The Freeze-Tag Problem: How to Wake Up a Swarm of Robots

Esther M. Arkin* Michael A. Bender † Sándor P. Fekete‡ Joseph S. B. Mitchell*
Martin Skutella §

Abstract

An optimization problem that naturally arises in the study of "swarm robotics" is to wake up a set of "asleep" robots, starting with only one "awake" robot. One robot can only awaken another when they are in the same location. As soon as a robot is awake, it assists in waking up other robots. The goal is to compute an optimal *awakening schedule* such that all robots are awake by time t^*, for the smallest possible value of t^*.

We consider both scenarios on graphs and in geometric environments. In the graph setting, robots sleep at vertices and there is a length function on the edges. An awake robot can travel from vertex to vertex along edges, and the length of an edge determines the time it takes to travel from one vertex to the other.

While this problem bears some resemblance to problems from various areas in combinatorial optimization such as routing, broadcasting, scheduling and covering, its algorithmic characteristics are surprisingly different. We prove that the problem is NP-hard, even for the special case of star graphs. We also establish hardness of approximation, showing that it is NP-hard to obtain an approximation factor better than 5/3, even for graphs of bounded degree.

These lower bounds are complemented with several algorithmic results. We present a simple on-line algorithm that is $O(\log \Delta)$-competitive for graphs with maximum degree Δ. Other results include algorithms that require substantially more sophistication and development of new techniques:

(1) The natural greedy strategy on star graphs has a worst-case performance of 7/3, which is tight.

(2) There exists a PTAS for star graphs.

(3) For the problem on ultrametrics, there is a polynomial-time approximation algorithm with performance ratio $2^{O(\sqrt{\log \log n})}$.

(4) There is a PTAS, running in nearly linear time, for geometrically embedded instances (e.g., Euclidean distances in any fixed dimension).

1 Introduction

The following natural problem arises in the study of *swarm robotics*. Consider a set of n *robots*, modeled as points in some metric space (e.g., vertices of an edge-weighted graph). Initially, there is one "awake" robot and all others are *asleep*, i.e., in stand-by mode. Our goal is to "wake up" all of the robots as quickly as possible. In order to wake up a sleeping robot, an awake robot must go to its location; once awake, this new robot is available to assist in awaking other robots. The objective is to minimize the "makespan", which is the time when the last robot is awakened.

This problem is reminiscent of the children's game of "freeze-tag", in which the person who is "it" tags a player, who becomes "frozen" until another player (who is not "it" and not "frozen") tags him to unfreeze him. Our problem arises in the situation in which there are a large number, n, of players that are frozen, and one (not "it") unfrozen player, whose goal it is to unfreeze the rest of the players as quickly as possible. Of course, as soon as a player gets unfrozen, he is available to assist in unfreezing other frozen players, so there is a cascading effect. Due to the similarity with this game, we have dubbed our problem the *freeze-tag problem* (FTP).

Other applications of the FTP arise in the context of distributing data (or some other commodity), where physical proximity is required to allow producing a copy. This may be because wireless communication is too costly in terms of bandwidth or because there is too much of a security risk. How does one propagate the data to the entire set of participants in the most efficient manner?

In this paper, we give the first algorithmic results on this problem, which arises naturally as a hybrid of various problems from the areas of broadcasting, routing, scheduling, and network design. As in *broadcasting problems*, the goal is to disseminate information in a network. The FTP also has elements of optimal *routing*, since robots must travel to awaken others or to pass off information. *Scheduling* issues arise in having to assign priorities to sleeping robots due to the limited but growing number of awake robots. One must decide whether to awaken a small nearby cluster to obtain a small number of helpers quickly or whether to awaken a distant but populous cluster to obtain many helpers, but after a longer delay. The relationship to *network design* is given by the fact that finding an optimal schedule can be interpreted as determining a spanning binary tree of

*Dept. of Appl. Math. and Statistics, SUNY Stony Brook, NY 11794-3600, {estie, jsbm}@ams.sunysb.edu.

†Dept. of Computer Science, SUNY Stony Brook, NY 11794-4400, bender@cs.sunysb.edu.

‡Abt. für Mathematische Optimierung, TU Braunschweig, 38106 Braunschweig, Germany, s.fekete@tu-bs.de.

§Institut für Mathematik, TU Berlin, 10623 Berlin, Germany, skutella@math.tu-berlin.de.

minimum depth in a complete weighted graph. Finally it should be noted that due to the practical motivation of the problem (e.g., in robotics), there is interest in considering *on-line versions*, where each robot can only see its immediate neighborhood in the graph.

There is an abundance of prior work on the dissemination of data in a graph. Most closely related to the FTP are the *minimum broadcast time problem*, the *multicast problem*, and the related *minimum gossip time problem*. See [5] for a survey; see [1, 8] for recent approximation results. However, there are significant differences: While the broadcast problem can be solved in polynomial time in tree networks, the FTP turns out to be NP-hard already on the seemingly easy class of weighted stars. Also, while it is fairly straightforward to obtain an on-line algorithm that is $O(\log n)$-competitive for the FTP, it is highly nontrivial to obtain an $o(\log n)$ approximation bound, even on some seemingly special graphs, such as trees.

Some of our results are specific to *star metrics* and *ultrametrics*, which arise as an important tool in obtaining approximation algorithms in more general metric spaces, as shown, e.g., in [2, 3, 7].

We also study a geometric variant of the problem where the robots are located at points of a geometric space and travel times are given by geometric distances. It turns out that geometry helps substantially in being able to compute good approximate solutions. For example, we give a nearly linear-time PTAS for robots embedded in any fixed-dimension L_p space.

1.1 Summary of Results:

(a) We prove that the freeze-tag problem is NP-hard, even for the case of star graphs with an equal number of robots at each vertex. Moreover, there exists a PTAS in this case and we analyze the greedy heuristic, showing a tight performance bound of 7/3.

(b) We give an $o(\log n)$-approximation algorithm for the FTP in ultrametrics. Specifically, we obtain an approximation ratio of $2^{O(\sqrt{\log \log n})}$, based on a series of nontrivial structural results. (Our approximation factor may be novel among existing approximation algorithms; it is a rather unusual term, asymptotically lying between $\log \log n$ and $\log n$.)

(c) We give a simple linear-time on-line algorithm that is $O(\log \Delta)$-competitive for the case of general weighted graphs of maximum degree Δ. On the other hand, we show for the off-line problem that finding a solution within less than 5/3 is NP-hard, even for graphs of maximum degree 5.

(d) We give a PTAS for geometric instances of the FTP in fixed-dimension domains endowed with an L_p metric. In fact, our algorithm runs in near-linear time, $O(n \log n + 2^{poly(1/\varepsilon)})$, with the nonpolynomial dependence on ε showing up only as an additive term in the time complexity.

1.2 Preliminaries. Let $R = \{v_0, v_1, \ldots, v_{n-1}\} \subset \mathcal{D}$ be a set of n *robots* in a *domain* $\mathcal{D}$. We assume that the robot at v_0 is the *source robot*, who is initially awake; all other robots are initially "asleep". We let $d(u,v)$ indicate the distance between two points, $u, v \in \mathcal{D}$. We study two cases, depending on the nature of the domain $\mathcal{D}$:

(i) The space $\mathcal{D}$ is specified by a graph $G = (V, E)$, with edge weights obeying the triangle inequality. The robots v_i correspond to a subset of the vertices, $R \subseteq V$, possibly with several robots at a single node. Distances $d(u,v)$ are measured according to the length of a shortest path in G. If G is a tree where all leaves have the same depth, then the shortest path lengths induce an ultrametric on V.

(ii) The space $\mathcal{D}$ is a d-dimensional geometric space with distances measured according to an L_p metric. (We concentrate here on Euclidean spaces, but our results apply more generally.) This is the case of a complete graph G, with edge weights given by the (L_p) distances between pairs of points V.

A solution to the FTP can be described by a *wake-up tree* $\mathcal{T}$ which is a directed binary tree, rooted at v_0, spanning all robots R. If a robot r is awakened by robot r', then the two children of robot r in this tree are the robots awakened next by r and r', respectively. Our objective is to determine an optimal wake-up tree, $\mathcal{T}^*$, that minimizes the *depth*, that is the length of the longest (directed) path from v_0 to a leaf (point of R). We also refer to the depth as the *makespan* of a wake-up tree. We let t^* denote the optimal makespan (the depth of $\mathcal{T}^*$). Thus, the FTP can also be succinctly stated as a graph optimization problem: In a complete weighted graph G (the vertices correspond to robots and edge weights represent distances between robots), find a binary spanning tree of minimum depth that is rooted at a given vertex v_0 (the initially awake robot).

We conclude the introduction by noting that one readily obtains an $O(\log n)$-approximation for the FTP. We say that a strategy is *nonlazy* if each awake robot claims and goes to an asleep unclaimed robot, if one exists, at the moment that he awakes.

PROPOSITION 1.1. *Any nonlazy strategy for the FTP achieves an approximation ratio of* $O(\log n)$.

Proof. Any nonlazy strategy can be transformed into a strategy having $\lceil \log n \rceil$ *stages*, such that at the end of stage $i < \lceil \log n \rceil$ there are exactly 2^i awake robots (some of which may have been temporarily waiting for the last one of the group to be awakened). Each stage involves each robot traveling a distance of at most the diameter, $diam(\mathcal{D})$, of the domain, to reach the next robot it is to awake. The claim follows by noting that a lower bound on the optimal makespan, t^*, is given by $diam(\mathcal{D})/2$ (or, more precisely, by the maximum distance from the source v_0 to any other point of $\mathcal{D}$).

2 Star Graphs

In this section we consider the FTP on weighted stars with one awake robot at the central node, v_0. In the simplest case, all edges of the star have the same length (we have a centroid *ultrametric*), while the number of robots at each node may vary. We show that the following greedy algorithm is optimal in this case: Each awake robot at the root v_0 lays claim to the leaf having the most (asleep) robots, and goes there to awaken them (and then all awake robots return simultaneously to v_0). The proof of optimality uses two exchange arguments and is omitted in this abstract.

LEMMA 2.1. *There is an exact greedy algorithm for awakening all the robots in a star with all edges of the same length.*

The rest of this section discusses the situation in which edge lengths of the star vary. As it turns out, this already makes the problem NP-hard, even if we have the same number of sleeping robots at each leaf. This nicely illustrates an important distinction between the FTP and broadcasting problems [8], which can be solved to optimality in polynomial time for the case of trees.

In the following subsections 2.1, 2.2, 2.3 we assume that there is an equal number, q, of robots at each leaf node. The general scenario will be discussed in subsection 2.4

We say that a robot has "visited" an edge and its leaf, if it has either been sleeping there or traveled there. We will often make use of the following observation, which follows from another simple exchange argument.

LEMMA 2.2. *For any instance of the FTP on stars, with an equal number of robots at each leaf vertex, there exists an optimal solution such that the lengths of the edges visited by a robot are non-decreasing over time.*

2.1 Performance of a greedy algorithm. Now we analyze a natural greedy algorithm: When an awake robot arrives at the root v_0, it chooses the shortest (unawakened and unclaimed) branch to awaken next. Interestingly, this natural greedy algorithm is *not* optimal. Consider a 2^{k+1}-edge example with one asleep robot at each leaf. There are $2^k - 1$ edges of length 1, 2^k edges of length k, and one edge of length $3k$. The greedy algorithm first awakens all robots at short edges. Thus, at time $2k$ exactly 2^k robots meet at the root, where they are matched to the edges of length k. Then, at time $3k$, one robot has to travel back to the root and is sent down to the last sleeping robot at the edge of length $3k$, where it arrives at time $7k$.

On the other hand, an optimal solution completes no later than time $3k + 4$: Send one robot down the longest edge at time 2, and another robot down an edge of length k at time 4, effectively awakening all short edges while those two long edges are traversed. At time $2k+4$ all short edges and one edge of length k have been awakened and 2^k robots have traveled back to the root (one robot is still traveling down the edge of length $3k$ and then arrives at time $3k+2$). We can thus use $2^k - 1$ of them to awaken the remaining edges of length k by the time $3k + 4$. Therefore, for large k the ratio of the greedy solution and the optimal solution tends to 7/3.

As it turns out, this is already a worst-case example for the greedy algorithm.

THEOREM 2.1. *For the FTP on stars with the same number of robots at each leaf, the performance guarantee of the greedy algorithm is 7/3, and this bound is tight.*

In order to prove the theorem, we first show the following theorem, which is of independent interest. We define the *completion time* of a robot to be the earliest point in time when it is awake and resting thereafter (no longer in motion).

THEOREM 2.2. *The greedy algorithm minimizes the average completion time of all robots.*

Proof. The proof is based on an exchange argument. Details are contained in the full paper.

Proof. [Theorem 2.1] Let n be the number of edges and q be the number of sleeping robots at each leaf. In particular, an instance contains $N := 1 + n \cdot q$ robots.

Since the average completion time is always a lower bound on the largest completion time, it follows from Theorem 2.2 that the average completion time $\bar{C}$ of the greedy solution is a lower bound on the optimal makespan.

Without loss of generality we may assume that at any point in time, a robot only moves if it will later awaken another robot. This implies that all robots will stop at leaves. Moreover, we can assume that for all

but one leaf, either all $q+1$ robots leave the leaf after awakening, or they all stay put. To be more precise, we can assume that there are $p := \lfloor N/(q+1) \rfloor$ leaves with $q+1$ robots, while the remaining $N - p(q+1) < q+1$ robots stay at one additional leaf.

We assume that the edges e_i, $i = 1, 2, \ldots, n$, are indexed in order of non-decreasing lengths $l(e_i)$, and the greedy algorithm visits the edges in this order. Thus, the greedy algorithm terminates as soon as a robot r arrives at the end of edge e_n and the previous leaf visited by this robot is given by edge e_{n-p}. Let T denote the time when robot r left the leaf of edge e_{n-p} in order to travel back to the root and then to the end of edge e_n. The makespan of the greedy solution is $T + l(e_{n-p}) + l(e_n)$. By construction, T is a lower bound on the completion time of each robot in the greedy solution. Thus, we get $T \leq \bar{C} \leq t^*$ and $l(e_n) \leq t^*$. It remains to be shown that $l(e_{n-p}) \leq t^*/3$.

At the end of the optimal solution there are p leaves with $q+1$ robots. Therefore, there must be an edge e_i with $i \in \{n-p, n-p+1, \ldots, n\}$ and less than $q+1$ robots at its end, because $|\{n-p, n-p+1, \ldots, n\}| = p+1$. Without loss of generality, one robot must have traveled down this edge in order to unfreeze the q robots at its end and then traveled back to the root in order to travel down another edge e_j. Lemma 2.2 yields $l(e_j) \geq l(e_i)$, so that the value of any solution is at least $2 \cdot l(e_i) + l(e_j) \geq 3 \cdot l(e_{n-p})$. Thus, $t^* \geq 3 \cdot l(e_{n-p})$, implying that the makespan of the greedy solution is at most $t^* + (t^*/3) + t^* = 7t^*/3$, completing the proof.

We conclude by noting a result that will be needed for the proof of Theorem 2.4.

COROLLARY 2.1. *The makespan of the greedy solution is at most* $t^* + 2l_{\max}$ *where* $l_{\max} := \max_i l(e_i)$.

2.2 NP-hardness.

THEOREM 2.3. *The FTP is strongly NP-hard, even for the special case of weighted stars with one (asleep) robot at each leaf.*

Proof. Our reduction is from NUMERICAL 3-DIMENSIONAL MATCHING (N3DM) [4]. **Instance:** Disjoint sets W, X and Y, each containing n elements, a size $a_i \in Z^+$ for each element $i \in W$, a size $b_j \in Z^+$ for each element $j \in X$, a size $c_k \in Z^+$ for each element $k \in Y$, and a target number $d \in Z^+$.
Question: Can $W \cup X \cup Y$ be partitioned into n disjoint sets $S_1, S_2, \ldots, S_n$, such that each S_h contains exactly one element from each of W, X, Y and such that for $1 \leq h \leq n$, $a_{i_h} + b_{j_h} + c_{k_h} = d$?

For technical reasons we assume without loss of generality that the size of each element from $W \cup X \cup Y$ is at most d. Moreover, we can assume without loss of generality that $n = 2^Q$ for some $Q \in N$ — the number of elements in W, X, and Y can be increased to the nearest power of 2 by adding elements of size d to W and elements of size 0 to X and Y; notice that this does not affect the value "yes" or "no" of the instance.

Let ε be a sufficiently small number ($\varepsilon < 1/(2Q)$ suffices), and let L be sufficiently large, e.g., $L := 15d$. Consider a designated root node with an awake robot, and attach the following edges to this root:

- $n-1$ edges of length ε; E denotes the robots at these leaves, along with the robot at v_0.
- n edges of length $\alpha_i := a_i/2 - \varepsilon Q + d$, $i = 1, \ldots, n$; A denotes the robots at these "A-leaves".
- n edges of length $\overline{\alpha_i} := L - a_i - 2d$, for $i = 1, \ldots, n$; $\overline{A}$ denotes the robots at these "$\overline{A}$-leaves".
- n edges of length $\beta_j := b_j/2 + 2d$, for $j = 1, \ldots, n$; B denotes the robots at these "B-leaves".
- $2n$ edges, two each of length $\gamma_k := L - 7d + c_k$, for $k = 1, \ldots, n$; C denotes the set of robots at these "C-leaves".

We claim that there is a schedule to awaken all robots within time L, if and only if there is a feasible solution for the N3DM instance.

It is straightforward to see that the "if" part holds.

To see that a feasible schedule implies a solution of the N3DM instance, we employ a combination of counting numbers of robots, local exchange arguments, and a careful case analysis to establish that the existence of a feasible solution guarantees one where each robot in E travels to one A-leaf, then to a B-leaf, and finally to a C-leaf. The time at which a robot in E who has visited edges of length α_i and β_j arrives at a C-leaf at distance γ_k is $L - d + a_i + b_j + c_k$. Therefore, a schedule that awakens all robots by time L implies a partition $S_1, \ldots, S_n$ with $a_{i_h} + b_{j_h} + c_{k_h} = d$ for all h.

Details can be found in the full paper.

2.3 A PTAS. We give a PTAS for the FTP on weighted stars with one awake robot at the central node, v_0, and an equal number q of sleeping robots at each leaf. The underlying basic idea is to partition the set of edges into "short" and "long" edges. The lengths of the long edges are rounded such that only a constant number of different lengths remains. The approximate positions of the long edges in an optimal solution can then be determined by complete enumeration. Finally, the short edges are "filled in" by a variant of the greedy algorithm discussed in subsection 2.1. During each step

we may lose a factor of $1+O(\varepsilon)$, such that the resulting algorithm is a $(1+O(\varepsilon))$-approximation algorithm, so we get a polynomial-time approximation scheme.

Similar techniques have been applied for other classes of problems before, e.g., in the construction of approximation schemes for machine scheduling problems (see [6]). However, the new challenge for the problem at hand is to cope with the awakened robots at short edges whose number can increase geometrically over time.

Let $T \le t^*$ be a good lower bound on the makespan, t^*, of an optimal solution. For our purpose, we can set T to 3/7 times the makespan of the greedy solution, which can be determined in polynomial time. For a fixed constant $\varepsilon > 0$, we partition the edges E into *short* edges, $S := \{e \in E \mid l(e) \le \varepsilon T\}$, and *long* edges $L := \{e \in E \mid l(e) > \varepsilon T\}$. We modify the given instance by rounding up the length of each long edge to the nearest multiple of $\varepsilon^2 T$.

LEMMA 2.3. *The optimal makespan of the rounded instance is at most* $(1+O(\varepsilon))t^*$.

Proof. See the full paper.

Any solution to the rounded instance induces a solution of the original instance that is not worse. Therefore, it completely suffices to construct a $(1+O(\varepsilon))$-approximate solution to the rounded instance. In the following we only work on the rounded instance.

LEMMA 2.4. *There exists a* $(1+\varepsilon^2)$*-approximate solution meeting the requirement of Lemma 2.2, such that for each long edge the point in time it is entered by a robot (its "start time") is a multiple of* $\varepsilon^2 T$.

Proof. An optimal solution can be modified to meet the requirement of the lemma as follows. Whenever a robot wants to enter a long edge, it has to wait until the next multiple of $\varepsilon^2 T$ in time. By Lemma 2.2, and since all lengths of long edges are multiples of $\varepsilon^2 T$, this modification increases the makespan of the solution by at most $\varepsilon^2 T$.

In the remainder of the proof we only consider solutions meeting the requirement of Lemma 2.4. Notice that both the number of different lengths of long edges and the number of possible start times are in $O(1/\varepsilon^2)$. The positions of long edges in an optimal solution can be described by specifying for each possible start time and each edge length the number of edges of this length that are started at that time. Since each such number is bounded by the total number of edges n, there are at most $n^{O(1/\varepsilon^4)}$ possibilities, which can be enumerated in polynomial time.

It remains to fill in all short edges. This can be done by a generalization of the greedy algorithm that only considers short edges after all long ones have been fixed. Technical details are involved, since we need to show that the generalized greedy procedure does not run out of robots traveling on short arcs before all short arcs have been taken care of. See the full paper for details. This yields the following:

LEMMA 2.5. *There is a generalized greedy procedure that yields a solution whose makespan is at most* $t^* + 4\varepsilon T \le (1+4\varepsilon)t^*$.

THEOREM 2.4. *There exists a polynomial-time approximation scheme for the FTP on stars with the same number of robots at each leaf.*

2.4 Any Number of Robots at Each Leaf. We give a constant-factor approximation algorithm for the case of a *centroid metric*, in which leaves may have various numbers of asleep robots and edge lengths of the star also vary. Interestingly, we obtain this $O(1)$-approximation algorithm by merging two "natural" but poorly performing algorithms:

(1) The "Shortest Edge First" (SEF) strategy, where an awake robot at v_0 considers the set of shortest edges leading to asleep robots, and chooses one with a maximum number of asleep robots.

(2) The "Repeated Doubling" (RD) strategy, where the edge lengths traversed repeatedly (roughly) double in size, and in each length class, edges are selected by decreasing number of asleep robots.

LEMMA 2.6. *The SEF strategy leads to a* $\theta(\log n)$*-approximation.*

Proof. Consider a tree having one edge of length $1+\varepsilon$ that leads to a leaf with $n-1$ robots, and $n-1$ edges, each of length 1, with one robot at each of the corresponding leaves. The Shortest-Edge-First strategy has makespan $\theta(\log n)$, whereas an optimal strategy has makespan $O(1)$.

Next we consider the Repeated Doubling strategy. A robot at v_0 faces the following dilemma: should the robot choose a short edge leading to a small number of robots (which can be awakened quickly) or a long edge leading to many robots (but which takes longer to awaken)? There are examples that justify both decisions, in which a wrong decision can be catastrophic.

We begin our analysis by assuming that all branches have lengths that are powers of 2. This assumption is justified because we can take an arbitrary problem

and stretch all the arms by at most a factor of 2. Now any optimal solution for the original problem becomes a solution to this modified problem, in which the makespan is increased by factor of 2. Thus, a k-approximation to the modified problem is a $2k$-approximation to the original problem.

Thus, we have reduced the problem on general stars to the problem on stars whose edge lengths are powers of 2. We partition the edges into *length classes*. Within each length class, it is clear which edge is the most desirable to awaken: the one housing the most robots. However, how can we choose which length class the robot should visit first? Suppose that an optimal algorithm chooses an edge of length 2^ℓ. We can visit edges of lengths $1, 2, 4, 8, \ldots, 2^\ell$ and only increase the makespan by a factor of 3. That is, we use repeated doubling to hedge our bets about what is the best path to take next. However, if the right choice to make is to awaken robots in a nearby length class, then we may suffer by sending robots on a repeated doubling trajectory to long edges.

Thus, consider the following algorithm. When a robot is first awakened, it awakens the most desirable edge in length class $1, 2, 4, 8, \ldots$. When it runs out of length classes, it starts the repeated doubling process anew. This algorithm may have poor performance:

LEMMA 2.7. *The RD strategy yields an $\theta(\log n)$-approximation.*

We now merge these two previous strategies to obtain what we call the *Tag-Team Algorithm*: When a robot is first awakened, it awakens one edge in each length class $1, 2, 4, 8, \ldots$. However, before each doubling step, the robot awakens the shortest edge that it can find. (When the robot runs out of length classes, it starts the repeated doubling process anew. Naturally, the robot skips any length class no longer containing edges.)

THEOREM 2.5. *The tag-team algorithm gives a 14-approximation for the FTP on centroid metrics (general stars).*

Proof. See the full paper.

3 An $o(\log n)$-Approximation Algorithm for Ultrametrics

An *ultrametric* for the set R of robots is defined by a rooted tree with the special property that every path from the root v_0 to a leaf has the same length (the *height* of the ultrametric), and all robots are placed at leaves.

One of our main results is an $O(2^{O(\sqrt{\log \log n})})$-approximation algorithm for ultrametrics. To appreciate how small this factor is, note that it is smaller than $O(\log^\epsilon n)$, for any constant $\epsilon > 0$. We rely on several lemmas, whose proofs are in the full paper. Lemma 5.2 (given later) shows that we can cluster leaves of the ultrametric together, resulting in the following:

LEMMA 3.1. *We can reduce the freeze-tag problem on an ultrametric to the freeze-tag problem on an ultrametric restricted as follows: if the ultrametric has height h, then the shortest distance from a leaf to an internal node is $O(h/\log^3 n)$.*

We next show how to convert the freeze-tag problem on an ultrametric to a freeze-tag problem on an ultrametric with at most ℓ levels. The degradation in the approximation ratio will depend on ℓ.

LEMMA 3.2. *Consider an ultrametric (as given by Lemma 3.1) where the height is h and the shortest distance between internal nodes is $O(h/\log^3 n)$. We can convert this ultrametric to an ultrametric having ℓ levels only increasingly the internal distances by a factor of $O(\log^{4/\ell} n)$.*

We now show how to solve the freeze-tag problem on ultrametrics having ℓ levels. Our approximation ratio will be a function of ℓ. We will ensure that our solutions have a specific form. To understand this restriction, first consider an ultrametric of height h with two levels (besides the root). Our solution is divided into *phases* of length $4h$, We require our solution to obey the following restrictions:

(1) At the end of each phase all the robots regroup at the root.

(2) Each robot can only visit one subtree in each phase.

LEMMA 3.3. *Suppose that a solution to the freeze-tag problem on an ultrametric of two levels and the root has length T. We can convert this solution into a solution obeying the restrictions (1) and (2) and having length at most $4T$.*

Now we consider the freeze-tag problem on an ultrametric with ℓ levels. The idea is to divide the problem into *phases*, *subphases*, *subsubphases*, etc, where the length of a subphase depends on the number of subtrees.

LEMMA 3.4. *Suppose that a solution to the freeze-tag problem on an ultrametric with ℓ levels has makespan T. Let the subtrees in the ultrametric have height $h_1, h_2, h_3, \ldots, h_\ell$, where w.l.o.g., we can assume that $h_1|h_2, h_2|h_3, \ldots, h_{\ell-1}|h_\ell$.[1] Then there exists a solution, with the proper choice of parameters, obeying restrictions (1) and (2) and having makespan $O(\ell 2^\ell T)$.*

[1] $x|y$ means that x divides y.

COROLLARY 3.1. *Any k-approximation to the optimal solution obeying restrictions (1) and (2) above will yield an $O(k\ell 2^\ell)$-approximation to the unrestricted optimal solution.*

Next we show how to find an approximate solution obeying these restrictions. Our algorithm uses both greedy and dynamic programming strategies. From now on, we assume that our solutions obey the previous restrictions.

The motivation for dividing into phases is so that we can solve each phase optimally, and then greedily concatenate the phases to obtain an optimal solution. Unfortunately, the concatenation of optimal solutions to the phases does not yield an overall optimal solution. Even for two-level ultrametrics, there exist problems for which every combination of optimal phases yields a suboptimal solution. In fact, it may be worthwhile awakening a suboptimal number of robots in one phase in order to set us up to awaken more robots in the next phase. However, we can say the following:

LEMMA 3.5. *The greedy algorithm that solves each subphase optimally has makespan at most $2t^*$.*

Now we show how to obtain an optimal solution within a phase using dynamic programming. At the beginning of each subphase, we have a certain number of awake robots and we have to partition these robots using the subtrees of the ultrametric. Suppose that recursively we can solve the freeze-tag problem on each of the subtrees optimally. Then we use dynamic programming to solve the whole subphase optimally. Even though there are an exponential number of ways of partitioning the robots among the subtrees, we can find the optimal partition in polynomial time. The idea is to arrange the subtrees of the ultrametric $S_1, S_2, \ldots, S_j$. Then for every prefix of subtrees $S_1, S_2, \ldots, S_i$, for $i = 1, \ldots, j$, calculate the optimal solution when we have r robots for $r = 1, \ldots, n$. Thus there are nj subproblems in our dynamic program.

Using Lemma 3.5, if we conclude that if we double the number of subphases compared to optimal, we have at least as many awake robots as in an optimal solution. However, by doubling the number of subphases recursively, the overall makespan is increased by $O(2^\ell)$.

LEMMA 3.6. *Our algorithm approximates the optimal restricted solution to within a factor of $O(2^\ell)$.*

Now finally we optimize for the number of levels. If ℓ is too large, the 2^ℓ dominates. If ℓ becomes too small, then the number of levels in the ultrametric is not enough for a good approximation of the original ultrametric, since we may lose a factor of $O(\log^{4/\ell} n)$. Setting these quantities equal, we obtain $\ell = O(2^{\sqrt{\log\log n}})$. Since we lose this factor at several stages of our approximation algorithm, we obtain a factor of $O(2^{O(\sqrt{\log\log n})})$.

THEOREM 3.1. *The freeze-tag problem on ultrametrics has an $O(2^{O(\sqrt{\log\log n})})$-approximation algorithm.*

Proof. (1) By Lemma 5.2 (given later) we can cluster leaves that are close together and put them in a single leaf; i.e., we can take the time to awaken small subtrees greedily without substantially increasing the makespan. Now we only have to consider ultrametrics in which the maximum distance between robots is $O(h)$ and the minimum distance between robots is $O(h/\log^3 n)$.

(2) By Lemma 3.2 we can modify our ultrametric so that internal nodes are on the small number ℓ of levels. We will choose ℓ to be a small increasing function, only slightly larger than a constant. This rounding allows us to approximate distances, and consequently the makespan, by a factor of $O(\log^{4/\ell} n)$.

(3) Next we restrict ourselves to solutions that are divided into phases, subphases, subsubphases, etc., where the recursive depth of the subphases is the number ℓ of levels. At the end of each subphase the robots are required to return to the root of the appropriate subtree. This restriction, by Lemma 3.4 and Corollary 3.1, will add an extra factor of $O(\ell 2^\ell)$ to the approximation ratio.

(4) Now we use an algorithm that obeys this restriction and combines greedy approaches and dynamic programming. Thus, we allocate robots among subtrees of the ultrametric so that they can awaken the maximum number of robots in each subphase. By Lemma 3.5 this means recursively increasing the number of subphases and working twice as hard as optimal at every recursive level, and thus this entails a loss of $O(2^\ell)$ compared to the optimal makespan, by Lemma 3.6.

(5) Finally we optimize for the number of levels setting $\ell = O(2^{\sqrt{\log\log n}})$. By combining all of the approximation ratios in the algorithm we obtain an approximation ratio of $O(2^{O(\sqrt{\log\log n})})$, as promised.

4 General Graphs

Now we discuss the FTP on general graphs $G = (V, E)$ with nonnegative edge weights $l(e)$. We let $\delta(v)$ denote the degree of v in G.

4.1 A Competitive On-Line Algorithm. As the hardness proof 2.3 illustrates, even the presence of a single vertex v with a high degree causes the problem to be NP-hard, showing that the resulting choices may be difficult to resolve. This makes it plausible that the complete absence of high-degree nodes could make the

problem more tractable. As we will see later on, this is not the case: Even for graphs of maximum degree 5, finding a solution within 5/3 of the optimum is NP-hard.

However, it is not hard to see that a sufficient number, $r(v_i)$, of robots at each vertex v_i yields an easy problem:

LEMMA 4.1. *Suppose $r(v_0) \geq \delta(v_0)$ for the source node, and $r(v_i) \geq \delta(v_i) - 1$ at any other node in G. Then the FTP can be solved by breadth-first search.*

This observation is based on the simple fact that any node in a BFS tree has minimal possible distance from the root, making the depth of this tree a general lower bound on the makespan of a wake-up tree. In the situation described in the lemma, we have sufficiently many robots available to use a BFS tree as the wake-up tree, and the claim follows.

As we noted in the introduction, the on-line version of the FTP is of some interest. Using the fact that BFS uses only local information, we can achieve some useful results for such a scenario. For easier notation, we write $\Delta_G := \max\{\frac{\delta(v_0)}{r(v_0)}, \frac{\delta(v_i)-1}{r(v_i)}, i = 1,\ldots,n-1\}$.

THEOREM 4.1. *There is a linear-time on-line algorithm for the FTP on a general weighted graph G that guarantees a competitive ratio of $O(\log \Delta_G)$.*

Proof. The basic idea is to simulate a breadth-first search at each node: At any vertex v_i, use a greedy strategy to wake up all robots at vertices adjacent to v_i. This can be achieved with a binary tree of depth $\lceil \log \frac{\delta(v_0)}{r(v_0)} \rceil$ for the root, and $\lceil \log \frac{\delta(v_i)-1}{r(v_i)} \rceil$ for any other vertex, as the vertex used to enter v_i does not need to be awakened. Moreover, the greedy strategy implies that any edge e in the resulting wake-up tree can only be placed below edges f that satisfy $w_f \leq w_e$. This implies the claim.

Since the greedy algorithm on stars actually has a constant performance guarantee, it is tempting to expect that a constant approximation is achieved by this more general approach. As the example in Figure 1 shows, this is not the case, and the estimate in Theorem 4.1 is tight. For the example, in a local greedy solution, all neighbors at distance 1 from the root are awakened in time $O(\log \Delta)$, before the vertex v_k is reached. On the other hand, an optimal solution does not exceed $3(1+\varepsilon)$: First wake the robot at v_k in time $1+\varepsilon$, then (using another period of ε) wake all robots close to v_k, and finally use all these robots to wake up the remaining robots within time $2+\varepsilon$.

4.2 Hardness of Approximation. As it turns out, there is no realistic hope for a PTAS on general graphs

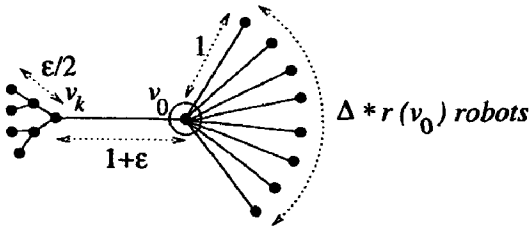

Figure 1: An example in which the local greedy procedure performs badly.

of bounded degree, even if we go beyond strictly local, (i.e., on-line) procedures:

THEOREM 4.2. *It is NP-hard to approximate the FTP on general weighted graphs within a factor less than 5/3, even for the case of $\Delta_G = 4$ and one robot at each node.*

Proof. The reduction is from 3SAT. See the full paper for the construction.

5 Freeze-Tag in Geometric Spaces

We now assume that the domain $\mathcal{D} = \Re^d$ and that the distance, $d(p_i, p_j)$, between the points $p_i, p_j \in P$ is the Euclidean distance (or any L_p metric). In this section, we give an efficient PTAS for the geometric freeze-tag problem. We begin by showing a constant-factor approximation, which will be one component needed in the PTAS; see the full paper for the proof.

THEOREM 5.1. *There is an $O(1)$-approximation algorithm, with running time $O(n \log n)$, for the geometric FTP in any fixed dimension d. The algorithm yields a wake-up schedule with makespan $O(diam(R))$.*

We give a $(1+\epsilon)$-approximation algorithm (PTAS) for the Euclidean (or L_p) freeze-tag problem in any fixed dimension. Our algorithm runs in nearly linear time, $O(2^{poly(1/\epsilon)} + n \log n)$. It is important to note that the exponential dependence on $1/\epsilon$ appears additively in our time bound, not multiplying $n \log n$.

We divide the plane into a constant number of square tiles or *pixels*. Specifically, we rescale the coordinates of the n input points (robots), R, so that they lie in the unit square, and we subdivide the square into a m-by-m grid of pixels, each of side length $1/m$. (We will select m to be $O(1/\epsilon)$.) We say that a pixel is *empty* if it contains no robots.

Our algorithm is based on approximately optimizing over a restricted set of solutions, namely those for which all of the robots within a pixel are awakened before any robot leaves that pixel. Note that by Theorem 5.1, once one robot in a pixel has been awakened, all of the robots in the pixel can be awakened within additional time $O(1/m)$.

We now describe the algorithm. We select an arbitrary *representative* point in each nonempty pixel. We pretend that all robots in the pixel are at this point, and we enumerate over all possible wake-up trees on the set, P, of representative points. (If there are ℓ robots in a given pixel, then we only enumerate wake-up trees whose corresponding outdegree at that pixel is at most $\min\{m^2-1, \ell+1\}$.) Since there are only a constant number of such trees (at most $2^{O(m^2 \log m)}$, since $|P| \leq m^2$), this operation takes time $2^{O(m^2 \log m)}$. We say that a wake-up tree is *pseudo-balanced* if each root-to-leaf path in the tree has $O(\log^2 m)$ nodes. Among those wake-up trees for P that are pseudo-balanced, we select one, $\mathcal{T}_b^*(P)$, of minimum makespan, $t_b^*(P)$. We convert $\mathcal{T}_b^*(P)$ into a wake-up tree for *all* of the input points R by replacing each $p \in P$ with an $O(1)$-approximate wake-up tree for points of R within p's pixel, according to Theorem 5.1. This step takes total time $O(n \log n)$. The total running time of the algorithm is therefore $O(2^{O(m^2 \log m)} + n \log n)$. Correctness is established in the following lemmas.

LEMMA 5.1. *There is a choice of representative points P such that the makespan of an optimal wake-up tree of P is at most $t^*(R)$.*

Proof. For each pixel, we select the representative point to be the location of the first robot that is awakened in an optimal solution, $\mathcal{T}^*(R)$, for the set of all robots. Then, for this choice of P, the spanning subtree of P within $\mathcal{T}^*(R)$ defines a feasible wake-up tree for P of makespan no greater than that of $\mathcal{T}^*(R)$ (namely, $t^* = t^*(R)$).

The following lemma is also important in proving Lemma 3.1:

LEMMA 5.2. *Suppose there exists an awakening tree, $\mathcal{T}$, having makespan t. Then, for any $\mu > 0$, there exists a pseudo-balanced awakening tree, $\mathcal{T}_b$, of makespan $t_b \leq (1+\mu)t$.*

Proof. First we perform a heavy path decomposition (see, e.g., [9]) of the tree $\mathcal{T}$. For each node v, let $d(v)$ be the number of descendents. Consider a node u with children $v_1, \ldots, v_k$. The edge (u, v_i) is *heavy* if v_i has more descendents than any other child of u; that is, $i = \arg\max_j d(v_j)$.If (u, v_j) is a *light* (non-heavy) edge, then at most half of u's descendents are v_j's descendents; that is, $d(v_j) \leq d(u)/2$. Thus, in any root-to-leaf path in $\mathcal{T}$ there are at most $\log n$ light edges. Also, heavy edges form a collection of disjoint paths. We say that a heavy path π' is a *child* of heavy path π if one end node of π' is the child of a node in π. The heavy path decomposition forms a balanced tree of heavy paths, since any root-to-leaf walk in $\mathcal{T}$ visits at most $\log n$ light edges, and therefore at most $\log n$ heavy paths.

We can use these heavy paths to refine the description of the wake-up tree. We can assume that each heavy path is awakened by one robot, the one that awakens its *head* (node closest to v_0) and that no robot awakens more than one heavy path.

Since $\mathcal{T}$ has makespan t, each heavy path has length at most t. We divide the heavy path into subpaths of length $\xi = \mu t / \log n$. Note that on any root-to-leaf path in $\mathcal{T}$, we visit at most $O((1 + 1/\mu) \log n)$ different subpaths. In the original wake-up tree, all nodes in one length ξ subpath are awakened by a single robot. Thus, by definition, a robot δ units from the beginning of the subpath is awakened δ units after the beginning (head) of the subpath. In our modified solution, the robots in a length ξ subpath share in the collective awakening of all the robots in the subpath. Thus, we guarantee that all of the robots are awake and back in their original (asleep) positions by time ξ units after the first robot in the subpath is originally awakened. Thus, a robot δ units from the beginning of the subpath is only guaranteed to be awake ξ units after the robot at the beginning of the subpath is awakened, which could entail a total delay of ξ over the original awakening.

We awaken a subpath as follows. We consider the subpath to be oriented from "left" (the head, closest to source v_0) to "right". The first robot r_1, at the left end of the subpath, travels along the subpath until the last (asleep) robot, r_2, before position $\xi/3$, if such a robot r_2 exists. If robot r_2 exists, then r_2 is sent leftwards with the responsibility to awaken all asleep robots in the interval (r_1, r_2), and this subproblem is solved recursively: Given an awake robot (r_2) at the end of the interval (r_1, r_2), he is responsible for initiating the awakening of all robots in the interval, and all robots must return to their initial positions. If no robot r_2 is encountered by r_1 before position $\xi/3$, then we use r_1 to solve recursively the subproblem $(\xi/3, \xi)$. When a subproblem's length drops below $\mu\xi/\log n$, we resort to a different wake-up strategy, as follows. The responsible robot, r, goes to the median robot of the subproblem and awakens it, and continues in his same direction. The robot he just awakened goes in the opposite direction

and recursively does the same thing, heading for the median in his subproblem, etc.

Consider a heavy path composed of α subpaths of length ξ. Consider any robot at position δ along the heavy path. The original wake-up tree will awaken this robot δ units after the first robot of the heavy path. The new solution may awaken this robot as much as $(1+\mu)(\delta+\xi)$ time units after the first robot of the heavy path. Since there are at most $\log n$ heavy paths on any root-to-leaf walk and the length of a subpath $\xi = \mu t/\log n$, the accumulated delay in the makespan is $(1+\mu)(t+\log n(\mu t/\log n)) = t(1+\mu)^2$.

On any root-to-leaf path in $\mathcal{T}$ there are at most $O(\log n)$ subpaths. Each of these subpaths in our new wake-up tree is transformed into a wake-up subtree of height $O(\log n)$. Thus, on any root-to-leaf path in the new wake-up tree there are at most $\log^2 n$ nodes, and therefore our wake-up tree is pseudo-balanced.

LEMMA 5.3. *For any two choices, P and P', of the set of representative points, $t_b^*(P) \leq t_b^*(P') + O((\log^2 m)/m)$.*

Proof. Pixels have size $O(1/m)$ and there are at most $O(\log^2 m)$ awakenings in each root-to-leaf path of a pseudo-balanced tree; thus, any additional wake-up cost is bounded by $O((\log^2 m)/m)$.

A similar proof yields:

LEMMA 5.4. *For any pseudo-balanced wake-up tree of P, there exists a wake-up tree, $\mathcal{T}(R)$, with makespan $t(R) \leq t_b(P) + O((\log^2 m)/m)$.*

THEOREM 5.2. *There is a PTAS, with running time $O(2^{O(m^2 \log m)} + n \log n)$, for the geometric FTP in any fixed dimension d.*

Proof. The time bound was already discussed. The approximation factor is computed as follows. By the lemmas above, the makespan, t, of the wake-up tree we compute obeys:

$$\begin{aligned} t &\leq t_b^*(P) + O((\log^2 m)/m) \\ &\leq t_b^*(P') + 2 \cdot O((\log^2 m)/m) \\ &\leq t_b(P') + O((\log^2 m)/m) \\ &\leq (1+\mu)t^* + O((\log^2 m)/m) \\ &\leq t^* \left(1 + \mu + \frac{C \log^2 m}{m}\right) \\ &\leq t^*(1+\epsilon), \end{aligned}$$

for appropriate choices of μ and m, depending on ϵ. (We also used the fact that $t^* \geq diam(R) \geq 1$.)

6 Conclusion

We conclude with some open problems suggested by our initial results:

1. Is there an $o(\log n)$-approximation algorithm for the freeze-tag problem in general graphs?
2. Is the FTP in low-dimensional geometric spaces NP-hard?
3. Is there an $o(\log n)$- (or, ideally, an $O(1)$-) approximation algorithm for the FTP for points in a polygon, where distances are measured according to the length of a shortest path in the polygon? Such an algorithm would apply also to the case of FTP in general trees.

Acknowledgements. We thank Tien-Ruey Hsiang, Nenad Jovanovic, and Marcelo Sztainberg for many helpful discussions on the topic of this paper.

References

[1] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Multicasting in heterogeneous networks. In *Proc. 30th Annual ACM Symposium on Theory of Computing*, pages 448–453, 1998.

[2] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proc. 30th Annual ACM Symposium on Theory of Computing*, pages 161–168, 1998.

[3] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 379–388, 1998.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[5] S. M. Hedetniemi, T. Hedetniemi, and A. L. Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *Networks*, 18:319–349, 1988.

[6] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM*, 34:144–162, 1987.

[7] G. Konjevod, R. Ravi, and F. Salman. On approximating planar metrics by tree metrics, Manuscript (submitted) 1997.

[8] R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Proc. 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 202–213, 1994.

[9] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*, SIAM, 1983.