Bounds on the quality of approximate solutions to the Group Steiner Problem *†

Edmund Ihler‡

Abstract

The Group Steiner Problem (GSP) is a generalized version of the well known Steiner Problem. For an undirected, connected distance graph with groups of required vertices and Steiner vertices, GSP asks for a shortest connected subgraph, containing at least one vertex of each group. As the Steiner Problem is NP-hard, GSP is too, and we are interested in approximation algorithms. Efficient approximation algorithms have already been proposed, but nothing about the quality of any approximate solution is known so far. Especially for the VLSI design application of the problem, bounds on the quality of approximate solutions are of great importance.

We present a simple polynomial time approximation algorithm that computes a tree with no more than $g-1$ times the length of a minimal tree, where g is the number of required groups. In addition, we propose an extended version of this algorithm, trading quality of the solution for computation time. Here, one extreme is just our proposed approximation, and the other is an optimal solution. Moreover, we will prove the quality bound $g-1$ for a modification of an efficient approximation algorithm proposed in the literature.

1 Introduction

Given a connected, undirected distance graph, the problem of finding a shortest connected subgraph spanning a required set of vertices, optionally using

*This work was partly supported by the Deutsche Forschungsgemeinschaft under grant Wid 810/5.

†Appeared in: *Graph-Theoretic Concepts in Computer Science, WG90*, No. 484 of *Lecture Notes in Computer Science*, pages 109–118, Springer, 1991.

‡Institut für Informatik, Universität Freiburg, Rheinstraße 10 - 12, D - 79104 Freiburg, Germany

vertices of a set of so-called Steiner vertices, is commonly called Steiner Problem (SP). A variety of applications is known. Since SP is NP-hard ([K]), several approximation algorithms have been proposed in the literature. In the routing phase of the physical VLSI design process, after the placement of components on a chip, pins of different components have to be connected, with the goal of constructing a shortest possible electrical network. This minimization aims at achieving high integration of the circuit and therefore is an essential aspect in practice. Let us take into account the freedom that results from rotating and mirroring components after placement, in order to shorten wire lengths. Here, any pin can be in any one out of several possible positions (for details, see [RW1]). By considering all possible positions of a pin to be a group of vertices, out of which just one needs to be connected, we are confronted with the following generalization of SP: For an undirected, connected distance graph with groups of required vertices and optional Steiner vertices, compute a shortest connected subgraph of the input graph, containing at least one vertex of each group. We call this generalized version of SP the Group Steiner Problem (GSP). As the Steiner Problem is NP-hard, GSP is too, and we are interested in approximation algorithms. Some efficient ones have already been given by [RW1], but nothing about the quality of the approximate solutions is known so far. In practice, guarantees for approximate solutions are an important first step in learning about the tradeoffs between efficiency of the computation and the optimality of the solution.

In this paper we present polynomial time algorithms, where the quality of the computed approximate solution is bounded. In Section 2 we introduce some definitions and formulate precisely the optimization problems SP and GSP. Section 3 presents a simple polynomial time approximation algorithm for GSP, computing a tree that is no more than $g-1$ times the length of a minimal tree, where g is the number of the required groups. We also prove $g-1$ to be the sharpest bound for the proposed approximation algorithm. In Section 4, we will give an extended version of this algorithm, tightening the quality bound to $g-h+1$, for an arbitrary positive integer $2 \leq h \leq g$. This increases the time complexity to a function exponential in h. Moreover, we prove in Section 5 the quality bound $g-1$ for a modification of an approximation algorithm proposed in [RW1] which is more efficient and computes better solutions in practice than the one given in Section 3.

2 GSP is a generalization of the Steiner Problem

We will consider *undirected distance graphs* only. A *graph* G is a tuple $G = (V, E, l)$, where V is the set of *vertices* in G, $E \subset \{(v, w) : v, w \in$

$V \wedge v \neq w\}$ is the set of *edges* in G and $l : E \rightarrow [0, \infty)$ is a *distance function*. We call a graph G_s a *subgraph* of G, writing $G_s \subset G$, if $V_s \subset V$, $E_s \subset E$, and l_s is a restriction of l to E_s. A $path(v, w)$ in G between vertices v and w where $v, w \in V$ is a subgraph G_s with $V_s = \{v, v_1, \ldots, v_k, w\}$ and $E_s = \{(v, v_1), (v_1, v_2), \ldots, (v_k, w)\}$. Writing $e \in G$ or $v \in G$ where e is an edge and v is a vertex means $e \in E$ or $v \in V$. The *total length* $l(G)$ of a graph G is defined as the sum $l(G) = \sum_{e \in E} l(e)$. If $l(G) \leq l(G')$, then we say G is shorter than G'. Consider $path(v, v)$ as a graph $(\{v\}, \emptyset, l)$ with a total length equal to zero. We call a graph G *connected* if $\forall v, w \in V \ \exists path(v, w) \subset G$. A graph G is a *tree*, if $\forall v, w \in V$ there is exactly one $path(v, w) \subset G$. A *leaf* of a tree T is a vertex $v \in T$ with $(v, w_1), (v, w_2) \in T \Rightarrow w_1 = w_2$. A *spanning* tree T for a set of vertices $\{v_1, \ldots, v_k\} \subset G$ of G is a tree where $v_1, \ldots, v_k \in T \wedge T \subset G$. For brevity, a spanning tree of G denotes a spanning tree for all vertices of G.

A connected graph G is a *Steiner Graph* (SG) if $V = S \cup R$ and $S \cap R = \emptyset$ with the set of *Steiner vertices* $S \neq \emptyset$ and the set of *required vertices* $R \neq \emptyset$. T is a *Steiner Tree* (ST) of G if T is a spanning tree for R of G. The *Steiner Problem* (SP) asks for a *minimal* ST of G, i.e., a ST of minimal total length among all ST of G.

Our generalized version of SP is the *Group Steiner Problem* (GSP), introduced by [RW1]. A connected graph G is a *Group Steiner Graph* (GSG) if $V = S \cup \bigcup_{i=1}^{g} R_i$ and g is a positive integer, $\forall i, j \in \{1, \ldots, g\}, i \neq j$: $R_i \cap R_j = \emptyset$, and $\forall i \in \{1, \ldots, g\} : (R_i \neq \emptyset \wedge S \cap R_i = \emptyset)$. We call the sets R_i *groups* of required vertices, and g the number of groups. Sometimes we will use the abbreviations $n := |V|$, $m := |E|$ and $n_i := |R_i|$. T is a *Group Steiner Tree* (GST) of G if $\exists (r_1, \ldots, r_g) \in R_1 \times \ldots \times R_g : T$ is a spanning tree for $\{r_1, \ldots, r_g\}$ of G. The Group Steiner Problem (GSP) asks for a *minimal* GST $T_M(G)$ of G, i.e., a GST of minimal total length among all GST of G. If there is no danger of confusion, we write T_M instead of $T_M(G)$.

We will sometimes write 'T is a ST of (G, R)' (or 'T is a GST of $(G, R_1, \ldots, R_g)$'). This will mean 'T is a ST of $G = (V, E, l)$ where $R \subset V$ and R is the set of required vertices' (or 'T is a GST of $G = (V, E, l)$ where $\bigcup_{i=1}^{g} R_i \subset V$ and $R_1, \ldots, R_g$ are the groups of required vertices'). All other vertices of V are Steiner vertices.

It is obvious, SP is a restriction of GSP, where $\forall i \in \{1, \ldots, g\} : n_i = 1$. SP is known to be NP-hard ([K]; for more information on NP-complete problems, see [GJ]). This involves GSP to be also NP-hard. [RW2] proved recently, that in contrast to the ordinary SP, GSP is NP-hard, even if the input graph is a tree and in addition, $S = \emptyset$ and $\forall i \in \{1, \ldots, g\} : n_i \leq 3$. Hence, unless P = NP, no polynomial time algorithm for GSP exists. We therefore search for approximation algorithms that are efficient and guarantee a good approximation. Efficiency of approximation algorithms has been considered in [RW1].

For a restricted version of GSP, called GSP(k), where the number of groups g for each input graph G is bounded by some fixed integer k, the minimization problem is solvable in polynomial time. Such a simple polynomial time algorithm, called *exact* algorithm, is the following: Compute for each tuple $(r_1, \ldots, r_g) \in R_1 \times \ldots \times R_g$ a minimal ST of $(G, \{r_1, \ldots, r_g\})$. Pick a shortest one of these. Obviously, this is a minimal GST of G. To compute a minimal ST, we use an exact polynomial time algorithm for SP(k), where SP(k) is a restriction of SP to input graphs with no more than k required vertices (see e.g. [DW]). Since the number of considered tuples is bounded by n^k, the algorithm runs in polynomial time.

In this paper, we are concerned with approximation algorithms that guarantee some quality of the approximation, without too much consideration for efficiency. From the VLSI application it should be clear that it may well be worthwhile to spend some extra computation time, in order to achieve a better solution. Besides aiming at an approximation algorithm that delivers a good solution on the average, we are interested in a worst-case upper bound on the total length of the computed tree, in comparison with a minimal GST. It turns out that GSP is far more complex than SP in this respect also: none of the approximation algorithms of [RW1] is guaranteed to produce acceptable solutions, and the guarantee for SP cannot simply be carried over to GSP. Nevertheless, let us see how an approximation algorithm for SP (respectively an exact algorithm for SP(k)) can be used for GSP.

If an algorithm A for SP (or GSP) computes in polynomial time a ST (or GST) T_A of a given SG (or GSG) G, and if there is a function $c : G \mapsto c(G) \in [1, \infty)$ binding the ratio $\frac{l(T_A)}{l(T_M)} \leq c(G)$ for each input graph G, then we call A a $c(G)$-approximation algorithm for SP (or GSP). For SP, approximation algorithms with $c(G) = c = constant$ have been found (see e.g. [M], [KMB] for $c \leq 2$).

The next section presents a simple polynomial time $(g-1)$-approximation algorithm for GSP, where g is the number of required groups.

3 A (g–1)-approximation algorithm for the GSP

We present a simple polynomial time approximation algorithm for the GSP:

Algorithm A:

input: a Group Steiner Graph $G = (S \cup \bigcup_{i=1}^{g} R_i, E, l)$

output: a Group Steiner Tree T_A of G

1. For each vertex r_1 of group R_1, find a shortest path to each other group and let G_{r_1} be the union of all these paths.

2. Let G_A be a shortest one of these n_1 graphs G_{r_1}.

3. Find a spanning tree T_A of G_A.

To precisely define the terms we will use in arguing about A in the following, we give a more detailed description of A. We formulate A as a function of a graph and its required groups:

function $T_A(G, R_1, \ldots, R_g)$

1. **foreach** $r_1 \in R_1$ **do**

   1.1. **foreach** $i \in \{2, \ldots, g\}$ **do**

   1.1.1. **foreach** $r_i \in R_i$ **do**

   $p_{r_1,r_i} := find\_a\_shortest\_path(r_1, r_i)$

   1.1.2. $p_{r_1,i} := min(\{p_{r_1,r_i} : r_i \in R_i\})$

   1.2. $G_{r_1} := \bigcup_{i=2}^{g} p_{r_1,i}$

2. $G_A := min(\{G_{r_1} : r_1 \in R_1\})$

3. $T_A := find\_a\_spanning\_tree(G_A)$

Several modifications are possible to optimize this algorithm, such as picking the smallest group R_i instead of R_1, or computing the minimal spanning tree of G_A in step $A.3$. Since this does not lead to a sharper bound for the worst case, it is not of any interest here.

Theorem 1 *Algorithm A is a polynomial time $(g-1)$-approximation algorithm for GSP with $2 \leq g$, and this is the best possible bound for A.*

Proof: First we show the polynomial time complexity of A. The function $find\_a\_shortest\_path$ can be computed in $O(n \log n + m)$ steps [FT]. With $n_1 \leq n$ and $n - n_1 < n$, $find\_a\_shortest\_path$ is executed $O(n^2)$ times, leading to $O(n^2(n \log n + m))$ steps. Since this execution time dominates that of all other operations in A, it is also the worst case time complexity of A.

Second we will prove that T_A is a GST. A computes $\forall r_1 \in R_1$ a graph G_{r_1} in a way that ensures $\forall i \in \{2, \ldots, g\}$ $\exists$ a path $p_{r_1,r_i} \subset G_{r_1}$ between r_1 and r_i where $r_i \in R_i$. Therefore G_A is connected and represents all groups. Hence the spanning tree T_A of G_A is a GST of G.

In the next step, we verify the asserted quality. T_M is a GST of G, especially the minimal one, therefore $\exists (r_1, \ldots, r_g) \in R_1 \times \ldots \times R_g : r_1, \ldots, r_g \in T_M$. We fix such a tuple and call it $(v_1, \ldots, v_g)$. Further $\forall i \in \{2, \ldots, g\}$ exists exactly one $path(v_1, v_i) \subset T_M$. Denote this path $p^M_{v_1,v_i}$. Obviously, for each $p^M_{v_1,v_i} \subset T_M$ follows $l(p^M_{v_1,v_i}) \leq l(T_M)$. As for all $r_1 \in R_1$, A computes a graph G_{v_1} for v_1, and Step A.2 ensures $l(G_A) \leq l(G_{v_1})$. To build G_{v_1}, Step A.1.1.1 finds shortest paths from v_1 to the vertices of each other group i, A.1.1.2 picks the shortest path $p_{v_1,i}$ of these. Therefore $l(p_{v_1,i}) \leq l(p^M_{v_1,v_i})$. A look at Steps A.3 and A.1.2 completes this part of the proof and we summarize:

$$\begin{array}{rcl} l(T_A) & \leq & l(G_A) \leq l(G_{v_1}) \leq \sum_{i\in\{2,\ldots,g\}} l(p_{v_1,i}) \leq \sum_{i\in\{2,\ldots,g\}} l(p^M_{v_1,v_i}) \\ & \leq & (g-1) l(T_M) \end{array}$$

To show $(g-1)$ is the best bound possible for A, we give a special family of graphs for $g > 2$ that achieves this bound. R_i are the groups of required vertices.

$$\begin{array}{rcl} G_\epsilon & := & (V_\epsilon, E_\epsilon, l_\epsilon) \\ V_\epsilon & := & \bigcup_{i=1}^{g} R_i \\ & & R_1 := \{r_1\};\ R_i := \{r_i^M, r_i^A\};\ i \in \{2,\ldots,g\} \\ E_\epsilon & := & \{(r_1, r_g^A), (r_1, r_g^M)\} \cup \{(r_1, r_i^A), (r_g^M, r_i^M) : i \in \{2,\ldots,g-1\}\} \\ l_\epsilon & : & (r_1, r_i^A) \mapsto 1;\ i \in \{2,\ldots,g\};\ (r_1, r_g^M) \mapsto 1 \\ & & (r_g^M, r_i^M) \mapsto \epsilon;\ i \in \{2,\ldots,g-1\};\ 0 < \epsilon < 1 \end{array}$$

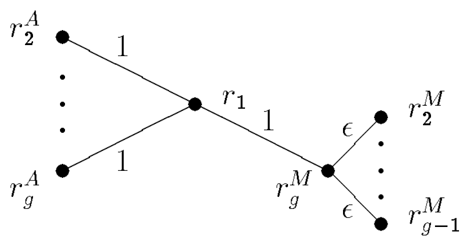

Figure 1: Worst case graph G_ϵ

Note that G_ϵ is already a tree. To compute a GST of G_ϵ, algorithm A starts with vertex r_1, the only one of group 1. Then it uses for each group i edge (r_1, r_i^A) as a nearest path to group i. But the minimal GST consists of the edges $(r_1, r_g^M), (r_g^M, r_i^M)$, for $i \in \{2,\ldots,g-1\}$. If ϵ tends to zero, we get

$$\frac{l(T_A)}{l(T_M)} = \frac{(g-1)}{1+(g-2)\epsilon} \quad \xrightarrow{\epsilon\to 0} \quad g-1. \qquad \square$$

Since g can be arbitrarily big, the question for a tighter bound is obvious. To achieve approximation algorithms with G-independent, constant bounds, as they do exist for SP, we cannot easily adapt the algorithms for SP suitably. Nevertheless, the following section presents an approach to use such results for SP, getting a tighter quality bound.

4 An extended version of algorithm A

It is possible to get a smaller approximation ratio, by extending the algorithm A given in the last paragraph. To this end, we distinguish between the first h groups of required vertices, with $2 \leq h \leq g$, and the remaining $g-h$ groups, and we apply two different methods to them. The first method is to compute a minimal ST for each combination of h vertices, with one vertex of the groups 1 to h. Each minimal ST is computed by an exact polynomial time algorithm. The second method is algorithm A, applied to the remaining groups $h+1$ to g, together with group 1 to achieve connectivity. The exact and the approximate partial solution combine to a total approximate solution. Clearly, the time complexity increases with increasing parameter h. A medium h is a compromise between an optimal solution and low order time complexity.

Algorithm A^h:

input: a Group Steiner Graph $G = (S \cup \bigcup_{i=1}^{g} R_i, E, l)$

output: a Group Steiner Tree T_{A^h} of G

1. For each vertex r_1 of group R_1 and each tuple $(r_2, \ldots, r_h) \in R_2 \times \cdots \times R_h$, use an exact polynomial time algorithm for SP (e.g. [DW]) to compute a minimal ST $T_{r_1,\ldots,r_h}$ of $(G, \{r_1, \ldots, r_h\})$. Let $T_{r_1}^h$ be a shortest one of these.
   Use algorithm A to get a GST $T_{r_1}^g$ of $(G, \{r_1\}, R_{h+1}, \ldots, R_g)$.
2. Let G_{A^h} be a shortest one of the subgraphs $T_{r_1}^h \cup T_{r_1}^g$.
3. Find a spanning tree T_{A^h} of G_{A^h}.

In the following more detailed version of algorithm A^h, defining the terms we will use in the following, function $find\_a\_minimal\_Steiner\_Tree$ stands for an implementation of an exact polynomial time algorithm for the ordinary Steiner Problem (e.g. [DW]), for an input graph and a set of required vertices.

function $T_{A^h}(G, R_1, \ldots, R_g)$

1. **foreach** $r_1 \in R_1$ **do**

   1.1. **foreach** $(r_2, \ldots, r_h) \in R_2 \times \cdots \times R_h$ **do**

   $T_{r_1,\ldots,r_h} := find\_a\_minimal\_Steiner\_Tree(G, \{r_1, \ldots, r_h\})$

   1.2. $T_{r_1}^h := min(\{T_{r_1,\ldots,r_h} : (r_2, \ldots, r_h) \in R_2 \times \cdots \times R_h\})$

1.3. $T^g_{r_1} := T_A(G, \{r_1\}, R_{h+1}, \ldots, R_g)$

2. $G_{A^h} := min(\{T^h_{r_1} \cup T^g_{r_1} : r_1 \in R_1\})$

3. $T_{A^h} := find\_a\_spanning\_tree(G_{A^h})$

Theorem 2 *Algorithm A^h is a polynomial time $(g-h+1)$-approximation algorithm for GSP with $2 \leq h \leq g$ (h is a fixed integer), and this is the best possible bound for A^h.*

Proof: Function $find\_a\_minimal\_Steiner\_Tree$ implemented as proposed in [DW] takes roughly $O(n^3 3^h)$ time. The number of tuples $(r_1, \ldots, r_h)$ is bounded by n^h, as $n_i \leq n$. This part of A^h dominates the computation time leading to a total time complexity of $O(n^{h+3}3^h)$.

To see that T_{A^h} is a GST of G, consider the trees $T^h_{r_1}$ and $T^g_{r_1}$ for each $r_1 \in R_1$, computed by A^h. $T^h_{r_1} \cup T^g_{r_1}$ is connected, since $T^h_{r_1}$ and $T^g_{r_1}$ are both connected graphs (namely trees), and $T^h_{r_1} \cap T^g_{r_1} \ni r_1$; also it contains one vertex of each group. As T_{A^h} is a spanning tree of G_{A^h} and G_{A^h} is $T^h_{r_1} \cup T^g_{r_1}$ for some $r_1 \in R_1$, T_{A^h} is a GST.

We prove the quality of the solution as follows: Recall that T_M is a minimal GST of $(G, R_1, \ldots, R_g)$. This implies $\exists (r_1, \ldots, r_g) \in R_1 \times \ldots \times R_g : r_1, \ldots, r_g \in T_M$. We fix such a tuple, denoting it $(v_1, \ldots, v_g)$. We call a minimal ST of $(G, \{v_1, \ldots, v_h\})$ T^h_M. Steps A^h.1 and A^h.1.1 ensure that we compute a tree $T_{v_1,\ldots,v_h}$ where $\{v_1, \ldots, v_h\} \in T_{v_1,\ldots,v_h}$. The use of the exact algorithm of [DW] in the Step A^h.1.1 guarantees $l(T_{v_1,\ldots,v_h}) = l(T^h_M)$. Because T^h_M is a shortest spanning tree for $\{v_1, \ldots, v_h\}$ of G, this leads to $l(T^h_M) \leq l(T_M)$. A look at step A^h.1.2 verifies $l(T^h_{v_1}) \leq l(T_{v_1,\ldots,v_h})$. Altogether, this yields:

$$l(T^h_{v_1}) \leq l(T_{v_1,\ldots,v_h}) = l(T^h_M) \leq l(T_M) \tag{1}$$

Furthermore, let T^g_M be a minimal GST of $(G, \{v_1\}, R_{h+1}, \ldots, R_g)$. Then by the use of algorithm A in step A^h.1.3, Theorem 1 ensures $l(T^g_{v_1}) \leq (g-h)l(T^g_M)$. The fact that T_M is a minimal GST for all the groups, i.e., the groups of T^g_M and $h-1$ more groups, implies $l(T^g_M) \leq l(T_M)$. Therefore

$$l(T^g_{v1}) \leq (g-h)l(T^g_M) \leq (g-h)l(T_M) \tag{2}$$

By a look at step A^h.2 and A^h.3 we get:

$$\begin{array}{rcl} l(T_{A^h}) & \leq & l(G_{A^h}) \leq l(T^h_{v1} \cup T^g_{v1}) \leq l(T^h_{v1}) + l(T^g_{v1}) \\ & \overset{(1),(2)}{\leq} & (g-h+1)l(T_M) \end{array}$$

To show $g-h+1$ is the sharpest bound for A^h, we use the family of graphs G_ϵ, given in the proof of Theorem 1 for $2 < h < g$ (case $h = g$ is evident). The usage of A^h leads to $l(T^h_{r_1}) = 1 + (h-1)\epsilon$, $l(T^g_{r_1}) = g - h$ and $l(T_M) = 1 + (g-2)\epsilon$. If ϵ tends to zero, this completes the proof. $\square$

It is possible to decrease the time complexity of A^h without too much increase of the quality bound, if we replace the function $find\_a\_minimal\_Steiner\_Tree$ in A^h by an implementation of the $(2(1-\frac{1}{n_{leaves}}))$-approximation algorithm for the SP in [M] with time complexity $O(n \log n + m)$, where n_{leaves} is the number of leaves in the minimal ST. An analogous proof is possible and leads to a total time complexity of $O(n^h(n \log n + m))$. For the quality bound, we have to replace the inequality (1) by

$$l(T^h_{v_1}) \le l(T_{v_1,\dots,v_h}) \le 2(1-\frac{1}{L})l(T^h_M) \le 2(1-\frac{1}{L})l(T_M) \le 2(1-\frac{1}{h})l(T_M)$$

where L is the number of leaves in T^h_M, and $L \le h$ holds, because there cannot exist a leaf of T^h_M not being member of $\{v_1,\dots,v_h\}$. This leads to a quality bound $(g-h+2(1-\frac{1}{h}))$. If $1 \ll (g-h)$, this version of A^h is a good alternative.

The algorithms A and A^h describe general methods and it is possible to refine these methods. For example, A adds shortest paths from a root vertex to each group, without checking whether there is a shorter path starting from another vertex connected to this root. An approximation algorithm proposed by [RW1] adds these shortest paths instead. We will extend this algorithm in the next section and prove it to be a $(g-1)$-approximation algorithm. In the worst case, this algorithm is not better than A, but for certain practically relevant classes of graphs, it achieves better solutions more efficiently.

5 A better (g–1)-approximation algorithm

In this section, we modify an efficient algorithm for GSP with time complexity $O(g(nlogn+m))$ given by [RW1]. We also prove this extended version to be a $(g-1)$-approximation algorithm, whereas the original has no l-independent bound. Let us call this algorithm B, presented as 'a group Steiner minimal tree approximation' in [RW1]. We briefly describe algorithm B; for more details see [RW1].

Algorithm B:

input: a GSG $G = (S \cup \bigcup_{i=1}^g R_i, E, l)$, an arbitrary vertex $r \in \bigcup_{i=1}^g R_i$

output: a GST T_B of G

function $T_B(G, r)$

1. $T^1_r := \{r\}$

2. **for** $i := 2$ **to** g **do**

   2.1. *find a shortest path, denoted* $p(T_r^{i-1}, r_i)$*, from* T_r^{i-1} *to a nearest required vertex* r_i *outside* T_r^{i-1}*, where the group of* r_i *is not yet represented in* T_r^{i-1}

   2.2. $T_r^i := T_r^{i-1} \cup p(T_r^{i-1}, r_i)$

3. T_B := *clean up tree* T_r^g (* deletes redundant vertices and edges of T_r^g *)

Lemma 1 *If* B *is a* $c(G)$*-approximation algorithm for GSP, then* $c(G)$ *is* l*-dependent.*

Proof: Assume $c \geq 1$ is an l-independent bound, and consider a GSG (V, E, l): $V = R_1 \cup R_2$; $R_1 = \{r_1^1, r_1^2\}$; $R_2 = \{r_2\}$ and $E = \{(r_1^1, r_2), (r_1^2, r_2)\}$ and $l((r_1^1, r_2)) = 1$; $l((r_1^2, r_2)) = 2c$. In the worst case, B picks arbitrarily r_1^2 to start, therefore $\frac{l(T_B)}{l(T_M)} = \frac{2c}{1} > c$, and this contradicts the assumption. □

To avoid this weak point, we extend B to algorithm B^+ which is similar to A. B^+ works better in practice than A, but respects no tighter bound in the worst case.

Algorithm B^+:

input: a GSG $G = (S \cup \bigcup_{i=1}^{g} R_i, E, l)$

output: a GST T_{B^+} of G

function $T_{B^+}(G, R_1, \ldots, R_g)$

1. **foreach** $r_1 \in R_1$ **do**

   $T_{r_1} := T_B(G, r_1)$

2. $T_{B^+} := min(\{T_{r_1} : r_1 \in R_1\})$

Comparing A with B^+, we recognize: Step A.2 is not necessary for B^+, because T_{r_1}, replacing G_{r_1} in A, is already a tree and also T_B replacing G_A. Step A.1 and all substeps are substituted by B^+.1, using B.

Theorem 3 *Algorithm* B^+ *is a polynomial time* $(g-1)$*-approximation algorithm for GSP with* $2 \leq g$*, and this is the best possible bound for* B^+*.*

Proof: As Step $B^+.1$ is the most time consuming step, and B can be computed with $O(g(n \log n + m))$ steps, B^+ has time complexity $O(gn(n \log n + m))$.

With T_B being a GST (see [RW1]), T_{B+} is a GST, too. Without loss of generality, let the vertex r_i, computed in step $B.2.1$ be a member of R_i. This renumbering of the groups of required vertices is possible, because the order of the groups has no effect on B. Recall that T_M is a minimal GST of G. This implies that $\exists (r_1, \ldots, r_g) \in R_1 \times \ldots \times R_g : r_1, \ldots, r_g \in T_M$. We fix such a tuple, denoting it $(v_1, \ldots, v_g)$. $T^g_{v_1}$ is a tree, incrementally computed in step $B.2.2$ for our fixed starting vertex v_1. Further $\forall i \in \{2, \ldots, g\}$ there is a unique $path(v_1, v_i) \subset T_M$, which we denote $p^M(v_1, v_i)$. Obviously, we have $l(p^M(v_1, v_i)) \leq l(T_M)$. Recall that $p(T^{i-1}_{v_1}, r_i)$ is this shortest path from $T^{i-1}_{v_1}$ to R_i found in Step $B.2.1$. This step ensures $l(p(T^{i-1}_{v_1}, r_i)) \leq l(p^M(v_1, v_i))$. Then the inequality

$$l(T^g_{v_1}) \leq (g-1)l(T_M) \tag{3}$$

is proved by the following induction over the number i of groups:

$$\begin{array}{lll} i = 2: & l(T^2_{v_1}) & = l(p(T^1_{v_1}, r_2)) \leq l(p^M(v_1, v_2)) \leq l(T_M) \\ i \rightsquigarrow i+1: & l(T^{i+1}_{v_1}) & = l(T^i_{v_1}) + l(p(T^i_{v_1}, r_{i+1})) \leq l(T^i_{v_1}) + l(p^M(v_1, v_{i+1})) \\ & & \leq ((i+1)-1)l(T_M) \end{array}$$

We complete this part of the proof by a look at B and B^+:

$$l(T_{B+}) \quad \leq \quad l(T_{v_1}) \leq l(T^g_{v_1}) \overset{(3)}{\leq} (g-1)l(T_M)$$

To show $(g-1)$ is the best bound, we can use the family of graphs G_ϵ and the arguments given for Theorem 1. This completes the proof. $\square$

6 Conclusion

For a natural generalization of the Steiner problem, where we have groups of required vertices instead of simply required vertices, we have studied the problem of approximating an optimal solution within some known bound. In a first attempt to solve this problem, we have presented three approximation algorithms, a simple one, a better one having the same quality bound in the worst case, and one with flexible quality guarantee. The approximation solutions are worse than the optimal solution roughly by a factor g, the number of groups of required vertices. Bounds independent of problem parameters, such as a constant, would be far more desirable. Our attempts at developing a corresponding approximation algorithm have so far been unsuccessfull. It can be shown, however, that the problem of constant factor approximation for graphs without Steiner vertices and constant distance function, i.e. all edges have unit length, is as hard as the problem with Steiner vertices and positive rational distance function.

References

[DW] S. E. Dreyfus, R. A. Wagner:
The Steiner Problem in Graphs, Networks, Vol. 1, 1971, 195 – 207.

[FT] M. L. Fredman, R. E. Tarjan:
Fibonacci Heaps and Their Uses in Improved Networks Optimization Algorithms, Journal of the ACM, Vol. 34, 1987, 596 – 615.

[GJ] M. R. Garey, D. S. Johnson:
Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

[K] R. M. Karp:
Reducibility among Combinatorial Problems, Complexity of Computer Computations, New York, 1972, 85 – 103.

[KMB] L. Kou, G. Markowsky, L. Berman:
A Fast Algorithm for Steiner Trees, Acta Informatica, Vol. 15, 1981, 141 – 145.

[M] K. Mehlhorn:
A Faster Approximation Algorithm for the Steiner Problem in Graphs, Information Processing Letters, Vol. 27, 1988, 125 – 128.

[RW1] G. Reich, P. Widmayer:
Beyond Steiner's problem: A VLSI oriented generalization, 15th International Workshop on Graph-Theoretic Concepts in Computer Science, Castle Rolduc, 1989.

[RW2] G. Reich, P. Widmayer:
Approximate Minimum Spanning Trees for Vertex Classes, Technical Report, Institut für Informatik, Universität Freiburg, Rheinstraße 10 – 12, 7800 Freiburg, West Germany