Balanced Partition of Minimum Spanning Trees

Mattias Andersson

Joachim Gudmundsson

Christos Levcopoulos

Giri Narasimhan

institute of information and computing sciences, utrecht university

technical report UU-CS-2002-037

www.cs.uu.nl

Balanced Partition of Minimum Spanning Trees *

Mattias Andersson† Joachim Gudmundsson ‡ Christos Levcopoulos*
Giri Narasimhan§

September 3, 2002

Abstract

To better handle situations where additional resources are available to carry out a task, many problems from the manufacturing industry involve dividing a task into a constant k number of smaller tasks, while optimizing a specific objective function. In this paper we consider the problem of partitioning a given set $\mathcal{S}$ of n points in the plane into k subsets, $\mathcal{S}_1, \ldots, \mathcal{S}_k$, such that $\max_{1 \leq i \leq k} |MST(\mathcal{S}_i)|$ is minimized. Variants of this problem arise in applications from the shipbuilding industry.

We show that this problem is NP-hard, and we also present an approximation algorithm for the problem. The approximation algorithm runs in time $O(n \log n)$ and produces a partition that is within a factor $(4/3 + \varepsilon)$ of the optimal if $k = 2$, and a factor $(2 + \varepsilon)$ otherwise.

1 Introduction

Scheduling problems [9] arise in a variety of settings. In general, scheduling problems involve m jobs that must be scheduled on $k \leq m$ machines subject to certain constraints while optimizing an objective function. In parallel computation [12], often the problem is to minimize the time complexity of the parallel algorithm.

In this paper we consider a geometric version of these problems, namely given a geometric task divide it into k subtasks such that the size of the largest subtask is minimized. Such problems arise in applications from the shipbuilding industry [13]. The task is to cut out a set of prespecified regions from a sheet of metal while minimizing the completion time. Typically the size of the sheet is 10×30 meters and the number of regions that are to be cut out can vary from a few regions to several hundreds. In most cases there is only one single robot to handle this task but lately there are also examples where the number of robots is greater. In the case when there is just one robot the problem is closely related to the problem known in the literature as the *Traveling Salesperson problem with Neighborhoods* (TSPN) and

*J.G. is supported by the Swedish Foundation for International Cooperation in Research and Higher Education

†Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. E-mail: *mattias, christos@cs.lth.se*

‡Department of Computer Science, Utrecht University, PO Box 80.089, 3508 TB, Utrecht, the Netherlands. E-mail; *joachim@cs.uu.nl*

§School of Computer Science, Florida International University, Miami, FL 33199, USA. E-mail: *giri@cs.fiu.edu*

it has been extensively studied [2, 4, 5, 8, 10, 11]. The problem asks for the shortest tour that visits all the regions, and it was recently shown to be APX-hard [4]. A variant of the Euclidean TSP when k robots are available was considered by Fredrickson *et al.* [6]. They showed that by computing a TSP-tour of the given point set and then partitioning the tour into k parts a $(2 + \varepsilon - 1/k)$-approximation could be obtained in the restricted case when the k robots must start and end at the same point. The need for partitioning the input set such that the optimal substructures are balanced gives rise to many interesting theoretical problems. In this paper we consider the problem of partitioning the input so that the sizes of the minimum spanning trees of the subsets are balanced. More formally, the *k-Balanced Partition Minimum Spanning Tree problem* (k-BPMST) is stated as follows:

Problem 1 Given a set of n points $\mathcal{S}$ in the plane, partition $\mathcal{S}$ into k sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest minimum spanning tree,

$$W = \max_{1 \leq i \leq k} (|M(\mathcal{S}_i)|)$$

is minimized. Here $M(\mathcal{S}_i)$ is the minimum spanning tree of the subset $\mathcal{S}_i$ and $|M(\mathcal{S}_i)|$ is the weight of the minimum spanning tree of $\mathcal{S}_i$.

We also formulate the following problem below, the k-BPTSP problem. This is relevant since, given a c-approximation for the k-BPMST problem, we can easily achieve a $2c$-approximation for the k-BPTSP problem, by traversing the produced minimum spanning trees (MSTs).

Problem 2 Given a set of n points $\mathcal{S}$ in the plane, partition $\mathcal{S}$ into k sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest traveling salesperson tour,

$$W = \max_{1 \leq i \leq k} (|TSP(\mathcal{S}_i)|)$$

is minimized. Here $TSP(\mathcal{S}_i)$ is the minimum traveling salesperson tour of the subset $\mathcal{S}_i$ and $|TSP(\mathcal{S}_i)|$ is the weight of the minimum traveling salesperson tour of $\mathcal{S}_i$.

From the formal definitions of these problems it is clear that they can be classifed as geometric k-clustering problems. As such they are very fundamental and have possible applications in a wide variety of settings such as for example statistics, image understanding, learning theory and computer graphics.

The paper is organized as follows. We first show, in Section 2, that the k-BPMST problem is NP-hard. We then present an approximation algorithm for the problem with approximation factor $(4/3 + \varepsilon)$ for the case $k = 2$, and with approximation factor $(2 + \varepsilon)$ for the case $k \geq 3$. The algorithm runs in time $O(n \log n)$.

2 NP hardness

In this section we show that the k-Bpmst problem is NP-hard. In order to do this we need to state the recognition version of the k-Bpmst problem:

Problem 3 Given a set of n points $\mathcal{S}$ in the plane, and a real number $\mathcal{L}$, does there exist a partition of $\mathcal{S}$ into k sets $\mathcal{S}_1, \ldots, \mathcal{S}_k$ such that the weight of the largest minimum spanning tree is bounded by $\mathcal{L}$, i.e., is it true that

$$W = \max_{1 \leq i \leq k} (|M(\mathcal{S}_i)|) \leq \mathcal{L}?$$

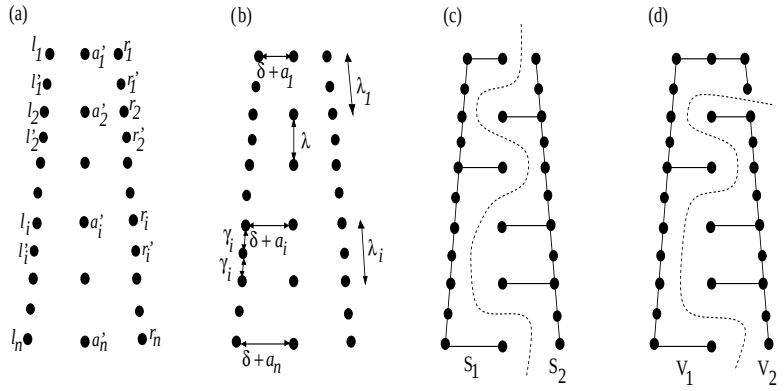

Figure 1: The set of points $\mathcal{S}$ created for the reduction. In Figure (a) all notations for the points are given. Similarly, in Figure (b) the notations for the distances between points are given. Figure (c) illustrates a class 1 partition, and (d) illustrates a class 2 partition.

Remark: The computational model used here is the real-RAM model in which we can handle square roots in polynomial time. Thus the above formulation of the problem is sufficient in order to show that the k-BPMST problem is NP-hard. Note, however, that it may be inadequate in stricter models, such as the algebraic decision tree model, where efficient handling of square roots is not possible. The computation of roots is necessary to determine the length of edges between points, which, in turn, is needed in order to calculate the weight of a minimum spanning tree. So in a stricter computational model the hardest part may not be to partition the points optimally, but instead to calculate precisely the length of the MST's. In such computational models, the combinatorial hardness of the k-BPMST problem is better captured by considering the rectilinear version of the problem. We do not discuss the rectilinear version of the problem here.

The NP-hardness proof is done by a polynomial-time reduction from the following recognition version of PARTITION.

Problem 4 PARTITION Given integers $A = \{a_1, \ldots, a_n\}$, such that $0 \leq a_1 \leq \ldots \leq a_n$, does there exist a subset $P \subseteq I = \{1, 2, \ldots, n\}$ such that

$$|P| = h = n/2 \quad \text{and} \quad \sum_{j \in P} a_j = \sum_{j \in I/P} a_j$$

The above version of PARTITION is NP-hard [7].

Lemma 2.1 *The k-BPMST problem is NP-hard.*

Proof: The reduction is done as follows. Given an instance of PARTITION, we create an instance of 2-BPMST in polynomial time, such that it is a yes-instance if and only if the PARTITION-instance is a yes-instance. Given that the PARTITION-instance contains n integers $a_1, \ldots, a_n$ in sorted order, we create the following 2-BPMST instance. A set of points $\mathcal{S}$, as shown in figure 1(a) is created, with interpoint distances as shown in figure 1(b). A closer description of these points and some additional definitions are given below:

- $A' = \{a_1', \ldots, a_n'\}$, where $a_i' = (0, i\lambda)$,

- $L = \{l_1, \ldots, l_n\}$, where $l_i = (-\delta - a_i, i\lambda)$,
- $R = \{r_1, \ldots, r_n\}$, where $r_i = (\delta + a_i, i\lambda)$,
- $L' = \{l'_1, \ldots, l'_{n-1}\}$, where l'_i is the midpoint on the line between l_i and l_{i+1}, and
- $R' = \{r'_1, \ldots, r'_{n-1}\}$, where r'_i is the midpoint on the line between r_i and r_{i+1}.

For any given partition $P = \{P[1], P[2], \ldots, P[h]\} \subseteq \{1, 2, \ldots, n\}$, we define $A_P = \{a_{P[1]}, \ldots, a_{P[n]}\}$. Furthermore, let $\lambda = 11n(a_n + n)$ and let $\delta = 7n(a_n + n)$. Note that $\lambda_i^2 \leq \lambda^2 + a_n^2$ which implies that $\lambda_i \leq 12n(a_n + n)$, which means that $\gamma_i = \lambda_i/2 \leq 6n(a_n + n)$. Finally let

$$\mathcal{L} = \frac{1}{2}\sum_{i \in I} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i.$$

Since the number of points in $\mathcal{S}$ is polynomial it is clear that this instance can be created in polynomial time. Next we consider the "if" and the "only if" parts of the proof separately.

If: If partition P exists and we have a yes-instance of PARTITION, then we will show that the corresponding 2-BPMST instance is also a yes-instance. This follows when the partition $\mathcal{S}'_1 = A_P \cup L \cup L'$, $\mathcal{S}'_2 = \mathcal{S} - \mathcal{S}_1$ (a class 1 partition, as defined below) is considered. The general appearance of $M(\mathcal{S}'_1)$ and $M(\mathcal{S}'_2)$ is determined as follows. The set of points $L \cup L'$ and the set of points $r \cup R'$ will be connected as illustrated in figure 1(c), which follows from the fact that $\gamma_i < \delta < \delta + a_1$. Next consider the remaining points A'. Any point a'_i will be connected to either l_i (in $M(\mathcal{S}'_1)$) or r_i (in $M(\mathcal{S}'_2)$), since r_i and l_i are the points located closest to a'_i (follows since $\lambda > \delta + a_n$). Thus,

$$|M(\mathcal{S}'_1)| = |M(\mathcal{S}'_2)| = \frac{1}{2}\sum_{i \in I} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i$$

and we have that the created instance is a yes-instance.

Only if: We have that P does not exist and we therefore want to show that the created 2-BPMST is a no-instance. For this we examine two classes of partitions referred to as Class 1 and Class 2 partitions.

Class 1: All partitions $\{\mathcal{V}_1, \mathcal{V}_2\}$ such that $L \cup L' \subseteq \mathcal{V}_1$ and $R \cup R' \subseteq \mathcal{V}_2$

Class 2: All other partitions $\{\mathcal{U}_1, \mathcal{U}_2\}$ not belonging to Class 1.

We start by examining class 1 – see figure 1(c). A simple examination of the edges that will be picked by Kruskal's algorithm on the entire set S shows that an optimal MST for S will contain the edges connecting the l_is and the l'_is, and also the edges connecting the r_is and the r'_is. Also, for each point $a'_i, i > 1$, it will contain an edge connecting it to either l_i or to r_i, Finally, for point a_1, it will contain edges connecting it to both l_1 and r_1. So clearly, $|M(S)| = 2 \cdot \mathcal{L} + \delta + a_1$. Its longest edge is of length $\delta + a_1$. Therefore, $|M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| \geq 2 \cdot \mathcal{L}$. Consequently, $max\{|M(\mathcal{V}_1)|, |M(\mathcal{V}_2)|\} \geq \mathcal{L}$.

Let P_1 and P_2 be the subset of points from A' that are in $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively. If $|P_1| = |P_2| = n/2$, then clearly for $j = 1, 2$, we have:

$$|M(\mathcal{V}_j)| = \sum_{i \in P_j} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i.$$

Since we have assumed that no solution to the instance of PARTITION exists, it is clear that $|M(\mathcal{V}_1)| \neq |M(\mathcal{V}_2)|$, implying that $max\{|M(\mathcal{V}'_1)|, |M(\mathcal{V}'_2)|\} > \mathcal{L}$, and that the instance of 2-BPMST is a no-instance.

If, on the other hand, $|P_1| \neq |P_2|$, then w.l.o.g., we assume that $|P_1| > n/2$. Then we know that

$$|M(\mathcal{V}_1)| \geq \delta + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i > \sum_{i \in I} a_i + \frac{n}{2} \cdot \delta + \sum_{i=1}^{n-1} \lambda_i > \mathcal{L},$$

again proving that the instance of 2-BPMST is a no-instance.

Next consider the class 2 partitions, illustrated in figure 1(d). There is always an edge of weight γ_i $(1 \leq i \leq n)$ connecting the two point sets of any such partition. This means that there can not exist a class 2 partition $\mathcal{U}_1, \mathcal{U}_2$ such that $\max\{|M(\mathcal{U}_1)|, |M(\mathcal{U}_2)|\} \leq \mathcal{L}$, because we could then build a tree with weight at most $2 \cdot \mathcal{L} + \gamma_i < |M(\mathcal{V}_1)| + |M(\mathcal{V}_2)| + \delta + a_1 = |M(\mathcal{S})|$, which is a contradiction. Thus, $\max\{|M(\mathcal{U}_1)|, |M(\mathcal{U}_2)|\} > \mathcal{L}$, which concludes this lemma. □

3 Approximating the k-BPMST

In this section a $(2+\varepsilon)$-approximation algorithm is presented. Before we present the approximation algorithm note that a straight-forward greedy algorithm that partitions $M(\mathcal{S})$ into k sets by removing the $k-1$ longest edges gives a k-approximation. The main idea of the $(2+\varepsilon)$-approximation algorithm is to partition $\mathcal{S}$ into a constant number of small components, test all valid combinations of these components and, finally, output the best combination. In order to do this efficiently, as will be seen later, one will need an efficient partitioning algorithm.

A partition of a point set $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ is said to be *valid* if $\max\{|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|\} \leq \frac{2}{3} \cdot |M(\mathcal{S})|$. The partition is denoted by the collection $\{\mathcal{S}_1, \mathcal{S}_2\}$. It is known that a valid partition always exists and can be computed efficiently [3]. For completeness we provide a detailed description of this algorithm.

3.1 ValidPartition

In this section we describe an algorithm, denoted VALIDPARTITION or VP for short; given a set of points $\mathcal{S}$, VP computes a valid partition. First the algorithm is described and then it is shown that it outputs a valid partition. We need the following definition.

Definition 3.1 *A point q is said to be a "hub" of $M(\mathcal{S})$ if and only if:*

1. *It has at least four (at most six) incident edges $e_1 = (q, v_1), \ldots e_s = (q, v_r)$ (in clockwise-order).*
2. *Every subtree of $M(\mathcal{S})$ that has q as a leaf has weight less than $1/3 \cdot M(\mathcal{S})$, see Fig. 2.*

Note that a spanning tree has at most one hub. We need the following notations. Let $M(\mathcal{T}_1), \ldots, M(\mathcal{T}_r)$ be the r maximal subtrees of $M(\mathcal{S})$, in clockwise order, that have q as a leaf (one for each incident edge of q), and let $\mathcal{T}_i$ be the set of points included in $M(\mathcal{T}_i)$.

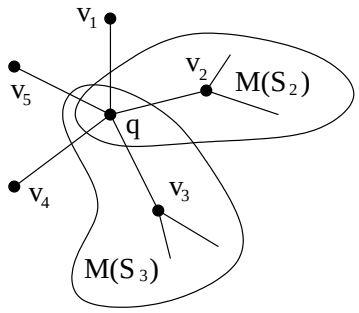

Figure 2: A possible hub q with five incident edges.

Finally, let $\mathcal{T}_i' = \mathcal{T}_i - \{q\}$. As mentioned earlier, a partition of a point set $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ is said to be valid if $\max\{|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|\} \leq 2/3 \cdot |M(\mathcal{S})|$.

Now, consider the following straight-forward algorithm, ValidPartition, for partitioning a set of points $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$. Select an arbitrary leaf ν of $M(\mathcal{S})$ as a starting point, and set $\mathcal{S}_1 = \{\nu\}$ and $\mathcal{S}_2 = \mathcal{S} - \{\nu\}$. During the whole process, vertices are added to $\mathcal{S}_1$ and deleted from $\mathcal{S}_2$, and both sets correspond to connected sets of vertices in $M(\mathcal{S})$. While $\{\mathcal{S}_1, \mathcal{S}_2\}$ is not a valid partition, expand $\mathcal{S}_1$ with points of $\mathcal{S}$ by following the tree in such a way that $M(\mathcal{S}_1) \subset M(\mathcal{S})$, $M(\mathcal{S} - \mathcal{S}_1) \subset M(\mathcal{S})$, and the weight of $M(\mathcal{S}_1)$ minimally increases in each iterative expansion. The algorithm will terminate after $O(n)$ steps. It is clear that the algorithm will either find a valid partition with $M(\mathcal{S}_1), M(\mathcal{S}_2) \subset M(\mathcal{S})$, or a "hub" of $M(\mathcal{S})$ would have been reached without finding a valid partition.

If, upon termination, $\mathcal{S}_1$ and $\mathcal{S}_2$ is not a valid partition then ValidPartition combines the $r+1$ trees $\{q, M(\mathcal{T}_1'), \ldots, M(\mathcal{T}_r')\}$ into an "optimal" partition, $\mathcal{S}_1$ and $\mathcal{S}_2$, by adding $r-1$ edges. Note that since a minimum spanning tree for points in the plane has maximum degree 6, $r+1 \leq 7$, which makes it possible for the above "optimal" partitioning of the $r+1$ trees to be done in constant time.

The following lemma can now be shown:

Lemma 3.2 *Given a set of points $\mathcal{S}$,* ValidPartition *partitions $\mathcal{S}$ into two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that (i)* $\max\{|M(\mathcal{S}_1)|, |M(\mathcal{S}_2)|\} \leq \frac{2}{3}M(\mathcal{S})$, *and (ii)* $|M(\mathcal{S}_1)| + |M(\mathcal{S}_2)| \leq |M(\mathcal{S})|$.

Proof: If a hub was not located by ValidPartition, then the lemma is clearly true. If a hub q was located, then we know that $\max\{|M(\mathcal{T}_1)|, \ldots, |M(\mathcal{T}_r)|\} < 1/3 \cdot |M(\mathcal{S})|$ and $r \geq 4$ (otherwise we would have a valid partition). Hence, it follows that there exists an $r' < r-1$ such that $M(\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_{r'})$ has weight between $1/3 \cdot |M(\mathcal{S})|$ and $2/3 \cdot |M(\mathcal{S})|$. We will have two cases (the other cases cannot occur):

$r' = 2$ **or** $r - r' = 2$**:** Assume for simplicity that $r' = 2$, then a valid partition is $\mathcal{S}_1 = \mathcal{T}_1' \cup \mathcal{T}_2'$ and $\mathcal{S}_2 = \mathcal{S} - \mathcal{S}_1 = \mathcal{T}_3 \cup \ldots \mathcal{T}_r$. We know that $|M(\mathcal{S}_2)| < 2/3 \cdot |M(\mathcal{S})|$ since $|M(\mathcal{T}_1)| + |M(\mathcal{T}_2)| \geq 1/3 \cdot |M(\mathcal{S})|$. Now, since the shortest edge connecting $M(\mathcal{T}_1')$ with $M(\mathcal{T}_2')$ is obviously shorter than $|e_1| + |e_2|$ it also holds that $|M(\mathcal{S}_1)| < 2/3 \cdot |M(\mathcal{S})|$. Note that in this case $|M(\mathcal{S}_1)| + |M(\mathcal{S}_2)| \leq |M(\mathcal{S})|$.

$r' = 3$ **and** $r = 6$**:** We have that $|M(\mathcal{T}_4 \cup \mathcal{T}_5 \cup \mathcal{T}_6)| = M(\mathcal{S}) - |M(\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3)| \leq \frac{2}{3}|M(\mathcal{S})|$. Further, it can be shown (see below) that $|M(\mathcal{T}_4' \cup \mathcal{T}_5' \cup \mathcal{T}_6')| \leq |M(\mathcal{T}_4 \cup \mathcal{T}_5 \cup \mathcal{T}_6)|$,

which means that $\mathcal{S}_1 = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$ and $\mathcal{S}_2 = \mathcal{T}_4' \cup \mathcal{T}_5' \cup \mathcal{T}_6'$ is a valid partition. Also, $|M(\mathcal{T}_4' \cup \mathcal{T}_5' \cup \mathcal{T}_6')| \leq |M(\mathcal{T}_4 \cup \mathcal{T}_5 \cup \mathcal{T}_6)|$, because if we consider the edges $e_4 = (q, v_4), e_5 = (q, v_5)$ and $e_6 = (q, v_6)$. The angle between $e_i = (q, v_i)$ and $e_{i+1} = (q, v_{i+1})$ is 60° since $M(\mathcal{S})$ is a Euclidean minimum spanning tree. It holds that $M(\mathcal{T}_4')$, $M(\mathcal{T}_5')$ and $M(\mathcal{T}_6')$ can be connected by two segments of total length less than $|e_4| + |e_5| + |e_6|$.

Thus the lemma follows. □

If a minimum spanning tree of $\mathcal{S}$ is given as input then it is easy to see that the time needed for VP to compute a valid partition is $O(n)$.

3.2 Repeated ValidPartition

One of the main ideas in the approximation algorithms (presented in Section 3.3) is to repeatedly use VP in order to create the many small components. The algorithm constructing these components is therefore called RepeatedValidPartition, or RVP for short.

RVP is described as follows: Given $M(\mathcal{S})$ and an integer m, first partitions $\mathcal{S}$ into two components using VP. Then repeatedly partition the largest component created thus far, again using VP, until m components have been created.

The following lemma describes an important property of RVP.

Lemma 3.3 *Given a minimum spanning tree of a set of points $\mathcal{S}$ and an integer m,* RVP *will partition $\mathcal{S}$ into m components $\mathcal{S}_1, \ldots, \mathcal{S}_m$ such that*

$$\max\{|M(\mathcal{S}_1)|, \ldots, |M(\mathcal{S}_m)|\} \leq \frac{2}{m}|M(\mathcal{S})|.$$

Proof: Consider the following alternative algorithm $\mathcal{A}$, which is similar to RVP. Given $M(\mathcal{S})$, algorithm $\mathcal{A}$ uses VP to divide $M(\mathcal{S})$ until all resulting components weigh at most $\frac{2}{m}|M(\mathcal{S})|$. The order in which the components are divided is arbitrary but when a component weighs at most $\frac{2}{m}|M(\mathcal{S})|$ it is not divided any further. Compare algorithm $\mathcal{A}$ with RVP, which always applies VP to the largest component and stops as soon as m components are computed. A component of weight at most $\frac{2}{m}|M(\mathcal{S})|$ will not be divided by RVP unless all other components created thus far also weigh at most $\frac{2}{m}|M(\mathcal{S})|$.

Now, if the number of resulting components of $\mathcal{A}$ is at most m then the lemma would follow. This is because RVP will first create the same components as $\mathcal{A}$ and then possibly divide these components further. Since these additional divisions are performed using VP we have that the resulting m components obviously will weigh at most $\frac{2}{m}|M(\mathcal{S})|$.

The process of $\mathcal{A}$ can be represented as a tree. In this tree each node represents a subset of $\mathcal{S}$ or a subtree of $M(\mathcal{S})$ on which VP is applied. The root represents $M(\mathcal{S})$, and the children of a node represent the components created when that node is divided using VP. We use the notation $M(v)$ to denote the subtree of $M(\mathcal{S})$ associated with node v. Note that the leaves of this tree represent the output components created by $\mathcal{A}$. We divide these leaves into two categories, where the first category consists of all leaves whose sibling is not a leaf and the second consists of all remaining leaves (that is, those whose siblings are also leaves). We let m_1 and m_2 denote the number of leaves of the first and second category correspondingly.

We start by examining the leaves of the first category. Consider any such leaf l_i, its sibling s_i, and its parent p_i. To each l_i we attach a weight $w(l_i)$ which is defined as $w(l_i) =$

$|M(p_i)| - |M(s_i)|$. Since s_i is not a leaf it holds that $|M(s_i)| > \frac{2}{m}|M(\mathcal{S})|$, and since VP was applied we know that $|M(s_i)| \leq \frac{2}{3}|M(p_i)|$. Thus $|M(p_i)| > \frac{3}{m}|M(\mathcal{S})|$, which implies that $w(l_i) \geq \frac{1}{3}|M(p_i)| > \frac{1}{m}|M(\mathcal{S})|$ and, hence, $\sum_{i=1}^{m_1} w(l_i) > m_1 \cdot \frac{1}{m}|M(\mathcal{S})|$.

Next the second category of leaves is examined. Denote any such leaf l_i' and its corresponding parent p_i'. Since there are m_2 leaves of this category and each leaf has a leaf sibling, these leaves have a total of $m_2/2$ parent nodes. Furthermore, for each of the $m_2/2$ parent nodes, the corresponding component $M(p_i')$ we have that $|M(p_i')| > \frac{2}{m}|M(\mathcal{S})|$, since they are not leaves. Thus, $\sum_{i=1}^{m_2} |M(p_i')| > \frac{m_2}{2} \cdot \frac{2}{m}|M(\mathcal{S})| = m_2 \cdot \frac{1}{m}|M(\mathcal{S})|$.

Finally, consider the total weight of the components examined. We have that $m_1 \cdot \frac{1}{m}|M(\mathcal{S})| + m_2 \cdot \frac{1}{m}|M(\mathcal{S})| < \sum_{i=1}^{m_1} w(l_i) + \sum_{i=1}^{m_2} |M(p_i')| \leq |M(\mathcal{S})|$, which implies that $m_1 + m_2 < m$. Thus, the number of leaves does not exceed m, and the lemma follows. □

3.3 The approximation algorithm

Now we are ready to present the approximation algorithm, which we will denote CA. As input we are given a set $\mathcal{S}$ of n points, an integer k and a positive real constant ε. The algorithm considers two cases: $k = 2$ and $k \geq 3$. First the case $k = 2$ is examined.

Case: $[k = 2]$

step 1: Divide $M(\mathcal{S})$ into $\frac{4}{\varepsilon'}$ components, using RVP, where $\varepsilon' = \frac{\varepsilon}{4/3+\varepsilon}$. The reason for the value of ε' will become clear below. Let w denote the weight of the heaviest component created.

step 2: Combine all components created in step 1, in all possible ways, into two groups.

step 3: For each combination tested in step 2, compute the MST for each of its two created groups.

step 4: Output the pair of MSTs with the least total weight.

Theorem 3.4 *For $k = 2$, the approximation algorithm* CA *has a time complexity of $O(n \log n)$, and produces a partition whose total weight is within a factor $(\frac{4}{3} + \varepsilon)$ of the optimal partition.*

Proof: Let $\{V_1, V_2\}$ be the partition obtained from CA. Assume that $\mathcal{S}_1$ and $\mathcal{S}_2$ is the optimal partition, and let e be the shortest edge connecting $\mathcal{S}_1$ with $\mathcal{S}_2$. In the following discussions, we assume that the weight of the output of CA is denoted by $|CA|$ and the weight of an optimal solution is denoted by $|opt|$. According to Lemma 3 it follows that $w \leq \frac{2}{4/\varepsilon'}|M(\mathcal{S})| = \frac{\varepsilon'}{2}|M(\mathcal{S})|$. We have two cases, $|e| > w$, and $|e| \leq w$, which are illustrated in figure 3(a) and 3(b), respectively. In the first case every component is a subset of either $\mathcal{S}_1$ or $\mathcal{S}_2$. This follows since a component consisting of points from both $\mathcal{S}_1$ and $\mathcal{S}_2$ must include an edge with weight greater than w. Thus, no such component can exist among the components created in step 1. Further, this means that the partition $\mathcal{S}_1$ and $\mathcal{S}_2$ must have been tested in step 2 of CA and, hence, the optimal solution must have been found.

In the second case, $|e| \leq w$, there may exist components consisting of points from both $\mathcal{S}_1$ and $\mathcal{S}_2$, see Fig. 3. To determine an upper bound of the approximation factor we start by examining an upper bound on the weight of the solution produced by CA. The dividing process in step 1 of CA starts with $M(\mathcal{S})$ being divided into two components $M(\mathcal{S}_1')$ and $M(S_2')$,

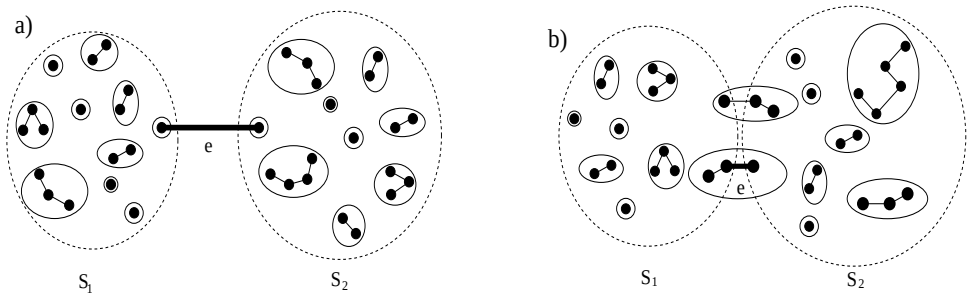

Figure 3: The two cases for CA, $k = 2$. The edge e (marked) is the shortest edge connecting $\mathcal{S}_1$ with $\mathcal{S}_2$.

such that $\max\{|M(\mathcal{S}_1')|, |M(\mathcal{S}_2')|\} \leq \frac{2}{3}|M(\mathcal{S})|$. These two components are then divided into several smaller components. This immediately reveals an upper bound for $|CA|$ of $\frac{2}{3}|M(S)|$. Next the lower bound is examined. We have:

$$|opt| \geq \frac{|M(\mathcal{S})| - |e|}{2} \geq \frac{|M(\mathcal{S})|}{2} - \frac{\varepsilon'/2 \cdot M(\mathcal{S})}{2} > (1 - \varepsilon')\frac{M(\mathcal{S})}{2}.$$

Then, if the upper and lower bounds are combined we get:

$$|CA|/|opt| \leq \frac{\frac{2}{3}|M(\mathcal{S})|}{(1-\varepsilon')\frac{M(\mathcal{S})}{2}} \leq \frac{4/3}{1-\varepsilon'} \leq 4/3 + \varepsilon.$$

In the third inequality we used the fact that $\varepsilon' \leq \frac{\varepsilon}{4/3+\varepsilon}$.

Next consider the complexity of CA. In step 1 $M(\mathcal{S})$ is divided into a constant number of components using VP. This takes $O(n)$ time, according to Lemma 3. Then, in step 2, these components are combined in all possible ways, which takes constant time since there are a constant number of components. For each tested combination there is a constant number of MST's to be computed in step 3. Further, since there are a constant number of combinations and $M(\mathcal{S})$ takes $O(n \log n)$ to compute, step 3 takes $O(n \log n)$ time. □

Next we consider the case $k \geq 3$. In this case the following steps are performed:

Case: $[k \geq 3\ ]$

step 1: Compute $M(\mathcal{S})$ and remove the $k-1$ heaviest edges $e_1, \ldots, e_{k-1}$ of $M(\mathcal{S})$, thus resulting in k separate trees $M(U_1'), \ldots, M(U_k')$.

step 2: Divide each of the trees $M(U_1'), \ldots M(U_k')$ into $\frac{4k}{\varepsilon'}$ components, using RVP. Set $\varepsilon' = \frac{\varepsilon}{2+\varepsilon}$. The reason for the value of ε' will become clear below. Denote the resulting components $M(U_1), \ldots, M(U_r)$, where $r = \frac{4k}{\varepsilon'} \cdot k$. Also, let $w = \max\{|M(U_1)|, \ldots, |M(U_r)|\}$.

step 3: Combine $U_1, \ldots, U_r$ in all possible ways into k groups.

step 4: For each such combination do:

- Compute the MST for each of its corresponding groups.

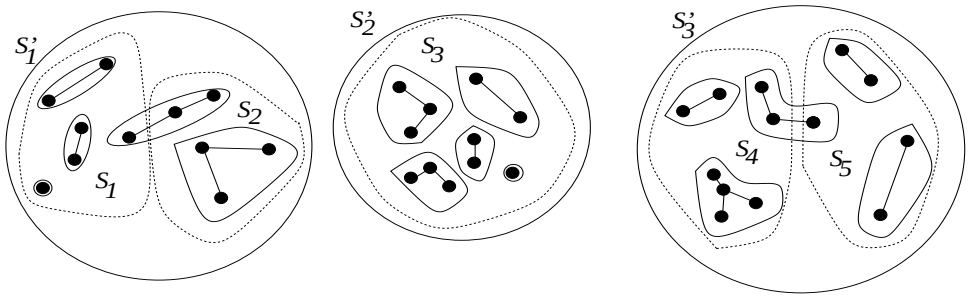

Figure 4: $\mathcal{S}_1, \ldots, \mathcal{S}_k$ is an optimal partition of $\mathcal{S}$. All subsets that can be connected by edges of length at most w are merged, thus creating the new set $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$.

- Divide each such MST in all possible ways, using RVP. That is, each MST is divided into $1, \ldots, i$, where $i \leq k$, components, such that the total number of components resulting from all the divided MST's equals k. Each such division defines a partition of $\mathcal{S}$ into k subsets.

step 5: Of all the tested partitions in step 4, output the best.

Theorem 3.5 *For $k \geq 3$ the approximation algorithm CA produces a partition which is within a factor of $(2+\varepsilon)$ of the optimal in time $O(n \log n)$.*

Proof: A constant number of components are created which means that the time complexity is the same as for the case when $k = 2$, that is $O(n \log n)$. To prove the approximation factor we first give an upper bound on the weight of the solution produced by CA and then we provide a lower bound for an optimal solution. Combining the two results will conclude the proof.

Consider an optimal partition of $\mathcal{S}$ into k subsets $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Merge all subsets that can be connected by edges of length at most w. From this we obtain the sets $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$, where $k' \leq k$ as illustrated in figure 4. Let m'_i denote the number of elements from $\mathcal{S}_1, \ldots, \mathcal{S}_k$ included in S'_i. The purpose of studying these new sets is that every component created in step 2 of CA belongs to exactly one element in $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. A direct consequence of this is that every possible partition of $\{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ into k' groups must have been tested in step 3.

Step 4 guarantees that $M(\mathcal{S}'_1), \ldots, M(\mathcal{S}'_{k'})$ will be calculated, and that these MSTs will be divided in all possible ways. Thus, a partition will be made such that each $M(\mathcal{S}'_i)$ will be divided into exactly m'_i components. This partitions $\mathcal{S}$ into k subsets $\mathcal{V}_1, \ldots, \mathcal{V}_k$. Let $\mathcal{V}$ be a set in $\mathcal{V}_1, \ldots, \mathcal{V}_k$ such that $|M(\mathcal{V})| = \max_{1 \leq i \leq k}\{|M(\mathcal{V}_i)|\}$. We wish to restrict our attention to exactly one element of the set $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$, hence we note that $\mathcal{V}$ is a subset of exactly one element $\mathcal{S}'$ in $\mathcal{S}'_1, \ldots, \mathcal{S}'_{k'}$. Assume that $M(\mathcal{V})$ was created in step 4 of the algorithm, when $M(\mathcal{S}')$ was divided into m' components using RVP, then it holds that $M(\mathcal{V}) \leq \frac{2}{m'}|M(\mathcal{S}')|$, according to Lemma 3. Since the partition $\mathcal{V}_1, \ldots, \mathcal{V}_k$ will be tested by CA we have that $|CA| \leq |M(\mathcal{V})| \leq \frac{2}{m'}|M(\mathcal{S}')|$.

Next a lower bound of an optimal solution is examined. Let $|opt'|$ be the value of an optimal solution for $\mathcal{S}'$ partitioned into m' subsets. Note that $\mathcal{S}'$ consists of m' elements from $\mathcal{S}_1, \ldots, \mathcal{S}_k$. Assume w.l.o.g. that $\#\mathcal{S}' = \#\mathcal{S}_1 + \ldots + \#\mathcal{S}_{m'}$. This means that $\mathcal{S}_1, \ldots, \mathcal{S}_{m'}$ is a possible partition of $\mathcal{S}'$ into m' subsets. Thus, $|opt| \geq \max_{1 \leq i \leq m'}\{|M(S_i)|\} = |opt'|$. Assume

w.l.o.g. that $e'_1, \ldots, e'_{m'-1}$ are the edges in $M(\mathcal{S})$ connecting the components in $\mathcal{S}'$. We have:

$$\begin{aligned} |opt| \geq |opt'| &\geq \frac{1}{m'}(|M(S')| - \sum_{i=1}^{m'-1} |e'_i|) \\ &\geq \frac{1}{m'}(|M(S')| - (m'-1)w) \end{aligned} \tag{1}$$

To obtain a useful bound we need an upper bound on w. Consider the situation after step 1 has been performed. We have $\max_{1\leq i\leq k}(|M(U'_i)|) \leq |M(\mathcal{S})| - \sum_{i=1}^{k-1}|e_i|$. Since each U'_i is divided into $\frac{4k}{\varepsilon'}$ components we have that the resulting components, and therefore also w, have weight at most $1/(\frac{2k}{\varepsilon'}) \cdot (|M(\mathcal{S})| - \sum_{i=1}^{k-1}|e_i|)$, according to Lemma 3. Using the above bound gives us:

$$\frac{w}{|opt|} \leq \frac{1/(\frac{2k}{\varepsilon'}) \cdot (|M(\mathcal{S})| - \sum_{i=1}^{k-1}|e_i|)}{\frac{1}{k}(|M(\mathcal{S})| - \sum_{i=1}^{k-1}|e_i|)} \leq \frac{\varepsilon'}{2} \quad \Longrightarrow \quad w \leq \frac{\varepsilon'}{2}|opt| \tag{2}$$

Note that $|opt| \leq |M(\mathcal{V})| \leq \frac{2}{m'}|M(\mathcal{S}')|$. Further, by combining (1) and (2) gives us:

$$|opt| \geq \frac{1}{m'}\left(|M(S')| - (m'-1)\frac{\varepsilon'}{2}|opt|\right) \geq (1-\varepsilon')\frac{|M(\mathcal{S}')|}{m'}.$$

Combining the two bounds together with the fact that $\varepsilon' \leq \varepsilon/(2+\varepsilon)$ concludes the proof of the theorem.

$$|CA|/|opt| \leq \frac{\frac{2}{m'}|M(\mathcal{S}')|}{(1-\varepsilon')\frac{|M(\mathcal{S}')|}{m'}} \leq \frac{2}{1-\varepsilon'} \leq 2+\varepsilon.$$

□

4 Conclusion and further research

In this paper it was first shown that the k-BPMST problem is NP-hard. After this was established, the next step was to design approximation algorithms for the problem. The algorithm is based on partitioning the point set into a constant number of smaller components and then trying all possible combinations of these small components. This approach revealed a $(4/3+\varepsilon)$-approximation in the case $k=2$, and a $(2+\varepsilon)$-approximation in the case $k \geq 3$. The time complexity of the algorithm is $O(n \log n)$.

For several generalizations of the k-BPMST problem it is straightforward to see that the results of this paper are immediately valid once a valid partition can be guaranteed. This is true, for example, in a higher-dimensional geometric setting. For other settings, such as metric graphs, it is obvious that we can't always guarantee a valid partition, since we have to consider non-complete graphs. In this case, however, the results of this paper are valid if we allow vertices to be included in more than one of the output subsets.

References

[1] M. Andersson. *Balanced Partition of Minimum Spanning Trees.* LUNDFD6/NFCS-5215/1–30/2001, Master thesis, Department of Computer Science, Lund University, Sweden, 2001.

[2] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55:197–218, 1994.

[3] R. I. Becker, S. R. Schach and Y. Perl. A shifting algorithm for min-max tree partitioning. *Journal of the Association for Computing Machinery*, 29(1):58–67, January 1982.

[4] M. de Berg, J. Gudmundsson, M. Katz, C. Levcopoulos, M. Overmars and F. van der Stappen. *Constant factor approximation algorithms for TSPN with fat objects.* To appear in Proc. of ESA'02, 2002.

[5] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[6] G. N. Fredrickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7, 1978.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[8] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for TSP with neighborhoods. *Nordic Journal of Computing*, 6:469–488, 1999.

[9] D. Karger, C. Stein and J. Wein. *Scheduling algorithms.* In Handbook on Algorithms and Theory of Computation, edited by M. J. Atallah. CRC Press, 1998.

[10] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. *Proc. 11th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1995.

[11] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[12] J. H. Reif and S. Sen. *Parallel computational geometry: An approach using randomization* In Handbook of Computational Geometry, edited by J.-R. Sack and J. Urrutia. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[13] B. Shaleooi. *Algoritmer för plåtskärning (*Eng. transl. *Algorithms for cutting sheets of metal).* LUNDFD6/NFCS-5189/1–44/2001, Master thesis, Department of Computer Science, Lund University, Sweden, 2001.