ELSEVIER Discrete Applied Mathematics 87 (1998) 117–137

DISCRETE APPLIED MATHEMATICS

Approximation algorithms for minimum tree partition

Nili Guttmann-Beck, Refael Hassin *

Department of Statistics and Operations Research, Tel Aviv University, Tel Aviv 69978, Israel

Received 29 October 1996; received in revised form 17 November 1997; accepted 6 April 1998

Abstract

We consider a problem of locating communication centers. In this problem, it is required to partition the set of n customers into subsets minimizing the length of nets required to connect all the customers to the communication centers. Suppose that communication centers are to be placed in p of the customers locations. The number of customers each center supports is also given. The problem remains to divide a graph into sets of the given sizes, keeping the sum of the spanning trees minimal. The problem is NP-complete, and no polynomial algorithm with bounded error ratio can be given, unless P = NP. We present an approximation algorithm for the problem assuming that the edge lengths satisfy the triangle inequality. It runs in $O(p^2 4^p + n^2)$ time ($n = |V|$) and comes within a factor of $2p-1$ of optimal. When the sets' sizes are all equal this algorithm runs in $O(n^2)$ time. Next, an improved algorithm is presented which obtains as an input a positive integer x ($x \leqslant n-p$) and runs in $O(f(p,x)n^2)$ time, where f is an exponential function of p and x, and comes within a factor of $2+(2p-3)/x$ of optimal. When the sets' sizes are all equal it runs in $O(2^{(p+x)}n^2)$ time. A special algorithm is presented for the case $p=2$. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Graph partitioning; Minimum spanning tree; Approximation algorithms

1. Introduction

Let $G=(V,E)$ be a complete undirected graph, with a node set V and an edge set E. The edges $e \in E$ have lengths $l(e)$ that satisfy the triangle inequality. We assume that each vertex represents a customer. The goal is to partition V into p subsets of given sizes, in order to locate a communication center in one node of each subset. The nodes of each subset will then be connected to this server through a subnetwork of minimum total length, that is, a minimum spanning tree (MST) of the subgraph induced by this subset of nodes. The *Minimum Tree Partition Problem* is to compute a partition with minimum total length.

More formally: Given $G=(V,E)$ with $|V|=n$, and p positive integers $\{k_i\}_{i=1}^{p}$ such that $\sum_{i=1}^{p} k_i = n$. *The Minimum Tree Partition Problem* is to find a partition of V into disjoint sets $\{P_i\}_{i=1}^{p}$ such that $\forall i \in \{1,\ldots,p\}$ $|P_i| = k_i$, and $\sum_{i=1}^{p} l(MST(P_i))$ is

* Corresponding author. E-mail: {nili,hassin}@math.tau.ac.il.

minimized, where $MST(P(i))$ is a minimum spanning tree in the graph induced on P_i and $l(E') = \sum_{e \in E'} l(e)$ for $E' \subseteq E$.

The problem is NP-hard [6]. In this paper we introduce approximation algorithms with bounded error ratio. First, we describe a general algorithm for dividing the graph into p sets of customers. It runs in $O(p^2 4^p + n^2)$ time where $n = |V|$, and comes within a factor of $2p-1$ of optimal. When the sizes of the sets are all equal it runs in $O(n^2)$ time. Next, we describe an algorithmic scheme, for any given value of a parameter $x \in \{1, 2, \ldots, n-p+1\}$ this algorithm runs in $O(f(p,x)n^2)$ time, where f is an exponential function of p and x, and comes within a factor of $2 + (2p-3)/x$ of optimal. When the sizes of the sets are all equal this algorithm runs in $O(2^{(p+x)}n^2)$ time.

For the case $p = 2$ we present an $O(n^2)$ time algorithm that comes within a factor of 2 of optimal. For dividing the graph into 2 equal-sized sets we prove that the optimal solution value is bounded by $3l(MST(G))/2$. For approximating the solution to this problem we define a 'K-centroid' which generalizes the concept of a centroid of a tree and prove its existence.

For small values of p these algorithms improve previously best-known performance of $4(1 - p/n)$ for partitioning the graph into p equal-sized sets, given by Goemans and Williamson in [4] (see also, [2, 10]). Our algorithms also improve for small p values the time requirement (from $O(n^2\sqrt{\log \log n})$) and generalize the problem allowing the sets to be of different sizes.

For a given $S \subset V$ consider all the partitions of S into p sets. For each partition compute the sum of lengths of the MSTs over the subgraphs induced by the partition. Denote $z(S)$ the minimum sum obtained over all the possible partitions. Our problem is to approximate $z(V)$, while Chandra and Halldórsson [1] present a 4-approximation algorithm for the problem of maximizing $z(S)$ over all subsets $S \subset V$, $|S| = k$, where k is given.

Imielińska et al. [7] and Goemans and Williamson [5] present polynomial algorithms with bounded performance guarantees for the following problem (without the triangle inequality assumption): Given $m \in \{2, \ldots, n\}$, find a minimum length spanning forest such that each of its trees spans **at least** m vertices. This is different from our problem in which the trees' **exact** sizes are given and for which it has been shown in [6] that no such approximation can be given unless P = NP.

Approximation algorithm with bounded performance guarantees for the related min-max tree partition problem in which the goal is to partition the node set into p sets of equal size $P_1, \ldots, P_p$ minimizing $\max_{i \in \{1,\ldots,p\}} l(MST(P_i))$, are described in [6].

Our algorithms can also be used to approximate the problem of covering the graph by cycles, (by doubling all the trees and using the triangle inequality to replace each tree by a cycle whose size is at most twice the size of the tree). The resulting error bound is twice the corresponding bound for the tree partition problem.

We will use the following notations: For an edge e, $l(e)$ is the length of e. For a set of edges E, $l(E) = \sum_{e \in E} l(e)$. For a graph $G = (V, E)$, $l(G) = l(E)$. For a set of nodes V', $MST(V')$ is a MST on the subgraph induced by V'. For a subgraph B we denote by V_B and E_B the sets of nodes and edges in B, respectively. We denote by *opt* the optimal solution value of the problem.

2. First approximation

2.1. The cycle procedure

Consider Cycle_Part given in Fig. 1. This procedure takes a *MST* on the given graph and doubles its edges, getting an Eulerian cycle. This cycle is changed into a simple one of shorter or equal length using the triangle inequality. Then we divide the nodes in the graph according to the order by which they appear in the cycle. Starting by removing the longest edge, then taking the first k_1 nodes into the first set, the next k_2 nodes to the second set, etc.

Lemma 2.1. *Let $\acute{e}$ be the longest edge in an MST T of G. Then the value r returned by Cycle_Part satisfies $r \leqslant 2l(T) - l(\acute{e})$.*

Proof. If $p=1$, $r=l(T)\leqslant 2l(T)-l(\acute{e})$.

Suppose $p>1$. For $i\in\{1,\dots,p\}$, if $k_i\geqslant 2$ then $\{(v_{m_i},v_{m_i+1}),\dots,(v_{m_i+k_i-2},v_{m_i+k_i-1})\}$ is a spanning tree of P_i. Thus,

$$l(MST(P_i))\leqslant \sum_{j=m_i}^{m_i+k_i-2} l(v_j,v_{j+1}) \quad \forall i\in\{1,\dots,p\}.$$

Note that when $k_i=1$ $l(MST(P_i))=0$ and the sum is also 0. It follows that

$$r=\sum_{i=1}^{p} l(MST(P_i))\leqslant \sum_{i=1}^{p}\sum_{j=m_i}^{m_i+k_i-2} l(v_j,v_{j+1})\leqslant l(C)-l(v_n,v_1).$$

Since (v_n,v_1) is the longest edge in C, $l(v_n,v_1)\geqslant l(\acute{e})$. (The longest edge in the MST appears in the cycle which was created by doubling the edges, when changing the cycle into a simple one this edge is either untouched, or is changed into a longer edge.) So the cycle contains an edge of length $\geqslant l(\acute{e})$.) Hence, $r\leqslant l(C)-l(\acute{e})\leqslant 2l(T)-l(\acute{e})$. □

To see that Cycle_Part may produce a bad approximation consider the graph with $V=\{v_1,v_2,v_3\}$, $l(v_1,v_2)=l(v_1,v_3)=1$ and $l(v_2,v_3)=0$. The desired sizes for partitioning are: $\{2,1\}$. A MST for this graph consists of the edges $(v_1,v_2),(v_2,v_3)$. The resulting simple cycle is $(v_1-v_2-v_3-v_1)$ with the numbering of the nodes giving that (v_3,v_1) is a longest edge of the cycle. In this case $P_1=\{v_1,v_2\},P_2=\{v_3\}$. giving $r=1$, while $opt=0$.

```
Cycle_Part
  input
  1. A graph $G=(V,E)$, $|V|=n$ and a MST $T$.
  2. A set of positive integers $\{k_1,\dots,k_p\}$ satisfying $\sum_{i=1}^{p} k_i=n$.
  returns
  1. $\{P_i\}_{i=1}^{p}$ where $\bigcup_{i=1}^{p} P_i=V$ and $|P_i|=k_i\ \forall i\in\{1,\dots,p\}$.
  2. A value $r$ satisfying $r=\sum_{i=1}^{p} l(MST(P_i))$.
  begin
  if $(p=1)$
      then
        $P_1:=V$.
        $r:=l(T)$.
        return $(\{P_1\},r)$
      end if
  Double all the edges in $E_T$. A cycle $C=(V,E_c)$ is created.
  Change it into a simple cycle of equal or smaller size,
  using the triangle inequality.
  Number the nodes in $V$ so that $E_c=\{(v_1,v_2),(v_2,v_3),\dots,(v_{n-1},v_n),(v_n,v_1)\}$,
  where $(v_n,v_1)$ is the longest edge in $C$.
  $m_1:=1$
  $m_i:=m_{i-1}+k_{i-1},\ i=2,\dots,p$.
  $P_i:=\{v_{m_i},\dots,v_{m_i+k_i-1}\},\ i=1,\dots,p$
  $r:=\sum_{i=1}^{p} l(MST(P_i))$.
  return $(\{P_i\}_{i=1}^{p},r)$
  end Cycle_Part
```

Fig. 1. The cycle partitioning routine.

2.2. *Algorithm Part_Alg*

To partition G into p sets of sizes $\{k_1,\dots,k_p\}$ call Part_Alg(G, $\{k_1,\dots,k_p\}$), where Part_Alg is defined in Fig. 2. This algorithm uses the previously defined Cycle_Part.

Step 2 of Part_Alg removes, during its jth application, a set of j longest edges from a MST of G, creating $j+1$ components. It then checks whether a partition of the components into subsets of sizes $k_1,\dots,k_p$ can be obtained. The step is repeated as long as such a partition exists. Finally, Step 3 applies Cycle_Part to each component obtained in the last iteration.

Lemma 2.2. *Let y be the value of e_cou when Part_Alg reaches Step* 3, *let $\{g_1,\dots,g_{y-1}\}$ be the $y-1$ longest edges in T, where g_1 is the longest of them. Let apx be the value returned by Part_Alg($G,\{k_1,\dots,k_p\}$). Then,*

$$apx\leqslant 2l(T)-l(g_1),$$

```
Part_Alg
  input
  1. A graph $G=(V,E)$, $|V|=n$
  2. A set of positive integers $\{k_1,\dots,k_p\}$ satisfying $\sum_{i=1}^{p} k_i = n$.
  returns
  1. $\{P_i\}_{i=1}^{p}$ where $\bigcup_{i=1}^{p} P_i = V$ and $|P_i| = k_i$.
  2. A value apx satisfying $apx = \sum_{i=1}^{p} l(MST(P_i))$.
  begin
  Step 1
        $T := MST(G)$.
        $PT := \{(T, \{k_1,\dots,k_p\})\}$.
        end Step 1
  Step 2
        $done := 0$. $e\_cou := 1$. ($e\_cou$ for edge-count).
        while ($done = 0$)
          Remove the $e\_cou$ longest edges in $E_T$.
          A set of connected components $\{T_i\}_{i=1}^{e\_cou+1}$ is created.
          ($T_i$ is a spanning tree of $V_{T_i}$).
          Compute a partition of $\{k_1,\dots,k_p\}$ into $e\_cou+1$ sets
          $\{K_1,\dots,K_{e\_cou+1}\}$, where $\sum_{k_j \in K_i} k_j = |V_{T_i}|$.
          if (a partitioning $\{K_1,\dots,K_{e\_cou+1}\}$ is found)
            then
                 $PT := \{(T_i, K_i)\ i = 1,\dots,e\_cou-1\}$.
                 $e\_cou := e\_cou + 1$.
                 if ($e\_cou = p$)
                   then $done := 1$.
                   end if
            else
                 $done := 1$.
            end if
          end while
        end Step 2
  Step 3 for every ($i = 1,\dots,e\_cou$)
            Call Cycle_Part($T_i, K_i$) where:
            $\{P_i^1,\dots,P_i^{|K_i|}\}$ is the returned partition,
            $r_i$ is the returned value.
            end for
        return ($\{P_1^1,\dots,P_1^{|K_1|},\dots,P_{e\_cou}^1,\dots,P_{e\_cou}^{|K_{e\_cou}|}\}$, $apx := \sum_{i=1}^{e\_cou} r_i$.)
        end Step 3
  end Part_Alg
```

Fig. 2. The partitioning algorithm.

and if $y>1$,

$$apx \leqslant 2\left(l(T) - \sum_{i=1}^{y-1} l(g_i) \right).$$

Proof. When entering Step 3 the set PT satisfies $PT=\{(T_1,K_1),\dots,(T_y,K_y)\}$, where the T_is are the connected components created from T when removing the edges $\{g_1,\dots,g_{y-1}\}$ from it, (if $y=1$ then $T_1=T$). Suppose that $y=1$. Activating the cycle routine on T according to Lemma 2.1 gives a value $r\leqslant 2l(T)-l(g_1)$. Hence for this special case $apx=r\leqslant 2l(T)-l(g_1)$.

Suppose now that $y>1$. From the way the T_is were obtained,

$$\sum_{i=1}^{y} l(T_i) = l(T) - \sum_{i=1}^{y-1} l(g_i). \tag{1}$$

For every $i\in\{1,\dots,y\}$, by Lemma 2.1, $r_i\leqslant 2l(T_i)$, and with Eq. (1) we get

$$apx=\sum_{i=1}^{y} r_i \leqslant \sum_{i=1}^{y} 2l(T_i) \leqslant 2\left(l(T) - \sum_{i=1}^{y-1} l(g_i) \right) \leqslant 2l(T)-l(g_1). \qquad \square$$

Let $\{O_i\}_{i=1}^{p}$ be an optimal partition, and denote the set of edges of $MST(O_i)$ as E_{O_i}, $i\in\{1,\dots,p\}$. For every $i\neq j$ $\{i,j\}\subset\{1,\dots,p\}$, define $e_{(i,j)}$ to be an edge satisfying

$$l(e_{(i,j)}) = \min_{v\in O_i, u\in O_j} \{l(v,u)\}.$$

Consider the graph G_0 in which nodes represent the sets O_i, and the length of the edge between the node representing O_i and the node representing O_j is $l(e_{(i,j)})$.

Define $\{e_\alpha^*\}_{\alpha=1}^{p-1}$ to be the $p-1$ edges of a MST in G_0. Rename the edges so: $l(e_1^*)\leqslant l(e_2^*)\leqslant l(e_3^*)\leqslant \cdots \leqslant l(e_{p-1}^*)$.

The set of edges $\bigcup_{i=1}^{p} E_{O_i} \cup \{e_1^*,\dots,e_j^*\}$ defines a set of $p-j$ connected components. Let $\{U_1^j,\dots,U_{p-j}^j\}$ be the sets of nodes in these components.

Lemma 2.3. *The shortest edge between U_i^j and U_k^j for $i\neq k$, $\{i,k\}\subset\{1,\dots,p-j\}$ is of length $\geqslant l(e_{j+1}^*)$.*

Proof. The set of edges $\{e_1^*,\dots,e_{p-1}^*\}$ is a MST in G_0. Suppose there is an edge g between U_i^j and U_k^j, such that $l(g)<l(e_{j+1}^*)$. Add the corresponding edge in G_0, $\hat{g}$, to $\{e_1^*,\dots,e_{p-1}^*\}$. A cycle has been created. This cycle contains at least one edge, $\hat{f}$, from $\{e_{j+1}^*,\dots,e_{p-1}^*\}$ (since $\{e_1^*,\dots,e_j^*\}$ are all edges inside the U_i^j sets). Then, $l(\hat{f})\geqslant l(e_{j+1}^*)$ and $\{e_1^*,\dots,e_{p-1}^*\}\setminus\{\hat{f}\}\cup\{\hat{g}\}$ is a strictly shorter spanning tree then $\{e_1^*,\dots,e_{p-1}^*\}$, contradicting the fact that the latter is a MST. $\square$

Theorem 2.4 (Gale [3], see also Lawler [9]). *Let $T_1=(V,H)$ be a MST of $G=(V,E)$, and let $T_2=(V,F)$ be any spanning tree of G. Suppose that $H=\{h_1,h_2,\dots,h_{n-1}\}$ is ordered so that $l(h_1)\leqslant\cdots\leqslant l(h_{n-1})$, and $F=\{f_1,f_2,\dots,f_{n-1}\}$ is ordered so that $l(f_1)\leqslant\cdots\leqslant l(f_{n-1})$. Then, $l(h_i)\leqslant l(f_i)$ $\forall i\in\{1,\dots,n-1\}$.*

Theorem 2.5. *$apx\leqslant(2p-1)opt$.*

Proof. When adding $\{e_1^*,\dots,e_{p-1}^*\}$ to $\bigcup_{i=1}^{p} E_{O_i}$ a spanning tree of G is created. Hence

$$l(T)\leqslant opt+\sum_{i=1}^{p-1} l(e_i^*). \tag{2}$$

Since T is a spanning tree it must contain at least $p-1$ edges between the sets of the optimal solution. Let the number of these edges in T be z. Consider a graph that contains p nodes (same nodes as in G_0), corresponding to $O_1,\dots,O_p$. Let the edges in this graph correspond to the z edges of T mentioned above, where such an edge connects a node corresponding to O_i to the node corresponding to O_j if the original edge connected in T a node from O_i with a node from O_j. Look at $p-1$ edges that create a spanning tree in this graph. Let these edges be $f_1,\dots,f_{p-1}$, $l(f_1)\leqslant l(f_2)\leqslant\cdots\leqslant l(f_{p-1})$. These edges satisfy that $\{f_1,f_2,\dots,f_{p-1}\}\subset E_T$ and that (by Theorem 2.4)

$$l(e_i^*)\leqslant l(f_i)\quad \forall i\in\{1,\dots,p-1\}. \tag{3}$$

We consider three cases:

1. $opt<l(e_1^*)$. The set of edges $\bigcup_{i=1}^{p-1} E_{O_i}\cup\{e_1^*,\dots,e_{p-1}^*\}$ is a spanning tree with at most $p-1$ edges of length $\geqslant l(e_1^*)$. Hence, by Theorem 2.4, T contains at most $p-1$ edges of length $\geqslant l(e_1^*)$. Removing from T its $p-1$ longest edges leaves p components with all of their edges inside the optimal solution's sets of nodes. (Because the shortest edge between two nodes from two different O_is has to be at least of length $l(e_1^*)$.) Therefore, these components are exactly the optimal solution, the value $e\_cou=p$ will be reached, and $apx=opt$.
2. $l(e_j^*)\leqslant opt<l(e_{j+1}^*)$ for some $j\in\{1,\dots,p-2\}$. We will show that in this case y, the value of e_cou when Step 3 is reached, satisfies $y\geqslant p-j$. The set of edges $\bigcup_{i=1}^{p-1} E_{O_i}\cup\{e_1^*,\dots,e_{p-1}^*\}$ is a spanning tree with at most $p-j-1$ edges of length $\geqslant l(e_{j+1}^*)$. Then, according to Theorem 2.4, T contains at most $p-j-1$ edges of this length. Removing from T its $p-j-1$ longest edges, will leave only edges of length $<l(e_{j+1}^*)$. Look at $\{U_1^j,\dots,U_{p-j}^j\}$ defined before. By Lemma 2.3, the shortest edge between U_i^j and U_k^j for every i,k is at least as long as $l(e_{j+1}^*)$. Thus after the removal of the $p-j-1$ longest edges from T there are no edges left between nodes from different U_i^js. T is disconnected into $p-j$ connected components, giving that this partitioning has to be $\{U_1^j,\dots,U_{p-j}^j\}$. So, $y\geqslant p-j$.

From $j \leqslant p-2$ it follows that $y \geqslant 2$. We now use Lemma 2.2, the fact that the f_i edges are in T, and Eq. (3),

$$\begin{aligned}
apx &\leqslant 2\left(l(T) - \sum_{i=1}^{y-1} l(g_i)\right) \leqslant 2\left(l(T) - \sum_{i=1}^{p-j-1} l(g_i)\right)\\
&\leqslant 2\left(l(T) - \sum_{i=1}^{p-j-1} l(f_{p-i})\right)\\
&\leqslant 2\left(l(T) - \sum_{i=1}^{p-j-1} l(e^*_{p-i})\right) = 2\left(l(T) - \sum_{i=1}^{p-1} l(e^*_i)\right) + 2\sum_{i=1}^{j} l(e^*_i).
\end{aligned}$$

By Eq. (2) and the assumption of this case, and since $j < p-2$

$$apx \leqslant 2opt + 2\sum_{i=1}^{j} l(e^*_i) \leqslant 2opt + 2jl(e^*_j) \leqslant 2opt + 2j\, opt \leqslant (2p-2)opt.$$

3. $l(e^*_{p-1}) \leqslant opt$. By Lemma 2.2 and since f_{p-1} is in T, $apx \leqslant 2l(T) - l(g_1) \leqslant 2l(T) - l(f_{p-1})$. By Eqs. (3) and (2), and the assumption of this case,

$$\begin{aligned}
apx &\leqslant 2l(T) - l(e^*_{p-1})\\
&= 2\left(l(T) - \sum_{i=1}^{p-1} l(e^*_i)\right) + 2\sum_{i=1}^{p-2} l(e^*_i) + l(e^*_{p-1})\\
&\leqslant 2\left(l(T) - \sum_{i=1}^{p-1} l(e^*_i)\right) + (2p-3)l(e^*_{p-1})\\
&\leqslant 2opt + (2p-3)l(e^*_{p-1})\\
&\leqslant (2p-1)opt. \qquad \square
\end{aligned}$$

2.3. *Tight example for Part_Alg*

Consider the graph with $p(p+1)$ nodes in Fig. 3(a). There are $p+1$ sets of nodes in this graph. $p-1$ of these sets contain $p+1$ nodes each. There is one more set of nodes containing p nodes and one more set containing only one node. The edges inside each one of these sets are of length 0. Edges between two sets have length 1. The objective is to divide the nodes into p sets of $p+1$ nodes each.

A MST is shown in Fig. 3(b). Step 2 tries to remove a longest edge. Let the chosen edge be $\hat{e}$ shown in the figure. There is no partitioning of $\{k_1, \ldots, k_p\}$ into sets of sizes $\{1, p(p+1)-1\}$ so $e\_cou = 1$ when Step 3 is reached.

The cycle routine is activated for the MST. The simple cycle achieved is shown in Fig. 3(c) and the resulting partitioning is shown in Fig. 3(d), giving $apx = 2p-1$. An optimal solution consists of using the original $p+1$-nodes sets as sets in the partition,

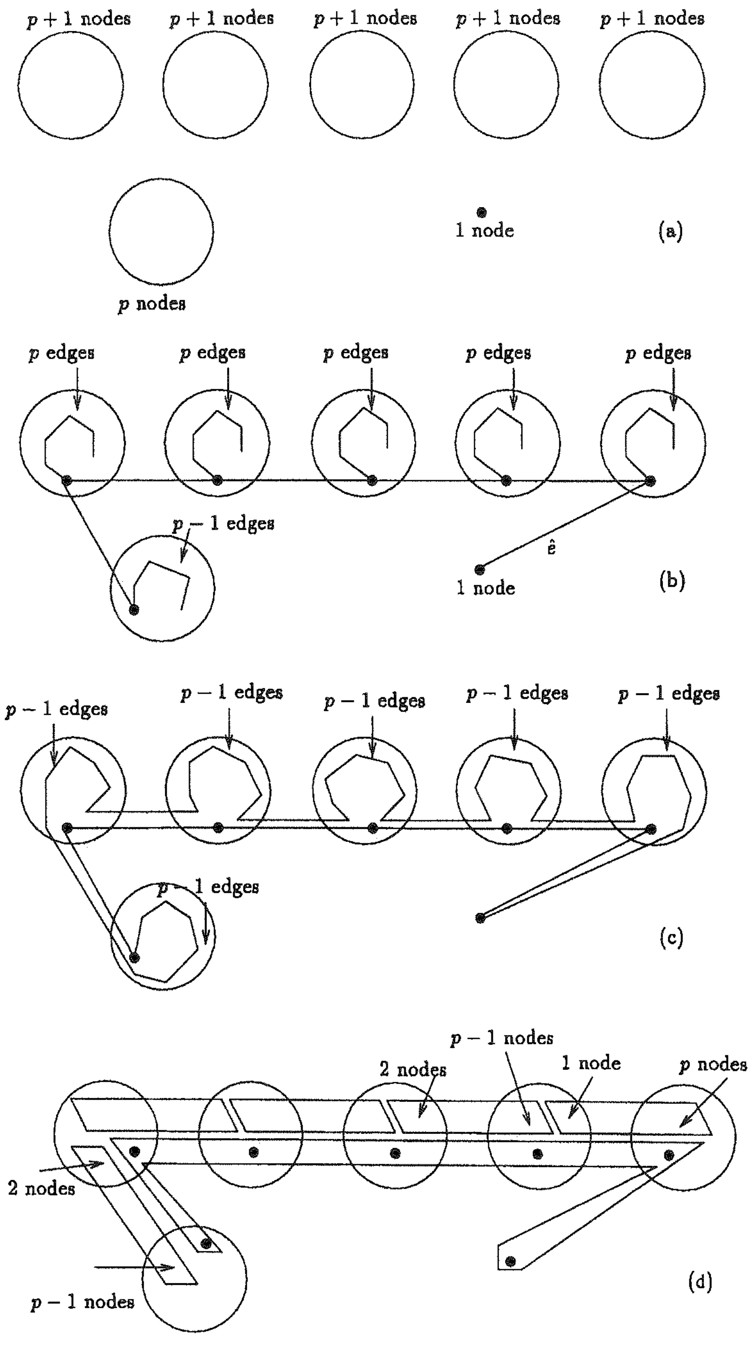

Fig. 3. A tight example for p partitioning.

and putting the original p nodes set together with the single node set, giving $opt = 1$. So, $apx = (2p-1)opt$.

2.4. Complexity

We now analyze the complexity of Part_Alg.

Step 1 takes $O(n^2)$.

Step 2 implements a loop which is activated at most p times. In each iteration the tree is scanned to find the next longest edge and the sizes of connected components when removing this edge. Next, a partitioning of $\{k_1,\dots,k_p\}$ is searched (only one is needed). Finding such a partitioning takes $O(4^p)$. To find whether there is a partitioning of $\{k_1,\dots,k_p\}$ into e_cou+1 sets of sizes $\{a_1,\dots,a_{e\_cou+1}\}$ define the next dynamic search:

$f_i(S)$ is defined for every $i \in \{1,\dots,e\_cou+1\}$ and every $S \subset \{k_1,\dots,k_p\}$. $f_i(S)$ receives the value *true* if there is a partitioning of S into i sets of sizes $\{a_1,\dots,a_i\}$.

$f_{i+1}(S) := true$ iff there is $T \subset S$ such that $f_i(S \setminus T) = true$ and $\sum_{k_j \in T} k_j = a_{i+1}$.

$f_1(S) := true$ if $\sum_{k_j \in S} k_j = a_1$. There are 2^p possible values of S and $e\_cou + 1 \leqslant p$ values of i. Hence there are $O(p2^p)$ values to compute. Every value takes $O(2^p)$ time to calculate (there are $O(2^p)$ possible subsets T). So each iteration takes $O(p4^p)$ time.

Thus Step 2 requires $O(p(n+p4^p))$ altogether. When $k_i = n/p\ \forall i \in \{1,\dots,p\}$ this step only requires to find the longest edge at each iteration and to compute the components' sizes, thus requiring only $O(pn)$.

Step 3 calls Cycle_Part for every pair in PT. This takes $O(|V_{T_i}|^2)$. So all the calls for this procedure take $O(n^2)$. Calculating r will take additional $O(p)$. Thus Step 3 takes altogether $O(n^2)$.

Altogether the algorithm takes $O(p(n+p4^p)+n^2)$. When $k_i = n/p\ \forall i \in \{1,\dots,p\}$ the algorithm will take $O(pn+n^2) = O(n^2)$.

3. Improving the bound

In this section we describe an algorithm that achieves a better bound at the expense of a higher complexity. It uses a parameter x ($x \leqslant n-p+1$) which determines the improvement in the bound, and the higher complexity.

To partition G into p parts with sizes $\{k_1,\dots,k_p\}$ call Part_Alg_x(G, $\{k_1,\dots,k_p\}$), where Part_Alg_x is defined in Fig. 4. This algorithm considers the $x+p-1$ components obtained when $x+p-2$ longest edges are removed from a MST of G. It considers all of the possible combinations to aggregate these components into sets of sizes that enable us to produce a solution by applying the cycle routine to each set. The case $x=1$ gives the same bound as Part_Alg, but with higher time complexity because it enumerates all of the possible combinations while Part_Alg only checks the existence of such a combination for each value of e_cou.

```
Part_Alg_x
  input
  1. A graph $G=(V,E)$.
  2. A set of positive integers $\{k_1,\dots,k_p\}$ satisfying $\sum_{i=1}^{p} k_i = n$.
  returns
  1. $\{P_i\}_{i=1}^{p}$ where $\bigcup_{i=1}^{p} P_i = V$ and $|P_i| = k_i$.
  2. A value apx satisfying $apx = \sum_{i=1}^{p} l(MST(P_i))$.
  begin
  Step 1
       $T := MST(G)$. $PT := \{(T, \{k_1,\dots,k_p\})\}$.
       $r := l(T) - \frac{l(g_1)}{2}$, where $g_1$ is the longest edge in $T$.
       end Step 1
  Step 2
       Remove the $x+p-2$ longest edges in $E_T$.
       A set of connected components $\{C_1,\dots,C_{x+p-1}\}$ has been created.
       Compute all the partitions of $\{k_1,\dots,k_p\}$ into $y$ sets $\{K_1,\dots,K_y\}$,
       and all the partitions of $V$ into $y$ sets $\{W_1,\dots,W_y\}$ such that:
       1. $2 \leqslant y \leqslant p$.
       2. $\forall j \in \{1,\dots,x+p-1\}\ \exists i \in \{1,\dots,y\}$ for which $V_{C_j} \subseteq W_i$.
       3. $\sum_{k_j \in K_i} k_j = |W_i|$.
       for every pair of such partitions $\{K_1,\dots,K_y\}$ and $\{W_1,\dots,W_y\}$ :
         $r_{temp} := \sum_{i=1}^{y} l(MST(W_i))$.
         if $(r_{temp} < r)$
           then $r := r_{temp}$.
                $PT := \{(MST(W_i), K_i)\ i=1,\dots,y\}$.
           end if
         end for
       end Step 2
  Step 3
       $y := |PT|$
       for every $(i=1,\dots,y)$.
         Call Cycle_Part$(T_i, K_i)$ where:
         $\{P_i^1,\dots,P_i^{|K_i|}\}$ is the returned partition,
         $r_i$ is the returned value.
         end for
       return $(\{P_1^1,\dots,P_1^{|K_1|},\dots,P_y^1,\dots,P_y^{|K_y|}\}$, $apx := \sum_{i=1}^{y} r_i$.)
       end Step 3
  end Part_Alg_x
```

Fig. 4. The improved partitioning algorithm.

3.1. Evaluating Part_Alg_x

The next 2 lemmas will be proved together.

Lemma 3.1. *Let g_1 be the longest edge in T, then $apx \leqslant 2l(T) - l(g_1)$.*

Lemma 3.2. *Let r_{temp} be a value calculated in Step 2. If in Step 3 $y > 1$ then $apx \leqslant 2r_{\text{temp}}$.*

Proof. When entering Step 3 the set PT is given by $PT = \{(T_1, K_1), \ldots, (T_y, K_y)\}$ and if $y > 1$ then $\sum_{i=1}^{y} l(T_i) = r$.

Suppose that $y = 1$. Activating the cycle routine of T gives, by Lemma 2.1, a value $r_1 \leqslant 2l(T) - l(g_1)$. Hence for this special case $apx = r_1 \leqslant 2l(T) - l(g_1)$.

Suppose now that $y > 1$. In this case, the value of r when entering Step 3 is different from the initial value $l(T) - l(g_1)/2$. For every r_{temp} along the algorithm

$$\sum_{i=1}^{y} l(T_i) = r \leqslant r_{\text{temp}}. \tag{4}$$

By Lemma 2.1, for every $i \in \{1, \ldots, y\}$ $r_i \leqslant 2l(T_i)$, implying

$$apx = \sum_{i=1}^{y} r_i \leqslant \sum_{i=1}^{y} 2l(T_i).$$

From Eq. (4), $apx \leqslant 2r_{\text{temp}}$. Since r at the end of the algorithm obviously satisfies that $r \leqslant l(T) - l(g_1)/2$, for this case too $apx \leqslant 2r \leqslant 2l(T) - l(g_1)$. □

Theorem 3.3. *$apx \leqslant (2 + (2p - 3)/x)opt$.*

Proof. By adding $\{e_1^*, \ldots, e_{p-1}^*\}$, to $\bigcup_{i=1}^{p} E_{O_i}$ a spanning tree of G is created. Hence,

$$l(T) \leqslant opt + \sum_{i=1}^{p-1} l(e_i^*). \tag{5}$$

1. $opt < xl(e_1^*)$. The set of edges $\bigcup_{i=1}^{p-1} E_{O_i}$ contains at most $x - 1$ edges of length $\geqslant l(e_1^*)$. It follows that the set of edges $\bigcup_{i=1}^{p-1} E_{O_i} \cup \{e_1^*, \ldots, e_{p-1}^*\}$ is a spanning tree with at most $(x - 1) + (p - 1)$ edges of length $\geqslant l(e_1^*)$. Hence, by Theorem 2.4, T contains at most $x + p - 2$ edges of length $\geqslant l(e_1^*)$. Removing from T its $x + p - 2$ longest edges leaves only edges inside the optimal solution set of nodes. (Because the shortest edge between two nodes from two different O_is has to be at least of length $l(e_1^*)$.) So there is a partitioning $\{W_1, \ldots, W_p\}$ which is exactly the optimal solution. That proves that when reaching Step 3, $r \leqslant opt$, and according to Lemma 3.2, $apx \leqslant 2r \leqslant 2opt$.
2. $xl(e_j^*) \leqslant opt < xl(e_{j+1}^*)$ for some $j \in \{1, \ldots, p - 2\}$. In this case the set of edges $\bigcup_{i=1}^{p-1} E_{O_i} \cup \{e_1^*, \ldots, e_{p-1}^*\}$ is a spanning tree with at most $x - 1 + p - j - 1$ edges of

length $\geqslant l(e^*_{j+1})$. Then according to Theorem 2.4, T contains at most $x+p-j-2$ edges of this length. Removing from T its $x+p-2$ longest edges, will leave only edges of length $< l(e^*_{j+1})$. Look at $\{U^j_1,\dots,U^j_{p-j}\}$ defined before. By Lemma 2.3, the shortest edge between U^j_i and U^j_k for every i,k is at least as long as $l(e^*_{j+1})$. Thus after the removal of the $x+p-2$ longest edges from T there are no edges left between nodes from different U^j_is. So there is a partitioning $\{W_1,\dots,W_{p-j}\}$ which is exactly $\{U^j_1,\dots,U^j_{p-j}\}$. Hence, when Step 3 is reached

$$r \leqslant \sum_{i=1}^{p-j} l(MST(U^j_i)).$$

By the way the U^j_i were defined

$$\sum_{i=1}^{p-j} l(MST(U^j_i)) \leqslant opt + \sum_{i=1}^{j} l(e^*_i).$$

$$\Rightarrow r \leqslant opt + \sum_{i=1}^{j} l(e^*_i).$$

By Lemma 3.2 $apx \leqslant 2r \leqslant 2(opt + \sum_{i=1}^{j} l(e^*_i)) \leqslant 2opt + 2jl(e^*_j)$. Since $j \leqslant p-2$ and according to the assumption of this case

$$apx \leqslant 2opt + (2p-4)l(e^*_j) \leqslant \left(2 + \frac{2p-4}{x}\right)opt.$$

3. $xl(e^*_{p-1}) \leqslant opt$. By Lemma 3.1 $apx \leqslant 2l(T) - l(g_1)$. Clearly $l(g_1) \geqslant l(e^*_{p-1})$, and with Eq. (5) and the assumption of this case,

$$\begin{aligned} apx &\leqslant 2l(T) - l(e^*_{p-1}) \\ &\leqslant 2\left(l(T) - \sum_{i=1}^{p-1} l(e^*_i)\right) + 2\sum_{i=1}^{p-2} l(e^*_i) + l(e^*_{p-1}) \\ &\leqslant 2\left(l(T) - \sum_{i=1}^{p-1} l(e^*_i)\right) + (2p-3)l(e^*_{p-1}) \\ &\leqslant 2opt + (2p-3)l(e^*_{p-1}) \leqslant \left(2 + \frac{2p-3}{x}\right)opt. \qquad \square \end{aligned}$$

3.2. Complexity

The complexity of this algorithm is $O(f(p,x)n^2)$ where f is an exponential function of p and x.

Step 1: Finding a MST takes $O(n^2)$.

Step 2: We can scan the tree to find the $x+p-2$ longest edges. This requires $O((x+p)n)$. Looking for all the partitions of $\{C_1,\dots,C_{x+p-1}\}$ requires $O(f_1(p,x))$, where f_1 is an exponential function of p and x. Then scan all the partitions of $\{k_1,\dots,k_p\}$ taking $O(f_2(p,x))$, where f_2 is an exponential function of p and x. For every acceptable pair of partitions finding all the MSTs and their lengths require $O(n^2)$. Altogether this step requires $O(f(p,x)n^2)$, where f is an exponential function of p and x. When $k_i=(n/p)\ \forall i\in\{1,\dots,p\}$ finding all the possible partitions takes $O(2^{(p+x)})$ time, and for each partitions $O(n^2)$ work is needed. Altogether this step requires $O(2^{(p+x)}n^2)$ time.

Step 3: As before this step calls Cycle_Part for every pair in PT, taking altogether $O(n^2)$.

So the complexity of Part_Alg_x is dominated by that of Step 2, that is, $O(f(p,x)n^2)$. When $k_i=(n/p)\ \forall i\in\{1,\dots,p\}$ the algorithm takes $O(2^{(p+x)}n^2)$ time.

4. Partitioning into 2 sets

In this section we treat the following case: Given a graph $G=(V,E)$, $|V|=n$, and a constant $K\leqslant n/2$. Partition V into disjoint sets P and Q such that $|P|=K$, $|Q|=n-K$, and $l(MST(P))+l(MST(Q))$ is minimized.

4.1. The K-centroid

For approximating the solution in the case $p=2$ we define a '*K-centroid*' and prove its existence.

Given a tree $T=(V,E_T)$, and a constant $K\leqslant n/2$. For a node $r\in V$ remove all the edges in E_T incident to r. A set of connected components is created. Let $\{C_1,C_2,\dots,C_m\}$ be all of these components which satisfy $|V_{C_i}|\leqslant K$. If $\sum_{i=1}^{m}|V_{C_i}|\geqslant K$ then r is a **K-centroid**.

For the special case $K=n/2$ the K-centroid is simply a centroid (a centroid is a node which when removing it form the T, each one of the connected components created contains at most $n/2$ nodes). The definition of a centroid of a tree, and a linear time algorithm for finding it are presented in [8].

Lemma 4.1. *A K-centroid exists for every tree and $K\leqslant n/2$. It can be found in* $O(n)$ *time.*

Proof. Consider Find_K-cent defined in Fig. 5. During this procedure, the spanning tree given to Find_K-cent as input contains at least $K+1$ nodes.

In each iteration, the number of nodes in the tree is no more than half the number of nodes in the previous one. Since the tree is always kept to contain at least $K+1$ nodes, Find_K-cent will always stop and find the required node.

In each iteration the most expensive operation is to find the centroid, which takes linear time. Since the number of nodes in each new iteration is no more than half the number of nodes in the previous iteration, the algorithm takes $O(n)$. □

```
Find_K-cent
  input
  1. A tree $T$.
  2. An integer $K$ $(1 \leqslant K \leqslant \frac{n}{2})$.
  returns
  1. A node $r$ which is a $K$-centroid.
  2. A forest $\{T_1,\dots,T_m\}$ such that $|V_{T_i}| \leqslant K$ $\forall i \in \{1,\dots,m\}$ and $\sum_{i=1}^{m} |V_{T_i}| \geqslant K$.
  begin
  $c := $ a centroid in $T$.
  Delete $c$ from $T$, a set of connected components $\{C_1,\dots,C_m\}$ is created.
  if $(|V_{C_i}| \leqslant K)$ $i = 1,\dots,m$
    then
      return $(c, \{C_1,\dots,C_m\})$
    else $S := V_{C_i}$ such that $|V_{C_i}| \geqslant K$
      $T_s := T$ induced on $S$.
      return (Find_K-cent $(T_s, K)$).
    end if
  end Find_K-cent
```

Fig. 5. Finding the K-centroid.

4.2. The approximation algorithm

To divide the graph into two sets of sizes K and $|V| - K$, call Part_2_Alg(G,K), where Part_2_Alg is defined in Figs. 6 and 7. This algorithm finds a MST of G. First it tries to find one edge whose removal divides the graph into sets of the desired sizes. If failed it doubles part of the tree's edges getting a graph that can be easily divided into sets of the desired sizes.

Note that when Step 3 is reached there is no edge whose removal creates a connected component of size exactly K. Hence for every $i \in \{1,\dots,m\}$ $|V_{T_i}| < K$. Also, since in Find_K-cent S was always kept to contain at least $K+1$ nodes, $m \geqslant 2$.

When $l(E_{T_2}) + l(e_2) \geqslant l(E_{TS})$ it is possible to find the defined above P since T_2 and the cycle contain all the nodes not in $V_{T_1} \cup \{u\}$, hence T_2 and the cycle contain at least $n - K \geqslant K$ nodes.

When $l(E_{T_2}) + l(e_2) < l(E_{TS})$ it is possible to find the defined above P since $|V_{T_1}| < K$, but T_1 and the cycle contain at least $K+1$ nodes. Also, in that case the nodes from the cycle that are inserted into P are obtained by walking on the cycle, starting at c and walking $K - |V_{T_1}| - 1$ nodes in one of the two possible directions.

4.3. Evaluating Part_2_Alg

Lemma 4.2. *If Step 3 is reached then* $apx \leqslant 2l(T) - (l(E_{T_1}) + l(e_1) + \max\{l(E_{T_2}) + l(e_2), l(E_{TS})\})$.

```
Cre_Cycle
  input
  1. A tree $T_0$.
  2. A set of edges $F \subset E_{T_0}$.
  returns
  1. A graph $H$.
  begin
  Double all the edges in $F$.
  A cycle has been created.
  Change this cycle into a simple cycle of equal or smaller length
  (using the triangle inequality).
  Let $H$ be the obtained graph.
  return ($H$)
  end Cre_Cycle
```

Fig. 6. Dividing the graph into 2 sets (*Cre_cycle* routine).

Proof. (1) If $l(E_{T_2})+l(e_2)\geqslant l(E_{TS})$ then the length of the part of graph which is doubled is $l(T)-(l(E_{T_1})+l(e_1)+l(E_{T_2}))$, and therefore,

$$apx\leqslant l(G_2)-l(e_2)\leqslant 2l(T)-(l(E_{T_1})+l(e_1)+l(E_{T_2})+l(e_2)).$$

By the assumption of this case this implies the claimed inequality.

(2) If $l(E_{T_2})+l(e_2)<l(E_{TS})$ then the length of the part of the graph which is doubled is $l(T)-(l(E_{T_1})+l(e_1)+l(E_{TS}))$. Hence, $apx\leqslant 2l(T)-(l(E_{T_1})+l(e_1)+l(E_{TS}))$. By the assumption of this case this implies the claimed inequality. □

Lemma 4.3. *If Step* 3 *is reached then* $apx\leqslant 2l(T)-(l(g_1)+l(g_2))$ *where* g_1 *and* g_2 *are two longest edges in* T.

Proof. (1) If $\{g_1,g_2\}\subset E_{T_1}\cup\{e_1\}\cup E_{T_2}\cup\{e_2\}$ then $l(E_{T_1})+l(e_1)+l(E_{T_2})+l(e_2)\geqslant l(g_1)+l(g_2)$. It follows from Lemma 4.2 that $apx\leqslant 2l(T)-(l(g_1)+l(g_2))$.

(2) If $\{g_1,g_2\}\subset E_{T_1}\cup\{e_1\}\cup E_{TS}$ then $l(E_{T_1})+l(e_1)+l(E_{TS})\geqslant l(g_1)+l(g_2)$. Again Lemma 4.2 gives the claimed result.

(3) If $\{g_1,g_2\}\subset E_{T_2}\cup\{e_2\}\cup E_{TS}$ then $l(E_{T_2})+l(e_2)+l(E_{TS})\geqslant l(g_1)+l(g_2)$. According to the algorithm $l(E_{T_1})+l(e_1)\geqslant l(E_{T_2})+l(e_2)$. Therefore, $l(E_{T_1})+l(e_1)+l(E_{TS})\geqslant l(g_1)+l(g_2)$, and from Lemma 4.2, $apx\leqslant 2l(T)-(l(g_1)+l(g_2))$. □

Theorem 4.4. $apx\leqslant 2opt$.

Proof. Consider an optimal solution. It divides V into two sets O_1 and O_2 with $|O_1|=K$ and $|O_2|=n-K$. Mark e^* to be a shortest edge between O_1 and O_2: The edges $E_{O_1}\cup E_{O_2}\cup\{e^*\}$ define a spanning tree of G, where E_{O_1} and E_{O_2} are the edges in $MST(O_1)$ and $MST(O_2)$ respectively. Therefore,

$$l(T)\leqslant l(MST(O_1))+l(MST(O_2))+l(e^*)=opt+l(e^*). \qquad (6)$$

```
Part_2_Alg
 input
 1. A graph $G$.
 2. An integer $(1 \leqslant K \leqslant n/2)$.
 returns
 1. $\{P,Q\}$ where $P \cup Q = V$, $|P| = K$ and $|Q| = n - K$.
 2. A value $apx = l(MST(P)) + l(MST(Q))$.
 begin
 Step 1
   $T := MST(G)$.
   end Step 1
 Step 2 if (There exists an edge $e_1$ whose removal from $T$ disconnects $T$
     into 2 connected components, $P$ and $Q$ such that $|P| = K$)
     then
       return $(\{P,Q\}, apx := l(MST(P)) + l(MST(Q)))$.
     end if
   end Step 2
 Step 3
   Call Find_K-cent $(T,K)$ where:
   $c$ is the returned $K$-centroid,
   $\{T_1,\ldots,T_m\}$ is the returned forest.
   $e_i :=$ the edge connecting $T_i$ to $c$ in $T$, $i = 1,\ldots,m$.
   W.l.o.g suppose that:
   $l(E_{T_1}) + l(e_1) \geqslant l(E_{T_2}) + l(e_2) \geqslant l(E_{T_i}) + l(e_i)\ \forall i \in \{3,\ldots,m\}$.
   $TS :=$ the subtree of $T$ induced by $V \backslash (\bigcup_{i=1}^{m} V_{T_i})$.
   if $(l(E_{T_2}) + l(e_2) \geqslant l(E_{TS}))$
     then
       $G_2 := Cre\_Cycle(T, E_T \backslash (E_{T_1} \cup E_{T_2} \cup \{e_1\}))$ (see Fig. 8).
       Delete $e_2$ from $G_2$.
       $P := V_{T_2} \cup \{$the first $K - |V_{T_2}|$ nodes from the path connecting
         $T_2$ to $c\}$.
       $Q := V \backslash P$.
     else
       $G_3 := Cre\_Cycle(T, \bigcup_{i=2}^{m}(\{e_i\} \cup E_{T_i}))$. (see Fig. 8).
       $P := V_{T_1} \cup \{c\} \cup \{K - 1 - |V_{T_1}|$ nodes that are adjacent to $u$
       on the cycle, found when walking from $c$
         on the cycle in one direction.$\}$
       $Q := V \backslash P$.
     end if
   return $(\{P,Q\}, apx := l(MST(P)) + l(MST(Q)))$.
   end Step 3
 end Part_2_Alg
```

Fig. 7. Dividing the graph into 2 sets.

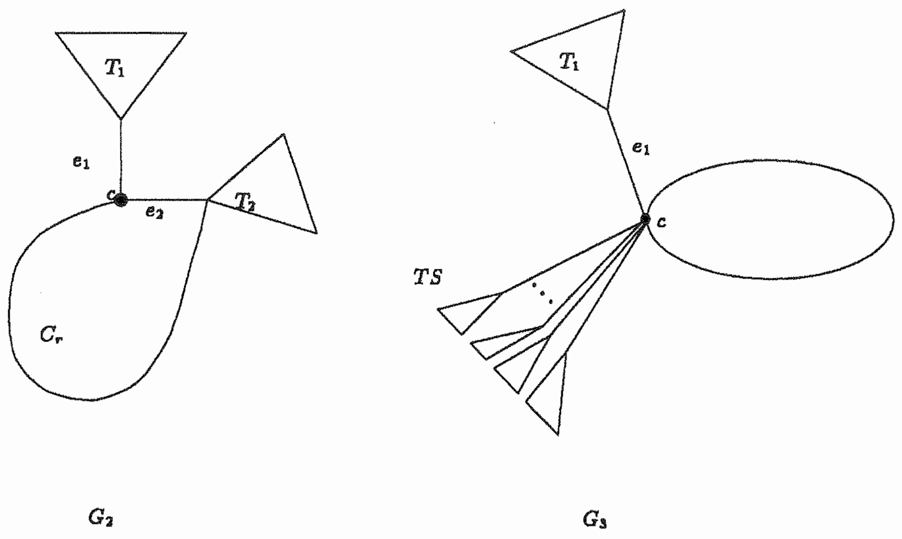

Fig. 8. G_2 and G_3.

1. $opt < l(e^*)$. The set of edges $E_{O_1} \cup E_{O_2} \cup \{e^*\}$ is a spanning tree of G with one edge of length $\geqslant l(e^*)$. Therefore, by Theorem 2.4, T contains at most one edge of length $\geqslant l(e^*)$. Removing this edge from T disconnects O_1 from O_2. Since this removal leaves 2 connected components, they must be O_1 and O_2. So Step 2 will find the edge e^* and $apx = opt$.
2. $l(e^*) \leqslant opt$. By Eq. (6), $l(T) \leqslant opt + l(e^*) \leqslant 2opt$. If the algorithm halts at Step 2 then $apx \leqslant l(T) \leqslant 2opt$. If Step 3 is reached then T contains at least 2 edges between O_1 and O_2. Hence, $2l(e^*) \leqslant l(g_1) + l(g_2)$. By Lemma 4.3 $apx \leqslant 2l(T) - (l(g_1) + l(g_2))$, so that, $apx \leqslant 2l(T) - 2l(e^*) = 2(l(T) - l(e^*))$, and by Eq. (6) $apx \leqslant 2opt$. □

4.4. Example

We show now that the bound of Theorem 4.4 is tight. Consider the graph in Fig. 9(a), and let $K = 2$. The optimal partition is $\{v_0, v_3\}, \{v_1, v_2\}$, with $opt = 1$.

The MST T, chosen in Step 1 of the algorithm, is described in Fig. 9(b) and $l(T) = 2$. Deleting any edge of T gives one set of 3 nodes and one set of 1 node. Therefore the algorithm continues to Step 3.

In this case, $K = n/2$ so that the K-centroid we are looking for is the centroid $c = v_0$. The algorithm finds T_1, T_2 and T_3 ($m = 3$). Let $V_{T_1} = \{v_1\}, V_{T_2} = \{v_2\}, V_{T_3} = \{v_3\}$, so that $E_{T_1} = E_{T_2} = E_{T_3} = \emptyset$, and $e_1 = (v_0, v_1)$, $e_2 = (v_0, v_2)$, $e_3 = (v_0, v_3)$. Then $l(E_{T_1}) + l(e_1) = l(E_{T_2}) + l(e_2) = 1$, and $l(E_{T_3}) + l(e_3) = 0$. Doubling $\{e_2\} \cup E_{T_3} \cup \{e_3\}$ gives the graph shown in Fig. 9(c), and creating the simple cycle gives the graph shown in Fig. 9(d).

Deleting $e_2 = (v_0, v_2)$ and the opposite edge (v_0, v_3) leaves the edges (v_0, v_1) and (v_2, v_3) so that the partitioning offered by Part_2_Alg consists of $\{v_0, v_1\}$ and $\{v_2, v_3\}$. Thus, $apx = l(v_0, v_1) + l(v_2, v_3) = 2 = 2opt$.

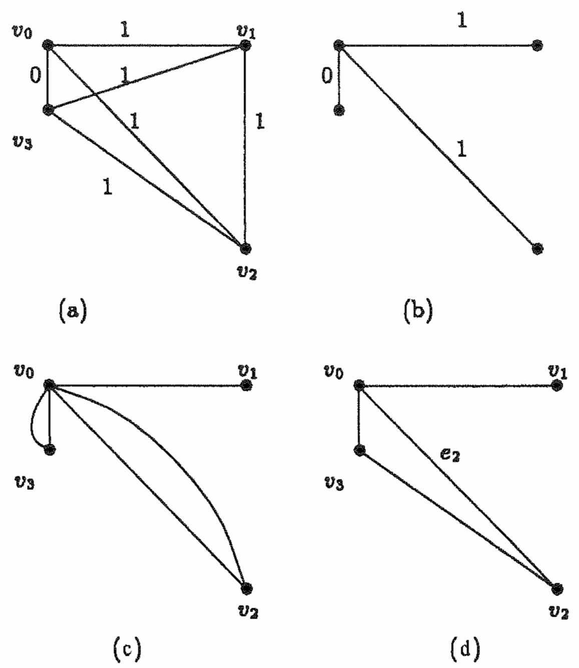

Fig. 9. A tight example for 2 partitioning.

4.5. *A bound on opt*

Theorem 4.5. *Let $T = MST(G)$, then $opt \leqslant 3l(T)/2$.*

To prove the bound we describe an algorithm that achieves $apx \leqslant 3l(T)/2$. Since $opt \leqslant apx$, the theorem is proved.

The algorithm is described as Part_2_Bound in Fig. 10. It calls Cre_Cycle defined in Fig. 6. The algorithm finds MST for G and a centroid c in this spanning tree. It then doubles part of the edges of the tree to find two spanning trees, each containing $n/2$ nodes, with lengths that sum up to $\leqslant 3l(T)/2$.

Note that if $l(E_{T_{i_0}}) + l(e_{i_0}) < l(T)/2$ then all the connected components satisfy this inequality, so that when n_1 is defined it must satisfy $n_1 \geqslant 2$.

4.5.1. *Evaluating Part_2_Bound*

To evaluate the algorithm we distinguish several cases:

1. $l(E_{T_{i_0}}) + l(e_{i_0}) \geqslant l(T)/2$. In this case $l(E_T \backslash (E_{T_{i_0}} \cup \{e_{i_0}\})) \leqslant l(T)/2$. This gives that the sum of the edges in the graph after creating the cycle is $\leqslant 3l(T)/2$. Spanning trees of P and Q can be obtained by deleting edges from this graph. Hence $apx \leqslant 3l(T)/2$.
2. $l(E_{T_{i_0}}) + l(e_{i_0}) < l(T)/2$.

```
Part_2_Bound
  input
  1. A graph $G$.
  returns
  1. $\{P,Q\}$ where $P \cup Q = V$ and $|P| = |Q| = n/2$.
  2. A value apx satisfying $apx = l(MST(P)) + l(MST(Q))$.
  begin
  Step 1
    $T := MST(G)$.
    $c :=$ a centroid of $T$.
    end Step 1
  Step 2
    Remove $c$ and the edges incident with it from $T$.
    A set of connected components $\{T_1, T_2, \ldots, T_m\}$ is created.
    (Since $c$ is a centroid $m \geqslant 2$ and $|V_{T_i}| \leqslant n/2\ \forall i$).
    $e_i :=$ the edge connecting $T_i$ to $c$ in $T$, $i = 1, \ldots, m$.
    Let $i_0$ be the index in $\{1, \ldots, m\}$ with the biggest $l(E_{T_{i_0}}) + l(e_{i_0})$.
    if $(l(E_{T_{i_0}}) + l(e_{i_0}) \geqslant l(T)/2)$
      then
        $G_2 := Cre\_Cycle(T, E_T \setminus (E_{T_{i_0}} \cup \{e_{i_0}\}))$.
        $P := V_{T_{i_0}} \cup \{c\} \cup \{$the adjacent $n/2 - |V_{T_{i_0}}| - 1$ nodes on the cycle$\}$.
      else
        $n_1 := \min\{i \in \{1, \ldots, m\} \mid \sum_{i=1}^{n_1} (l(E_{T_i}) + l(e_i)) \geqslant l(T)/2\}$.
        if $(\sum_{i=1}^{n_1} |V_{T_i}| + 1 \leqslant n/2)$
          then
            $G_2 := Cre\_Cycle(T, \bigcup_{i=n_1+1}^{m} (E_{T_i} \cup \{e_i\}))$.
            $P := \bigcup_{i=1}^{n_1} V_{T_i} \cup \{c\} \cup \{$adjacent $\frac{1}{2}n - \sum_{i=1}^{n_1} |V_{T_i}| - 1$ nodes on the cycle$\}$
          else
            $G_2 := Cre\_Cycle(T, \bigcup_{i=1}^{n_1-1} (E_{T_i} \cup \{e_i\})$.
            $P := V_{T_{n_1}} \cup \{c\} \cup \{$the adjacent $\frac{1}{2}n - |V_{T_{n_1}}| - 1$ nodes on the cycle$\}$.
          end if
        end if
      $Q := V \setminus P$.
      return $(\{P,Q\}, apx := l(MST(P)) + l(MST(Q)))$.
      end Step 2
  end Part_2_Bound
```

Fig. 10. Dividing the graph into 2 sets, an algorithm to bound opt.

- $\sum_{i=1}^{n_1} |V_{T_i}| + 1 \leqslant n/2$. By the way n_1 was defined the sum of edges in the part of the graph which is doubled $\leqslant l(T)/2$ and the bound is achieved.
- $\sum_{i=1}^{n_1} |V_{T_i}| + 1 > n/2$. By the definition of n_1:

$$\sum_{i=1}^{n_1-1} (l(E_{T_i}) + l(e_i)) \leqslant \frac{l(T)}{2}.$$

This is the part of the graph which is doubled. So the length of the graph after the simple cycle was created is $\leqslant 3l(T)/2$.

Thus, in both cases $apx \leqslant 3l(T)/2$ giving that $opt \leqslant 3l(T)/2$.

This algorithm however does not improve the two bounds achieved before, since $l(T)$ may be bigger then *opt*.

To see that the bound $3l(T)/2$ can be (asymptotically) achieved consider a graph with $n+1$ (n odd) nodes: a node u and the nodes $\{v_1, \ldots, v_n\}$. The distances between the nodes are: $l(v_i, u) = 1\ \forall i \in \{1, \ldots, n\}$. $l(v_i, v_j) = 2\ \forall i \neq k \in \{1, \ldots, n\}$. A MST, T, has $l(T) = n$, while $opt = (n-1)/2 + 2((n+1)/2 - 1) = 3(n-1)/2$.

References

[1] B. Chandra, M. Halldórsson, Facility dispersion and remote subgraphs, Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT), Lecture Notes in Computer Science, vol. 1097, Springer, Berlin, 1996, pp. 53–65.

[2] H.N. Gabow, M.X. Goemans, D.P. Williamson, An efficient approximation algorithm for the survivable network design problem, Mathematical Programming 82 (1998) 13–40.

[3] D. Gale, Optimal assignments in an ordered set: An application of matroid theory, J. Comb. Theory 4 (1968) 176–180.

[4] M.X. Goemans, D.P. Williamson, A general approximation technique for constrained forest problems, SIAM J. Comput. 24 (1995) 296–317.

[5] M.X. Goemans, D.P. Williamson, Approximating minimum-cost graph problems with spanning tree edges, Oper. Res. Lett. 16 (1994) 183–189.

[6] N. Guttmann-Beck, R. Hassin, Approximation algorithms for min-max tree partition, J. Algorithms 24 (1997) 266–286.

[7] C. Imielińska, B. Kalantari, L. Khachiyan, A greedy heuristic for a minimum-weight forest problem Oper. Res. Lett. 14, (1993) 65–71.

[8] O. Kariv, S.L Hakimi, An algorithmic approach to network location problems, Part II: p-medians, SIAM J. Appl. Math. 37 (1979) 539–560.

[9] E.L Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[10] D.P. Williamson, On the design of approximation algorithms for a class of graph problems, Ph.D. Thesis, MIT, Cambridge, MA, 1990.