |X|, k \Leftrightarrow 1) + g_{G,N}(m \Leftrightarrow 1, d, k) \\ &\leq (d+1) + (m \Leftrightarrow 1)^{k-1}(2(d \Leftrightarrow |X|) + 1) + (m \Leftrightarrow 1)^k (2d+1) \\ &\leq (d+1) + (m \Leftrightarrow 1)^{k-1}(2d \Leftrightarrow 4 + 1) + (m \Leftrightarrow 1)^k (2d+1) \\ &\leq (m \Leftrightarrow 1)^{k-1}(d + 1 + 2d \Leftrightarrow 3 + (m \Leftrightarrow 1)(2d+1)) \\ &\leq (m \Leftrightarrow 1)^{k-1}(m+1)(2d+1) \\ &= (m \Leftrightarrow 1)^{k-2}(m \Leftrightarrow 1)(m+1)(2d+1) \\ &\leq (m \Leftrightarrow 1)^{k-2}(m^2)(2d+1) \;\leq\; m^k(2d+1) \end{aligned}$$

If $k = 1$ instead, then

$$\begin{aligned} g_{G,N}(m,d,k) &= (d+1) + g_{G,N}(m \Leftrightarrow 1, d \Leftrightarrow |X|, k \Leftrightarrow 1) + g_{G,N}(m \Leftrightarrow 1, d, k) \\ &\leq (d+1) + (d \Leftrightarrow |X|) + 1 + (m \Leftrightarrow 1)(2d+1) \\ &\leq (d+1) + (d \Leftrightarrow 2) + 1 + (m \Leftrightarrow 1)(2d+1) \\ &= 2md + m \Leftrightarrow 1 \;<\; 2md + m \;=\; m(2d+1) \;=\; m^k(2d+1) \end{aligned}$$

In case (c) we have

$$\begin{aligned} g_{G,N}(m,d,k) &= (d+1) + g_{G,N}(m \Leftrightarrow 1, d, k) \\ &\leq (2d+1) + (m \Leftrightarrow 1)^k (2d+1) \;\leq\; (1 + (m \Leftrightarrow 1)^k)(2d+1) \;\leq\; m^k(2d+1) \end{aligned}$$

Therefore, the induction step holds for $i = m$ for each of the scenarios (a) to (d). □

Theorem 10 The `IA/G/aux`() function (one iteration of the k-IA/G algorithm) has complexity $O(|V|^k|N|)$, where k is the number of allowed Steiner mergers.

Proof: Since there are at most $2|N| \Leftrightarrow 1$ nodes removed from the peer set (all terminals and Steiner nodes), the complexity of the algorithm is equal to $g_{G,N}(|V|, 2|N| \Leftrightarrow 1 + 1, k) = |V|^k(2(2|N|) + 1) = O(|V|^k|N|)$. □