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, s/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

*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, D-38106 Braunschweig, Germany, s.fekete@tu-bs.de.

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

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 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 [7] for a survey; see [1, 10] for recent approximation results. However, that 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. And although it is fairly straightforward to obtain an on-line algorithm that is $O(\log n)$-competitive, 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, 9].

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.

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 non-polynomial dependence on ε showing up only as an additive term in the time complexity.

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 $O(\log n)$.*

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.

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 [10], 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 2.3 *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 2.4 *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 Appendix. □

Proof: [Theorem 2.3] 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 2.4 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, implying 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 2.10.

Corollary 2.5 *The makespan of the greedy solution is at most* $t^* + 2l_{\max}$ *where* $l_{\max} := \max_i l(e_i)$.

2.2 NP-hardness.

We now prove that the FTP is hard.

Theorem 2.6 *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) [6]. Details can be found in the Appendix. □

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, e.g., see [8]. 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 $T \leq t^*$ 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, $S := \{e \in E \mid l(e) \leq \varepsilon T\}$, 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 2.7 *The optimal makespan of the rounded instance is at most* $(1 + O(\varepsilon))t^*$.

Proof: See the Appendix. □

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 2.8 *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 2.8. 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 Appendix for details. This yields the following:

Lemma 2.9 *There is a generalized greedy procedure that yields a solution whose makespan is at most* $t^* + 4\varepsilon T \leq (1 + 4\varepsilon)t^*$.

A proof can be found in the Appendix. We summarize:

Theorem 2.10 *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 2.11 *The SEF strategy leads to a* $\theta(\log n)$*-approximation.*

Lemma 2.12 *The RD strategy yields an* $\Omega(\log n)$*-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 2.13 *The tag-team algorithm gives a 14-approximation for the FTP on centroid metrics (general stars).*

Proof: See the Appendix. □

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 Appendix. Lemma 5.3 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 $h/\log^3 n$.*

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 $4T$.*

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 $h_1|h_2, h_2|h_3, \ldots, h_{\ell-1}|h_\ell$. Then there exists a solution, with the proper choice of parameters, that obeys the restrictions (1) and (2) from above and has makespan $O(\ell 2^\ell T)$.*

Corollary 3.5 *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, this solution approach does not work because a concatenation of optimal phases does not yield an 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 3.6 *If the makespan of an optimal solution is t^*, then 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 3.6, 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 3.7 *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 3.8 *The freeze-tag problem on trees has an $O(2^{O(\sqrt{\log \log n})})$-approximation algorithm.*

Proof: (1) By Theorem **??**, it suffices to consider ultrametrics. We round the ultrametric so that it approximates a cascade of ultrametrics, whose heights are powers of 2, all connected at the root v_0.

(2) By Lemma 5.3 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 $O(h/\log^3 n)$.

(3) 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 $O(\log^{4/\ell} n)$.

(4) 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 3.5, will add an extra factor of $O(\ell 2^\ell)$ to the approximation ratio.

(5) 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 3.6 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 3.7.

(6) 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 2.6 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 $\Delta_G := \max\{\frac{\delta(v_0)}{r(v_0)}, \frac{\delta(v_i)-1}{r(v_i)}, i = 1,\dots, n-1\}$.

Theorem 4.2 *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 a constant approximation is achieved by this more general approach. As Example A.2 in the Appendix shows, this is not the case, and the estimate in Theorem 4.2 is tight.

4.2 Hardness of Approximation

As it turns out, there is no realistic hope for a PTAS on general graphs of bounded degree, even if we go beyond strictly local, (i.e., on-line) procedures:

Theorem 4.3 *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. Without loss of generality, we assume that we have $n = 2^Q$ variables. For technical reasons, we add n clauses of size 2 of the form "x or not x", one for each variable.

This instance will be mapped to an FTP instance on a weighted graph of bounded degree with one robot per vertex, such that we have a solution of makespan $3/2 + O(\varepsilon \log n)$ if there is a satisfying truth assignment, and a makespan of at least 5/2 if there is no such truth assignment. By choosing $\varepsilon = o(\log n)$, this implies the claim.

See the Appendix for details of 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 Appendix 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 size $1/m$. (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 5.2 *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 5.3 *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., [11]) 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. $\square$

Lemma 5.4 *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 $O((\log^2 m)/m)$. $\square$

A similar proof yields:

Lemma 5.5 *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 5.6 *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 $t^* \geq diam(R) \geq 1$.) $\square$

References

[1] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Multicasting in heterogeneous networks. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 448–453, New York, May 23–26 1998. ACM Press.

[2] Bartal. On approximating arbitrary metrics by tree metrics. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 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 *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages –, 1998.

[4] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[5] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[6] 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.

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

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

[9] G. Konjevod, R. Ravi, and F. Salman. On approximating planar metrics by tree metrics, 1997.

[10] R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In S. Goldwasser, editor, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 202–213, Los Alamitos, CA, USA, Nov. 1994. IEEE Computer Society Press.

[11] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

Appendix: Proof Details

Proof of Theorem 2.4. The proof is based on an exchange argument. Consider an arbitrary solution minimizing the average completion time. Assume that at some point in time a robot enters an edge e_1 that has a length larger than that of a shortest available edge e_2. Therefore, e_2 is chosen at a later point in time. If, in the tree corresponding to the given solution, e_2 lies in the subtree below e_1, an exchange of the two edges decreases the average completion time, contradicting the optimality of the solution under consideration.

For $i = 1, 2$, let n_i denote the number of edges in the subtree T_i that is formed by e_i and all its descendants. If $n_1 > n_2$, then exchanging the two edges does not increase the average completion time. On the other hand, if $n_1 \leq n_2$, then exchanging the two subtrees T_1 and T_2 within the whole tree does not increase the average completion time. By iterating this exchange argument, we finally arrive at the greedy solution, which must therefore be optimal with respect to the average completion time. □

Proof of Theorem 2.6. We give a reduction of NUMERICAL 3-DIMENSIONAL MATCHING (N3DM), as stated in [6]:

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 $S_1, S_2, ..., S_n$, 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 $\alpha_i := a_i/2 - \varepsilon Q + d$, $i = 1, \dots, n$; A denotes the robots at these "A-leaves".
- n edges of length $\overline{\alpha_i} := L - a_i - 2d$, for $i = 1, \dots, n$; $\overline{A}$ denotes the robots at these "$\overline{A}$-leaves".
- n edges of length $\beta_j := b_j/2 + 2d$, for $j = 1, \dots, n$; B denotes the robots at these "B-leaves".
- $2n$ edges, two each of length $\gamma_k := L - 7d + c_k$, for $k = 1, \dots, n$; 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: Using a binary tree on the set E (a "greedy cascade on E"), we can bring all n robots in E to the root at time $2\varepsilon Q = 2\varepsilon \log n$. These n robots are sent to the A-leaves. Now there are $2n$ robots available, two each will get back to the root at time $a_i + 2d$, $i = 1, \dots, n$. One of each pair is sent down the edge of length $\overline{\alpha_i}$, so that the whole set $\overline{A}$ gets woken up just in time L. The remaining n robots (one for each a_i, call this robot A_i) get sent to wake up the robots of set B, such that A_{i_h} is assigned to an edge of length β_{i_h}, if a_{i_h} and b_{j_h} belong to the same set S_h. This gets two robots for each h (say, $A_{i_h}^{(1)}$ and $A_{i_h}^{(2)}$) back to the root at time $a_{i_h} + b_{j_h} + 6d$. Send those two robots down the two edges of length γ_{k_h}. If $a_{i_h} + b_{j_h} + c_{k_h} = d$, then all robots in C are awake at time L.

To see that a feasible schedule implies a solution of the N3DM instance, first observe that no robot in $F = \overline{A} \cup C$ can wake up any other robot, as the corresponding edges are longer than $L/2$. Moreover, the same argument implies that no two robots in F can be awakened by the same robot. Since the total number of robots is precisely $2|F| = 6n$, we conclude that each robot in $E \cup A \cup B$ must wake up a different robot in F.

Clearly, no robot in B can wake up a robot in $\overline{A}$ by the deadline L. Thus, the robots in $\overline{A}$ are awakened by a set of n robots $\tilde{A} \subset E \cup A$. Notice that a robot in $\tilde{A}$ can neither visit a B-leaf nor two A-leaves and still meet the deadline.

The $2n$ robots in C must be awakened by the $2n$ robots in $B \cup (E \cup A) \setminus \tilde{A}$. Since none of them has enough time to visit two B-leaves, each must visit exactly one B-leaf and then, by Lemma 2.2, travel immediately to a C-leaf. We can assume without loss of generality (by a simple exchange argument) that each pair of robots that has visited the same B-leaf is assigned to a pair of C-leaves at the same distance.

As explained above, each robot in $\tilde{A}$ can visit at most one A-leaf; the same is true for all robots in $(E \cup A) \setminus \tilde{A}$, because each must visit one B-leaf and one C-leaf afterwards. Since there are $2n$ robots in $E \cup A$ and also $2n$ visits to A-leafs, each robot in $E \cup A$ must visit exactly one A-leaf.

We next argue that, without loss of generality, a feasible solution uses a greedy cascade in the beginning to bring all n robots in E to the root at time $2\varepsilon Q$. As described above, the greedy cascade guarantees that, later, each pair of robots returning from an A-leaf at distance α_i arrives at the center node at time $a_i + 2d$. On the other hand, such a pair cannot arrive before time $2\alpha_i = a_i + 2d - 2\varepsilon Q > a_i + 2d - 1$. Since the deadline L as well as all remaining travel times to $\overline{A}$-leaves or B and C-leaves are integral, the claim follows.

A simple exchange argument yields that, without loss of generality, the robots in $\overline{A}$ are woken up by the robots in A, i. e., $\tilde{A} = A$. Thus, 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. □

Proof of Lemma 2.7. Consider the tree corresponding to an optimal solution of the original instance. Notice that on any path starting at the root node of the tree, there can be at most $O(1/\varepsilon)$ long edges. In the rounding step we increase the length of a long edge by at most $\varepsilon^2 T$. Therefore the length of any path, and thus the completion time of any robot in the solution given by the tree, is increased by at most $O(\varepsilon) \cdot T$. Since T is a lower bound on the optimum, the claim follows. □

Proof of Lemma 2.9. Consider the following generalization of the greedy procedure that proceeds over time.

We set $S_{p+1} := \infty$ and $i := 1$ in the beginning. Assume that a robot, coming from a short edge, gets to the central node at time t.

- If $t \geq S_i + 2\varepsilon T$, then send the robot into the i-th subtree and set $i := i + 1$.
- Else, if there are still short edges to be visited, then send the robot down the shortest of those edges.
- Else, if $i \leq p$, then send the robot into the i-th subtree and set $i := i + 1$.
- Else stop.

The most difficult task in the proof is to show that the greedy procedure yields a feasible solution. Unfortunately, it is not obvious that the greedy procedure does not run out of robots traveling on short arcs before every short arc has been taken care of.

While the greedy procedure is running, we simultaneously construct a modified instance and a (partial) solution to this instance. When the greedy procedure sends a robot into the i-th subtree, we replace the first edge of this subtree in the modified instance by a new edge whose length equals the length of the shortest currently available edge and assign the robot to it. Moreover, we delete all long edges of the i-th subtree in the modified instance. Then, whenever a robot from the modified subtree arrives at the center node, we add a new edge to the modified instance whose length equals the length of the shortest currently available short edge (if any) and assign the robot to it.

Notice that the constructed (partial) solution for the modified instance is a greedy solution. Moreover, the only difference to the (partial) solution for the original instance is that we have replaced each subtree of long edges by a subtree of short edges.

Claim A.1 *In the solution constructed by the greedy procedure for the original instance, there is always a robot traveling on short edges as long as there are still short edges to be visited.*

In particular, this claim implies that the greedy procedure yields a feasible solution.

Proof: [Claim A.1] Assume by contradiction that at time t all awake robots work in subtrees of long edges although there are still short edges to be visited.

Let k be the number of awake robots in the greedy solution for the modified instance at time t. Notice that, by our choice of t, all k robots belong to one of the modified subtrees. The optimal solution to the original instance induces a solution σ for the modified instance by replacing each subtree of long edges by the corresponding subtree of short edges from the greedy solution of the modified instance.

We claim that there are at least $k + 1$ awake robots in σ at time $t - 2\varepsilon T$. Since each subtree is started at least $2\varepsilon T$ units of time earlier than in the greedy solution (by construction of the greedy procedure), the number of awake robots in these subtrees at time $t - 2\varepsilon T$ is at least k. Moreover, since all short edges from the original instance must be awakened in σ, there must be at least one more awake robot at time $t - 2\varepsilon T$.

In order to get $k + 1$ awake robots, $\lceil k/q \rceil$ leaves must have been visited. Consider the instance that we get by taking the $\lceil k/q \rceil$ shortest edges of the modified instance. It follows from the discussion in the last paragraph that an optimal solution to this subinstance takes at most a time of $t - 2\varepsilon T$. On the other hand, the makespan of the greedy solution is larger than t. This is a contradiction to Corollary 2.5, because $l_{\max} \leq \varepsilon T$. $\square$

Notice that the length of the time interval between two visits to the central node of a robot who travels on short edges is at most $2\varepsilon T$. It thus follows from Claim A.1 that a robot is sent into the i-th subtree before time $S_i + 4\varepsilon T$, for $i = 1, \ldots, p$. As a consequence, the robots at each long edge are awake before time $opt + 4\varepsilon T$. Finally, the same argument as in the proof of Claim A.1 shows that all robots at short edges are awake at time $opt + 2\varepsilon T$. This completes the proof of the Lemma 2.9. $\square$

Proof of Theorem 2.13. We loose a factor of 2 by assuming that all edge lengths are powers of 2.

Now we restrict ourselves to this special case and consider an optimal solution, given by a wake-up tree $\mathcal{T}^*$. (For ease of argument, we will assign priorities to branches that have the same number of robots.) We show by induction that if $\mathcal{T}^*$ awakens an edge e at time t, then the tag-team algorithm awakens this edge e at or before time $7t$.

Suppose that in $\mathcal{T}^*$ at time t a robot r awakens all the robots $r_1, r_2 \ldots r_k$ at the end of an edge e, where e has length $L(e)$. Consider the next edge that each of $r_1, r_2 \ldots r_k$ awakens in $\mathcal{T}^*$. Specifically, suppose that in $\mathcal{T}^*$, r_i travels to an edge e_i of length $L(e_i)$. That is, in $\mathcal{T}^*$ at time $t + L(e) + L(e_i)$, r awakens the robots at the end of edge e_i.

Now we consider when these robots are awakened in the Tag-Team Algorithm. By induction, suppose that robot r was awakened at time $7t$. In the tag-team algorithm each of the awakened robots $r_1, r_2 \ldots r_k$ performs a repeated doubling trajectory, ultimately awakening the edge in the appropriate length class. That is, either r_i awakens e_i by time $7t + L(e) + 2 \cdot 2 \cdot (1 + 2 + 4 + \cdots L(e_i)/2) + 2L(e_i) + L(e_i) \leq 7t + L(e) + 7L(e_i)$ or edge e_i' was already awakened. That is, even without knowing the edge class where r_i should go directly, it gets there eventually. But what about the original robot r, which continues its repeated doubling trajectory to larger edges when in $\mathcal{T}^*$ r visits a smaller edge? Robot r plays tag-team awakening at least one robot on the smallest edge possible, and this new robot performs r's duties. Thus, this robot in time $\leq 7t + L(e) + 7L(e_i)$ awakens the edge class that r would awaken in $\mathcal{T}^*$. $\square$

Proof of Lemma 3.1. We modify an ultrametric of height h as follows: consider any subset of leaves that form an ultrametric of height $h/\log^3 n$, and collapse or cluster all of these leaves into a single leaf containing all of the robots. Now we have an ultrametric satisfying the requirements of the lemma. Now we show that a k-approximation algorithm for this modified ultrametric yields an $O(k + O(1))$-approximation algorithm for the original ultrametric. Whenever we awaken a leaf in the modified ultrametric we will awaken the associated subtree in the original ultrametric. Since this subtree has height $h/\log^3 n$, it can be awakened (by Proposition 1.1) in time $O(h/\log^2 n)$. Therefore, the awakening of a single node in the modified ultrametric corresponds to awakening a small subtree of the original ultrametric (time $O(h/\log^2 n)$). However, by Lemma 5.3 there exists an approximately optimal solution to the freeze-tag problem on an ultrametric, such that each robot awakens at most $\log^2 n$ nodes. Therefore, this expansion will only increase the makespan by $O(h)$. Since there is a lower bound of $O(h)$ on the makespan, the lemma follows. $\square$

Proof of Lemma 3.2. We create the internal nodes at heights $h/\log^{4i/\ell} n$, for $i = 0 \ldots \ell - 1$. We now modify the ultrametric by gradually raising the heights of the LCAs of the leaves. Specifically, we gradually move the LCA of two nodes closer to the root until the LCA is that one of the allowed heights for internal nodes. (Note that the LCA may merge with other LCAs that are moving toward the root.) Therefore the maximum factor that any distance between two nodes will be increased is $O(\log^{4/\ell} n)$. $\square$

Proof of Lemma 3.3. We first modify our optimal solution into a solution that obeys the first restriction. Thus, every $2h$ steps we splice in a trip to the root. That is, every $2h$ steps the robots stop whenever they are doing, travel to the root, and then $2h$ steps later are back where they started. This extra travel increases the makespan by a factor of 2. Note that the robots still may not obey the second restriction. However, we know that now in each phase (consisting of $2h$ worth of active work and $2h$ worth of trips to and from the root) the robots can be in at most two subtrees. Therefore, we can increase the makespan by another factor of 2 by splitting each phase into two phases and guaranteeing that each robot is in at most one subtree per phase. $\square$

Proof of Lemma 3.4. We first transform the solution so that it obeys the first restriction. We begin with the smallest trees and make our solution obey the restriction for these trees, and then work outward to the largest subtrees until restriction (1) is obeyed at each subtree. For the smallest trees we choose the constants as before, where a phase has $2h_1$ of active work and $2h_1$ work traveling to and from the appropriate root. In other phases for subtrees of height h_i, we have $2h_i$ active work and an amount of active work that depends on the number of smaller subtrees. The lengths are determined so that in the original solution, for any subphase corresponding to a tree of height h_i, there is exactly $2h_i$ work from the original solution and the rest of the work in the phase corresponds to the time traveling to and from roots in the subtrees. Thus, at the bottom level, this creates a subphase of length $4h_1$, at the next level, $6h_2$, then $8h_3$, etc. As in the two-level solution, we start with the original solution but every time some subphase approaches its end, each robot travels to the appropriate roots and then back to where they were originally working. Since the size of the largest phase is $O(\ell h)$, we have a solution obeying restriction (1) with makespan $O(\ell T)$.

Now we modify the solution so that it also obeys restriction (2). For the same reason as in Lemma 3.4, and by induction, we only visit two subtrees in any subphase. (We should verify that this is true. This is not obvious because the phase is now $O(\ell h)$, but is made of a fixed number of subphases. It should be true by induction, but if not is no big deal because we can just change the lengths of the phases and use a slightly different argument.) We convert this solution into one obeying the second restriction by recursively splitting one subphase into two subphases and adding idle time. Since there are ℓ levels, we recursively increase the makespan to $O(\ell 2^\ell T)$. $\square$

Proof of Lemma 3.6. [sketch] We argue that we need at most twice the number of phases as a greedy strategy that awakens as many robots per phase as possible. The proof is by induction: We claim that after an optimal strategy has performed k phases, resulting in N_k awake robots, the greedy strategy will, within its first $2k$ phases, awaken at least N_k robots. $\square$

Example A.2 *The estimate in Theorem 4.2 is tight.*

See Figure 1. 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 $2+\varepsilon$.

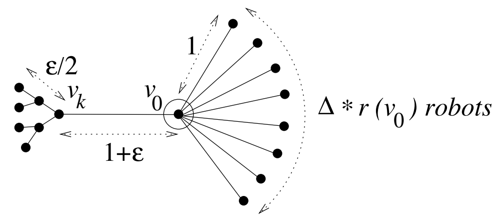

Figure 1: An example in which the local greedy procedure performs badly.

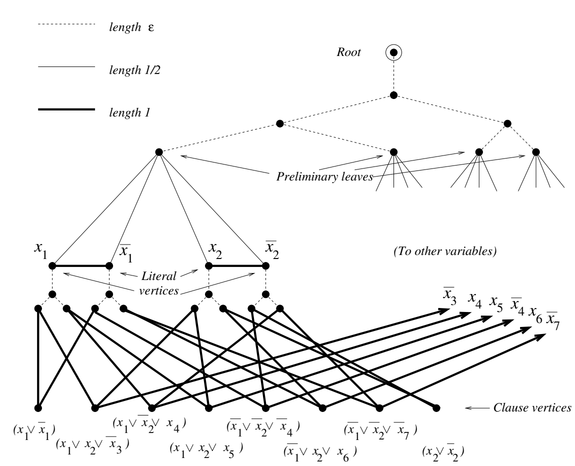

Figure 2: NP-hardness of 5/3-approximation of freeze tag in general graphs.

Details of Theorem 4.3. See Figure 2. From the root, build a binary tree of depth $\varepsilon \log(n/2)$, resulting in two alive robots at each of the $n/2$ *preliminary leaves*, after time $\varepsilon \log(n/2)$.

Next group the n variables in an arbitrary way to $n/2$ pairs, and assign a pair to each preliminary leaf. Each pair of variables is represented by four more vertices, two corresponding to "true", two corresponding to "false". All get connected to the respective preliminary leaf, using an edge of length 1/2. The two vertices for the same variable get connected to each other, using an edge of length 1.

If $c(x)$ is the number of clauses in which some literal x occurs, attach a small binary tree of height $O(\varepsilon)$ to allow $c(x)+1$ alive robots at cost $O(\varepsilon)$ when the literal node is reached. (This does not affect the overall Δ.)

Finally, add one vertex per clause (including the artificial ones stated above.) Using an edge of length 1, connect each literal vertex to the vertices representing the clauses in which the literal occurs.

Now a truth assignment induces a wake-up tree of makespan $3/2 + O(\varepsilon \log n)$. After going through the initial binary tree, we have n alive robots at the preliminary leaves. For each variable, pick the node corresponding to the literal in the given truth assignment. Use $c(x)$ of the robots close to this literal to wake up all corresponding clause nodes, and the remaining robot to wake up the counterpart $\overline{x}$ of x.

Conversely, it is not hard to see that any solution of makespan $3/2+O(\varepsilon \log n)$ induces a satisfying truth assignment. Furthermore, if there is no such truth assignment, it follows from our construction that a makespan of at least 5/2 is necessary, proving our claim. $\square$

Proof of Theorem 5.1. For each of the points $v \in R$, we consider K sectors defined by rays emanating from the points at angles $0, 2\pi/K$, $2(2\pi/K)$, $3(2\pi/K)$,.... Let $u_j(v)$ denote the point (if any) of R in the jth sector that is closest to v; if there are no points of R in the jth sector of v, then $u_j(v)$ is undefined. We can compute the $u_j(v)$ points, for all j and all $v \in R$, in total time $O(Kn \log n)$, using standard Voronoi diagram-based methods [4].

We sort the points $u_j(v)$ by distance from v; let these points be $u_1, u_2, \ldots, u_{K'}$, in sorted order by distance from v. The wake-up strategy we employ is as follows: Once the robot at v is unfrozen, he follows the path $v, u_1, u_2, \ldots, u_{K'}$, awakening the nearest robot in each of the $K' \leq K$ nonempty sectors about him. (Of course, some of these robots may have already been awakened before he gets to their (initial) positions. This potentially saves him the effort of going to all of these nearby neighbors, allowing for some possible further improvement in our constant factor.)

We now analyze the performance of this algorithm. Let $G_K = (R, E_K)$ be the graph that links each point v to the points u_j that are its nearest neighbors in the K sectors about v. Such a graph G_K is known as a *θ-graph*, for $\theta = 2\pi/K$, and is known to be a t-spanner for values of $K \geq 9$ (e.g., see [5]). This means that distances in the graph G_K approximate to within a constant factor the Euclidean lengths of the edges in the complete graph G on R.

Assume that the robot at point v_ℓ is the last one to be unfrozen by our algorithm. What is the path length of the "signal" (the unfreezing tag) in getting from v_0 to v_ℓ? We know that if some point v is reached by the signal by distance t, then any neighbor, $u_j(v)$, of v in the graph G_K is reached by distance $\leq t + \xi$, where ξ is the length of the path $v, u_1, u_2, \ldots, u_j$; thus, $\xi \leq (2j-1) \cdot d(v, u_j) \leq (2K-1) \cdot d(v, u_j)$. Thus, the signal will reach v_ℓ within a distance of at most $(2K-1)$ times the distance from v_0 to v_ℓ in G_K. Since, for constant $K \geq 9$, distances in G_K approximate distances in the Euclidean plane, up to a constant depending on K, we have shown that the signal gets to v_ℓ within distance $O(d(v_0, v_\ell))$, showing that we have an $O(1)$-approximation, since $d(s, p_\ell)$ is a lower bound on the optimal makespan, t^*. $\square$