A Neighborhood Search Technique for the Freeze Tag Problem

Dan Bucantanschi[1], Blaine Hoffmann[2], Kevin R. Hutson[1], R. Matthew Kretchmar[1]

[2]College of Information Sciences and Technology, Penn State University, University Park, PA 16802
Email: bhoffman@ist.psu.edu
[1]Department of Mathematics & Computer Science, Denison University, Granville, Ohio 43023
Email: hutsonk@denison.edu, kretchmar@denison.edu, bucata_d@denison.edu

April 17, 2006

Abstract

The Freeze Tag Problem arises naturally in the field of swarm robotics. Given n robots at different locations, the problem is to devise a schedule to activate all robots in the minimum amount of time. Activation of robots, other than the initial robot, only occurs if an active robot physically moves to the location of an inactive robot. Several authors have devised heuristic algorithms to build solutions to the Freeze Tag Problem. Here, we investigate an update procedure based on a hill-climbing, local search algorithm to solve the Freeze-Tag Problem.

Subject Classifications: Metaheuristics, degree-bounded minimum diameter spanning trees, swarm robotics, neighborhood structure, improvement graph, combinatorial optimization

1 Introduction

Consider the following problem that arises in the field of swarm robotics ([6]). Suppose there are n robots placed in a d-dimensional space. Starting with one initially active robot and the other $n-1$ robots inactive, the goal is to "awaken" the inactive robots so that all n robots are awakened in the fastest time possible. Robot x can awaken robot y only by physically moving to the location of robot y. The cost for robot x to awaken robot y is determined by the straight-line, geometric distance between x's current position and y's position; though not considered here, other variants of this problem constrain robots to travel only on weighted edges of a prescribed graph. Once a robot becomes active, it can assist in awakening the remaining dormant robots. The goal is to compute an optimal awakening schedule, i.e. a schedule that minimizes the time to activate all robots, also known as the *makespan*. Arkin, et. al. [6] dubbed the problem the Freeze-Tag Problem (FTP) for its similarities to a children's game of the same name.

The problem can also be described in the following context. A telecommunications company would like to build a communications network for dissemination of information from a single source r to all of nodes of minimum total cost. In wanting to achieve a desired level of service quality, the company constrains itself to build a network with minimum longest path from r to allow for fast, reliable communication links between the source and the customers. Also, the company wants to limit the degree of each node in the network so as to more equally distribute the workload in routing information. A spanning tree network design is a minimum cost alternative for the company because it allows for the desired communcation without redundant network links. Hence the company desires to build a spanning tree network with bounded vertex degrees and minimum longest path from the source. Note that a solution to an instance of the FTP is a spanning tree respresenting the path that robots take with minimum longest path from the initially-awaken robot. At each subsequent awakening, two robots are able to disperse to activate other robots resulting in a degree bound of 3 on the spanning tree solution.

1.1 Related Work

As seen in the second example, the problem of waking a set of sleeping robots in the manner described by the FTP has similarities to various problems arising in broadcasting, routing, scheduling, and network design ([6]). These broadcasting and network design problems share elements of trying to determine how to efficiently disseminate information through a network. Similar problems have arisen in the past to model this data dissemination, such as the minimum broadcast time problem (see [10] for a survey), the multicast problem ([7, 8]), and the minimum gossip time problem ([17]). However, as shown in [6],while the minimum broadcast time problem can be solved in polynomial time in tree networks, the FTP remains intractible even on weighted star graphs.

In fact, much of the prior research on the FTP has focused on proving that it is NP-hard and on designing heuristic algorithms to solve it. Arkin, et. al. ([5]) prove that the FTP is NP-hard in unweighted graphs. These authors show that any "nonlazy" strategy yields an $O(\log n)$ approximation but that an $O(logn)$ approximation under the same strategy in general metric spaces is difficult to obtain. Sztainberg, et. al. ([19]) prove that a natural greedy heuristic applied to a geometric instance gives an $O((logn)^{1-\frac{1}{d}})$ approximation in d-dimensions. Their experiments using several heuristics described in Section 2, shows that this greedy approach performs well on a broad range of data sets and yields a small constant-factor approximation.

The FTP is closely related to a more general problem called the *Bounded Degree Minimum Diameter Spanning Tree Problem* (BDST). Introduced in [13], the BDST problem is stated as follows. Given an undirected complete graph $G = (V, E)$ with metric lengths c_{ij} for each $(i, j) \in E$ and bounds $B_v > 0$ on the degree of each $v \in V$, find a minimum-diameter spanning tree T so that for each $v \in V$ the tree-degree of each node v is not greater than B_v. A solution to a Freeze-Tag instance with makespan ρ corresponds to finding a degree-3-bounded spanning tree with longest root-to-leaf path ρ. We should note, however, that the FTP will always have an initial root node with degree 1 which, in general, does not apply to the BDST.

Konemann, et. al. ([12]) provide an $O(\sqrt{log_B n}) * \Delta$ approximation algorithm for the BDST, where B is the max-degree in the spanning tree and Δ is the minimum diameter of any feasible T. This algorithm provides the best general bound for the FTP as well. This algorithm will also be described in Section 2. Arkin, et. al. ([5]) propose an algorithm that performs better than the algorithm of [12] in some special cases. Namely, for the case where the graph is unweighted, these authors provide an $O(1)$-approximation.

1.2 Outline

The motivation of this paper is to propose a local hill-climbing strategy based on an update graph and search algorithm. We refer to this as the Alternating Path Algorithm for the way it searches a local neighborhood; we will show how this algorithm finds neighboring awakening schedules by alternately adding and removing edges from the current schedule. We compare the performance of the Alternating Path Algorithm on the Freeze-Tag problem against previously published results based on heuristics and combinatorial search strategies.

The paper is outlined as follows. In Section 2, we describe the existing heuristic methods (Greedy Fixed, Greedy Dynamic, BTFB, Opposite Cone, and Konemann's BDST algorithm) to build approximate solutions for the FTP and BDST. We also review two combinatorial search strategies based on genetic algorithms and ant algorithms. We then propose, in Section 3, the Alternating Path Algorithm that employs an improvement graph to take a given awakening schedule and update this solution to a schedule with decreased makespan that still satisfies the degree bounds on each vertex. Finally in Section 4 we present our experimental results.

2 Preliminaries

2.1 Notation

In this section, we describe the basic notation used in the rest of the paper. Some of the graph notation used, such as the definitions of graphs, trees, degrees, cycles, walks, etc., are omitted here, and the reader is referred to the book of Ahuja, Magnanti, and Orlin ([3]). Let $G = (V, E)$ be an undirected network where associated with each edge $e = (i, j)$ is a weight (perhaps metric distance) c_{ij}. Suppose that T is a rooted spanning tree of G with node r specially designated as the root node. Each arc $(i, j) \in E(T)$ denotes a parent-child relationship where i is the parent and j is the child of i. Under this terminology, r is an ancestor to every node in T. Let T_i denote the subtree of T rooted at i.

Associated with each $v \in V(G)$, let $\delta_v(G)$ (also $\delta_v(T)$) denote the degree of v in G (also T) and a number B_v to be a degree bound on v. That is, if T is a spanning tree solution for the FTP, we require $\delta_v(T) \leq 3$, $v \neq r$. Unique to the FTP, is the requirement that the root have degree 1, $B_r = 1$, since the root is the initially active robot and it can only travel to one other robot to awaken that. From that point forward, there will always be two robots (the already active one, and the newly active one) which can leave a node hence the requirement $\delta_v(T) \leq 3$ representing the path of the incoming robot and the two paths of the outgoing robots.

Recall, a path $P = \{v_0, e_0, v_1, e_1, \ldots, v_k - 1, e_{k-1}, v_k\}$ in G is a sequence of nodes and arcs with $e_i = (v_i, v_{i+1}) \in E$ so that no node is repeated. Given a path P, let $dist(P) = \sum_{i=0}^{k-1} c_{v_i, v_{i+1}}$.

2.2 Heuristic Solutions to the FTP

We now review some of the existing approaches for building a heuristic solution for the FTP and BDST. Since minimum spanning trees can be built by a greedy approach ([3]), it makes sense to apply this approach to the FTP. Simply stated, under a greedy awakening strategy, once a robot is awakened, it locates the nearest asleep robot to it and attempts to wake it. Sztainberg et. al. [19] explain though that any heuristic that attempts to build a solution from scratch in a greedy fashion must specify how conflicts among robots are resolved, since more that one robot might desire to wake the same neighboring robot. One way to avoid this conflict is to allow robots to claim their next target, and once an inactive robot is claimed by an active robot, it cannot be claimed by another. This greedy approach will be called Greedy Fixed (GF). Alternatively, one could allow claims to be refreshed as needed. Using a greedy approach, a newly active robot could renegotiate the claim of another robot if the newly awakened robot is closer to the claimed inactive robot. This approach combined with an offline delayed target choice to avoid physical oscillations of robots will be refered to as Greedy Dynamic (GD). Experimental results ([19]) show that the greedy dynamic outperforms other methods over a variety of data sets.

Konemann et. al. [12] design an algorithm based on clustering the vertices of the graph. Their idea is to partition the nodes into low-diameter components. They then form a balanced tree which spans the clusters and has a small number of long edges. This ensures that the components are connected by a low-diameter tree. Finally, for each component, a low-diameter spanning tree is found with max-degree B. This divide and conquer approach improves upon the theoretical runtime of the approaches in [19].

2.3 Genetic Algorithm Solution to the FTP

The genetic algorithm leverages the principles of Darwinian evolution to stochastically search a complex solution space. These algorithms begin with an initial population of awakening schedules.

Each schedule is evaluated by computing its makespan. Based on the makespan, solutions are probabilistically removed from the population (there is a higher probability of retaining schedules with better makespans). Some of the remaining awakening schedules are copied into the new population; others are combined to form new awakening schedules. Finally, mutations alter some of the solutions in the new population. As the cycle repeats with each subsequent generation, the overall fitness of the population increases and very good solutions are increasingly more likely to be discovered. We discuss those details of the genetic algorithm specific to the freeze tag problem. The interested reader is directed to [1] for complete details on the genetic algorithm.

For the cross-over operator, we employ a variant of genetic algorithm cross-over operators used in similar problems [11, 16]. Suppose two parent solutions, T_1 and T_2, are to be combined in cross-over to create a child solution. The child's root node is selected to be the root node in the parent with the smaller makespan. The child also contains all edges common to both parents. This results in a graph with k connected components, $T_1, T_2, \ldots, T_k$. For each i, if $|T_i| = 1$, the component is connected to other components by adding in the smaller of the two parent's edges used to connect this node. Lastly, if any robots were not connected to the awakening schedule in the prior step, typically due to incompatibilities of the parents' schedules that would result in cycles or disconnected schedules, the algorithm searched top-down for the first edge to connect a node to the forest. This is equivalent to trying to place robots in the awakening schedule earlier rather than later.

The child now replaces the parent with the larger makespan in the next generation of solutions while the parent with the smaller makespan is retained. The mutation operator randomly swaps edges between nodes in an awakening schedule.

For these series of experiments, the population size is either 200 or $2n$ where n is the number of robots in the problem instance (two series of experiments are performed with each population size). The number of generations is ?. Cross-over rate is 0.7 and mutation rate is set to 0.1. A solution's fitness is computed as $f(T) = \frac{\bar{\rho}}{\rho(T)}$ where $\rho(T)$ is the solution's makespan and $\bar{\rho}$ is the average makespan of the current population. For selection, each solution received at least $\lfloor f(T) \rfloor$ (the integer of its fitness); the remaining slots in the fixed-size generation were filled probabilistically using roulette selection on the fractional values of each solution's fitness (i.e. $f(T) - \lfloor f(T) \rfloor$).

2.4 Ant Colony Optimization Solution to the FTP

Ant algorithms derive their concept from the ways ants search for food. As described in [9], real-life ants use pheromone trails to guide themselves back and forth from food sources. Ants are attracted to pheromone trials with high levels of pheromone. Those trails which find the shortest path to food sources get traversed faster and therefore have their phermone levels reinforced more often.

For the Freeze-Tag problem, each of m ants are sent out to form an awakening schedule. Each ant will leave a pheromone trial in two places: on the edges of the graph to reflect the chosen awakening schedule and on the nodes of the graph to reflect whether both or only one robot left the node to awaken other robots. Each ant will choose the next robot to be awakened at node j from node i, at iteration t with some probability depending on the following:

1. Which nodes ant k has left to visit. Each ant k keeps track of a set J_i^k of nodes that still need to be processed.

2. The inverse distance d_{ij} from node i to node j called the visibility and denoted $\eta_{ij} = \frac{1}{d_{ij}}$.

3. The amount of virtual pheromone at iteration t representing the learned desirability of choosing node j when transitioning from node i.

4. The amount of virtual pheromone representing the learned desirability to choose to send $r = 2$ robots from node i to awaken other robots or to send only $r = 1$ robots.

3 A Neighborhood Search for the FTP

In this section, we introduce an improvement graph structure, similar to [4] and a search method to indicate attractive edge exchanges for solutions of the FTP. Given a solution T to the FTP, an improved solution $\overline{T}$ constitutes a degree-3 bounded tree whose maximum root-to-leaf path is smaller than that of T. More concretely, given any feasible solution $T = (V, E')$ to the FTP problem, with $E' \subseteq E$ and $|E| = |V| - 1$, let $N^{(k)}(T)$ be the set of feasible trees $\overline{T}$ which differ from T in exactly k edges. The sequence $N^{(1)}(T), N^{(2)}(T), \ldots, N^{(k_{max})}(T)$, $k_{max} \leq |V| - 1$, defines a so-called *neighborhood structure* ([14]) relative to T for the FTP problem. We seek to explore this neighborhood structure for a tree $\overline{T}$ whose makespan is less than T.

Many authors have introduced techniques to perform large-scale neighborhood searches in graphs, see for instance [14] and [2], [4],[14] , [18] . These methods have been applied to similar problems such as the degree-constrained MST problem [18] and the capacitated MST problem [4]. In the latter, the authors define a corresponding graph, called an improvement graph, that is used to indicate profitable exchanges of vertices and subtrees among multiple subtrees in a current solution to produce an improved solution. Unlike [14] which randomly generates trees in $N^{(i)}(T)$, using this improvement graph gives a deterministic approach to finding improved solutions in the neighborhood structure.

3.1 The Search Method

Our goal is to define an improvement graph and a search technique that allows us to find a tree $\overline{T} \in N^{(i)}(T)$, $i = 1, \ldots, k_{max}$, such that the maximum root-to-leaf path in $\overline{T}$ is less than that of the current solution T. To this end, let $v \in T$ with degree less than B_v be called a *candidate* vertex. Note for the Freeze Tag problem $B_v = 3$, $\forall v \in V$, and a vertex that has 0 or 1 children is a candidate. Let P_j^l be the path and $d^l[j]$ be the distance from vertex j in the tree T to the farthest leaf from j in T_j. Let $d^r[i]$ be the distance along edges in T from the root vertex r to the vertex i. Let $\rho(T)$ denote the length of the longest path in T from r We wish to find a tree T' with $\rho(T') \leq \rho(T)$.

We now define the specifics of this improvement graph. Let $G^1(T)$ be a directed graph with vertex set the same as T. Let (i, j) be a directed edge in $G^1(T)$ if either of the following conditions hold. If i is the parent of j in T, then (i, j) is a directed edge in $G^1(T)$ called a *tree edge*. If $(i, j) \notin T$, $j \neq r$, then (i, j) is a directed edge in $G^1(T)$ if a distance condition is satisfied; we call this type of edge a *nontree edge*. This distance condition will be discussed later. The case of $j = r$ is more complicated and will not be considered here.

We wish to show that an exchange of edges/subtrees in T corresponds to traversing an alternating path $AP = \{v_0, e_0, v_1, e_1, \ldots, v_k, e_k, v_{k+1}\}$ between tree edges ($E^T = \{e_0, e_2, \ldots, e_{k-1}\}$) and nontree edges ($E^N = \{e_1, e_3, \ldots, e_k\}$) in $G^1(T)$ beginning at a vertex v_0 in $P_l(T)$ and ending at a candidate vertex. If such an alternating path exists, in forming $\overline{T}$ from T we delete those traversed tree edges in $G^1(T)$ from T and add edges (j, i) to $\overline{T}$ when edges $(i, j) \in E^N$ are traversed. Define $\overline{E}^N$ to be such that $(j, i) \in \overline{E}^N$ whenever $(i, j) \in E^N$. We claim that $\overline{T} = T - E^T + \overline{E}^N$ is a spanning tree, rooted at r, such that for all $v \in \overline{T}$, $\delta_v \leq B_v$. To illustrate this, consider the following operations for exchanging edges.

Example 1: Child/Subtree Promote Let T_j be a subtree of T where $\delta_j(T) < B_j$, and let i (with parent $k \neq j$ be a grandchild of j connected to j by tree path $P_T(j, i)$ of distance $dist(j, i)$ such that $d^l[i] + c_{ij} + d^r[j] < \rho(T)$. Note, if the Triangle Inequality is satified by the edge weights, $c_{ij} < dist(j, i)$. The child-promote method would delete (k, i) and make j the parent of i. In the

improvement graph, since $d^l[i] + c_{ij} + d^r[j] < \rho(T)$, the link (i, j) would be a nontree edge in $G^1(T)$. Hence the alternating path $AP = \{k, (k, i), i, (i, j), j\}$ exists in $G^1(T)$.

Example 2: Child/Subtree Exchange In this operation, two vertices b and c (nonancestors) swap one of their children. This operation is shown in Figure 2 along with alternating cycle $(b - e - c - f - b)$ in the improvement graph. This operation can be extended easily to multiple edge swaps.

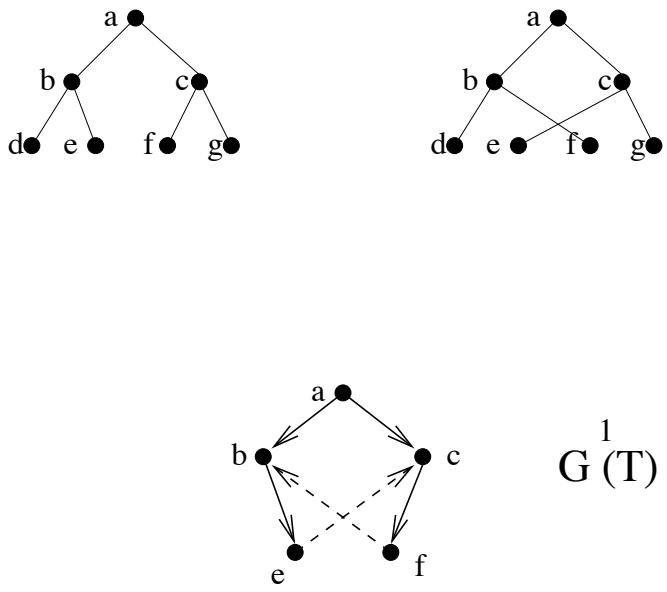

Figure 1: Illustration of Child Exchange and Corresponding Improvement Subgraph

We now wish to show by exchanging edges in this fashion that $\overline{T} = T - E^T + \overline{E}^N$ is a rooted spanning tree that satisfies all degree constraints.

Lemma 1 *$\overline{T}$ is a spanning tree rooted at r.*

Proof. Since $(j, r) \notin AP$ by construction, the root node cannot change. Thus, we need only to show that $\overline{T}$ is a graph with $n - 1$ edges and no cycles. By the construction of AP, $|E^T| = |\overline{E}^N|$, and hence $|E(\overline{T})| = |E(T)| - |E^T| + |\overline{E}^N| = |E(T)| = n - 1$.

To show $\overline{T}$ is cycle-free, suppose to the contrary that $\overline{T}$ contains a cycle C. Then C consists of edges in $T - E^T \subset T$ and edges in $\overline{E}^N$. If $C \cap \overline{E}^N = \emptyset$ then we have a contradiction since T is a spanning tree. Otherwise, there exists an edge $a_0 = (i, j) \in C \cap \overline{E}^N$. By the construction AP, there exists edge $e_0 = (m, j)$ that directly preceeds a_0 in AP and $e_0 \in E^T$. Since AP contains vertex j at most once (because it is a path), the edge $e_j = (l, j)$ that preceeds a_0 in C must be in $T - E^T$. Since $e_0, e_j \in E(T)$, j is a child of l and m indicating a cycle in T, a contradiction. Hence $\overline{T}$ is cycle-free, and thus $\overline{T}$ is a spanning tree. □

Lemma 2 *$\overline{T}$ satisfies $\delta_v \leq B_v$ for all $v \in V(\overline{T})$.*

Proof. Note that if $v \notin AP$, then $\delta_v(\overline{T}) = \delta_v(T) \leq B_v$. Also, if v_i is an interior vertex of AP, it gains a child (v_{i-1}) and loses a child (v_i) and thus $\delta_v(\overline{T}) = \delta_v(T) \leq B_v$. Hence, we need only to consider the end vertices of the path AP: v_0 and v_{k+1}. The vertex v_0 loses a child (v_1) so

$\delta_{v_0}(\overline{T}) < \delta_{v_0}(T) \leq B_{v_0}$. Further, by construction of AP, in T, $\delta_{v_{k+1}}(T) < B_{v_{k+1}}$. Thus, since v_{k+1} gains only one child, v_k, $\delta_{v_{k+1}}(\overline{T}) = \delta_{v_{k+1}}(T) + 1 \leq B_{v_{k+1}}$, and the result is shown. □

Given that the improvement graph can be used to generate alternating paths ending at a candidate vertex, one option for generating an improved solution is to successively find such paths and test whether such a path produces an edge exchange that would result in $\rho(\overline{T}) \leq \rho(T)$. Under this method, the only criteria for a nontree edge e to be included in $G^1(T)$ would be $e \in E(G)$. Another option would be to establish a distance criteria for nontree edge inclusion in $G^1(T)$ so that the improvement graph might be used to indicate attractive edge exchanges. One obvious criteria would be that if $(i,j) \notin T$, $j \neq r$, then (i,j) is a directed nontree edge in $G^1(T)$ if $d^l[j] + c_{ij} + d^r[i] < \rho(T)$.

This criteria, though, is not reliable under all search methods of $G^1(T)$. Consider a nontree edge $(u,v) \in G^1(T)$. If a search method for $G^1(T)$ traverses (u,v) then T_u will be attached to v in T'. However, this attachment could change the distance labels $d^l[j]$ for v and its ancestors and $d^r[i]$ for u and its descendants making their distance labels unreliable. To combat this, we attempt to search edges $(i,j) \in G^1(T)$ where $d^l[j] + c_{ij} + d^r[i] < \rho(T)$ is guaranteed to hold. To accomplish this, we restrict movement in $G^1(T)$ to be between disjoint subtrees. This restriction is enforced in [4] under a natural condition of a capacited minimum spanning tree. Here, more care is needed. Let P be an alternating path (or cycle) in $G^1(T)$, we say $P = \{v_0, e_0, v_1, e_1, \ldots, v_k, e_k, v_{k+1}\}$ between tree edges $E^T = \{e_0, e_2, \ldots, e_{k-1}\}$ and nontree edges $E^N = \{e_1, e_3, \ldots, e_k\}$ in $G^1(T)$. Then P is a *reliable path* if for each $e_i = (v_i, v_{i+1}) \in E^N$, every v_k, $k \geq i+1$, is neither an ancestor of v_{i+1} nor a descendant of v_i. Given a reliable path (cycle) in $G^1(T)$, we can show $\overline{T}$ is a degree-bounded rooted spanning tree with no greater maximum root-to-leaf path than T.

The improvement graph takes $O(n^2)$ time to construct since each vertex pair must be checked to determine whether the conditions for edge inclusion have been satisfied. The complexity of finding such an alternating path or cycle of length less than $2 \cdot k_{max}$ (indicating $\overline{T}$ has k_{max} edges different from T) is $O(k_{max} \cdot n)$ per node used as v_0. Searching for tree/nontree edges from each node involves just a scan of the edges emanating from the node taking $O(n)$ time per node. To check whether a vertex is an ancestor or descendant of a previously visited vertex, a $0-1$ ancestor and descendent array is employed. This makes this check excutable in $O(1)$ time. To produce an alternating path of length $2k_{max}$ then is accomplished in breadth-first fashion in $O(k_{max} \cdot n)$ time. We limit ourselves to choosing v_0 to lie on the path defining $\rho(T)$. So our search takes $O(k_{max} \cdot n^2)$ time. In implementation, we limit $k_{max} \leq 4$.

4 Experimental Results

In this section we illustrate the operation of the Alternating Path Algorithm by showing the edges that are added and removed along the alternating path. We then discuss implementation details of the algorithm such as search depth (previously referred to as k_{max}) and iterative improvement. We compare the results of the Alternating Path Algorithm against the other benchmarks noted in the previous work discussion. Finally, we note a quirk of the neighborhood structure searched by this particular hill climbing algorithm.

4.1 Illustrating the Alternating Path Algorithm

For this discussion, we use the Eil51 problem from the TSPLIB [20] which contains 51 robots dispersed in 2D space with $\{x,y\}$ coordinates in the range of $(0,0)$ to $(70,70)$. The location of these robots is shown in Figure 2. The numbered labels of these robots is arbitrary (set by the Eil51.tsp file) but are useful for referring to specific robots.

Since the Alternating Path Algorithm improves upon an existing solution to the FTP, we need an awakening schedule to use as a "seed" value. We use Greedy Dynamic (see Heuristic solutions

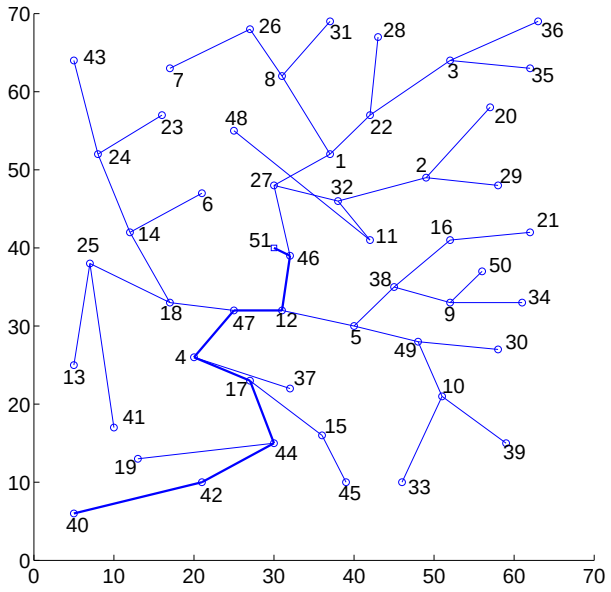

Figure 2: Greedy Dynamic's Solution for Eil51.tsp

discussed earlier) as an algorithm that provides a starting point with a good, but not nearly optimal awakening schedule. The structure of Greedy Dynamic's solution is shown in Figure 2. This awakening schedule has a makespan of $\rho(T) = 66.07$ seconds with the root node in the middle (rootID = 51). The longest path, indicated by the shaded edges, traverses robots 51, 46, 12, 47, 4, 17, 44, 42, and 40.

Mechanically, the Alternating Path Algorithm starts near the root at robot 46 (the root is not considered for edge swapping in this algorithm). It then tries to severe (remove) the edge between robot 46 and robot 12 along the longest path. After removing this edge, it will now have to add an edge from robot 12 to some other already-connected robot. Thus it continues constructing the alternating path by removing and adding edges until it adds a final edge at a candidate robot (one with 0 or 1 children). The algorithm exhaustively considers all such alternating paths; the search space is reduced considerably by eliminating obvious choices that would induce cycles and other undesirable features.

After searching for alternating paths starting between robots 46 and 12, it then progresses to the next edge along the longest path. It will next consider alternating paths formed between robots 12 and 47. Then robots 47 and 4. And so on until the last edge between robots 42 and 4 is considered. Of all these new trees generated, the best is recorded and is used as the solution returned by the Alternating Path Algorithm.

For the specific case in Figure 2, the best alternating path structure was found using the connection between robots 44 and 42. In Figure 3 we see the newly created awakening schedule. The edges which have been added are shown with heavy dashed lines. The edges which were removed are shown in light dotted lines. The alternating path discovered here consists of $\{ (44, 42),\ (42, 4),\ (4, 37),\ (37, 5),\ (5, 38),\ (38, 11)\}$ where the odd edges are the ones removed and the even edges are the ones added. Note, by the proof in the prior section, we do indeed end up

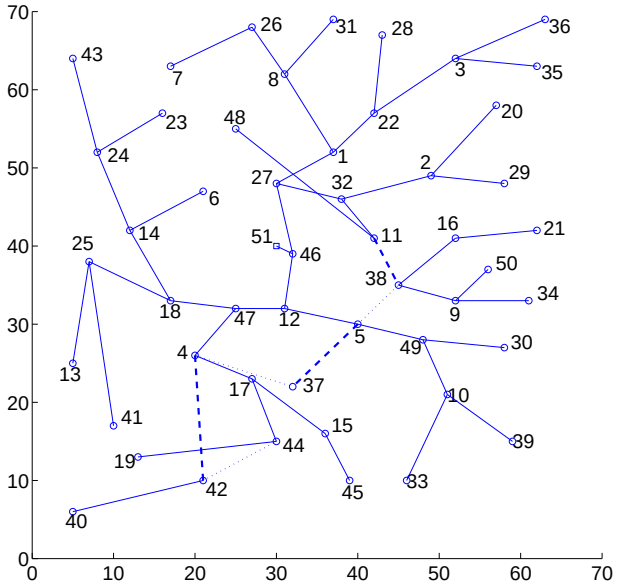

Figure 3: Illustration of Alternating Path Algorithm

with a structurally sound spanning tree.

The makespan of this new tree is $\rho(T') = 56.80$ seconds (compared with 66.07 seconds of the seed tree). Note that the longest path has changed, it now starts at the root and progresses toward the upper left corner: $(51, 46, 12, 47, 18, 14, 24, 43)$.

4.2 Depth Control and Iteration

As discussed previously, $N^{(k)}(T)$ is the neighborhood of all trees differing from T by k edges. This actually infers a pair of edges (one removed and one added). Thus the example awakening schedule illustrated in Figure 3 belongs to $N^{(3)}(T)$. By controlling the k parameter, we can limit the number of edge exchanges in our search process. Naturally, a larger k generally finds better solutions but at the cost of more computation and a quirky side-effect that we discuss in the next subsection.

Looking back at Figure 3, we again mention that the longest path has changed from the lower right corner to the upper left corner. An obvious question is why not apply the algorithm again to the new longest path to see if further gains can be realized? The *Iterative Alternating Path Algorithm* attempts just this process. It repeated applies the update search to successive solutions of improvement graphs until no further improvement can be realized. Thus Figure 3 is just the first iteration of this process.

We actually find that an additional 14 iterations are possible with incremental improvements at each step. The final awakening schedule, after 15 iterations of Alternating Path, is shown in Figure 4. It features a makespan of 51.57 and has fewer than half the original edges of the seed tree.

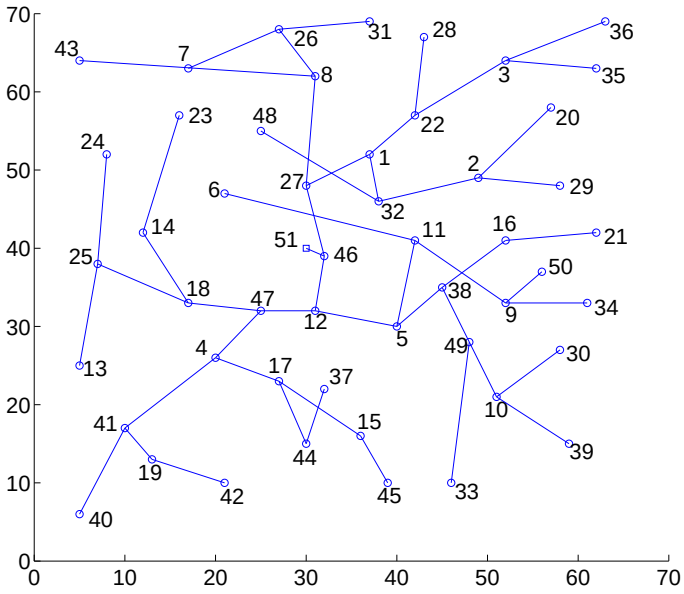

Figure 4: Iterative Alternating Path Algorithm

4.3 Results

The following table summarizes the best solutions found using each method on seven different FTPs. These seven problems were selected from the TSPLIB (Traveling Salesperson Problem Library) [20]. While the ant algorithms typically perform well, it should be noted that the ant algorithm actually employed a primitive version of Alternating Path to further improve upon the solutions that it had discovered; thus the results for ant algorithm are a bit better than would normally be accomplished with the algorithm alone. The Genetic Algorithm search also performs well. We can see in general that Iterative Alternating Path finds solutions which are significantly better than Greedy Dyanmic, but still falls short of the gains realized by these two combinatorial search algorithms (ants and GAs).

DataSet → **Algorithm** ↓	eil51	eil76	kroA100	d198	lin318	att532	rat783
Greedy Fixed	75.03	68.7	3857	3087	4612	9720	463.3
Greedy Dynamic	61.63	56.64	2515	2433	2806	5096	340.8
Center of Mass	55.92	59.65	2591	2446	2944	5367	362.6
Genetic Algorithm	54.79	51.72	2393	2445	2759	5064	475.7
Ant Algorithm	49.20	50.28	2396	2380	2700	5073	336.0
Alternating Path	50.56	52.77	2442*	2773*	2717	5115*	403.3*

Our first comment on Alternating Path concerns its search methodology. AP is clearly a local hill-climbing method; it finds solutions within a neighborhood of its current best solution and then chooses the most optimal among these. Furthermore, it is limited to only improving steps and thus will never escape a local minima in its neighborhood of search. The more robust searching algorithms (ants and GAs) are more adapt at escaping local minima and exploring a larger portion

of the solution space.

The IAP algorithm is also computationally intensive. Running on a fast workstation, a 200 robot problem with depth=4 requires more than 24 hours of computation time. Increasing the problem size even modestly (say `lin318.tsp` with 318 robots) prohibits full computation; we are limited to computing only a subset of initial seeds. These computations are marked with an asterisk in the table.

This suggests a more proper role for Alternating Path. Instead of using AP as a basic search algorithm, it is better to use a more exploratory algorithm first. Then AP can be used as a final step operating on the best (or best few) solution found with the first search algorithm. Thus IAP acts more like a last-step improvement procedure to further realize gains on an already good solution. It is hypothesized that IAP would also work well in concert the Simulated Annealing. The SA algorithm is adept at escaping the local minima that limit IAP.

4.4 Local Search Topology

Our last experimental remarks concern the way in which IAP searches. The Alternating Path Algorithm is essentially a local hill-climbing method. The $N^{(k)}(T)$ function specifies a set of local solutions in the neighborhood of tree T. The Alternating Path Algorithm searches a subset of this neighborhood and selects the best new solution as the next tree, T'. Iterative AP will continue to make incremental improvements to the tree until no further improvements can be found. Thus the final solution is a local minima (with respect to makespan) in its neighborhood.

We can conjecture about the topology of the makespan function over the space of trees searched. It could, for example, be a well-shaped basin with a distinct local minima that will inevitably be discovered regardless of the route taken. Alternatively, the landscape could be full of many hills and valleys with multiple local minima; in this latter case, the final solution will depend highly on the path of the search algorithm in exploring the space of solution trees.

The parameter, k_{max}, determines the number of edge exchanges we may consider to construct the neighborhood of a solution T. Naturally, increasing the parameter k_{max} expands the neighborhood thereby allowing us to consider a greater number of potential next solutions and possibly discovering a better local makespan that would not have been found in a single step with a lower k_{max}.

If the topology is a smooth basin, then larger k_{max} is generally better. We'll take faster steps toward the inevitable discovery of the locally optimal awakening schedule. But if the topology is more varied, then larger steps may take us out of of local basin where a very good solution might lie; or alternately, allow us to skip into a better basin with a better local minima.

An intriguing result allows us to hypothesize that the topology is probably more varied than smooth. On the Eil51 problem, we ran Iterative Alternating Path with a depth of $k_{max} = 3$ and achieved a best solution of 51.57 after 15 iterations. Increasing $k_{max} = 4$ drops the number of iterations down to just seven. However, the locally optimal solution found has a makespan of only 53.69. The larger neighborhood structure of this latter run allowed us to accept an intermediate solution that was not available with the smaller neighborhood, but by doing so, it stepped out of a basin that contained ultimately a better solution (51.57).

References

[1] E. Aarts and J. Lenstra. Local Search in Combinatorial Optimization, John Wiley and Sons, Ltd. 1997.

[2] R. Ahuja, O. Ergun, J. Orlin, and A. Punnen. A Survey of Very Large-Scale Neighborhood Search Techniques. Fill in.

[3] R. Ahuja, T. Magnanti, and J. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, Englewood Cliffs, NJ, 1993.

[4] R. Ahuja, J. Orlin, and D. Sharma. Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. Mathematical Programming 91, 71-97, 1997.

[5] E. Arkin, M. Bender, G. Dongdong, S. He, and J. Mitchell. Improved Approximation Algorithms for the Freeze-Tag Problem.

[6] E. Arkin, M. Bender, S. Fekete, J. Mitchell, and M. Skutella. The Freeze-Tag Problem: How to wake up a swarm of robots. In Proc. 13th ACM-SIAM Sympos. Discrete Algorithms, pp. 568-577, 2002.

[7] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. IEEE Infocom, 2003.

[8] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Multicasting in heterogeneous networks. In Proc. 30th ACM Sympos. Theory comput., pp. 448-453, 1998.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence: From Natuaral to Artificial Systems. Oxford University Press, 1999.

[10] S. Hedetniemi, T. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. Networks, 18:319-349, 1988.

[11] B. Julstrom and G. Raidl. A Permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In 2003 Genetic and Evolutionary Computation Conference's Workshop Proceedings, Workshop on Analysis and Design of Representations, pp. 2-7, 2003.

[12] J. Konemann and R. Ravi. A matter of degree: Improved approximation algorithms for the degree-bounded minimum spanning trees. SIAM J. Comput., 31(6), pp. 1783-1793, 2002.

[13] J. Konemann and R. Ravi. Primal-dual algorithms meets local search: Approximating MST's with nonuniform degree bounds. In Proc. of the 35th ACM Symposium on Theory of Computing, 2003.

[14] N. Mladenovic and P. Hansen. Variable neighbourhood search. Comput. Oper. Res., Vol. 24, 1097-1100, 1997.

[15] W. Press, S. Teukolsky, W. Vetterling and B. Flannery. Numerical Recipes in C: The Art of Scientific Computing, 2nd Ed.. Cambridge University Press. 1998.

[16] G. Raidl. An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. In Proc. of the 2000 Congress on Evolutionary Computation CEC00, 2000.

[17] R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In Proc. 35th Sympos. Found. Computer Science, pp. 202-213, 1994.

[18] C. Ribeiro and M. Souza. Variable neighborhood search for the degree-constrained minimum spanning tree problem. Discrete Applied Mathematics, Vol. 118, pp. 43-54, 2002.

[19] M. Sztainberg, E. Arkin, M. Bender, and J. Mitchell. Analysis of heuristics for the Freeze-Tag Problem. In Proc. Scandianvian Workshop on Algorithms, Vol. 2368 of Springer-Verlag LNCS, pp. 270-279, 2002.

[20] TSPLIB. ¡http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/¿