Computational Geometry 47 (2014) 377–380

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

# An optimal algorithm for the Euclidean bottleneck full Steiner tree problem ☆

Ahmad Biniaz, Anil Maheshwari, Michiel Smid *

*School of Computer Science, Carleton University, Ottawa, Canada*

ARTICLE INFO

*Article history:*
Received 1 May 2013
Accepted 2 October 2013
Available online 9 October 2013
Communicated by T.M. Chan

*Keywords:*
Minimum spanning tree
Steiner tree
Bottleneck length
Yao graph

ABSTRACT

Let $P$ and $S$ be two disjoint sets of $n$ and $m$ points in the plane, respectively. We consider the problem of computing a Steiner tree whose Steiner vertices belong to $S$, in which each point of $P$ is a leaf, and whose longest edge length is minimum. We present an algorithm that computes such a tree in $O((n+m)\log m)$ time, improving the previously best result by a logarithmic factor. We also prove a matching lower bound in the algebraic computation tree model.



## 1. Introduction

Let $P$ and $S$ be two disjoint sets of $n$ and $m$ points in the plane, respectively. A *full Steiner tree* of $P$ *with respect to* $S$ is a tree $\mathcal{T}$ with vertex set $P \cup S'$, for some subset $S'$ of $S$, in which each point of $P$ is a leaf. Such a tree $\mathcal{T}$ consists of a *skeleton tree*, which is the part of $\mathcal{T}$ that spans $S'$, and *external edges*, which are the edges of $\mathcal{T}$ that are incident on the points of $P$.

The *bottleneck length* of a full Steiner tree is defined to be the Euclidean length of a longest edge. An *optimal bottleneck full Steiner tree* is a full Steiner tree whose bottleneck length is minimum. In [1], Abu-Affash shows that such an optimal tree can be computed in $O((n+m)\log^2 m)$ time. In this paper, we improve the running time by a logarithmic factor and prove a matching lower bound. That is, we prove the following result:

**Theorem 1.** *Let $P$ and $S$ be disjoint sets of $n$ and $m$ points in the plane, respectively. An optimal bottleneck full Steiner tree of $P$ with respect to $S$ can be computed in $O((n+m)\log m)$ time, which is optimal in the algebraic computation tree model.*

If $n = 2$, i.e., the set $P$ only consists of two points, say $p$ and $q$, then an optimal bottleneck full Steiner tree can be obtained in the following way: In $O(m\log m)$ time, compute a Euclidean minimum spanning tree of the set $P \cup S$ and return the path in this tree between $p$ and $q$. The correctness of this algorithm follows from basic properties of minimum spanning trees.

In the rest of this paper, we will assume that $n \geqslant 3$. This implies that any full Steiner tree of $P$ with respect to $S$ contains at least one vertex from $S$; in other words, the skeleton tree has a non-empty vertex set $S'$.

☆ Research supported by NSERC.
* Corresponding author.
*E-mail address:* michiel@scs.carleton.ca (M. Smid).

0925-7721/$ – see front matter 
http://dx.doi.org/10.1016/j.comgeo.2013.10.001

## 2. The algorithm

### 2.1. Preprocessing

We compute a Euclidean minimum spanning tree $MST(S)$ of the point set $S$, which can be done in $O(m \log m)$ time. Then we compute the bipartite graph $\Upsilon_6(P, S)$ with vertex set $P \cup S$ that is defined as follows: Consider a collection of six cones, each of angle $\pi/3$ and having its apex at the origin, that cover the plane. For each point $p$ of $P$, translate these cones such that their apices are at $p$. For each of these translated cones $C$ for which $C \cap S \neq \emptyset$, the graph $\Upsilon_6(P, S)$ contains one edge connecting $p$ to a nearest neighbor in $C \cap S$. (This is a variant of the well-known Yao-graph as introduced in [5].) Using an algorithm of Chang et al. [3], together with a point–location data structure, the graph $\Upsilon_6(P, S)$ can be constructed in $O((n+m)\log m)$ time.

The entire preprocessing algorithm takes $O((n+m)\log m)$ time.

### 2.2. A decision algorithm

Let $\lambda^*$ denote the *optimal bottleneck length*, i.e., the bottleneck length of an optimal bottleneck full Steiner tree of $P$ with respect to $S$.

In this section, we present an algorithm that decides, for any given real number $\lambda > 0$, whether $\lambda^* < \lambda$ or $\lambda^* \geqslant \lambda$. This algorithm starts by removing from $MST(S)$ all edges having length at least $\lambda$, resulting in a collection $T_1, T_2, \ldots$ of trees. The algorithm then computes the set $J$ of all indices $j$ for which the following holds: Each point $p$ of $P$ is connected by an edge of $\Upsilon_6(P, S)$ to some point $s$, such that (i) $s$ is a vertex of $T_j$ and (ii) the Euclidean distance $|ps|$ is less than $\lambda$. As we will prove later, this set $J$ has the property that it is non-empty if and only if $\lambda^* < \lambda$. The formal algorithm is given in Fig. 1.

Observe that, at any moment during the algorithm, the set $J$ has size at most six. Therefore, the running time of this algorithm is $O(n+m)$.

Before we prove the correctness of the algorithm, we introduce the following notation. Let $j$ be an arbitrary element in the output set $J$ of algorithm CompareToOptimal($\lambda$). It follows from the algorithm that, for each $i$ with $1 \leqslant i \leqslant n$, there exists a point $s_i$ in $S$ such that

- $s_i$ is a vertex of $T_j$,
- $(p_i, s_i)$ is an edge in $\Upsilon_6(P, S)$, and
- $|p_i s_i| < \lambda$.

We define $\mathcal{T}_j$ to be the full Steiner tree with skeleton tree $T_j$ and external edges $(p_i, s_i)$, $1 \leqslant i \leqslant n$. Observe that, since each edge of $T_j$ has length less than $\lambda$, the bottleneck length of $\mathcal{T}_j$ is less than $\lambda$. Therefore, we have proved the following lemma.

```
Algorithm CompareToOptimal(λ);
remove from MST(S) all edges having length at least λ;
denote the resulting trees by T_1, T_2, ...;
number the points of P arbitrarily as p_1, p_2, ..., p_n;
J := ∅;
for each edge (p_1, s) in Υ_6(P, S)
do j := index such that s is a vertex of T_j;
    if |p_1 s| < λ
    then J := J ∪ {j}
    endif
endfor;
for i := 2 to n
do for each j ∈ J
    do keep(j) := false
    endfor;
    for each edge (p_i, s) in Υ_6(P, S)
    do j := index such that s is a vertex of T_j;
        if j ∈ J and |p_i s| < λ
        then keep(j) := true
        endif
    endfor;
    J := {j ∈ J: keep(j) = true}
endfor;
return the set J
```

**Fig. 1.** This algorithm takes as input a real number $\lambda$ and returns a set $J$. This set $J$ is non-empty if and only if $\lambda^* < \lambda$.