Report Number: WUCS-89-15

1989-05-01

New Approximation Algorithms for the Steiner Tree Problem

Bernard M. Waxman

In this paper we consider several approximation algorithms for the Steiner tree problem in an attempt to find one whose worst case performance is better than two times optimal. We first examine Rayward-Smith's algorithm to gain insight into why it has worst case performance no better than two. Based on these ideas we propose several new algorithms (approximation schemes). We eliminate from further consideration those which we have bene able to showed have worst case performance that is still no better than two. Then we conjecture that one of these schemes not only has improved worst case performance, but... **Read complete abstract on page 2.**

New Approximation Algorithms for the Steiner Tree Problem

Bernard M. Waxman

Complete Abstract:

In this paper we consider several approximation algorithms for the Steiner tree problem in an attempt to find one whose worst case performance is better than two times optimal. We first examine Rayward-Smith's algorithm to gain insight into why it has worst case performance no better than two. Based on these ideas we propose several new algorithms (approximation schemes). We eliminate from further consideration those which we have bene able to showed have worst case performance that is still no better than two. Then we conjecture that one of these schemes not only has improved worst case performance, but is also the basis for a polynomial scheme. That is, given an e greater than 0 we conjecture that this scheme specifies a polynomial time approximation algorithm with worst case performance within 1 + e times optimal.

NEW APPROXIMATION ALGORITHMS FOR THE STEINER TREE PROBLEM

Bernard M. Waxman

WUCS-89-15

May 1989

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130-4899

ABSTRACT

In this paper we consider several approximation algorithms for the Steiner tree problem in an attempt to find one whose worst case performance is better than two times optimal. We first examine Rayward-Smith's algorithm to gain insight into why it has worst case performance no better than two. Based on these ideas we propose several new algorithms (approximation schemes). We eliminate from further consideration those which we have been able to show have worst case performance that is still no better than two. Then we conjecture that one of these schemes not only has improved worst case performance, but is also the basis for a polynomial scheme. That is, given an $\epsilon > 0$ we conjecture that this scheme specifies a polynomial time approximation algorithm with worst case performance within $1 + \epsilon$ times optimal.

This work supported by NSF grant DCI-86000947, Bellcore, BNR, Italtel SIT, and NEC.

New Approximation Algorithms for the Steiner Tree Problem

Bernard M. Waxman

1. Steiner Tree problem

In this paper we consider polynomial time approximation algorithms for the Steiner tree problem in graphs, which was shown to be NP-complete by Karp in 1972 [3]. Formally the Steiner problem in graphs can be stated as follows. Given

- a graph $G = (V, E)$,
- a cost function $C : E \rightarrow \Re^+$,
- and a set of vertices $D \subseteq V$, called the set of terminal nodes,

find a subtree $T = (V_T, E_T)$ of G which spans D, such that the cost$(T) = \sum_{e \in E_T} C(e)$ is minimized. Throughout this paper we assume that each graph $G(V, E)$ is connected, and let

- $n = |V|$
- $m = |E|$
- and $k = |D|$.

We say that an approximation algorithm for the Steiner tree problem has worst case performance within B times optimal if the solution produced for any instance of the Steiner tree problem has cost no more than B times the cost of a minimum Steiner tree. There are a large number of approximation algorithms [6] for deriving solutions to the Steiner tree problem in graphs that run in polynomial time, though none of these algorithm is know to have worst case performance better than twice optimal. In fact, a number of researchers believe that no polynomial time approximation algorithm can do better that twice optimal assuming that P $\neq$ NP. On the other hand, we do not take this view and discus our search for an algorithm whose worst case performance is better than two.

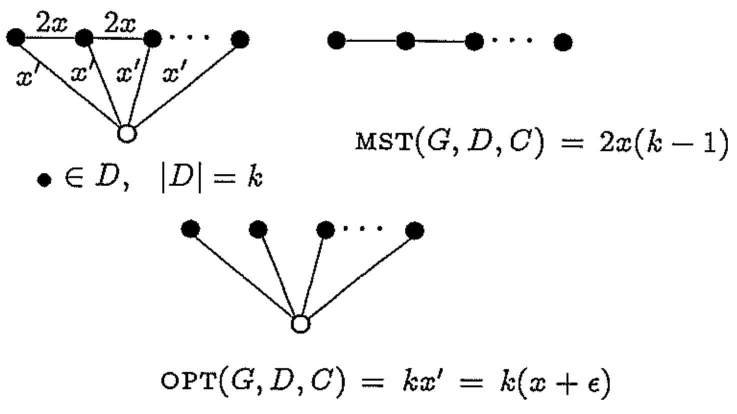

$2x$
$2x$
x'
x'
x'
x'
$\bullet \in D, \quad |D| = k$
$\text{MST}(G, D, C) = 2x(k-1)$
$\text{OPT}(G, D, C) = kx' = k(x+\epsilon)$

Figure 1: MST approaches two times optimal

2. Approximation Algorithms Based on Distance

There are a number of algorithms whose operation is based only on the distance between each pair of nodes in the terminal set. Of these algorithms MST [2] is perhaps the most well known. It can easily be seen that no algorithm based only on this information can have worst case performance better than two [1]. The simplest example of an instance that illustrates this point consists of a graph with one nonterminal node which when included in the Steiner tree yields a tree whose cost is reduced by nearly a factor of two. (See Figure 1.) In the next section we consider in some detail an approximation algorithm proposed by Rayward-Smith [4] (RS), which attempts to find such nonterminal nodes, but as shown in [5] RS still does not have worst case performance better than two. We note that for the collection of instances used in [5], to show that RS can have performance as bad as two, RS makes its choices based only on the distances between nodes in the terminal set. This leads to the idea that if there exists an approximation algorithm with better worst case performance it must take advantage information in addition to the distances between terminal nodes.

3. Rayward-Smith's Approximation Algorithm

The problem with MST and other similar algorithms is that they only consider the distances between nodes in the terminal set D and do not consider the value to a final solution of nodes not in D. Nodes outside of D important to a solution of the Steiner tree problem are called Steiner points. More formally, given an instance of the Steiner problem (G, D, C), we say that a node s is a Steiner point of $T = (V_T, E_T)$ if

- $s \in V_T$,
- $s \notin D$,
- and $\deg_T(s) \geq 3$ (degree in T).

RS is of particular interest since it makes a systematic attempt to find potential Steiner points. Even though, RS does not yield improved worst case performance we will use it as the basis for a new collection of polynomial time approximation algorithms. Thus, we look at RS in some detail.

The operation of RS is analogous to Kruskal's algorithm in that at each stage RS joins two subtrees. RS starts with a collection of singleton trees corresponding to the set of terminal nodes D. Then at each stage RS joins a pair of trees by a path between them. This is repeated until only one tree remains. In order to choose a pair of subtrees and a path joining them Rayward-Smith defines a collection of functions $f_\ell : V \to \Re^+$ for each stage ℓ, $0 \leq \ell < k-1$. Using this function RS chooses a node v such that $f(v)$ is minimum and then joins two subtrees closest to v with a shortest path through v. When it causes no confusion, we will drop the subscript. The nodes $v \notin D$ for which f has a minimum value are potential Steiner points.

In the algorithmic definition that follows the implied subscript ℓ, $0 \leq \ell < k-1$ of f indicates the ℓ-th iteration of the **do** statement.

```
Derive the function dist: $V \times V \to \Re_0^+$;
Initialize $\mathcal{T} \; = \; \{(\{v\}, \emptyset) | v \in D\}$;
$\forall v \in V$ create a list $L_v$ of trees $T = (V_T, E_T) \in \mathcal{T}$ in ascending order by $\mathrm{dist}(v, T)$;
        Here $\mathrm{dist}(v, T) = \min_{u \in V_T}\{\mathrm{dist}(v, u)\}$.
do $|\mathcal{T}| > 1 \to$
        $\forall v \in V \to \; f[v] := \mathrm{val}(v)$;
        Choose a vertex $v$ that gives a minimum value for $f(v)$;
        $T_0 := L_v[0]$;   $T_1 := L_v[1]$;
        Join $T_0$ and $T_1$ with a shortest path through $v$ to form $T'$;
        $\mathcal{T} := (\mathcal{T} - \{T_0, T_1\}) \cup \{T'\}$;
        $\forall v \in V$ calculate $\mathrm{dist}(v, T')$;
        $\forall v \in V$ update $L_v$
od;
```

The total running time for RS is $O(n^3)$.

Of course the crucial part of this heuristic is the calculation of f. We first define f_ℓ.

$$f_\ell(v) \; = \; \min_{0 < r < |\mathcal{T}|} \left\{ 1/r \sum_{i=0}^{r} \mathrm{dist}(v, T_i) \right\} \tag{1}$$

where $T_i \; = \; L_v[i]$, after ℓ executions of the **do** statement. Note that L_v is a list of subtrees currently in $\mathcal{T}$ listed in nondecreasing order by their distance from v.

To calculate $f(v)$ we can evaluate the sums beginning with $r = 1$ continuing until $\text{dist}(v, T_{r+1}) \geq 1/r \sum_{i=0}^{r} \text{dist}(v, T_i)$. Note that to reorder each L_v after the set T has been modified we need only remove two trees from the list and insert the new tree in the appropriate spot. We now give an algorithmic definition for the function val.

```
function val(node $u$)
    $f := g := \text{dist}(u, L_u[0]) + \text{dist}(u, L_u[1])$;   $r := 2$;
    do $f > \text{dist}(u, L_u[r])$ and $r < |T| \rightarrow$
        $g := g + \text{dist}(u, L_u[r])$;
        $f := g/r$;
        $r := r + 1$
    od;
    return f
end val;
```

The total running time for *val* is $O(k)$

When we talk about the value of r associated with the function f, we are referring to the value of r in Equation 1. Note that the value of r on exiting function val is one more than this value.

4. Approximation Algorithms with Improved Worst Case Performance?

In this section we present several schemes for creating polynomial time approximation algorithms for the Steiner problem in an attempt to find ones which have worst case performance better than two. We first look at two exact algorithms which we employ as a subroutine. After that we consider several approximation algorithm schemes which looked promising but turn out to have worst case performance no better than two. These algorithms are worth considering in order to discover what techniques will not work and for insight into other promising techniques. Finally we present a scheme which we conjecture has worst case performance better than two.

4.1. EMST and EXACT

MST can be used to create an exact algorithm, EMST, if MST is given the Steiner points of an optimal solution along with the set of terminal nodes. The following remark is used as the basis for EMST.

Remark 4.1 *If $k = |D|$, then any minimum Steiner tree T for D will contain at most $k - 2$ Steiner points.*

This can be shown by using a few simple facts. First, given a graph $T(V, E)$ the sum of the degree of each node in V is $2m$, where, as usual, $m = |E|$. Thus, if T is a tree, this sum is $2(n-1)$, $n = |V|$. Second, if T is a minimum Steiner tree, T can have at most k leaf nodes, i.e., nodes from D. Thus, all other nodes in T will have degree at least 2. If we let s be the number of Steiner points (degree at least 3) we get the following inequality

$$k + 3s + 2(n-k-s) \quad \leq \quad 2(n-1)\,.$$

Solving for s we get $s \leq k-2$.

We now give an informal description of EMST. Given an instance of the Steiner problem ($G(V, E)$, D, C), apply MST to sets $D \cup X$ for all possible $X \subseteq V - D$ where $|X| \leq k-2$. Of all trees constructed in this manner those with minimum cost will be minimum Steiner trees.

Based on EMST we define a function EXACT (algorithm) which we use in the approximation algorithms below. Given a set of subtrees S generated as partial solutions to an instance of the Steiner problem, EXACT finds a minimum tree connecting the members of S. That is EXACT treats the subtrees of S as if they were single nodes. In the following the function **dist** has three meanings depending on its parameter types.

1. dist(**node** u, **node** v) is just the length of a shortest path.
2. dist(**node** u, **subtree** T) is the minimum distance between u and any node in T as defined for RS.
3. dist(**subtree** T_i, **subtree** T_j) is is the minimum distance between any pair of nodes, one from each tree, i.e.,

$$\text{dist}(T_i, T_j) \quad = \quad \min_{u \in T_i, v \in T_j} \text{dist}(u, v)\,.$$

Types (2) and (3) of the function **dist** can be maintained incrementally in any algorithm which uses EXACT.

We now proceed with an algorithm for EXACT. Let $V(S)$ represent the set of all nodes in the elements of S. Assume that the function **dist** and the graph $G(V, E)$ are global to EXACT, and assume that there exists a bijection $M : V(G_{W,S}) \overset{1:1}{\leftrightarrow} W \cup S$ where $G_{W,S}$ is a complete graph with $|W| + |S|$ nodes.

```
Tree function EXACT(foreset $S$)
    do $W \subseteq V - V(S)$ and $|W| \leq |S| - 2$ $\rightarrow$
        Create the complete graph $G_{W,S}$;
            *where each node is identified with a distinct element of* $W \cup S$
            and the $\mathrm{dist}(u, v)$, $u, v \in V(G_{W,S}) = \mathrm{dist}(M(u), M(v))$.
        $T_{W,S}$ := *minimum spanning tree* of $G_{W,S}$
    od;
    Choose a tree $T'$ with minimum cost among all $T_{W,S}$;
    Expand $T'$ to a subtree $T$ of $G$;
    return $T$
end EXACT;
```

The running time for EXACT is $O(kn^k)$, which is polynomial if we bound the maximum value of k. Note that here $k = |S|$.

Along with EXACT we define a new cost function

$$\mathrm{cost}'(T, S) \quad = \quad \mathrm{cost}(T) - \sum_{t \in S} \mathrm{cost}(t)$$

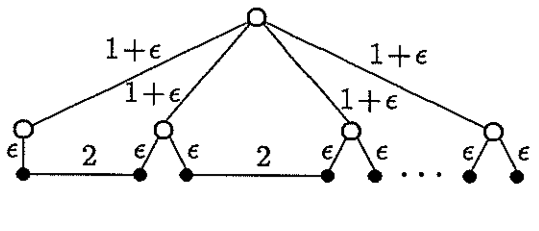
which returns the cost of joining the subtrees of S without including the cost of the edges within each subtree.

4.2. N-TREE Approximation Algorithms

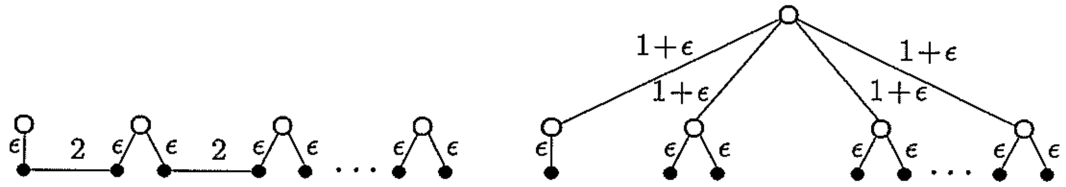
As a first attempt to find polynomial time approximation algorithms which have worst case performance better than two, we define a collection of N-TREE algorithms. The basic idea here is to begin with a collection of singleton trees $\mathcal{T}$ consisting of the nodes in D. At each stage of the algorithm we find a minimum tree joining each subset of N subtrees from the collection $\mathcal{T}$.

```
Initialize $\mathcal{T}$ := $\{(\{v\}, \emptyset) | v \in D\}$;
do $|\mathcal{T}| \geq N$ $\rightarrow$
    $\forall S \subseteq \mathcal{T}$ such that $|S| = N$ $\rightarrow$    $T_S$ := EXACT($S$);
    Select $S_m$ so that $\mathrm{cost}'(T_{S_m}, S_m) = \min_{S \subseteq \mathcal{T}}(\mathrm{cost}'(T_S, S))$;
    $T'$ := $T_{S_m}$;
    $\mathcal{T} := (\mathcal{T} - S_m) \cup \{T'\}$
od;
if $|\mathcal{T}| > 1$ $\rightarrow$ $\mathcal{T}$ := {EXACT($\mathcal{T}$)} fi;
```

The running time for N-TREE is $O(\frac{k^{N+1}}{N} O(\text{EXACT}))$, and for $|S| \leq N$ EXACT can be implemented with $O(Nn^N)$ running time. Therefore, N-TREE is $O(k^{N+1}n^N)$. Of course if we are clever we can do better, but this enough to show that N-TREE runs in polynomial time for fixed N. If we set $N = 2$, we have the 2-TREE algorithm which is a variation of MST. So, the 3-TREE algorithm is the first algorithm we considered in our search for improved worst case performance. As illustrated in Figure 2 the 3-TREE heuristic has worst case performance as bad as two. If each pair of terminal nodes adjacent with a distinct nonterminal is expanded to include $N-1$ terminal nodes, this example will then indicate that N-TREE has worst case performance as bad as two for any fixed $N \geq 2$.



Graph G



3-TREE solution

cost $= k(1+\epsilon) - 1$

Minimum Steiner tree

cost $= \frac{(k+1)(1+3\epsilon)}{2} - \epsilon$

Figure 2: 3-TREE approaches two times optimal

The problem with the N-TREE algorithm is that a number of very short edges can result in a situation in which N-TREE is essentially equivalent to MST. In effect, the $N-1$ nodes in each cluster act as a single node negating any advantage that N-TREE has over MST.

A variation N-TREE$'$ is the same as N-TREE except that instead of setting $T' := T_{S_m}$ we let T' be the tree formed by joining two elements of S_m which are connected by a shortest path in T_{S_m}. 3-TREE$'$ finds an optimal solution for the example in Figure 2, but still has worst case performance as bad as twice optimal as illustrated in Figure 3. In this example edges $i = 1, 2, \ldots, k$, in the upper part of graph G, have cost $1 + \epsilon_i$, and all edges in the lower part have cost $1+\epsilon_{k+1}$. Here $\epsilon_i = 2\epsilon_{i-1}$ for $1 < i \leq k+1$, and $0 < \epsilon_1 \ll 1/2^k$. This example can be extended to any value of $N \geq 2$ by substituting groups of N terminal nodes for each group of three terminal nodes adjacent with a distinct nonterminal node.

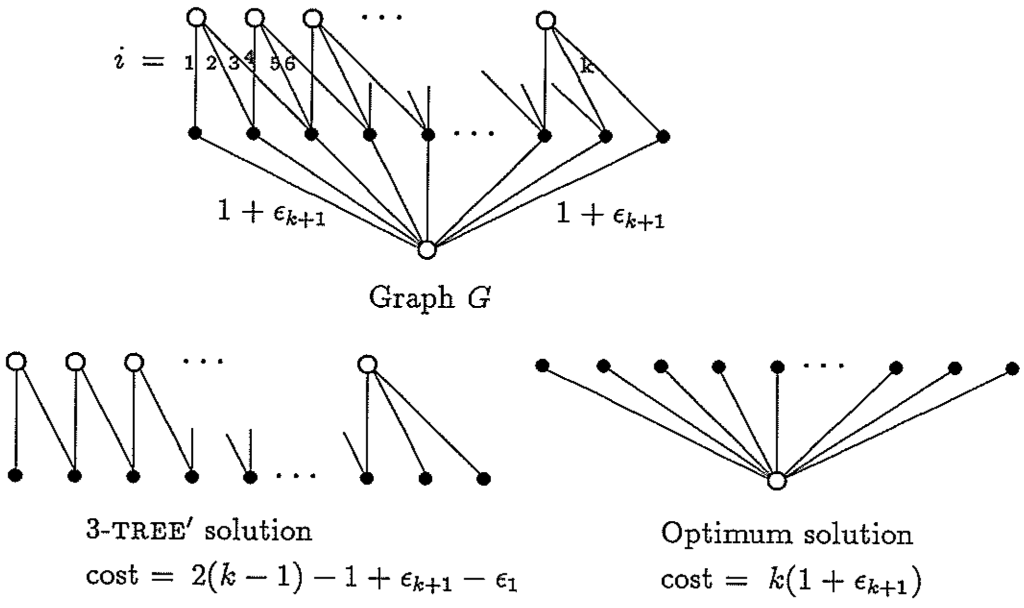


Figure 3: 3-TREE′ approaches two times optimal

One other variation of N-TREE we call N-TREE″ is similar to N-TREE′ except that we maintain a list L of paths from each tree T_{S_m} selected by N-TREE″ which connect distinct subtrees of $\mathcal{T}$. At each stage of N-TREE″ we check the cost of the minimum path in L. If one of these paths has a cost that is less than $\text{cost}(T_{S_m})/(N-1)$ we join the two subtrees connected by that path otherwise we make the same choice as is made by N-TREE′. Of course, we maintain the list by deleting any path that does not join two distinct subtrees in $\mathcal{T}$. For $N \geq 3$ we know of no example to show that the worst case performance of N-TREE″ is not bounded by a value less than two. We believe that N-TREE″ warrants further investigation as a possible candidate for an approximation algorithm with worst case performance better than two.

4.3. RS-N

In this section we define another collection of approximation algorithms, this time based on RS. The basic idea is to restrict the value of r in RS so that each decision to join a pair of subtrees is based on a collection of at least $N \geq 2$ subtrees. Informally, at each stage we evaluate the normalized cost for joining each subset of size N along with the RS function for $r \geq N$. Among all of these choose the minimum value and join two subtrees from $\mathcal{T}$ based on this choice. Next contract the resulting subtree to a single node. When $\mathcal{T}$ consists of a single tree T, expand T to create a Steiner tree. We believe that RS-N has significant potential of having worst case performance better than two.

RS-N makes use of two functions to determine which two subtrees to join at each stage of the algorithm. The function $f^{[N]}$ is very similar to that used by RS except that we require $r \geq N$. We define $f^{[N]}$ as follows

$$f^{[N]}(v) \;=\; \min_{N \leq r < |\mathcal{T}|} \left\{ 1/r \sum_{i=0}^{r} \text{dist}(v, T_i) \right\} \tag{2}$$

where each $T_i = L_v[i]$. Refer to the definition of RS for details. The second function $h : S \rightarrow \Re^+$, where $S \subseteq \mathcal{T}$ gives the cost of the tree returned by the algorithm EXACT.

$$h(S) \;=\; \frac{\text{cost}'(\text{EXACT}(S), S)}{N-1} \tag{3}$$

We are now ready to define RS-N for $N \geq 2$.

```
Create a collection of single node trees $\mathcal{T}$ consisting of the nodes in $D$;
Initialize dist and $L_v$, $\forall v \in V$ exactly as is done in RS;
do $|\mathcal{T}| \geq N$ $\rightarrow$
    Choose a vertex $v \in V - V(\mathcal{T})$ that gives a minimum value for $f^{[N]}(v)$;
    $h_{\min} := \infty$;  $S_{\min} := \emptyset$;
    do $S \subseteq \mathcal{T}$ and $|S| = N$ $\rightarrow$
        if $h(S) < h_{\min}$ $\rightarrow$
            $h_{\min} := h(S)$;    $S_{\min} := S$
        fi
    od;
    if $h_{\min} < f^{[N]}(v)$ $\rightarrow$
        Set $T_{\min} := \text{EXACT}(S_{\min})$;
        Select trees $T_0$, $T_1 \in S_{\min}$ which are closest in $T_{\min}$;
        Join $T_0$ and $T_1$ by the path in $T_{\min}$ to form $T'$
    | $f^{[N]}(v) \leq h_{\min}$ $\rightarrow$
        $T_0 := L_v[0]$; $T_1 := L_v[1]$;
        Join $T_0$ and $T_1$ with a shortest path through $v$ to form $T'$
    fi;
    $\mathcal{T} := (\mathcal{T} - \{T_0, T_1\}) \cup \{T'\}$
    $\forall v \in V$ calculate dist$(v, T')$;
    $\forall v \in V$ update $L_v$;
    $\forall T \in \mathcal{T}$ calculate dist$(T, T')$
od;
if $|\mathcal{T}| > 1$ $\rightarrow$ $\mathcal{T} := \{\text{EXACT}(\mathcal{T})\}$;
```

RS-N has running time $O(k^{N+1}n^N + n^3)$, which is polynomial for fixed N. Note that RS-2 is equivalent to RS even though RS-2 is not as efficient as RS. At this point we are not concerned with finding an efficient implementation but only with finding a polynomial time approximation algorithm which has improved worst case performance.

We note that MST and RS are guaranteed to give an optimal solution when $k = 2$ but not for any larger values. On the other hand, RS-N, N-TREE and its variants are guaranteed to give optimum performance for values of k up to and including N. Of course this does not gives tell us anything about the worst case performance, as the results for N-TREE have demonstrated.

5. If P $\neq$ NP is Two the Best Bound?

Several researchers including ourselves have tried to prove the no polynomial time approximation algorithm for the Steiner tree problem in graphs can have worst case performance better than twice optimal if P $\neq$ NP. At this time, as far as we know, no one has been able to prove such a result. In fact we suspect that there are approximation algorithms which have worst case performance better than two. We believe that RS-N has worst case performance better than two for $N > 2$. We present the following conjecture.

Conjecture 5.1 *For any instance of the Steiner tree problem* (G, D, C)*, the approximation algorithm* RS-N *produces a solution that is within* $\frac{N}{N-1}$ *times optimal for* $N \geq 2$*. Further more,* $\frac{N}{N-1}$ *is a tight bound.*

This conjecture is clearly true for $N = 2$. In addition we have not found any counter example to the conjecture for $N = 3$ and have found a class of problem instances for which the performance of RS-3 approaches 3/2. Of course this is a long way from proving the conjecture or even proving that any algorithm has worst case performance better than twice optimal.

References

[1] J.M. Jaffe. "Distributed multi-destination routing: the constraints of local information." *SIAM J. Comput*, 14(4):875–88, November 85.

[2] L. Kou, G. Markowsky, and L. Berman. "A fast algorithm for Steiner trees." *Acta Informatica*, 15:141–5, 1981.

[3] R.E. Miller and J.W. Thatcher. *Complexity of Computer Computations*, chapter Reducibility among combinatorial problems - by R.M. Karp, pages 85–103. Plenum Press, 1972.

[4] V.J. Rayward-Smith. "The computation of nearly minimal Steiner trees in graphs." *International Journal of Math. Ed. Sci. Tech.*, 14(1):15–23, 1983.

[5] B. Waxman and M. Imase. "Worst-case performance of Rayward-Smith's Steiner tree heuristic." *Information Processing Letters*, 29(6):283–287, 1989.

[6] P. Winter. "Steiner problem in networks: a survey." *Networks*, 17:129–167, 1987.