Power-Aware Collective Tree Exploration*

Miroslaw Dynia[1], Miroslaw Korzeniowski[2], and Christian Schindelhauer[3]

[1] DFG Graduate College "Automatic Configuration in Open Systems", University of Paderborn, Germany
mdynia@uni-paderborn.de
[2] International Graduate School of Dynamic Intelligent Systems, University of Paderborn, Germany
rudy@uni-paderborn.de
[3] Institute of Computer Science, University of Paderborn, Germany
schindel@uni-paderborn.de

Abstract. An n-node tree has to be explored by a group of k mobile robots deployed initially at the root. Robots traverse the edges of the tree until all nodes are visited. We would like to minimize maximal distance traveled by each robot (e.g. to preserve the battery power). First, we assume that a tree is known in advance. For this NP-hard problem we present a 2-approximation. Moreover, we present an optimal algorithm for a case where k is constant.

From the 2-approximation algorithm we develop a fast 8-competitive online algorithm, which does not require a previous knowledge of the tree and collects information during the exploration. Furthermore, our online algorithm is distributed and uses only a local communication. We show a lower bound of 1.5 for the competitive ratio of any deterministic online algorithm.

1 Introduction

Suppose, we conduct a Mars expedition by sending a group of robots to this distant planet. The team lands at the bottom of an unknown crater. Each mobile robot is equipped with a wireless communication device and batteries for energy supply. The goal of the first mission is to explore the unknown terrain minimizing the energy consumption of each robot.

Since it takes many minutes for a signal to reach the Earth, the remote coordination of the exploration is impossible. So the robots organize themselves as a team and using local distributed strategies complete the mission and return to the landing zone in order to send results (the map of the terrain) to the Earth.

In this paper we investigate the problem of exploring graphs by a group of wireless mobile robots. We assume that there is no central authority, which could coordinate the robots. So the team has to organize itself in order to jointly

* This research is partially supported by the DFG-Sonderforschungsbereich SPP 1183: "Organic Computing. Smart Teams: Local, Distributed Strategies for Self-Organizing Robotic Exploration Teams".

W. Grass et al. (Eds.): ARCS 2006, LNCS 3894, pp. 341–351, 2006.

explore the terrain. All decisions are made locally without any prior knowledge of the terrain's complexity, as it is always the case for such online problem. The global (optimal) solution emerges from decisions made locally by the robots.

We investigate a terrain with a tree topology. An n-node tree has to be explored by a group of k mobile robots initially deployed at the root. Robots traverse the edges of the tree until all nodes are visited. The goal is to minimize the maximal distance traveled by one robot.

The standard approach in the competitive analysis [1] is to compare results of our *online* algorithm to the results of the optimal *offline* algorithm which knows the tree in advance.

1.1 The Model

We assume we are given a tree T with a root r and D is the maximal distance (number of edges) from r to any node in T, i.e. the height of the tree. The team of k robots has to explore the tree in such a way that robots start and finish at r and jointly traverse all edges of the tree. We minimize the maximal distance traveled by each robot. This problem can be defined in the following way [2]:

Definition 1 (of *k-MIN-RE* problem).

Instance:	*an undirected tree* $T = (V, E)$, $\lvert V\rvert = n$, *a fixed node* $r \in V$, *an integer* $k > 0$
Solution:	*tours* $C_1, C_2, \ldots C_k$, *where* $\bigcup_{i=1}^{k} C_i = E$ *and each tour contains the node* r
Goal:	*minimize* $c_{\max} = \max\{\lvert C_i\rvert : i = 1, \ldots k\}$

For the *offline* approximation, we assume that the tree T is known in advance and we construct tours C_i in polynomial time, so that $c_{\max}$ is close to the optimal value.

In the *online* model we assume that robots initially have no knowledge of T and a robot in a node v of T sees only the outgoing *ports*, i.e. beginnings of edges adjacent to v. It does not see the other node adjacent to any edge leaving v. Moreover, we have discrete and synchronous time and each time step consists of the following events:

1. a robot placed at some node v gets the information on all outgoing ports of that node,
2. a robot can communicate with other robots at distance at most 1 (or it can read or write some information on the landmark at node v),
3. a robot may choose either to wait or to traverse an edge from the chosen port.

Waiting and communicating does not induce any energy cost, while traversing an edge costs one unit. The overall cost of the exploration is the maximal distance traveled by one robot, which describes maximal energy used by one

robot. We compare this cost to the cost of the optimal algorithm, which knows T in advance.

1.2 Related Work

An exploration of unknown environments is widely studied (see [3] for a survey). The simplest case is when a mobile robot is initially placed at some node of a graph. If the graph is unlabeled, a robot cannot explore the graph alone. Hence, in [4] the robot marks the nodes of the graph with the *pebbles* in order to recognize visited nodes.

In [5] one robot traverses an arbitrary graph in a piecemeal manner and in [6] some natural exploration algorithms are presented (DFS) and lower bounds for exploration are established. In [7] the cost (*penalty*) is measured as a ratio between the number of edge traversals of an algorithm and the overall number of edges m in a graph. There, the authors develop an algorithm with penalty $O(m)$.

The single robot approach can be extended by introducing a larger number of robots and exploring the graph by a *group* of them. In [8] a team of two robots explores an unlabeled directed graph in time $O(d^2n^5)$ with high probability, where d is the maximal degree in the graph.

Profits from a collective exploration of trees are investigated in [2]. They prove $O(D + n/\log k)$ running time of their algorithm, which gives a multiplicative overhead of $O(k/\log k)$ for the time of exploration comparing to the time of an optimal algorithm which knows the tree. They also prove lower bound of $2-1/k$ for this ratio.

In [9] they present an $(2-2/(k+1))$-approximation algorithm for k-traveling salesman problem for edge-weighted trees with running time $O(k^{k-1} \cdot n^{k-1})$. In [10] the running time is improved to $O((k-1)! \cdot n)$. The k-traveling salesman problem is similar to our model, but it does not constrain the the starting positions of the robots.

The problem of minimizing the maximal distance traveled by a single robot in a collective exploration of an *unweighted* tree is shown to be NP-hard. For trees with *weighted edges* it is NP-hard too and remains NP-hard even for a constant number of robots [11].

1.3 Our Results

In this paper we focus on the exploration of an arbitrary, *unweighted* tree. Unlike in [9] and [10] we constrain positions of robots to one node in the tree (i.e. to the root). As a first result we show how to compute in polynomial time the optimal tours for the problem of the exploration with a fixed number of robots k.

Moreover, we show a 2-approximation algorithm for the problem with an arbitrary number of robots which runs in $O(k \cdot \max\{D, (n-1)/k\})$ time. This significantly improves the time comparing to results of [9] and [10].

Furthermore, from our 2-approximation we develop an 8-competitive algorithm with the same running time for the online problem (Sect. 1.1). We also prove a lower bound of 1.5 for the online problem.

2 The Complexity of *k-MIN-RE*

We start with an observation that *k-MIN-RE* is intractable [2]. The following remark is a result of a simple reduction of *k-MIN-RE* to the *3-PARTITION* problem.

Remark 1. The decision version of the *k-MIN-RE* problem is NP-hard.

We show that for unweighted trees the problem becomes easy when we assume that the number of robots is fixed. The algorithm *DPExplore* based on the dynamic programing technique constructs a k-dimensional array A_v of size n in a bottom-up fashion for each node v. The element $A_v(a_1, a_2, \ldots, a_k)$ of the array is the sequence $(T_1, T_2, \ldots, T_k)$ of subtrees, such that $\bigcup T_i = T_v$, $T_i \subset T_v$, $v \in T_i$ and $a_i = |T_i|$ is the number of edges in T_i. Each subtree defines a tour of a single robot.

The array A_v is capable of storing all possible *reasonable* traversals of T_v (and clearly includes the optimal one) by a group of k robots. The *reasonable* traversals are those, which are optimal or can be used to construct the optimal traversal. If there are many such traversals for fixed $a_1, a_2, \ldots, a_k$, then we take any of them.

The algorithm *DPExplore*(v, T) computes A_v for any v and thus after computing A_r, the *OptFixed*(v, T) algorithm can easily find the optimal solution for $T = T_r$ by finding the cell of an array A_r with the best content. Pseudo-codes for the algorithm and its subprocedures are depicted in Fig. 1, 2 and 3.

```
Extend(v, A_s)
    foreach b_1, ..., b_k where A_s(b_1, ..., b_k) ≠ ∅ do
        (T_1, ..., T_k) ← A_s(b)
        foreach b_i > 0 do
            b'_i ← b_i + 1
            T'_i ← T_i ∪ {v}
        A'(b') ← (T'_1, ..., T'_k)
    return A'

Combine(A_x, A_y)
    foreach a, b where A_x(a) ≠ ∅ and A_y(b) ≠ ∅ do
        foreach 1 ≤ i ≤ k do c_i ← a_i + b_i
        A'(c) ← A_x(a) ∪ A_y(b)
    return A'
```

Fig. 1. The *Extend* and *Combine* procedures

Lemma 1. *The algorithm* DPExplore(v,T_v) *computes in* $O(n^{2k+2})$ *time steps the array* A_v*, which contains all* reasonable *traversals of* T_v*.*

Proof. For $D = 0$ the algorithm computes A_r such that $A_r(0, \ldots, 0)$ is a set of k trees each containing only one node. Clearly this is an optimal solution for a tree in height of 0.

```
DPExplore(v, T)
  if v has no children
    A_v(0, ..., 0) ← the sequence of k trees consisting only of node v
  else
    foreach child s of v do A_s ← DPExplore(s, A_s)
    foreach child s of v do A'_s ← Extend(v, A_s)
    foreach child s of v do A_v ← Combine(A_v, A'_s)
  return A_v
```

Fig. 2. The *DPExplore* algorithm

```
OptFixed(v,T)
  A ← DPExplore(v, T)
  find (a_1, ..., a_k) minimizing max_i{a_i} for which A(a_1, ..., a_k) ≠ ∅
  (T_1, ..., T_k) ← A(a_1, ..., a_k)
  return sequence of C_i described by a tree T_i
```

Fig. 3. The optimal polynomial algorithm for *k-MIN-RE* and fixed k

Suppose that T is $h > 0$ in height and *DPExplore* computes properly arrays for all smaller subtrees. The root r of T has children $s_1, s_2, \ldots, s_d$ where d is the degree of r (clearly $d > 0$). Assume that $A_{s_1}, A_{s_2}, \ldots, A_{s_d}$ have already been computed and contain all *reasonable* traversals of subtrees T_{s_i}. The procedure *Extend*(v, A_{s_i}) builds an array A'_{s_i}, which contains all *reasonable* traversals of subtree $T_{s_i} \cup \{v\}$ for each i. The next d calls of procedure *Combine*(A_v, A_{s_i}) construct all *reasonable* traversals of T.

The upper bound for the number of nonempty elements in A_v is n^k. So, the procedure *Extend* needs $O(n^k)$ time steps. The procedure *Combine* needs $O(n^{2k})$ time steps for the same reason, thus the algorithm *DPExplore* terminates after $O(n^{2k+2})$ time steps. □

Lemma 2. *The algorithm* OptFixed(r, T) *computes an optimal solution to the problem* k-MIN-RE *for a fixed k in a polynomial time.*

Proof. The first step of *OptFixed* is a call to subalgorithm *DPExplore* which computes in time $O(n^{2k+2})$ the array A containing all reasonable traversals of T (Lemma 1).

Then, the optimal traversal of T can be easily found in $O(n^k)$ time. This implies that the *OptFixed* finds an optimal solution in $O(n^{2k+2})$ time steps, which is polynomial in the size of the tree. □

3 The Approximation

In this section we present the algorithm which produces tours C_i in such a way that $c_{\max}$ is only two times larger than the cost of the optimal tours. We need

LeftWalker(r,t)
Perform $t/2$ steps of DFS using only unmarked edges and preferring ports with the smallest IDs and then return to r. Mark each traversed edge which does not lead to unmarked edges.

Fig. 4. The *LeftWalker* algorithm

the following simple procedure which is similar to the well-known Depth First Search algorithm. It traverses the tree and marks some edges, which is described in details on Fig. 4.

Let T_v denote the subtree rooted in v and S_v be the sequence of children of v, such that for each $s \in S_v$ the tree T_s contains an unmarked edge. We assume that S_v is sorted in increasing order of IDs.

Furthermore, $B_v = \{(v, w) : w \in S_v\}$ is a set of outgoing edges which lead to subtrees containing unexplored edges. The following fact is straight-forward for a DFS-kind algorithm.

Remark 2. For any v and $s_1, s_2 \in S_v$ where $id(s_1) < id(s_2)$ the *LeftWalker* will not enter the T_{s_2} before completely marking all edges of T_{s_1}

Let $t = 4 \cdot \max\{D, (n-1)/k\}$ and we sequentially run *LeftWalker*(r, t) on the same tree k times. We show two lemmas concerning subsequent calls of *LeftWalker*(r, t) (Fig. 5).

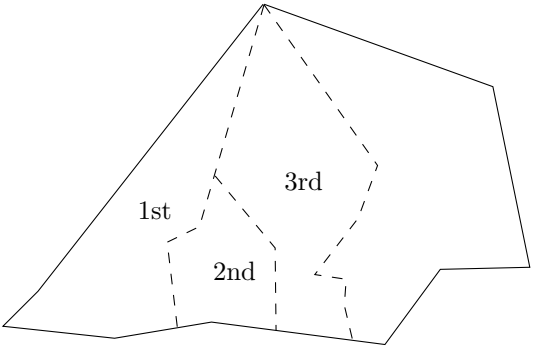

Fig. 5. Subsequential calls of *LeftWalker* algorithm

Lemma 3. *At the beginning of any subsequent call of* LeftWalker(r,t) *the following assertion holds. For each $v \in V$ at most one edge from the current set B_v has been traversed by a robot. If there is such an edge in B_v, it must be (v, w) where w is the node from the current S_v with the smallest ID.*

Proof. Assume that for some v we have two traversed edges in the current B_v. We denote these edges by (v, w_1) and (v, w_2) where $w_1, w_2 \in S_v$ and (w.l.o.g) the ID of the port connecting to w_1 is smaller than the ID of the port connecting

to w_2. Then, there is an unexplored edge in both T_{w_1} and T_{w_2}. It means that in some round in the past, *LeftWalker* started to explore 'right' subtree T_{w_2} before finishing T_{w_1}, which contradicts the Remark 2. □

Lemma 4. *If the next subsequent call of* LeftWalker(r,t) *does not finish the exploration of T, it discovers at least* $\max\{D, (n-1)/k\}$ *'new' edges which were not traversed by any robot before.*

Proof. Suppose that *LeftWalker*(r, t) explores the tree for the first time (first subsequent call) and it will not finish the exploration in this round. Before it starts returning to the root, its energy units are limited to $t/2$. It can traverse each edge at most twice, so it will traverse at least $t/4$ 'new' edges, before reaching the limit.

Suppose that it is a further subsequent call of *LeftWalker*, i.e. there was at least one call before. Assume that the exploration will not be finished in this round, so it will traverse exactly $t/2$ edges. We claim that at least $t/4$ of these edges are 'new', since at most $t/4$ of these edges were traversed before.

Indeed, from Lemma 3, we have that there is at most one such edge at arbitrary distance from the root r. This implies that there exist at most $D \leq t/4$ such edges which have been already traversed and are not marked. □

Basing on the *LeftWalker* and its properties we define the algorithm depicted in Fig. 6, and in the following lemma we show that it produces a feasible solution.

```
2-ApproxAlg(T)
    t=max{D, (n − 1)/k}
    for i=1 to k do
        C_i ← LeftWalker(r, 4t)
```

Fig. 6. The approximation algorithm

Lemma 5. *For any tree $T(V, E)$ with root r the algorithm produces a sequence of paths C_i such that*

$$\bigcup C_i = E$$

and r belongs to each path described by C_i.

Proof. Clearly, r belongs to each path described by C_i, since *LeftWalker*$(r, 4t)$ starts and ends in the root. Furthermore, Lemma 4 guarantees that in each pass *LeftWalker*$(r, 4t)$ explores at least $\max\{D, (n-1)/k\}$ 'new' edges or finishes the exploration. This implies that k passes suffice to completely explore the tree ($k \cdot \max\{D, (n-1)/k\} \geq n-1$ 'new' edges traversed). □

In the next lemma we show that the algorithm is an approximation of an optimal solution, i.e. we show a lower bound of any feasible solution.

Lemma 6. *For every algorithm for* k-MIN-RE *there exists an index* i *such that* $|C_i| \geq 2 \cdot t$*, where* $t = \max\{D, (n-1)/k\}$.

Proof. Assume that for each i the size $|C_i| < 2D$. Then, the bottom of the tree cannot be reached and at least one leaf cannot be explored. In the case, where for each i, $|C_i| < (2n-2)/k$, we have $\sum_{i=1}^{k} |C_i| < 2n-2$ steps made, which is not sufficient to explore any tree consisting of n nodes. □

Our algorithm outputs in $O(k \cdot \max\{D, (n-1)/k\})$ time steps k feasible sequences C_i (Lemma 5), such that $|C_i| \leq 4 \cdot t$ and Lemma 6 states the lower bound of $2 \cdot t$ for the optimal algorithm, which proves the approximation factor of 2 for *k-MIN-RE*.

4 The Online Problem

As described in Sect. 1.1 in the online setting, we assume no previous knowledge of the tree. The robots have to gather information on the terrain's complexity during the exploration. First, we show that no deterministic online algorithm can be optimal and then, we develop the strategy which gives the close to optimal solution. We apply the standard approach and we search in a binary way for the appropriate value of s for the tree.

4.1 Lower Bound

The following lemma states a lower bound on the competitive ratio of any deterministic algorithm.

Lemma 7. *Any deterministic online algorithm for* k-MIN-RE *has the competitive ratio* $\delta \geq 1.5$.

Proof. For an arbitrary deterministic algorithm A we will present a tree for which A needs at least $6 \cdot p$ energy units to explore and for which the optimal offline solution needs only $4 \cdot p$ energy units.

We construct a tree T which is an union of $2k-2$ paths of length p and one path of length $2p$. The tree $T = (V, E)$, depicted in Fig. 7, is such that:

$$V = \{r\} \cup \{v_{i,j} : 1 \leq i \leq p, 1 \leq j \leq 2k-1\} \cup \{v_{i,q} : p+1 \leq i \leq 2p+2\}$$

and

$$\begin{aligned} E = &\{(r, v_{1,j}) : 1 \leq j \leq 2k-1\} \\ &\cup \{(v_{i-1,j}, v_{i,j}) : 2 \leq i \leq p, 1 \leq j \leq 2k-1\} \\ &\cup \{(v_{i-1,q}, v_{i,q}) : p+1 \leq i \leq 2p+2\} \end{aligned}$$

The parameter q is defined for A in such a way, that the node $v_{p,q}$ is visited by a robot which has already visited another node on level p (a node from the set $\{v_{p,j} : 1 \leq j \leq 2k-1\}$). We assume that q is the smallest number with this property.

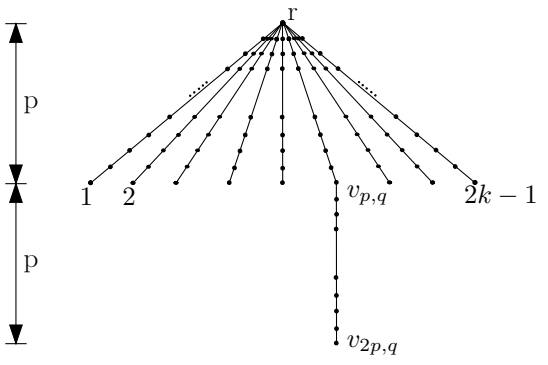

Fig. 7. The tree of lower bound

The offline algorithm may assign one robot to traverse the long path and all other robots to traverse two short paths each. Thus, the tree is explored with an energy of $4p$ per robot. Below we show that the deterministic online algorithm A will need at least $6p$ energy units for at least one robot.

By the definition of q, A uses $q-1$ robots and $2p$ energy per robot to visit the first $q-1$ dead ends and then return to r. Then one of these $q-1$ robots is sent to explore the long path. After the robot reaches node $v_{p,q}$, it discovers that this is the long path. It can not decide to explore this path (its total energy consumption would grow up to $6p$). It comes back to the root, thus some other robot explores the long path. The yet unexplored long path has to be traversed by the robot which has explored no edge before.

Let us sum up the energy the robots used so far. We have three groups of robots:

2 robots with energy consumption of at least $4p$ (which are useless now),
$q-1$ robots with energy consumption of at least $2p$,
$k-q-1$ robots which have used no energy.

There are $2k-1-q$ unexplored dead ends left, and even if the second group explores $q-1$ of them, there are still $2k-2q$ unexplored short paths and only $k-q-1$ robots which can explore them. Even if each robot takes two short paths, running out the whole available energy, there will be at least one unexplored path. Thus, the team will fail to explore the tree with an energy smaller than $6p$. □

4.2 The Online Algorithm

Now we introduce a distributed online algorithm which uses *LeftWalker* routine and explores an unknown tree.

Assume that robots have unique IDs from set $[0, 1, \ldots, k-1]$. The algorithm to explore the tree is depicted in Fig. 8. In the following two lemmas we prove the correctness and a good performance of the algorithm.

```
WarningBee(r,k)
  energy ← 1
  while (tree is not completely explored){
    energy ← energy · 2
    wait( id · energy )
    get list of the most left ports to unmarked edges from the robot id − 1
    LeftWalker( r, energy )
    wait( (k-id) · energy )
  }
```

Fig. 8. The online parallel algorithm for a group of robots

Lemma 8. WarningBee *terminates after* $\lceil \log(2t) \rceil + 1$ *rounds, and thus eventually it completely explores the tree with energy consumption of at most* $16t$ *units per robot.*

Proof. As we know from Sect. 3, running $LeftWalker(r, 4t)$ sequentially k times will completely explore the tree. This is the case in the $\lceil \log(2t) \rceil + 1$ round of *WarningBee*, where $energy = 2^{\lceil \log(t) \rceil + 2} \geq 4t$. After all robots have completed this round the whole tree is explored. This implies that *WarningBee* will terminate after $\lceil \log(2t) \rceil + 1$ rounds, where $energy \leq 8t$. Summing up the overall energy used in all rounds, we get at most $16t$ energy units per robot. □

Lemma 9. WarningBee *is 8-competitive for the online model.*

Proof. Since the *WarningBee* uses at most $16t$ energy units (Lemma 8) and optimal algorithm uses at least $2t$ energy units (Lemma 6) we have the bound of 8 for the competitive ratio. □

5 Conclusion

We have presented algorithms for exploration of trees with the goal of minimizing maximal energy used by each robot.

The first two algorithms deal with the situation where the tree is known in advance. The third one is online and it efficiently explores a tree not known in advance. The online algorithm is distributed and uses only local communication. Both algorithms are optimal up to a constant factor.

It turns out that the 8-competitive (energy) online algorithm, *WarningBee*, uses at most $O(k \cdot D + n)$ time steps. *WarningBee* moves only one robot at a time and therefore is not capable of optimizing the time of exploration, achieving k-competitiveness in this model (time).

It is known that there is $O(k/\log k)$ competitive online algorithm for optimizing time [2]. At the moment there is a lower bound of $2 - 1/k$, so there is a huge gap left for further research.

References

1. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York, NY, USA (1998)
2. Fraigniaud, P., Gasieniec, L., Kowalski, D., Pelc, A.: Collective tree exploration. In: Proc. LATIN 2004. Volume 2976. (2004) 141–151
3. Rao, N., Kareti, S., Shi, W., Iyenagar, S.: Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical report (1993)
4. Bender, M., Fernández, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: exploring and mapping directed graphs. In: Proc. 30th Symp. Theory of Computing, ACM (1998) 269–278
5. Awerbuch, B., Betke, M., Rivest, R., Singh, M.: Piecemeal graph exploration by a mobile robot. Information and Computation **152**(2) (1999) 155–172
6. Dessmark, A., Pelc, A.: Optimal graph exploration without good maps. In: Proc. ESA 2002. Volume 2461. (2002) 374
7. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. In: SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1998) 316–322
8. Bender, M., Slonim, D.: The power of team exploration: two robots can learn unlabeleddirected graphs. In: Proc. FOCS 1994. (1994) 75–85
9. Averbakh, I., Berman, O.: (p - 1)/(p + 1)-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective. Discrete Applied Mathematics **75**(3) (1997) 201–216
10. Nagamochi, H., Okada, K.: A faster 2-approximation algorithm for the minmax p-traveling salesmen problem on a tree. Discrete Applied Mathematics **140**(1-3) (2004) 103–114
11. Averbakh, I., Berman, O.: A heuristic with worst-case analysis for minimax routing of two traveling salesmen on a tree. Discrete Applied Mathematics **68** (1996) 17–32