INSTYTUT INFORMATYKI
UNIWERSYTETU WROCŁAWSKIEGO

INSTITUTE OF COMPUTER SCIENCE
UNIVERSITY OF WROCŁAW

ul. Joliot-Curie 15
50–383 Wrocław
Poland

Report 05/08

Jakub Lopuszanski

On-Line Exploration of Trees of Known Depth by a Team of Mobile Robots

November 2008

On-Line Exploration of Trees of Known Depth by a Team of Mobile Robots

Jakub Lopuszanski*

University of Wroclaw, Poland jlo@ii.uni.wroc.pl

Abstract. This paper deals with the on-line problem of exploring an unknown labeled tree by a team of robots. All robots start at the root and can traverse one edge per step. The goal is to discover the shape of the whole tree and return to the root. The cost equals the maximum over all robots of the number of traversed edges. The main result is that prior knowledge of the tree's depth, allows for a solution only 2.4 times worse than the optimum computed off-line, even if robots communicate only locally.
This is shown by a reduction to the proposed yeti@home problem. The yeti@home problem is simpler than tree exploration, but exposes its main difficulty, i.e. the trade-off between the overhead of switching robots and uneven distribution of work. This paper contains the proof of $e/(e-1)$ competitive ratio for this problem, where e is the base of the natural logarithm.

1 Introduction

Imagine a group of k robots emerging from a door of a space ship, which has just landed on an unknown planet. The planet contains many interesting places, some of them connected by convenient paths. We model the territory by an undirected graph and ask for the fastest way the robots can visit all of the sites and return to the space ship. That is we want each node to be visited by at least one robot.

If there is only one robot, and the graph is known in advance (*off-line setting*), then the problem is equivalent to Traveling Salesperson Problem (TSP) [7]. For k robots, the problem is known as Multiple Traveling Salesman Problem (mTSP) or k-TSP, where k is the number of salesmen. The goal is to find k cycles jointly covering all nodes, such that the longest is as short as possible. See [8] for this and many other variants of TSP.

This paper focus on the *on-line* setting[10], in which the shape of the graph (V, E) is unknown to the algorithm. Exploration starts from the node $v_0 \in V$ chosen by the adversary and is performed in synchronous time steps. Initially all robots can see all the edges $\{v_0, u\} \in E$ incident to v_0. In each time step each robot can communicate using short range radio signals with other robots located at the same node. Neither the amount of exchanged information, nor

* Supported by MNiSW grant number N206 001 31/0436, 2006-2008.

computational power of a robot are restricted. Every robot decides which of neighboring nodes to visit in this time step, or chooses to stay. Then at the end of the time step, all robots, that decided to move, simultaneously traverse edges, and the adversary reveals neighborhoods of newly visited nodes. This way a robot located at vertex $u \in V$ knows all edges $\{u, w\} \in E$, but the adversary has a freedom of defining parts of the tree unseen by robots later on.

All robots are deterministic, have same capabilities, but can distinguish among themselves by unique ids. The adversary has infinite processing power and knows the algorithm, thus can predict future actions of the algorithm.

Although this is not required, the algorithm presented in this paper will use communication only between robots located in v_0, which makes it applicable in many different models of communication.

The cost can be measured in two slightly different ways, which are indistinguishable in the off-line setting.

- In the *time model* one would like the whole exploration to take as short as possible.
- In the *energy model* one would like the longest path of a robot to be as short as possible.

This paper focus on the later model, which is motivated by a problem of minimizing supplies required per robot, such as fuel. Also, parts of a robot can wear off, so one can be more worried about a length of a trip, than about the time it will take.

For an on-line algorithm we are usually interested in its *competitive ratio*, that is a largest possible ratio between the cost of the algorithm and the cost of the optimum computed off-line for a same input[9].

Please note, that algorithms for the energy model, can be viewed as approximation algorithms for a NP-hard problem of k-TSP. As this problem seems very difficult, one can consider a restricted version in which the graph is a tree. This paper focus on this variant, which is called Tree Exploration Problem (TEP)[4].

Of course algorithms for TEP can be applied not only in motion planing, but for example to search a tree of possible proofs for a theorem using multiple processors in a PROLOG-like environment, or to browse a network, or in other areas, where an unknown tree has to be traversed concurrently.

1.1 Related Work

The k-TSP problem was studied in [5]. There are also many results for unlabeled graphs and/or finite automata which are out of the scope of this paper.

An off-line version of TEP was shown to be NP-hard in [4], by a reduction from the 3-PARTITION problem. An algorithm pseudo-polynomial in k for it was shown in [2].

The time model for TEP seems very mysterious, as the best known lower bound for the competitive ratio is $\Omega(\frac{\log k}{\log \log k})$ [1], while the upper bound is $O(\frac{k}{\log k})$ [4]. Please note, that $\frac{k}{\log k}$ is only slightly better, than performing the

whole exploration using one robot only. A very intuitive algorithm from [4] introduced very interesting model of communication using yellow POST-it stickers.

For the energy model, an algorithm can take more time for scheduling task more efficiently. This resulted in algorithms with constant competitiveness. There are algorithms known with competitiveness better than $3.5 - \frac{6}{k+1}$ for $k \leq 5$ and $4 - \frac{9}{k+1}$ for $k \geq 5$ [1]. There is also a simple lower-bound of 1.5 for this problem [2]. In the same paper, 2-competitive algorithm was shown, requiring a number of nodes to be known in advance. That was very close to the spirit of this paper, as it assumed some a priori knowledge given to the robots.

In [6] there was only a single robot, equipped with an oracle. Authors investigated amount of information necessary for a robot to complete the exploration of a tree faster. They did not require the robot to return to the root, thus allowing for algorithms better than DFS.

1.2 New Results

This paper considers the TEP in the context of competitive analysis in the energy model. Moreover on-line algorithm is given the size of the unknown tree as a hint.

In Sect. 2 we show an algorithm with competitive ratio arbitrary close to 1 for trees of constant depth. This is result just emphasize that the problem gets interesting only if we treat the depth as a part of input and do not allow for arbitrary large additive constants.

In Sect. 4 the yeti@home problem is defined, which is much easier to analyze than the TEP, as it deals with numbers rather than with trees. This problem has two flavors: yeti@home$_{sum}$and yeti@home$_{max}$. Finding the exact solution to the latter variant is the main subject of this paper, as there exists a reduction from TEP to it.

In Sect. 5 a simple algorithm for yeti@home$_{max}$along with short analysis of it's competitive ratio is given. This gives an upperbound of $\phi + 1 \approx 2.6$ for yeti@home$_{max}$problem.

In Sect. 6 lower bound and upper bound for the yeti@home$_{sum}$problem meet together at $e/(e-1)$, where e is the basis of the natural logarithm. Unfortunately this neat result does not apply directly to TEP. However similar techniques are used in One has to slightly change Sect. B to prove better upperbound for yeti@home$_{max}$.

The analysis presented in this paper shows that knowledge of the depth of the tree can be used by an algorithm to reach 2.4-competitive ratio for the TEP in the energy model. This number is just a safe approximation, which follows from a numerical analysis of yeti@home$_{sum}$problem found in Sect. 6.

Although for yeti@home$_{max}$the best on-line algorithm can be easily constructed using binary search, it's unclear what is it's competitive ratio. Moreover, even a precise lowerbound could not be applied to TEP, as the reduction works only in one direction.

2 A Short Note on Trees of Constant Depth

If we are interested in exploring trees of depth bounded by a constant D, then we can abuse a trade-off between an additive constant and a competitive ratio. For a large real constant f consider an algorithm that always picks the least exhausted robot for a trip, which is the one that has moved the least number of times so far. In case of ties, robot with the smallest identificator is chosen. The trip consists of three phases:

1. reaching the place in which the previous robot finished exploration
2. performing moves along a standard DFS path, until the number of moves in this trip is equal to $f * d$, where $d < D$ is the depth of the seen part of the tree
3. getting back to the root and communicating the location of the node where Step 2 has ended to the others

Step 3 of a round followed by Step 1 of another, will be called *switching explorers* throughout this paper. Note that this can not take more than $d + d$ steps, so at least $\frac{f}{f+2}$ of a time spent by the algorithm is the actual DFS and less than $\frac{2}{f+2}$ is wasted for switching between explorers. Since the DFS takes exactly $2m$ steps, where m is the number of edges in the tree, it follows that there must be a robot that traveled at most $2m\frac{f+2}{fk}$ units. Let r be the most exhausted robot after the execution finishes. Consider the last trip of the robot r. Since it was chosen for this trip, it must have been the least exhausted one at that moment. In particular it had had traversed at most $2m\frac{f+2}{fk}$ edges so far. The last trip could not take longer than $(f + 2)D$ steps, thus the total cost of the algorithm, which is determined by r, is not larger than $2m\frac{f+2}{fk} + (f + 2)D$. Observe that the optimum can't be smaller than $\frac{2m}{k}$ as each edge must be covered in both directions. As the deepest leaf must be visited, the optimum must be at least $2D$, as well. This gives $\frac{f+2}{f} + \frac{f+2}{2}$ competitive ratio, but if you treat D as a constant, then you can think about it as $\frac{f+2}{f}$-competitive algorithm with a huge additive constant $(f + 2)D$.

3 Trees of Known Depth

In a more interesting version of the problem, D is a part of an input.

This may be motivated by applications in which description of a whole tree would be to large to be given to all robots. This may seem silly, as in the original problem statement the computational power of robots was unrestricted, but algorithms presented in this paper are quite space- and time-efficient.

Other motivation is purely theoretical - one can be interested, if partial information about tree's shape can be useful. Number of tree nodes is an example of hint, for which better upper bound was reached, than for the general case[2]. However it is still unclear, if the advantage is real, as the upper bound is still higher then the best known lower bound for the general case. Since optimal solution for a tree is usually estimated in terms of the number of edges, and the

depth of the tree, author of cited paper asked, if knowing something about the optimum an algorithm could act better.

From the previous section one can learn that there is a trade-off between an average energy consumption, and a difference between the least and the most exhausted robot. The two extreme strategies illustrating this are:

- use only one robot to explore whole tree
- always explore only one new edge and get back to the root

First strategy has the optimal average power consumption per robot, but it is totally unfair. Second strategy splits a cost among robots as fair as possible, but has a huge maintenance cost.

When robots know D, it is easier for them to estimate the maintenance cost of switching explorers. For the sake of analysis we assume that it always takes D units of energy to get from the root to the place where the previous robot finished. This assumption is not true in general, and certainly harms the algorithm, so it reduces TEP to a new problem. However it is much easier analysis to analyze, as one can forget about the shape of the tree and focus on the number of it's edges only. Therefore new problem consists only of three numbers : $k, D, 2m$, of which only $2m$ is not known to the algorithm. This was the original motivation for the yeti@home$_{max}$problem.

4 The yeti@home Problem

Imagine a large distributed science project such as SETI@home or folding@home[1], in which k participants repeatedly, one at the time:

- contact the server to download a current state of the project,
- perform some amount of computations on a local machine,
- then finally submit the new state to the server.

It does not really matter what is the exact problem participants are solving. The important issue is that computations can not be performed in parallel. We assume that downloading and uploading the state takes together one unit of time, and that each time a user can choose any number $\in \mathbb{R}_+$ of units of time to spend on computations before submitting partial results.

Personally I was thinking about searching for the Yeti in the dangerous Himalayas by a team of frighten zoologists, who prefer to spend nights in the city and avoid long, risky trips to the mountains. Here, going uphill is like downloading the state of the project, spending several nights in the mountains is like performing computations, returning to the safe city is like submitting new state to the server. Reader is encouraged to focus only on the issue of determining a schedule of contributions and not the underlying problem (the project). The schedule should determine the (infinite) order in which users connect, and

[1] Name "'yeti@home"' was made up. See http://setiathome.berkeley.edu/ and http://folding.stanford.edu/ for real problems.

amount of work that a user is willing to perform. The order does not have to be cyclic, and same user can contribute uneven resources at different occasions.

The experiment stops when one of the users finally solves the problem, which requires n units of processing time. Unfortunately n is unknown to the participants.

The goal for an on-line algorithm is to assure for a small real constant α nobody spends more than $\alpha * OPT$ units of time on the project. Smallest such α, but large enough for all instances, is called competitive ratio of the algorithm. The optimum solution for this problem is to assign exactly $\frac{n}{k}$ units of work to each user, which gives $OPT = 1 + \frac{n}{k}$.

Note that a particular deterministic algorithm $\mathcal{A}$ for this problem acts exactly in the same way regardless of n, as $\mathcal{A}$ is stopped as soon as it learns n. Therefore for each k we can describe strategy of an algorithm as an infinite sequence of nonnegative real numbers $\beta_0, \beta_1, \ldots$. To simplify notation, instead of asking an algorithm to pick next participant, algorithm is asked for each user, one by one, in round robin fashion, how much contribution should the user make. Zeros in the sequence mean that algorithm wants to pick different user, and $\beta_{rk+i} > 0$ means that in r-th round i-th user connects to the server and spends $1 + \beta_{rk+i}$ units of time for I/O and computations.

The sum of sequence β must not be finite, as it would imply infinite loop for large values of n. Since the sequence β describes all that adversary needs to know about the algorithm, through the rest of this paper terms *sequence* β and *algorithm* β will be used interchangeably.

A reduction from the TEP to yeti@home is quite simple. Lets call a distance of 2D edges a *unit*. This way switching explorers never takes longer than 1 unit. It is sufficient for the i-th robot in r-th round to traverse distance equal to the β_{rk+i}. However there are two issues we have to address here.

First is the impossibility of traversing a number of edges that is not natural by a mere robot. This can be solved if each robot will keep track of the sum of numbers that were assigned to it, and will always try to make as many moves as necessary, to make total distance traveled so far not smaller than this sum.

Second is that the only known lower bound for the optimum solution to TEP is $\max\{1, \frac{n}{k}\}$ rather than $1 + \frac{n}{k}$. That is why we need to distinguish between the two very similar problems yeti@home$_{sum}$and yeti@home$_{max}$. The first has very natural description, the later is more applicable in tree exploration but has unintuitive definition of the cost. In both problems, cost of an algorithm is measured in the same way, however we compare this cost to different values in yeti@home$_{sum}$and yeti@home$_{max}$. In yeti@home$_{sum}$we assume OPT to be $1 + \frac{n}{k}$, but just $\max\{1, \frac{n}{k}\}$ in yeti@home$_{max}$. The latter problem is artificial in that it's actually impossible for an algorithm to reach OPT, as this value has no corresponding solution.

Example 1. Let say for $k = 4$, an algorithm generates infinite sequence β, of which first several elements are $4, 0, 2, 3, 5$. Let say, the adversary chooses $n = 8$. So, the cost of the algorithm would be: $(1+4)+0+(1+2)+(1+2) = 11$, as fourth user solves the problem one unit earlier than she was willing to participate.

Optimum for yeti@home$_{sum}$would be $OPT = 1 + \frac{8}{4} = 3$, which can be achieved by a schedule $2, 2, 2, 2, \ldots$.

Optimum for yeti@home$_{max}$would be $OPT = \max\{1, \frac{8}{4}\} = 2$, which has no representation, as explained before.

In what follows, both adversary and algorithm will be restricted in such a way, as to make input and output more regular and thus easier to analyze. Of course care must be taken not to favor any of them.

4.1 Regular Form Of Schedules

Lemma 1. *W.l.o.g. adversary can choose n which is the sum of some prefix of sequence β.*

Proof. This is because for every input adversary can simulate a deterministic algorithm and identify the user u with the largest cost. Part of the experiment that takes place after the last contribution of u does not affect the cost of the algorithm, so we can assume u was the one who solved the problem. If it happened earlier than u was willing to participate, then the adversary could wait longer, as the competitive ratio converges to k, while u is working. □

Moreover we can change the definition of algorithm's cost – from now on we will analyze the last user instead of the most exhausted one.

For each $\beta_t, t \in \mathbb{N}$, let[2]

$$ALG(t) = \sum_{\substack{0 \le j \le t \\ j \bmod k = t \\ \beta_j > 0}} (1 + \beta_j) = \sum_{\substack{0 \le j \le t \\ j \bmod k = t}} [\beta_j > 0] + \beta_j$$

$$OPT(t) = \begin{cases} \max\left\{1, \frac{1}{k}\sum_{i=0}^{t} \beta_i\right\} & \text{for yeti@home}_{max} \\ 1 + \frac{1}{k}\sum_{i=0}^{t} \beta_i & \text{for yeti@home}_{sum} \end{cases}$$

$$\alpha(t) = \frac{ALG(t)}{OPT(t)} \ ,$$

then $\alpha = \sup_t \alpha(t)$ is the competitive ratio of the algorithm β.

Lemma 2. *If there exists algorithm β with competitive ratio α, then there exists β' such that:*

$$\forall t \in \mathbb{N} \quad \beta'_t > 0 \implies \alpha(t) = \alpha \ .$$

Proof. Let β be the α-competitive algorithm. Let τ be the first index for which $\beta(\tau) > 0$ and $\alpha(\tau) < \alpha$. Let ϵ be such that increasing β_τ by ϵ would give $\alpha(\tau) = \alpha$. Let $s = \sum_{i>0} \beta_{\tau+ik}$ be the remaining work assigned to the user.

We shall build the new algorithm β' in which $\beta'_i = \beta_i$ for the most of i except that $\beta'_\tau = \beta_\tau + \epsilon$. If $s > \epsilon$ we also decrease the first few elements of the form

[2] $[\phi]$ is equals 1 if ϕ is true, 0 otherwise.

$\beta'_{\tau+ik}$ for $i > 0$ as much as we can and as long as necessary to get ϵ change in total. Otherwise we set $\beta'_{\tau+ik} = 0$ for $i > 0$.

In both cases the change we introduce does not affect $\alpha(t)$ nor β_t for $t < \tau$. It makes $\alpha(\tau)$ equal to α and for $t > \tau$ the change might help to increase $OPT(t)$ without increasing $ALG(t)$, thus it can decrease $\alpha(t)$. Thus, β' is α-competitive.

We can iterate the procedure above to generate elements of the sequence β' one by one. □

Lemma 2 says that there is no point in working less than possible. In other words, element of β is either 0 or can be computed as:

$$\beta_t = \begin{cases} \max\left\{\alpha - 1 - ALG(t-k), \frac{(\alpha \sum_{i<t} \beta i) - k(1+ALG(t-k))}{k-\alpha}\right\} & \text{for yeti@home}_{max} \\ \frac{(\alpha \sum_{i<t} \beta_i) - k(1+ALG(t-k)-\alpha)}{k-\alpha} & \text{for yeti@home}_{sum} \end{cases} . \tag{1}$$

Which simply speaking means that, if we are sure we want user u to contribute, then we assign to him this much work:

$$todo_u = \begin{cases} \max\left\{\alpha - (1 + done_u), \frac{(\alpha progress) - k(1+done_u)}{k-\alpha}\right\} & \text{for yeti@home}_{max} \\ \frac{(\alpha progress) - k(1+done_u-\alpha)}{k-\alpha} & \text{for yeti@home}_{sum} \end{cases} . \tag{2}$$

In this equation *progress* is the sum of contributions made by all users so far. The amount of work done by user u so far (including I/O) is denoted $done_u$.

Lemma 2 implies that for given k, an algorithm can be described by it's competitive ratio and the order in which the users connect to the server. Now, we would like to restrict this order.

Lemma 3. *If there exists an algorithm with competitive ratio α, then there exists β such $\forall t \quad \beta_t = 0 \vee \alpha(t) = \alpha$, which always selects the least exhausted user to connect to the server.*

Proof. We already know that we can require from each user to work as hard as possible according to (2). We can assure this condition step by step. In the same manner we will try to improve the algorithm to select in each step the least exhausted user.

Assume that τ is the first step when the algorithm selects the user d instead of user e, even though $done_d > done_e$. For both problems, from (2) it follows that $todo_e \geq todo_d + done_d - done_e$. This means that we can ask user e to replace user d at step τ and ask him to perform $done_d + todo_d - done_e$ units of computations without increasing the competitive ratio.

Imagine this switch from the point of view of other users, which can identify d and e only by how much they are exhausted. They see that the *progress* after the step τ has increased by $\epsilon = done_d - done_e$ compared to the old algorithm. It will also seem that the user e increased his value of $done_e$ by ϵ without even connecting to the server!

From now on we just need to pretend that the user d is the user e and vice versa. Moreover the *new* user e has already done *magically* some amount of work

he planned to do in the future. This is similar situation to the on in the proof of Lemma 2, and we resolve it in the same manner.

After this local change we will probably get an algorithm in which not all $\alpha(t) = \alpha$, but we have already seen how to resolve this. Note that there is no circularity here, as we always perform only these three steps:

- maximize work at step τ for user d.
- choose the proper user for step τ
- maximize work at step τ for user e.

□

Lemma 4. *If the users always do as much as they can, then we can break the ties in such a way that the sequence generated by picking least exhausted user is exactly the same as round robin.*

Proof. Imagine a priority queue of users, in which the *top* is the least exhausted user, and *bottom* is the most exhausted one. In τ-th step the algorithm pops the user number $e = top$ from the queue, assigns some amount of work $todo_e$, increases $done_e$, and inserts the user back to the priority queue. The values of $done_i$ variables do not change while in queue and value of OPT is monotone in time, therefore it always holds that $done_{bottom} \leq \alpha OPT$. Since the algorithm assigns to e as much work as possible, then after the τ-th step the value of $done_e$ is equal to αOPT, so the user e ends up in the bottom of the queue. Therefore the queue can be implemented as FIFO, which gives the thesis. □

Theorem 1. *If there exists an algorithm with competitive ratio α, then there exists β such that all β_t are positive and are given by (1).*

Proof. Combine Lemma 4 and Lemma 3. □

Fact 2. *If there is an α-competitive algorithm, then for every $\alpha' > \alpha$ there is an algorithm with competitive ratio α'.*

Theorem 1 further simplifies description of algorithms - we just need to specify α. For too small values of α generated sequence β will contain negative numbers, or will have a finite sum. Actually, one can see from (1) that the latter case implies the former. Fact 2 states that we can pick too large value of α with no such problems.

Algorithm 1 Algorithm aiming at α for *yeti@home*

```
t ← 0
while there is something to do do
  β_t ← solve(ALG(t) = αOPT(t))
  user t mod k spends β_t + 1 units on computation and I/O
  t ← t + 1
end while
```

Note that if we replace $OPT(t)$ by a smaller $OPT'(t)$ in the call to $solve()$, then the algorithm will still be α-competitive, as long as it will work at all.

5 The Analysis of yeti@home$_{max}$

For yeti@home$_{max}$we can use the following OPT' which has a nice property of being constant in each round r of k consecutive connections, thus makes picking correct α easier.

$$OPT'(t) = \begin{cases} 1 & \text{if } t < k \\ \sum_{r<\lfloor \frac{t}{k} \rfloor} \min_{0 \le i < k} \beta_{rk+i} & \text{if } t \ge k \end{cases}$$

$$\beta_{rk} = \begin{cases} \alpha - 1 & \text{if } r = 0 \\ (\alpha - 1)\left(\sum_{i<r} \beta_{ik}\right) - r - 1 & \text{if } t \ge k \end{cases}$$

For all $r > 0$ we have to enforce:

$$\begin{aligned} 0 \le \beta_{rk} &= \left((\alpha-1)^2 - 2 - \frac{1}{\alpha - 1}\right)\alpha^{r-1} + \frac{1}{\alpha - 1} \ , \\ 0 &\le (\alpha-1)^2 - 2 - \frac{1}{\alpha-1} \ , \\ \alpha &\ge \frac{3+\sqrt{5}}{2} = 1 + \phi \approx 2.618 \ . \end{aligned}$$

Theorem 3. *For all k, the algorithm given by*

$$\beta_t = \begin{cases} \phi & \text{if } t < k \\ \phi - 1 & \text{if } t \ge k \end{cases} \ ,$$

solves the yeti@home$_{max}$problem with competitive ratio $\phi + 1$.

6 The Analysis of yeti@home$_{sum}$

For yeti@home$_{sum}$problem β can be computed straightforward from (1).

$$\beta_t = \begin{cases} \frac{\alpha \sum_{i<t} \beta_i + (\alpha-1)k}{k-\alpha} & \text{if } 0 \le t < k \\ \frac{\alpha \sum_{i=t-k+1}^{t-1} \beta_i - k}{k-\alpha} & \text{if } \quad t \ge k \end{cases}$$

$$\beta_t = \begin{cases} (\alpha-1)\left(\frac{k}{k-\alpha}\right)^{t+1} & \text{if } 0 \le t < k \\ \left(\frac{\alpha}{k-\alpha}\right)\sum_{i=t-k+1}^{t-1} \beta_i - \left(\frac{k}{k-\alpha}\right) & \text{if } \quad t \ge k \end{cases} \tag{3}$$

Lemma 5.

$$\beta_t < 0 \implies \beta_{t+1} < 0$$

Proof. Consider the first τ for which $\beta_\tau < 0$. Clearly $\tau \ge k$, so the previous $k-1$ elements where positive. Since $\tau + 1 \ge k$, $\beta_{\tau+1}$ is computed from the sum of $k-1$ elements, which is smaller than it was while computing β_τ. Same holds for $\beta_{\tau+i}$ for $i = 2, \ldots, k-1$. Once $k-1$ consecutive elements are negative, further elements will also be negative. □

Fact 4. *β_t for $t \geq k$ are computed as a linear function of average of previous $k-1$ elements. Fixed point of this linear function is $\frac{1}{\alpha-1}$.*

Let $\Delta_t = \beta_t - \frac{1}{\alpha-1}$.

$$\Delta_t = \begin{cases} (\alpha-1)\left(\frac{k}{k-\alpha}\right)^{t+1} - \frac{1}{\alpha-1} \text{ if } 0 \leq t < k \\ \left(\frac{\alpha}{k-\alpha}\right)\sum_{i=t-k+1}^{t-1}\Delta_i \quad \text{if} \quad t \geq k \end{cases} \tag{4}$$

The characteristic polynomial for the sequence Δ is:

$$w(x) = x^{k-1} - \left(\frac{\alpha}{k-\alpha}\right)(x^{k-2} + \ldots + 1)$$

Yet, it is convenient to analyze the following polynomial as well:

$$v(x) = (x-1)w(x) = \left(x - \left(\frac{k}{k-\alpha}\right)\right)x^{k-1} + \left(\frac{\alpha}{k-\alpha}\right)$$

$$v'(x) = (x - \frac{k-1}{k-\alpha})kx^{k-2}$$

The polynomial $v(x)$ has no multiple roots, as its derivative has only two roots: 0 and $\left(\frac{k-1}{k-\alpha}\right)$, none of which is a root of $v(x)$. Thus $w(x)$ has $k-1$ distinct roots.

$$\left.\begin{array}{l} v(1) = 0 \\ v'(1) < 0 \\ v'\left(\frac{k-1}{k-\alpha}\right) = 0 \\ v\left(\frac{k}{k-\alpha}\right) = \left(\frac{\alpha}{k-\alpha}\right) > 0 \end{array}\right\} \implies w\left(\frac{k-\gamma_{k,\alpha}}{k-\alpha}\right) = 0 \text{ for some } \gamma_{k,\alpha} \in (0,1)$$

As all of $w(x)$ roots are distinct, sequence Δ must be a linear combination of geometric sequences of the form $1, q_i, q_i^2, \ldots$ where $w(q_i) = 0$ for $i = 1, \ldots, k-1$. Without loss of generality let $q_1 = \left(\frac{k-\gamma_{k,\alpha}}{k-\alpha}\right)$.

Let $g : \mathbb{R} \to \mathbb{R}^{k-1}$, $g(x) = (x^{k-2}, \ldots, x, 1)$ be the function which simply generates prefixes of geometric sequences.

Let $\boldsymbol{d} = (\Delta_{k-1}, \ldots, \Delta_2, \Delta_1) = (\alpha-1)\left(\frac{k}{k-\alpha}\right)^2 g\left(\frac{k}{k-\alpha}\right) - \frac{1}{\alpha-1}g(1)$.

Let A be $k-1 \times k-1$ the matrix describing recursion (4).

$$A = \begin{vmatrix} \left(\frac{\alpha}{k-\alpha}\right) & \left(\frac{\alpha}{k-\alpha}\right) & \cdots & \left(\frac{\alpha}{k-\alpha}\right) & \left(\frac{\alpha}{k-\alpha}\right) \\ 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \end{vmatrix} \tag{5}$$

Fact 5. *By the Lemma 5, α is too small if and only if there exists n such that $A^n \boldsymbol{d} < \mathbf{0}$.*

Let $\boldsymbol{u_i} = g(q_i)$, be the eigenvector of matrix A with eigenvalue q_i, for $i = 1, \ldots, k-1$.

Fact 6. *A dot product of vectors $g(x)$ and $(a_{k-1}, \ldots, a_1, a_0)$ is equal to the value of the polynomial $a_{k-1}x^{k-1} + \ldots + a_1 x + a_0$.*

Fact 7. *$w(q_i)/(q_i - q_1) = 0$ for $i = 2, \ldots, k-1$. Therefore the coefficients of the polynomial $w(x)/(x - q_1)$ form a vector $\boldsymbol{z}$ that is perpendicular to $u_2, \ldots, u_{k-1}$.*

Lemma 6.

$$\boldsymbol{z} > \mathbf{0}$$

Proof. One can simply compute the coefficients of $\boldsymbol{z}$ by performing the division of polynomials $w(x)/(x - q_1)$:

$$z_{k-2} = 1 \tag{6}$$

$$z_i = q_1 z_{i+1} - \left(\frac{\alpha}{k-\alpha}\right) \text{ , for } i = k-3, k-4, \ldots, 0 \tag{7}$$

Resolving (7) gives:

$$z_i = (q_1^{k-2-i} - \left(\frac{\alpha}{k-\alpha}\right)(q_1^{k-3-i} + \ldots + 1))$$

$$q_1^{i+1} z_i = (q_1^{k-1} - \left(\frac{\alpha}{k-\alpha}\right)(q_1^{k-2} + \ldots + q_1^{i+1}))$$

$$z_i > w(q_1) q_1^{-i-1} = 0$$

□

Lemma 7.

$$\boldsymbol{z} A^n \boldsymbol{d} = q_1^n \boldsymbol{z} \boldsymbol{d}$$

Proof. Since vectors u_i form the basis, one can express any vector as $\sum_{i=1}^{k-1} c_i u_i$ and deduce from the Fact 7 that the dot product with $\boldsymbol{z}$ depends on $c_1 u_1$ only. On the other hand the coefficient c_1 is multiplied by q_1 with each application of the transformation A. □

Lemma 8. *The α is too small if and only if $\boldsymbol{z}\boldsymbol{d} < 0$.*

Proof. Combine Fact 5 with Lemma 6 and Lemma 7. □

Theorem 8. *The yeti@home$_{sum}$ problem has $1/(1 - (1 - \frac{1}{k})^k)$ competitive ratio, which converges from $4/3$ to $e/(e-1)$.*

Proof. Omitted due to space limitations.

A Omitted Proof of Theorem 8

Proof. Combining the Fact 6 with the definition of $\boldsymbol{z}$ one can express the condition in Lemma 8 as

$$0 > \boldsymbol{zd} = (\alpha - 1)\left(\frac{k}{k-\alpha}\right)^2 \boldsymbol{z}g\left(\frac{k}{k-\alpha}\right) - \frac{1}{\alpha - 1}\boldsymbol{z}g(1) \ .$$

This leads to the following conclusions :

$$\begin{aligned}
\boldsymbol{z}g\left(\frac{k}{k-\alpha}\right) &= \frac{w\left(\frac{k}{k-\alpha}\right)}{\left(\frac{k}{k-\alpha}\right) - q_1} = \frac{1}{\left(\frac{k}{k-\alpha}\right) - q_1} \ , \\
\boldsymbol{z}g(1) &= \frac{w(1)}{1 - q_1} = \left(\frac{k}{k-\alpha}\right)\frac{\alpha - 1}{q_1 - 1} \ , \\
0 &> (\alpha - 1)\left(\frac{k}{k-\alpha}\right)\frac{1}{\left(\frac{k}{k-\alpha}\right) - q_1} - \frac{1}{q_1 - 1} \ , \\
q_1 &< \left(\frac{k}{k-1}\right) \ , \\
0 < v\left(\frac{k}{k-1}\right) &= \left(\left(\frac{k}{k-1}\right) - \left(\frac{k}{k-\alpha}\right)\right)\left(\frac{k}{k-1}\right)^{k-1} + \left(\frac{\alpha}{k-\alpha}\right) \ , \\
\alpha &< \frac{1}{1 - (1 - \frac{1}{k})^k} \ .
\end{aligned}$$

□

B The Asymptotic Analysis of yeti@home$_{max}$

Fact 9. *A simple lower bound for yeti@home$_{max}$ is 2, as for $n = k$, optimum is 1, and one of the users has to perform not below the average.*

The application to the tree exploration problem requires that optimum should be computed as $\max\{1, \frac{n}{k}\}$, which is nonlinear function. However it is monotone, and as long as it is equal to 1, β will be equal to $\alpha - 1$. Since $\alpha \geq 2$, at most after k rounds the optimum exceeds 1. Let $s = \left\lceil \frac{k}{\alpha - 1} \right\rceil = \frac{k}{\alpha - 1} + \iota$, be the number of rounds necessary to rise OPT above 1. This leads to the following recursion for OPT and β:

$$OPT_t = \begin{cases} 1 & \text{if } 0 \le t < s-1 \\ \frac{1}{k}\sum_{i\le t}\beta_i & \text{if } t \ge s-1 \end{cases}$$

$$\beta_t = \begin{cases} \alpha - 1 & \text{if } 0 \le t \le s-2 \\ \left(\frac{\alpha}{k-\alpha}\right)(s-1)(\alpha-1) - \left(\frac{k}{k-\alpha}\right) & \text{if } t = s-1 \\ \beta_{s-1}\left(\frac{k}{k-\alpha}\right)^{t-s+1} & \text{if } s \le t \le k-1 \\ \beta_{s-1}\left(\frac{k}{k-\alpha}\right)^{t-s+1} - \alpha\left(\frac{k}{k-\alpha}\right)^{t-k+1} & \text{if } k \le t \le k+s-2 \\ \left(\frac{\alpha}{k-\alpha}\right)\sum_{i=t-k+1}^{t-1}\beta_i - \left(\frac{k}{k-\alpha}\right) & \text{if } k+s-1 \le t \end{cases}$$

We are now interested in computing the limit of best α, as k goes to infinity. For simplicity, assume from now on that this limit exists and is equal to α.

By the same reasoning as for yeti@home$_{sum}$one can see that β is good if and only if $\beta_k > 0$ and the dot product of $\boldsymbol{z}$ and $\boldsymbol{d} = (\Delta_{k+s-2}, \ldots, \Delta_{s+1}, \Delta_s)$ is nonnegative.

The first condition assures $\beta_t > 0$ for $t = 0, \ldots, k+s-2$, and gives:

$$\alpha < \lim_{k\to\infty}\beta_{k-1} = (\alpha-1)e^{\alpha\frac{\alpha-1}{\alpha-2}} \iff \alpha > 2.32 \ .$$

Now, for the second condition, observe that

$$\boldsymbol{d} = \beta_{s-1}\frac{k}{k-\alpha}g\left(\frac{k}{k-\alpha}\right) - \frac{g(1)}{\alpha-1} - \alpha\left(\left(\frac{k}{k-\alpha}\right)^{s-1}, \ldots, \frac{k}{k-\alpha}, 0, \ldots, 0\right)$$

$$\boldsymbol{z}\boldsymbol{d} = \frac{k}{k-\alpha}\left(\beta_{s-1}\frac{1}{\left(\frac{k}{k-\alpha}\right) - q_1} - \frac{1}{q_1 - 1} - \alpha\sum_{t=k}^{k+s-2} z_{t-s}\left(\frac{k}{k-\alpha}\right)^{t-k}\right)$$

$$= \frac{k}{k-\alpha}\left(\frac{\alpha(\alpha-1)(s-1)+k}{k-(k-\alpha)q_1} - \frac{1}{q_1-1} - \alpha\frac{q_1^{s-1}-1}{q_1-1}\right) \ .$$

Using the fact that $w(q_1) = 0$, one can show that $\boldsymbol{z}\boldsymbol{d} > 0$ if and only if

$$0 < (q_1-1)\iota(\alpha-1) + \alpha - 1 - \alpha q_1^{k\frac{2-\alpha}{\alpha-1}+\iota} \ .$$

To connect q_1 to k somehow, notice that

$$w\left(\frac{k-\gamma_{k,\alpha}}{k-\alpha}\right) = 0 \iff \gamma_{k,\alpha}\left(\frac{k-\gamma_{k,\alpha}}{k-\alpha}\right)^{k-1} = \alpha \ .$$

Fact 10. *The term $\gamma\left(\frac{k-\gamma}{k-\alpha}\right)^{k-1}$ is decreasing with respect to k and increasing with respect to γ for interesting values of γ, α, k. Thus, $\gamma_{k,\alpha} < \gamma_{k+1,\alpha}$ for all $k > 2$. Since $\gamma_{k,\alpha}$ is monotone and bounded, it must converge.*

Let $\gamma_\alpha = \lim_{k\to\infty} \gamma_{k,\alpha}$. Let $f(x) = x - \ln x$.

Fact 11. *If* $\lim_{k\to\infty} \gamma_\alpha \left(\frac{k-\gamma_\alpha}{k-\alpha}\right)^{k-1} = \alpha$, *then* γ_α *is a solution to* $f(\gamma_\alpha) = f(\alpha)$.

Fact 12.

$$f(x) < f(\alpha) \iff x \in (\gamma_\alpha, \alpha)$$

$$\begin{aligned}
\boldsymbol{zd} \geq 0 &\iff \alpha - 1 \geq \alpha \lim_{k\to\infty} q_1^{k\frac{2-\alpha}{\alpha-1}+\iota} = \alpha e^{(\alpha-\gamma_\alpha)\frac{2-\alpha}{\alpha-1}} = \alpha \left(\frac{\gamma_\alpha}{\alpha}\right)^{\frac{\alpha-2}{\alpha-1}} \\
&\iff \alpha \left(\frac{\alpha-1}{\alpha}\right)^{\frac{\alpha-1}{\alpha-2}} \geq \gamma_\alpha \\
&\iff f\left(\alpha \left(\frac{\alpha-1}{\alpha}\right)^{\frac{\alpha-1}{\alpha-2}}\right) \leq f(\alpha) \\
&\iff \alpha \geq 2.3635
\end{aligned}$$

Theorem 13. *The yeti@home$_{max}$problem for large k has asymptotic competitive ratio* ≈ 2.3635.

Conjecture 1. The competitive ratio for yeti@home$_{max}$problem is better than ≈ 2.3635 for all k.

References

1. M. Dynia, J. Lopuszanski and C. Shindelhauer: *Why Robots Need Maps.* Proceedings of Sirocco 2007
2. M. Dynia, M. Korzeniowski and C. Shindelhauer: *Power-Aware Collective Tree Exploration.* Proceedings of ARCS06
3. M. Dynia, J. Kutyowski, F. Meyer auf der Heide and C. Shindelhauer: *Smart Robot Teams Exploring Sparse Trees.* MFCS2006, LNCS4162, 327–338
4. P. Fraigniaud, L. Gasieniec, D. Kowalski and A. Pelc: *Collective tree exploration.* Proceedings of LATIN 2004, Volume 2976, 141151
5. I. Averbakh and O. Berman: *$(p-1)/(p+1)$-approximate algorithms for p-traveling salesmen problems on a tree with minmax objective.* Discr.Appl.Mathematics, 75, (1997), 201–216
6. P. Fraigniaud, D. Ilcinkas and A. Pelc.: *Tree Exploration with an Oracle..* In Proc. of MFCS, pages 24-37, 2006
7. N.L. Biggs, E.K. LLoyd, and R.J. Wilson: *Graph Theory 1736-1936.* Clarendon Press, Oxford, 1976.
8. Bektas, Tolga (2006): *The multiple traveling salesman problem: anoverview of formulations and solution procedures..* Omega: TheInternational Journal of Management Science, 34, (3), 209-219.
9. D. Sleator e R. E. Tarjan: *Amortized efficiency of list update and paging rules..* Communications of the ACM, 1985, 28, 202–208
10. A. Borodin and R. El-Yaniv: *Online computation and competitive analysis.* Cambridge University Press, New York, USA(1998)

11. S. Albers and M. R. Henzinger: *Exploring unknown environments.* SIAM Journal on Computing, 29 (2000), 1164–1188
12. B. Awerbuch, M. Betke, R. Rivest and M. Singh: *Piecemeal graph learning by a mobile robot.* COLT95, 321–328
13. X. Deng and C.H. Papadimitriou: *Exploring an unknown graph.* Journal of Graph Theory, 32, (1999), 265–297
14. A. Dessmark and A. Pelc: *Optimal graph exploration without good maps.* ESA02, 374–386
15. K. Diks, P. Fraigniaud, E. Kranakis and A. Pelc: *Tree exploration with little memory.* 13th Ann. ACM-SIAM SODA02, 588–597
16. C.A. Duncan, S.G. Kobourov and V.S.A. Kumar: *Optimal constrained graph exploration.* 12th Ann. ACM-SIAM SODA01, 807–814
17. Cristina Bazgan, Refael Hassin, Jerome Monnot: *Approximation Algorithms for Some Vehicle Routing Problems.*
18. A. Lopez-Ortiz and S. Schuierer: *On-line Parallel Heuristics and Robot Searching under the Competitive Framework.* SWAT02, 260–269
19. P. Panaite and A. Pelc: *Exploring unknown undirected graphs.* Journal of Algorithms, 33, (1999), 281–295
20. C.H.Papadimitriou and M.Yannakakis: *Shortest paths without a map.* Theoretical Computer Science, 84, (1991), 127–150
21. N.S.V. Rao, S. Hareti, W. Shi and S.S. Iyengar: *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms.* Tech. Rep. ORNL/TM-12410, Oak Ridge National Laboratory, July 1993
22. S. Schuierer: *On-line searching in geometric trees.* Sensor Based Intelligent Robots LNAI 1724, (1999), 220–239
23. R. Fleischer and G. Trippen: *Exploring an unknown graph efficiently.* 13th Annual European Symposium, Volume 3669, Springer(2005) 11–22