LINEAR TIME ALGORITHM FOR MINIMUM WEIGHT
STEINER TREE IN GRAPHS WITH BOUNDED
TREE-WIDTH

by

Ephraim Korach and Mir Solo

Technical Report #632

June 1990
LINEAR TIME ALGORITHM FOR MINIMUM WEIGHT STEINER TREE IN GRAPHS WITH BOUNDED TREE-WIDTH

Ephraim Korach and Nir Solel

Department of Computer Science
Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

We give a linear time algorithm for the minimum weight Steiner tree problem in graphs with tree-width bounded by $k$, a constant, given a tree decomposition with tree-width $\leq k$. The algorithm remains polynomial if $k = O(\log n / \log \log n)$.

1. INTRODUCTION.

The minimum weight Steiner tree problem (MSTP) is defined as follows: Given a graph $G=(V,E)$, a weight function $c: E \rightarrow \mathbb{Z}^+$, a subset of target vertices $T \subseteq V$, find a tree in $G$ that spans all the target vertices $T$, such that the sum of the weights of the edges in the tree is minimum.

Karp [K] proved that this problem is NP-complete (see [GI], problem ND12). The problem has numerous applications in many areas such as VLSI layout design, communication networks and others. In the literature there are exponential algorithms for the problem on general graphs (e.g. the Dreyfus-Wagner algorithm [DW]), polynomial approximation algorithms, and many algorithms for restricted versions of the problem. For a survey see [W].

The tree-width of a graph is defined as follows (RS):

A tree decomposition of a graph $G=(V,E)$ is a pair $((X_i; i \in I), T=(I,A))$, where $\{X_i; i \in I\}$ is a family of subsets of $V$ and $T=(I,A)$ is a tree, such that:

1.1) $\bigcup_{i \in I} X_i = V$.

1.2) Every edge of $G$ has both its ends in some $X_i$ ($i \in I$).

1.3) For all $i,j \in I$, if $f$ lies on the path from $i$ to $k$ in $T$, then $X_i \cap X_k \subseteq X_j$.

The tree-width of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The tree-width of $G$ is the minimum tree-width of a tree decomposition of $G$, taken over all possible tree-decompositions of $G$. 
Deciding the tree-width of a graph is NP-complete [ACP], however for a given constant \( k \) it was shown in [ACP] that deciding whether a graph \( G, \forall (G) = k \), has tree-width \( \leq k \) and constructing an appropriate tree decomposition can be done in \( O(n^{k+2}) \) time, and in [B3] a non-constructive polynomial algorithm for the problem is given, and it’s time complexity is \( O(n^3) \). Moreover, for \( k=2 \) vs \( k=3 \) the decision and construction problems can be solved in \( O(n^3) \) time (see [B3]). Some known families of graphs with bounded tree-width are: trees and forests (tree-width \( k \leq 1 \)), series-parallel graphs (\( k \leq 2 \)), grid \( d^\ast n \) (\( k \leq 2d \)), d-outerplanar graphs (\( k \geq 3d-1 \)) and many others (see e.g. [B1]). Many NP-complete problems have been shown to have polynomial algorithms when restricted to graphs with constant tree-width, and for some of them linear time algorithms are known, provided that a tree decomposition with constant tree-width is given [AP][B2][BLW]. To show that a problem can be solved by the methods in these papers may require much effort, and our method is designed especially for solving directly MSTP, and thus our algorithm remains polynomial also if the tree-width is \( O(\log n/\log \log n) \). In this paper, as in other algorithms for decomposable graphs, the algorithm is based on directing the parse tree and then repeatedly combining partial solutions by dynamic-programming methods.

In [WC] a linear time algorithm for MSTP on graphs with tree-width \( \leq 2 \) is given. Bodlaender [B2] reported that MSTP can be solved in polynomial time using his general method for graphs with constant tree-width. However, his general method has a high degree polynomial complexity, and the polynomial degree is dependent upon \( k \) - the tree-width. In this paper we solve MSTP in linear time on graphs with constant bounded tree-width, when a tree decomposition with constant tree-width is given. Thus if the tree-width of the input graph is 2 or 3 the whole algorithm (finding a tree decomposition and solving MSTP) takes linear time. Furthermore, our algorithm runs in polynomial time on graphs with tree-width \( d \), for any \( d \) that satisfies the equation \( d^{2d^2+1}(d+2)^{d+2}(d+1)! \leq n \) for some constant \( c \), where \( n = |V(G)| \). Thus if \( d = O(\log n/\log \log n) \) the algorithm is polynomial, given a graph with an appropriate tree decomposition.

2. THE ALGORITHM.

In this section we give an overview of our method for solving MSTP. In section 3 a detailed description of the main procedures of the algorithm is given.

For a graph \( G=(V,E) \) denote \( E(G) = E \), \( V(G) = V \) and for \( X \subseteq V \) let \( G(X) \) denote the subgraph of \( G \) induced by the vertices of \( X \). A separation of a graph \( G=(V,E) \) is a triport \((A,B,C)\) of subsets of \( V \), where \( A \cup B \cup C = V \), such that no edge of \( G \) joins a vertex in \( A-B \) to a vertex in \( C-B \). Let \( ST \) be a Steiner tree in \( G \) where \( TS \) is the set of target vertices. Let \((A,B,C)\) be a separation of \( G \). Denote by \( A_1, \ldots , A_n \) the trees of the forest \( F_T = ST \cap G \{A \} \) and by \( B_1, \ldots , B_j \) the trees of the forest \( F_B = ST \cap G \{B \} \).
LEMMA 1: Given a separation \((A, B, C)\) of a graph \(G = (V, E)\) and \(ST\) a Steiner tree in \(G\) where \(T_s\) is the set of target vertices, the following hold:

1. \((\cup \{V(B_i) : i = 1, \ldots, l\}) \cap A = (\cup \{V(A_i) : i = 1, \ldots, m\}) \cap B.\)
2. \(E(F_A(A \cap B)) = E(F_B(A \cap B)).\)
3. If \(F_B = \emptyset\) then either \(F_A = \emptyset\) and \(A \cap T_s = \emptyset\) or \(F_A\) is the Steiner tree \(ST\).
4. For any two vertices \(u, v \in A \cap B\), there exists a path in \(F_A[A \cap B]\) (which equals \(F_B[A \cap B]\)).
5. Let \(A_1, \ldots, A_N, B_1, \ldots, B_N \subseteq \{V(A_i), \ldots, V(B_i)\}\), \(N \geq 1\) be a collection of disjoint subsets of vertices, and similarly \(B_1, \ldots, B_N \subseteq \{B_1, \ldots, B_i\}\); then there is no set of distinct vertices \(\{v_0, \ldots, v_{2N-1}\} \subseteq V\) such that: \(v_0 \in B_1 \cap A_1, \ldots, v_{2N-1} \in B_1 \cap A_1\) (part 5).

Proof: By observing the Steiner tree one can easily see that the above properties hold. In particular property (3) holds since the Steiner tree is connected, and properties (4) and (5) hold since \(ST\) has no circuits: In (4) \(u\) and \(v\) are in the same tree in \(F_A\) and in the same tree in \(F_B\); there is a unique single path in \(ST\) connecting \(u\) and \(v\) and thus this path must be in the intersection of the two forests. In (5) note that if the condition does not hold then \(\{v_0, \ldots, v_{2N-1}\}\) are on a circuit in \(F_A \cup F_B\). 

The input to the algorithm is a connected graph \(G = (V,E)\), a target set of vertices \(T_s\), and \((\{X_i : i \in I\}, \mathbb{T} = (r, A))\) a tree decomposition of \(G\) with node width bounded by a constant. Initially choose any vertex \(r \in I\), and direct \(T\) such that \(r\) is the root. Denote \(0(i)\) as the father of \(i\) in \(T\). The algorithm proceeds as follows: For each \(i \in I\) we maintain a table \(tb(i)\) and each entry in the table defines a collection of disjoint subsets of the vertices of \(X_i\) and a collection of edges between vertices in \(X_i\). The idea behind this is the following: We assume that there is a separation \((A, B, C)\) of \(G\) and a Steiner tree \(ST\) in \(G\). Yes \(j \neq 0(i)\), \(X_j \subseteq X_i\) and \(X_j \cap X_i\). The edges in the entry of \(X_j\) (or of \(X_i\)) are the edges of \(ST\{X_j\}\) \((ST\{X_i\})\) and each subset of vertices corresponds to a tree in the forest \(ST\{B_i\}\) \((ST\{A_i\})\). Initially, the table for each \(X_i, i \in I\) is constructed according to the separation \((\mathcal{G}, X_i, V, X_i)\), therefore an entry in the table describes a forest in \(X_i\). The details of the initialization appear in the next section. Then leaves are recursively deleted from \(T\), until one vertex is left - the root of \(T\). When deleting a leaf \(i\) from \(T\) two tables \(tb(i)\) and \(tb(j)\), \(j \neq 0(i)\) are combined in order to get a new updated table for \(j\). A solution to MSTP can be found from the final table of the root, after deleting all it's children.
Let \( des(X_j) = \cup (X_k, k \cdot \text{a descendant of } i \text{ in } T \text{(including } i \text{ itself)}) \). For each \( j \in \mathcal{I} \), define a variable \( X_{ij} \), which depends only on \( i \), where \( \mathcal{I} \) is the set of the children of \( j \) in \( T \), as follows: 
\[ X_{ij} = X_j \cup (des(X_j) : i \in \mathcal{I}). \]
In the algorithm initially \( X_{ij} = X_j \), i.e. \( \mathcal{I} = \emptyset \). Then each time during the algorithm a leaf \( i \) which is a child of \( j \) in \( T \) is being deleted \( \mathcal{I} \leftarrow \mathcal{I} \cup \{i\} \); therefore \( X_{ij} \) is assigned the set of vertices in \( X_j \) together with the vertices of \( V \) corresponding to the deleted descendants of \( j \). Each Steiner tree \( F \) in \( G \) induces a forest in each \( G[X_j] \), denote \( F[X_j], F[X_j] \) restricted to \( X_j \), defines the following: 1. The edges in \( G[X_j] \) participating in \( F \). 2. Partition of \( X_j \) to the following: 2.1. Vertices that are not in \( F \). 2.2. Collection of disjoin subsets of vertices such that for every tree \( r \in F[X_j] \) corresponds one subset of vertices which includes exactly the vertices from \( X_j \) in \( r \). We keep an entry for each possible restriction of a forest \( F[X_j] \) to \( X_j \) that includes all target vertices in \( G[X_j] \), and thus the solution induced by any such entry is a candidate to be completed to a Steiner tree in \( G \). To each entry in the table we associate a cost function \( w \) which is the cost of a minimum forest \( F[X_j] \), that includes all target vertices in \( G[X_j] \), and that corresponds to the entry.

When combining two tables \( th(i) \) and \( th(j) \), \( j = \{i\} \), every entry in the new table \( th(j) \) represents a solution that can be extended to a forest \( F[X_j] \) in the new \( G[X_j] \) (which is \( G[X_j] \cup \mathcal{I} \)). \( w \) is updated accordingly. Based on lemma 1 any two entries, one from each table, are tested, and consequently may be combined to give a new entry in the new table of \( j \). In section 3 we explain which pair of entries may be combined, and describe the resulting entries in the new \( th(j) \). It may happen that some entries in the combined table are the same. In that case only one copy of the entry is kept (the rest are omitted) and the cost \( w \) corresponding to that entry would be the minimum \( w \) taken over all the identical entries. Finally a solution to MSTP can be found in the table of the root \( r \), after deleting all its children. At that stage \( G[X_j] = G \). Only entries that correspond to exactly one connected component (i.e. \( F[X_j] \) consists of a single tree) may be extended to a Steiner tree in \( G \), and therefore one such entry with the minimum cost \( w \) corresponds to a minimum Steiner tree in \( G \).

### 3. PROCEDURES OF THE ALGORITHM

In this section we give the procedures used by the algorithm. The procedures are based on lemma 1 and the fact that \((X_j, X_j, G - (X_j, \cup X_j))\) is a separation of \( G \). Let \( X_j = X_j \cup X_j \), \( \mathcal{I}(X_j) = \mathcal{I}(G[X_j]) \). Every entry in \( th(j) \) corresponds to a forest \( F[X_j] \), and the entry contains a collection of disjoint subsets of vertices, denote \( \{ST_1, \ldots, ST_{m}\} \), which are the vertices of the disjoin sets of \( F[X_j] \) restricted to \( X_j \), and also the entry contains a set of edges, denote \( E ST_i \), which are the edges of \( F[X_j] \) restricted to \( G[X_j] \). Let \( V ST = \cup \{ST_1, \ldots, ST_{m}\} \) be the set of all vertices from \( X_j \) participating in \( F[X_j] \). Let \( w \) denote the cost corresponding to
the entry (the weight of $F(\tilde{X}_i)$).

3.1. Initialization.

3.1.1. Initialization of $\text{tb}(i)$: For each $i \in I$ we initialize $\text{tb}(i)$ at the beginning of the algorithm. The initial table must include entries for all possible forests $F(\tilde{X}_i)$ in $\tilde{X}_i = X_i$ which are candidates to be extended to a minimum Steiner tree in $G$. Note that at the beginning for each $i$ $G(\tilde{X}_i) = G(X_i)$. Therefore each entry in the initial table must satisfy the following conditions:

- $T = \text{ST}(\tilde{X}_i, X_i)$; i.e all the target vertices from $X_i$ must be included in the solution.
- If $(u, v) \in \text{EST}_i$ then $u, v \in \text{VST}_i$.
- The graph $(\text{VST}_i, \text{EST}_i)$ is a forest, and the subsets $\text{ST}_{i,j}, j = 1, \ldots, n_i$ are exactly the sets of vertices in the disjoint trees in that forest.
- For each entry, the weight $w_i$ is set to be $\sum_{e \in \text{EST}_i} c(e)$.

3.1.2. First note that property (3) in lemma 1 implies the following:

OBSERVATION 2: Let $G$ be a graph given with a tree decomposition $\langle \{X_i: i \in I\}, T = \{A_i\} \rangle$. Let $F$ be a Steiner tree in $G$, with $T$ the set of target vertices and $j = \text{tb}(i)$ in the tree decomposition. Assume that $X_i \cap T \neq \emptyset$ then $F \cap X_i \neq \emptyset$ if and only if $F = F(\tilde{X}_i - X_i)$. □

Based on observation 2, we make use of the following set $\text{SPECIAL} \subseteq E(T)$:

$e = (j, k) \in \text{SPECIAL}$ if and only if $T_k \subseteq \text{des}(X_i)$ and $T_k \cap X_i = \emptyset$. In a preprocessing procedure we find the set $\text{SPECIAL}$ in linear time and it is used in the combination procedure.

3.2. Combination of two tables.

In the following we describe the procedure for combining two tables. Let $\text{tb}(i)$ be the table of $i$, $\text{tb}(j)$ be the table of $j$, where $j = \text{tb}(i)$. For every pair of entries one from $\text{tb}(i)$, and the other from $\text{tb}(j)$, check from if the conditions in 3.2.1 are met. If yes then create a new entry in the new $\text{tb}(j)$ according to 3.2.1. Otherwise, check if all the conditions in 3.2.2 are met and if so create the new entry according to the details in 3.2.2. For every new entry the cost $w_i$ of the new entry would be: $w_i + w_j - \sum_{e \in \text{EST}_i} c(e)$ where $w_i, w_j$ are the costs corresponding to the entries of $X_i$ and $X_j$ respectively prior to the combination, and $\text{EST}_{i,j} = \text{EST}_i \cap \text{EST}_j$. Note that we subtract the cost of the edges with both ends in $\text{VST}_i$ since their cost was added both in $w_i$ and in $w_j$.

3.2.1. The edge $(j, k) \in \text{SPECIAL} \subseteq \text{VST}_i \cap X_j = \emptyset$, $\text{VST}_j = \emptyset$, and the number of components in the entry in $\text{tb}(j)$ is no more than one.

The entry in the combined table would remain $\text{VST}_j = \emptyset$ and $\text{EST}_j = \emptyset$. This new entry corresponds to a possible solution to MSTP that all its vertices are in $\text{des}(X_i)$, but none in $X_j$. 

Technion - Computer Science Department - Technical Report CS0632 - 1990
3.2.2. The entry in \( b(i) \) defines a forest \( F[X_i] \) in \( G[X_i] \) restricted to the vertices of \( X_i \); similarly the entry in \( b(j) \) defines a forest \( F[X_j] \) in \( G[X_j] \) restricted to the vertices of \( X_j \).

From properties (1) and (2) in lemma 1 we have the following conditions:

(C1) \( VST_i \cap VX_j = VST_i \cap VX_j \).

(C2) \( EST_i \cap EX_j = EST_i \cap EX_j \).

Here we assume 3.1 does not hold, and therefore by lemma 1:

(C3) \( \forall i = 1 \ldots m, ST_{i,j} \cap VX_j = \emptyset. \)

As we recall each subset of vertices \( ST_{i,j} \) is connected by edges of \( G[X_j] \), and each subset of vertices \( ST_{j,m} \) is connected by edges of \( G[X_j] \) when combining the two entries we need to make sure that no circuit is formed, therefore we demand that two vertices would not be connected both by edges of \( G[X_j] \) and by edges of \( G[X_i] \) unless the edges are in both subgraphs. We also need to determine the new disjoint subsets in the created entry, and it is clear that two vertices are in the same set if they are connected either in \( G[X_i] \) or in \( G[X_j] \). To formalize the above discussion the following condition must hold: Let \( ST_{i,0} \ldots ST_{i,m} \in \{ ST_{i,0} \ldots ST_{i,m} \} \), \( N \geq 1 \) be a collection of disjoint subsets of vertices, and similarly \( ST_{j,0} \ldots ST_{j,m} \in \{ ST_{j,0} \ldots ST_{j,m} \} \).

(C4) For any two subsets \( ST_{i,j} \) and \( ST_{j,m} \), the vertices of their intersection are connected by edges in \( EST_i \cap EST_j \), and there is no set of distinct vertices \( \{ v_0, \ldots, v_{2N-1} \} \in V \) such that:

\[ v_0 \in ST_{i,j} \cap ST_{j,m}, v_{2N} \in ST_{i,j} \cap ST_{j,m} \text{ (mod } N) \text{ where } N \geq 1, r \geq 0. \]

The new entry in \( b(j) \) corresponding to the above combination of two entries is as follows: The set of edges \( EST_j \) would be the same set \( EST_j \) prior to the combination. The set of vertices \( VST_j \) would be the same set \( VST_j \) prior to the combination. The new collection of disjoint subsets is the following: Consider the collection of subsets \( \{ ST_{i,0} \ldots ST_{i,m} \} \cup \{ ST_{j,0} \ldots ST_{j,m} \} \). Repeatedly replace two subsets having non-empty intersection by their union. As a result we get a new collection of disjoint subsets, and this collection restricted to the vertices of \( X_j \) is the collection of disjoint subsets in the new entry.

3.3. Result.

The cost of a minimum solution to MSTD can be found from the final table of the root \( r \); it would be the minimum cost \( \omega \) of an entry among the following: (i) Entries that consist of one non-empty subset which corresponds to one connected component in \( G[X_j] = G_0 \). (ii) The entry that consists of an empty set of vertices which corresponds to a solution that includes none of the vertices of \( X_j \).
OBSERVATION 3: Let \( G \) be a graph with \( |X| = n \), and \( (X_0, \ldots , X_l), T=(I, A) \) a tree decomposition of \( G \) with tree-width \( \leq d \).

1. The size of \( T \) is linear in \( n \).
2. The size of each table is bounded by \( S = 2^{2(d+1)+(d+2)^{d+1}}(d+1)! \).
3. Initializing each table takes \( O(S) \) time, and combining any two tables takes \( O(S^2) \) time. These procedures are called \( O(n) \) times during the algorithm.

Thus we have the following theorem:

THEOREM 4: Let \( G \) be a graph with \( n \) vertices, with tree-width bounded by \( d \), given with a tree decomposition with tree-width \( \leq d \).

(i). If \( d \) is constant then the algorithm solves MSTP in linear time.
(ii). If \( d = O(\log n/\log \log n) \), then the algorithm solves MSTP in polynomial time.

4. CONCLUDING REMARKS.

In this paper we have considered graphs with constant bounded tree-width. Given such a graph, with an appropriate tree decomposition, we have presented a linear time nonserial dynamic programming algorithm for finding the cost of a minimum Steiner tree in the graph. The construction of the minimum Steiner tree is implicit in the algorithm. In fact it can be done easily by keeping back pointers during the algorithm. Each table was either created in the initialization procedure, or formed by combining two tables. Thus in total there are \( O(n) \) tables which may be viewed as organized in a binary tree structure induced by the process of their creation. For each non-initial entry we keep two pointers to the entries from which it was created from. Therefore we can find all the entries involved in the creation of the final entry corresponding to the minimum Steiner tree, and so we can construct the minimum Steiner tree, which is the union of all edges corresponding to these entries. The solution can be extended to a more general problem, where the vertices are also weighted. To do so we need only to remember for each solution the appropriate cost, which includes the weight of the vertices. Our method is also applicable if we allow negative edge weights. In general the complexity of the algorithm for graphs with tree-width \( \leq d \) is \( O(n + S^3) \) where \( S = 2^{2(d+1)+(d+2)^{d+1}}(d+1)! \) is a bound on the size of each table, hence the algorithm remains polynomial if \( d = O(\log n/\log \log n) \) and subexponential for \( d = O(\log n) \).

Technion - Computer Science Department - Technical Report CS0632 - 1990
REFERENCES


