Algorithms and Complexity Results for Graph-Based Pursuit Evasion

Richard Borie, Craig Tovey, Sven Koenig*

June 14, 2011

Abstract

We study the classical edge-searching pursuit-evasion problem where a number of pursuers have to clear a given graph of fast-moving evaders despite poor visibility, for example, where robots search a cave system to ensure that no terrorists are hiding in it. We study when polynomial-time algorithms exist to determine how many robots are needed to clear a given graph (minimum robot problem) and how a given number of robots should move on the graph to clear it with either a minimum sum of their travel distances (minimum distance problem) or minimum task-completion time (minimum time problem). The robots cannot clear a graph if the vertex connectivity of some part of the graph exceeds the number of robots. Researchers therefore focus on graphs whose subgraphs can always be cut at a limited number of vertices, that is, graphs of low treewidth, typically trees. We describe an optimal polynomial-time algorithm, called CLEARTHETREE, that is shorter and algorithmically simpler than the state-of-the-art algorithm for the minimum robot problem on unit-width unit-length trees. We then generalize prior research to both unit-width arbitrary-length and unit-length arbitrary-width graphs and derive both algorithms and time complexity results for a variety of graph topologies. Pursuit-evasion problems on the former graphs are generally simpler than pursuit-evasion problems on the latter graphs. For example, the minimum robot and distance problems are solvable in polynomial time on unit-width arbitrary-length trees but NP-hard on unit-length arbitrary-width trees.

1 Introduction

Pursuit evasion or, synonymously, graph searching is an important problem in robotics [36], agents [34] and artificial intelligence [15]. As described by an

*R. Borie is with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487, USA. His e-mail address is borie@cs.ua.edu. C. Tovey is with the Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. His e-mail address is cat@gatech.edu. S. Koenig is with the Computer Science Department, University of Southern California, Los Angeles, CA 90089, USA. His e-mail address is skoenig@usc.edu.

empirical robotics researcher, consider "... the problem of closing a museum for the night. In order to be sure that no thieves or other malcontents remain inside after closing, the guards must perform a thorough search of the building. They must keep in mind that any intruders may try to avoid the guards. For example, if a guard is checking each room along a hall, an intruder might sneak behind the guard while he is checking one room and hide in a room that was already checked. In this case, one solution might be to use two guards, with one always keeping watch on the hall. ... we are interested in scalable techniques for coordinating the actions of teams of robots to clear entire buildings." [14] Pursuit evasion is important for many other practical applications as well [17], and teams of robots can solve pursuit-evasion problems automatically in a secure, reliable and efficient way, which explains the interest of empirical robotics researchers in the topic. For example, several robotics groups have implemented pursuit-evasion algorithms on actual robots; see the section on "Real Systems" in [23] and the workshop on "Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees" at the IEEE International Conference on Robotics and Automation in 2010.

There are two main classes of pursuit-evasion algorithms. Visibility-based pursuit-evasion algorithms still have severe limitations for practical applications [23]. Graph-based pursuit-evasion algorithms depend on being able to construct an appropriate graph from the terrain but several approaches exist for this purpose [24] [27] [28], making them interesting for practical applications. Many different assumptions about the capabilities of the robots and evaders are possible. Pursuit-evasion algorithms often make worst-case assumptions about the evaders, such as their number and movement capabilities, making them interesting if no information about the evaders is available and it is important not to miss any evader. In terms of the other assumptions, we use a classic (and the oldest) version of the pursuit-evasion problem in the literature, which provides a clean formulation of the pursuit-evasion problem that is amenable to rigorous analysis. In particular, we study a formulation of pursuit-evasion problems where a number of pursuers have to clear (decontaminate) a given graph of fast-moving evaders despite poor visibility [33]. Applications, besides our motivating application above, include finding confused elderly people who wander off, capturing fleeing animals, locating lost team members of first response teams or survivors in disaster scenarios, and finding terrorists in cave systems. They also include containing poisonous gas in a building, and tracking and destroying a computer virus on a computer network. Consider, for example, a scenario where robots search a known but twisty cave system to ensure that no terrorists are hiding in it. The robots are the pursuers, and the terrorists are the evaders. The cave system can be modeled as a graph with vertices that have widths and edges that have both widths and lengths. Figure 1 illustrates the concepts of edge widths and lengths for robots of the size of the grey squares in narrow cave tunnels. The robots and evaders move on the edges of this graph and are able to stop or change directions anywhere on the edges. The robots know the graph but not how many evaders are present. They must catch all evaders despite zero visibility, that is, robots are unable to detect the locations of evaders prior

to capturing them. Initially, the entire graph is potentially contaminated. The evaders can hide anywhere on the vertices or edges. They cannot be seen by the robots and can move faster than them, with unbounded speed. They get caught only if they collide with a number of robots at a vertex or the same point on an edge that is no smaller than the width of the vertex or edge, respectively. Thus, a vertex of width k can be guarded by at least k robots that wait at the vertex, and an edge of width k can be cleared by at least k robots that move along the edge in the same direction. The robots move at unit speed. Their travel times or distances are thus equal to the lengths of their paths. A solution of the pursuit-evasion problem is a movement strategy for a given number of robots with given start vertices on a given graph that enables them to clear the graph, that is, either ensure that no evaders are present or catch them all. This is a common and very general model of pursuit-evasion problems on graphs. For example, a solution remains a solution even if the evaders can be seen by the robots over longer distances or can move only slowly.

We study three optimization problems, namely how many robots are needed to clear a given graph (minimum robot problem) and how a given number of robots should move on the graph to clear it with either a minimum sum of their travel distances (minimum distance problem) or minimum task-completion time (minimum time problem), depending on the desired cost objective. These three problems can be defined more precisely as follows:

- *Minimum robot problem*: Given input graph G, determine the fewest robots r such that a solution exists for clearing G using r robots.
- *Minimum distance problem*: Given input graph G and number of robots r, determine the shortest total distance d such that a solution exists for clearing graph G using r robots so that the sum of travel distances is at most d. Special case: If it is not possible to clear graph G using r robots, then the minimum distance is ∞.
- *Minimum time*: Given input graph G and number of robots r, determine the shortest time t such that a solution exists for clearing graph G using r robots within at most t time units. Special case: If it is not possible to clear graph G using r robots, then the minimum time is ∞.

All the algorithms in this paper will be constructive, that is, they can be used to construct a solution whenever one exists.

Consider, for example, the graph in Figure 2. All vertices and edges have width one, with one exception: e_1 has width 2. Four robots starting at vertex t_1 can clear the graph in many ways. One possible solution is the following one: At time 0, send two robots from t_1 to t_2 along e_1 and one robot from t_1 to t_2 along e_4 to e_8. At time 5, send one robot from t_2 to t_1 along e_2 and one robot from t_2 to t_1 along e_3. The resulting task-completion time for clearing the graph is 6 and the sum of travel distances is 9. Another possible solution is the following one: At time 0, send two robots from t_1 to t_2 along e_1 and one robot from t_1 to t_2 along e_2. At time 1, send one robot from t_2 to t_1 along e_2.

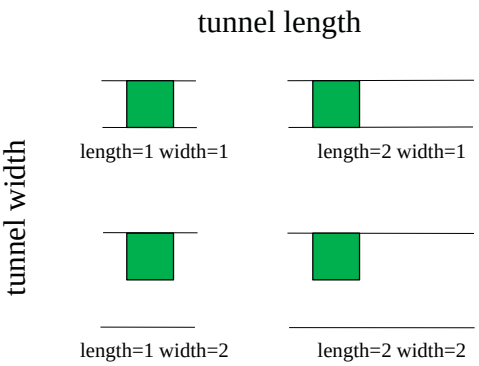

Figure 1: Edge Widths and Lengths

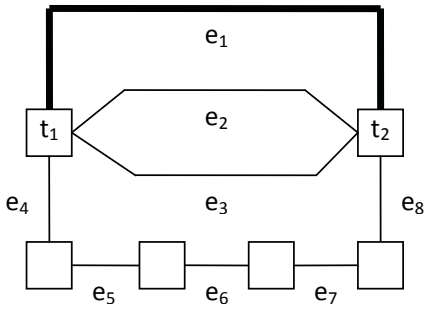

Figure 2: Example Graph

At time 2, send one robot from t_1 to t_2 along e_3, one robot from t_2 to t_1 along e_3, one robot from t_1 to t_2 along e_4 to e_6 and one robot from t_2 to t_1 along e_8 to e_6. The resulting task-completion time for clearing the graph is 4.5 and the sum of travel distances is 10.

The robots cannot clear a graph if the vertex connectivity of some part of the graph exceeds the number of robots. For instance, suppose that the graph contains a K_7 subgraph, as shown in Figure 3. Between any two vertices in that subgraph there are six vertex-disjoint connecting paths, so five robots can not catch an evader. For example, if an evader is currently located at vertex A and the five pursuers are at vertices B,C,D,E and F, then the evader could escape to vertex G. Prior research has therefore rightfully studied graphs whose subgraphs can always be cut at a limited number of vertices, that is, graphs of low treewidth. It has focused on the minimum robot problem on unit-width

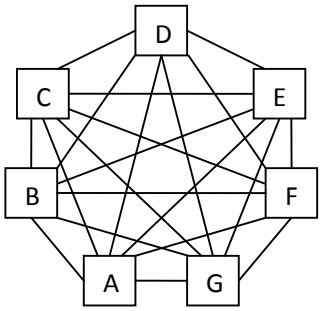

Figure 3: K_7 Subgraph

unit-length graphs (where all vertices and edges have width one and all edges have length one). It has shown that the pursuit-evasion problem is NP-hard on such graphs in general and can be solved in polynomial time on trees.

Our objective is to provide a catalog of algorithmic and complexity results for several variations of pursuit-evasion problems on several classes of graphs. In Section 3, we describe a linear-time algorithm, called CLEARTHETREE, that solves the minimum robot problem on unit-width unit-length trees and is shorter and algorithmically simpler than the state-of-the-art algorithm [31]. In Section 4, we then either develop polynomial-time algorithms or prove the NP-hardness of the minimum robot, distance and time problems on both unit-width arbitrary-length and arbitrary-width unit-length graphs of a variety of simple topologies, including paths, cycles, stars, trees, two-vertex multi-graphs, series-parallel graphs, and cliques. Pursuit-evasion problems on unit-width arbitrary-length graphs are often simpler than pursuit-evasion problems on arbitrary-width unit-length graphs. For example, the minimum distance problem is solvable in polynomial time on unit-width arbitrary-length trees but NP-hard on arbitrary-width unit-length trees. In this paper we only consider graphs that are connected; Figure 4 illustrates the different graph topologies. A *path* has two vertices with degree 1, and each remaining vertex has degree 2. In a *cycle*, every vertex has degree 2. A *star* has a central vertex, and an edge from the central vertex to each other vertex. In a *tree*, no subset of vertices forms a cycle. A *clique* (or *complete graph*) has an edge between every pair of vertices. In a *two-vertex multi-graph*, multiple edges may exist between the two vertices. Finally, a *series-parallel graph* is constructed inductively from individual edges by applying series, parallel, and/or jackknife operations as illustrated in Figure 5. Each operation takes two subgraphs G_1, G_2, each having two special vertices (shown with double boxes), and joins them by merging these special vertices as pictured. Both the series and parallel operations can be generalized to arities larger than two. So for example, the series-parallel graph in Figure 4 can be constructed as $p(s(e,e,e,e),e,s(e,p(s(e,e),s(e,e,e)),e))$; and the star in Figure 4, which

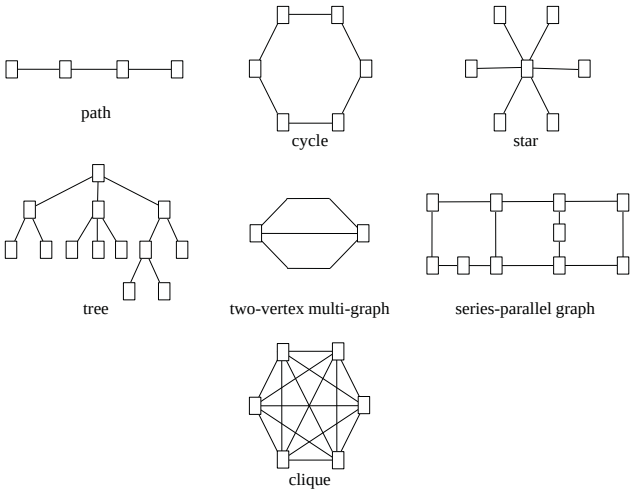

Figure 4: Graph Topologies

is also a series-parallel graph, can be constructed as $j(j(j(j(j(e,e),e),e),e),e)$. Within each expression, each occurrence of e denotes a new distinct edge.

Our main contribution is to generalize the assumptions made by prior research to make them more realistic, while maintaining a clean formulation of the pursuit-evasion problem that is amenable to rigorous analysis. Our time complexity results go beyond unit-width unit-length graphs by applying to more realistic graphs, namely unit-width arbitrary-length graphs and arbitrary-width unit-length graphs. They also go beyond the minimum robot problem by applying also to the minimum time and distance problems, an important extension as pointed out in [28]. Given that all of these problems turn out to be NP-hard on general graphs, it is important to understand whether they remain NP-hard even for restricted classes of topologies that are subsets of topologies encountered for practical applications. If so, then one needs to concentrate on developing heuristic or approximation algorithms for practical applications since robots need to solve time-sensitive problems with a small sum of planning and plan-execution time.

2 Related Work

Our version of the pursuit-evasion problem, called edge searching in the literature, is the oldest one [33]. The minimum robot problem has been studied extensively for edge searching on unit-width unit-length graphs. The smallest number of robots needed to clear a given graph is called its edge search number. The decision version of the minimum robot problem is in NP since there

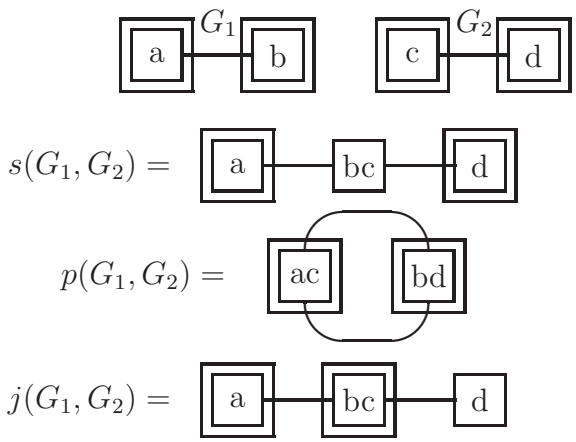

Figure 5: Series, Parallel, and Jackknife Operations

exists a solution for the smallest number of robots where no edge gets recontaminated [29] [4]. Each step of a solution can have the form "send k robots along edge e". If any solution exists, then there must be one with a polynomial number of steps. One can thus guess a polynomial-size solution and verify its correctness in polynomial time. This result can be extended to show that the decision versions of all pursuit-evasion problems in this paper are in NP. The decision problem can be solved in linear time on arbitrary graphs for any constant bound (in time exponential in the bound) [8] [39] but is NP-complete otherwise [31]. It remains NP-complete for chordal graphs [16], star-like graphs [16] and planar graphs with maximum degree three [30] [32], as argued in [37] by exploiting that the edge search number, node search number, vertex separation, path width and interval thickness of a graph differ by at most a small constant [10] [20] [4] [22]. It can be solved in polynomial time on cographs [7], permutation graphs [6], interval graphs [21], k-trees for fixed k [5] and grid graphs [11]. For example, an m-by-n grid graph can be cleared by $\min(m, n) + 1$ robots. On trees, both the search number [31] [10] and the corresponding solution [9] [37] can be determined in linear time. For example, the edge search number of a graph is identical to the vertex separation of the two-expansion of the graph (where every edge is replaced with three consecutive edges), an optimal layout of the two-expansion of the graph can be transformed into an optimal solution [10] in linear time [37], and an optimal layout can be computed in linear time for a tree [37].

Many variations of the pursuit-evasion problem other than edge searching have been considered in the literature, often making different assumptions about movement or capture. Node searching, for example, assumes that robots and evaders jump from vertex to adjacent vertex in a fraction of or a single time

step, and an evader is captured if it is at the same vertex as a robot at some time [22] [4]. Edge searching is more suited to robot, vehicle or human movement for which an edge traversal has a physical meaning. Many other variations have been studied as well [12], but the typical results are the same: NP-hardness proofs for the minimum robot problem on unit-width unit-length graphs in general, optimal polynomial-time algorithms for the minimum robot problem on unit-width unit-length trees, no results for either the minimum time or minimum distance problems, and very few results on graphs other than unit-width unit-length graphs.

The concepts of arbitrary vertex and edge widths were introduced in [3], but the problems studied in that paper differ from the current paper in several ways: In that publication, the graph must be a tree, all robots must start at the same location, and that the solution must preserve contiguity (the cleared portion of the graph must always form a connected subgraph). Based on these assumptions, that publication develops a linear-time algorithm for the minimum robot problem. Arbitrary vertex and edge widths are also used in [25] and [26]. In those publications, the operations performed during a solution are called "sweep" (clear a vertex or region) and "block" (clear an edge or doorway between regions). The graph must be a tree, but contiguity is not required. These papers provide polynomial-time algorithms for the minimum robot problem, but the solutions are not guaranteed to be optimal.

Experimental validations on graphs obtained by discretizing real-world environments are performed in [18] and [19]. The algorithm in [19] produces a solution that clears a graph by decreasing the set of contaminated vertices until this set becomes empty. However, it is not always guaranteed that such a solution can be found. The algorithm in [18] produces a solution that performs well both in the worst-case (when the evader is adversarial) and in the average-case (when the evader moves randomly, assuming a Markovian motion model).

A close relative of the pursuit-evasion problem is the perimeter-guarding problem [1] [2], where guards also move along edges but intruder speeds are bounded and intruder movement may not be restricted to the graph. Capture is different since guards detect intruders within a given radius of capture. In these problems, it is usually impossible to guarantee that the perimeter will not be breached. Instead, the objective is to minimize the probability or maximize the expected time to success for the intruders, usually by means of a randomized strategy that is not predictable and has desirable game-theoretic properties [35] [40].

3 Minimum Robots on Unit-Width Unit-Length Trees

In this section, we study the minimum robot problem on unit-width unit-length trees (treewidth-1 graphs), the classical setting. We describe a linear-time algorithm, called CLEARTHETREE, that solves the minimum robot problem on

Label the vertices and edges of T as follows while doing a postorder traversal.

- If v is a leaf then $S(v) = \{1\}$. Also $Q(v, \mathit{parent}(v)) = 1$ if $v \neq \mathit{root}(T)$.
- If v is not a leaf then v has children $c_1, \ldots, c_k$, where $k \geq 1$. Let x be the largest value that appears in at least two of the $S(c_i)$, or 0 if no such value exists. Let $y = \max_i \min(S(c_i))$, that is, the largest value that is the minimum of any $S(c_i)$.
  - If $x < y$ then $S(v) = \bigcup_i S(c_i) - \{1, 2, \ldots, y-1\}$. Also if $v \neq \mathit{root}(T)$ then $Q(v, \mathit{parent}(v)) = y$.
  - If $x = y$ and this value appears in exactly two of the $S(c_i)$ and is the minimum in both sets, then let $S' = \bigcup_i S(c_i) - \{1, 2, \ldots, y-1\}$.
    - If $v = \mathit{root}(T)$ then $S(v) = S'$.
    - If $v \neq \mathit{root}(T)$ then let z be the smallest positive integer that is not in S'. $S(v) = S' - \{1, 2, \ldots, z-1\} \cup \{z\}$. Also $Q(v, \mathit{parent}(v)) = z$.
  - Otherwise, either $x > y$ or ($x = y$ and the conditions in the previous case do not hold). Let z be the smallest integer exceeding x and that is not in $\bigcup_i S(c_i)$. $S(v) = \bigcup_i S(c_i) - \{1, 2, \ldots, z-1\} \cup \{z\}$. Also if $v \neq \mathit{root}(T)$ then $Q(v, \mathit{parent}(v)) = z$.

Figure 6: Algorithm CLEARTHETREE

unit-width unit-length trees and is shorter and algorithmically simpler than the state-of-the-art algorithm [31].

Recall that the minimum robot problem can be solved in linear-time on such trees [31]. Let T_r denote the smallest tree that requires r robots to clear. So T_1 has only one edge, and T_2 is a star with three edges. The following results were obtained in [31]:

- For $r \geq 2$, the smallest tree T_r that requires r robots can be obtained by taking 3 copies of T_{r-1} and fusing one leaf from each copy. So T_r has $n = 3^{r-1} + 1$ vertices, and $r = 1 + \log_3(n-1) = O(\lg n)$.
- If a given tree T can be cleared by r robots, then T can be cleared by r robots in such a way that, at any instant, all robots lie along a common path.
- A tree T can be cleared by r robots iff it contains a path P such that splitting each degree-d vertex of P into d vertices of degree 1 produces a forest of trees that can each be cleared by $r-1$ robots.

We now present a new linear-time algorithm for the minimum robot problem on trees T, called CLEARTHETREE, that is shorter and algorithmically simpler than the existing algorithm because it uses a postorder traversal, see Figure 6. We label each vertex v with a subset $S(v) \subseteq \{1, 2, \ldots, r\}$ where r is the minimum number of robots. The intuition behind CLEARTHETREE is as follows: First, when robot number $\min(S(v))$ is at vertex v, the robots numbered in the range $\{\min(S(v)), \ldots, \max(S(v))\}$ will be located within the subtree rooted at v. Also, any robots numbered above $\max(S(v))$ will be located at ancestors of v. Second, a robot that clears exactly one child of v should continue up the path to also

clear v. Third, a robot that clears three or more children of v must not also clear v, because this would contradict the path conditions stated earlier. Instead, v must be cleared by some higher-numbered robot, and among such robots that are available, we choose a robot with the lowest number. Finally, a robot that clears exactly two children of v might also be able to clear v, thus forming a single path that changes direction at v. If so, then we must also select another robot to clear the edge leading upward from v, unless v is the root of the tree. However, in certain circumstances given in Figure 6, merging two paths at v would cause a violation of the stated path conditions, and, if so, then this case is handled identically to the previous case (when a robot clears three or more children of v).

During execution of CLEARTHETREE, each edge $(v, \mathit{parent}(v))$ is labeled with a value $Q(v, \mathit{parent}(v)) \in \{1, 2, \ldots, r\}$ where again r is the minimum number of robots. The label $Q(v, \mathit{parent}(v))$ denotes which one of the r robots will be responsible for clearing the edge $(v, \mathit{parent}(v))$.

CLEARTHETREE first locates the path P along which robot number r moves. As robot r visits each vertex v along P, robots $\{1, 2, \ldots, r-1\}$ recursively clear each subtree of v. Also, when robot r visits the vertex v of P nearest to $\mathit{root}(T)$, robots $\{1, 2, \ldots, r-1\}$ recursively clear the portion of T that lies above v, unless $v = \mathit{root}(T)$.

After CLEARTHETREE runs, T can be cleared using $r = \max(S(\mathit{root}(T)))$ robots. (Note that sometimes a robot is listed in a vertex label S but not in any incident edge labels.)

Theorem 1 *Algorithm CLEARTHETREE solves the minimum robot problem in linear time for unit-width unit-length trees (indeed, also unit-width arbitrary-length trees).*

Proof sketch: Most of the ideas to prove correctness have been discussed above. CLEARTHETREE maintains the following invariant: Consider any feasible set of robots that satisfies the same conditions required of $S(v)$. Among all possible such sets, when sorted into descending order, $S(v)$ is lexicographically minimum.

To establish the running time, first observe that CLEARTHETREE runs in $O(n \cdot r)$ time, where the number of robots needed is $r = O(\lg n)$. This is because the time used to determine $S(v)$ at each node v is proportional to the product of r and the number of v's children. This assumes each set $S(v)$ is maintained as a sorted doubly-linked list, so that each union can be performed in $O(r)$ time. However, a more careful analysis of CLEARTHETREE shows that it actually runs in $O(n)$ time. The union operation can be implemented to require at most j comparison steps, where j is the lesser of the maxima of the two sets whose union is being performed. (Such a union is destructive, that is, it might destroy the two sets whose union is being taken.) An induction shows that for $1 \leq j \leq r$, the number of unions that involve two sets that each contain a value j or greater is at most $2n/2^j$. Therefore the total time for all the unions is at most proportional to

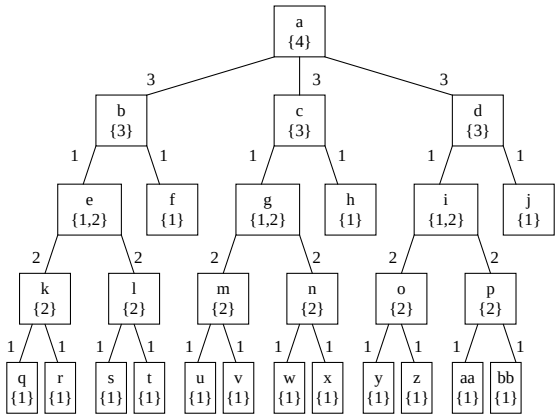

Figure 7: First Example for CLEARTHETREE

$$\Sigma_{1\le j\le r}[j * 2n/2^j] = \Sigma_{1\le j\le r}\Sigma_{1\le i\le j}[2n/2^j]$$
$$= \Sigma_{1\le i\le r}\Sigma_{i\le j\le r}[2n/2^j] \le \Sigma_{1\le i\le r}[4n/2^i] \le 4n = O(n).$$

□

Next we trace algorithm CLEARTHETREE using two example trees, which together illustrate all the significant cases that occur within the algorithm.

First see Figure 7 for an example of CLEARTHETREE using tree T_4. Sets $S(v)$ are shown at the vertices, and labels $Q(v, \mathit{parent}(v))$ are shown along the edges. The number of robots needed is 4, as computed at the root. During the solution, robot 4 remains stationed at the root. Robot 3 clears edges (a,b), (a,c), and (a,d). While robot 3 guards vertex b, robot 2 clears path k→e→l. When robot 2 guards vertex k, robot 1 clears path q→k→r; when robot 2 guards vertex e, robot 1 clears edge (e,b); and when robot 2 guards vertex l, robot 1 clears path s→l→t. The other two subtrees are cleared analogously while robot 3 guards vertices c and d.

Also see Figure 8 for another example of CLEARTHETREE. This is the same tree T_4 except that one leaf (bb) is now missing, and so only 3 robots are needed. During this solution, robot 3 clears path b→a→c. The first two subtrees are cleared the same as for Figure 11. While robot 3 guards vertex a, robot 2 clears path o→i→d→a. When robot 2 guards vertex o, robot 1 clears path y→o→z; when robot 2 guards vertex i, robot 1 clears path aa→p→i; and when robot 2 guards vertex d, robot 1 clears edge (j,d).

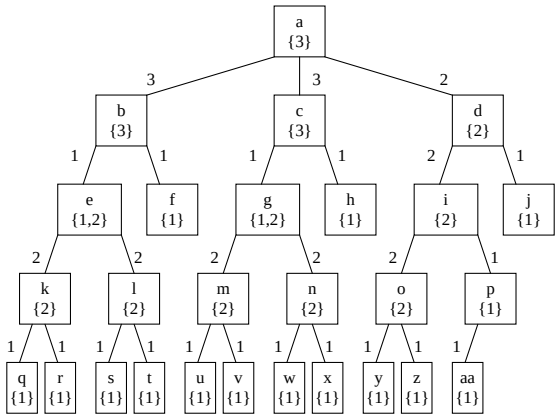

Figure 8: Second Example for CLEARTHETREE

	Arbitrary-Width Unit-Length Graphs			Unit-Width Arbitrary-Length Graphs		
	Minimum Robot	Minimum Distance	Minimum Time	Minimum Robot	Minimum Distance	Minimum Time
Paths	P	P	Pseudo-P	P	P	P
Cycles	P	Pseudo-P	Pseudo-P	P	P	P
Stars	NP-c, Pseudo-P	NP-c	Strongly NP-c	P	P	Strongly NP-c
Trees	NP-c	NP-c	Strongly NP-c	P*	(Open)	Strongly NP-c
Two-vertex multi-graphs	NP-c, Pseudo-P	NP-c	Strongly NP-c	P	P	Strongly NP-c
Series-parallel graphs	NP-c	NP-c	Strongly NP-c	(Open)	(Open)	Strongly NP-c
Cliques	NP-c	NP-c	NP-c	P	P	Strongly NP-c
General graphs	NP-c*	NP-c	Strongly NP-c	NP-c*	NP-c	Strongly NP-c

Figure 9: Summary of Results

4 Minimum Robots, Distance and Time on Unit-Width Arbitrary-Length and Arbitrary-Width Unit-Length Graphs

In this section, we study the minimum robot, distance and time problem on unit-width arbitrary-length and arbitrary-width unit-length graphs. Each robot can start and finish at any vertex of the graph. We either develop optimal polynomial-time algorithms or prove the NP-hardness for a variety of simple topologies, including paths, cycles, stars, trees, two-vertex multi-graphs (where multiple edges may exist between the two vertices), series-parallel graphs, and cliques. We prove NP-hardness by reduction from well-known NP-hard problems, such as partition (a weakly NP-hard problem) and 3-partition (a strongly NP-hard problem). Weakly NP-hard problems might be solved optimally with pseudo-polynomial-time algorithms, while strongly NP-hard problems cannot

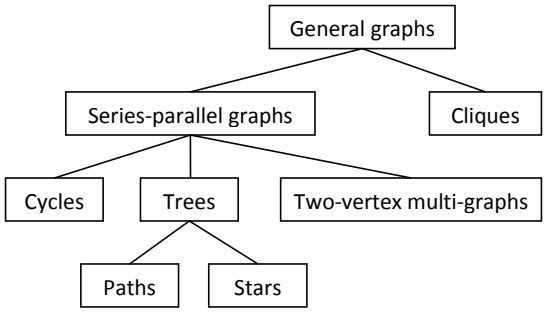

Figure 10: Hierarchy of Graph Classes

unless P=NP. Pseudo-polynomial-time algorithms can solve pursuit-evasion problems with a small number of robots fast and will thus often be acceptable in practice, which is why we develop optimal pseudo-polynomial-time algorithms for NP-hard problems, where possible.

Figure 9 summarizes the results to be obtained in this section. *P* denotes that a deterministic polynomial-time algorithm exists, and *NP* denotes that a non-deterministic polynomial-time algorithm exists (that is, any proposed solution can be verified in polynomial time). *NP-hard* denotes that the problem is at least as hard as any problem in NP, and *NP-c* is shorthand for *NP-complete* which denotes that the problem is both NP and NP-hard (and hence the best possible algorithms are likely to require exponential time). *Pseudo-P* denotes that the problem is solvable in pseudo-polynomial time, which assumes that numerical values in the input are represented in unary rather than binary in order to achieve polynomial-time. *Strongly NP-c* denotes that the problem would remain NP-complete even if the input numbers were represented in unary rather than binary. The complexity on arbitrary-width arbitrary-length graphs is always at least as difficult as that on both unit-width arbitrary-length and arbitrary-width unit-length graphs. The starred entries are found in [31]; CLEARTHETREE provides an alternative proof for one of them. Each other result is either derived directly by a theorem in Section 4.1 or 4.2, or can be inferred as a corollary of a derived result by applying Propositions 2 and/or 3 below.

Proposition 2 pertains to subclass and superclass relations between graph classes, as illustrated in Figure 10. That is, paths and stars are subclasses of trees; trees, cycles, and two-vertex multi-graphs are subclasses of series-parallel graphs; and series-parallel graphs and cliques are subclasses of general graphs. Membership in each of these classes can be determined in linear time.

Proposition 2 *If algorithm A solves problem Q on graph class C, then algorithm A also solves problem Q for any subclass of C. Conversely, if problem Q is NP-hard for some graph class C, then problem Q remains NP-hard for any*

superclass of C.

So, if some problem Q is polynomial-time solvable on trees, then problem Q is also polynomial-time solvable on paths and on stars. (For example, the CLEARTHETREE algorithm also applies to paths or to stars.) Alternatively, if some problem Q is NP-hard on cliques, then problem Q is also NP-hard on general graphs.

Proposition 3 *Suppose the minimum robot problem is NP-hard for some graph class C. Then, the minimum distance and time problems are also NP-hard for graph class C. Moreover, there cannot exist any polynomial-time approximation algorithms for these latter problems unless P=NP.*

Proof: Minimum distance and minimum time are each finite iff the number of robots is sufficient to clear the graph. So, if any such approximation algorithm A exists, then the minimum robot problem could be solved in polynomial time by checking whether A returns a finite value. □

For example, we know that the minimum robot problem is NP-hard on general graphs [31]. Therefore, both the minimum distance problem and the minimum time problem are also NP-hard on general graphs.

Section 4.1 describes our algorithms and time complexity results for arbitrary-width unit-length graphs, and Section 4.2 describes our results for unit-width arbitrary-length graphs. We use the following notation. Let $G = (V, E)$ denote a graph, where $n = |V|$ and $m = |E|$. Each edge e has a length $L(e)$, that represents both the traversal distance and time. Each edge e also has a width $w(e)$, that denotes the least number of robots needed to clear it by traversing it simultaneously, and each vertex v has a width $w(v)$, that denotes the least number of robots needed to guard it when it is incident upon both cleared and contaminated edges. We consider both decision and optimization versions of the minimum robot, distance and time problems. For example, the decision version of the minimum robot problem is to determine whether a given number of robots can clear a given graph, and the optimization version is to determine the minimum number of robots.

In this paper we assume that each robot may start at any location, and that each robot may end at any location. However, many other constraints could also be deemed appropriate for certain applications, such as: the starting location and/or ending location of each robot might be specified as part of the instance; perhaps it is required that each robot must end at the same location where it originally started; maybe all the robots must start at the same location and/or all must end at the same location; or any combinations of these. Note that the version we consider in this paper always yields the smallest distance and the smallest time, and therefore may serve as a lower bound for all other variations based on the initial and/or final deployment of the robots. We leave such remaining variations for future work.

4.1 Arbitrary-Width Unit-Length Graphs

This section considers only graphs for which all lengths $L(e) = 1$. However, note that the widths are permitted to be arbitrary, that is, $w(v) \geq 1$ and $w(e) \geq 1$. For conciseness, we denote such graphs as AWUL (arbitrary-width unit-length). In this respect the assumptions are similar to those in [3] [25] [26].

4.1.1 Time Complexity Results

In the following theorems, we will reduce from the NP-hard *partition problem* [13]. An instance of partition is defined by positive integers $(a_1, \ldots, a_n)$. Let $k = \sum_{i=1}^{n} a_i/2$. The partition problem asks whether there exists any $X \subseteq \{1, \ldots, n\}$ such that $\Sigma\{a_i : i \in X\} = k = \Sigma\{a_i : i \notin X\}$.

Theorem 4 *The minimum robot problem is NP-hard for AWUL stars.*

Proof: Let $(a_1, \ldots, a_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} a_i/2$. Construct a star $G = (V, E)$ as follows. Let $V = \{v_i : 1 \leq i \leq n\} \cup \{z\}$, and let $E = \{(v_i, z) : 1 \leq i \leq n\}$. Let $w(v_i) = 1$ for $1 \leq i \leq n$, and let $w(z) = k$. Also let $w(v_i, z) = a_i$ for $1 \leq i \leq n$. We claim that G can be cleared using k robots iff the partition instance has a solution.

"If" direction: Let the partition problem have solution $X \subseteq \{1, \ldots, n\}$. For all $i \in X$, a_i robots start at v_i, clear edge (v_i, z), and simultaneously arrive at z. Then, for all $i \notin X$, a_i robots exit z simultaneously, clear edge (v_i, z), and arrive at v_i.

"Only if" direction: Suppose the pursuit-evasion problem has a solution. Since there are only $w(z) = k$ robots, no edge can be cleared while z is guarded. Let $X = \{a_i : (v_i, z)$ is cleared before k robots reach $z\}$. Then $\Sigma_{i \in X} a_i = k = \Sigma_{i \notin X} a_i$, and the partition instance must have a solution. □

Corollary 5 *The minimum robot problem is NP-hard for AWUL trees and AWUL series-parallel graphs. Also, the minimum distance and minimum time problems are NP-hard for AWUL stars, AWUL trees, and AWUL series-parallel graphs.*

Theorem 6 *The minimum robot problem is NP-hard for AWUL two-vertex multi-graphs.*

Proof: Let $(a_1, \ldots, a_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} a_i/2$. Construct a two-vertex multi-graph $G = (V, E)$ as follows. Let $V = \{y, z\}$, and let $E = \{e_1, e_2, \ldots, e_n\}$. Let $w(y) = 1$ and let $w(z) = k$. Also let $w(e_i) = a_i$ for $1 \leq i \leq n$. We claim that G can be cleared using $k + 1$ robots iff the partition instance has a solution.

"If" direction: Suppose the partition problem has a solution $X \subseteq \{1, \ldots, n\}$. Start all $k + 1$ robots at vertex y. One robot will remain at y throughout. For

all $i \in X$, a_i robots clear edge e_i and simultaneously arrive at z. Then, for all $i \notin X$, a_i robots exit z simultaneously, clear edge e_i, and arrive at y.

"Only if" direction: Suppose the pursuit-evasion problem has a solution. Because $w(z) = k$ and there are only $k+1$ robots, the edges cleared before z is cleared and after z is cleared must form a solution to the partition problem. □

Corollary 7 *The minimum distance and minimum time problems are NP-hard for AWUL two-vertex multi-graphs.*

Theorem 8 *The minimum robot problem is NP-hard for AWUL cliques.*

Proof: Let $(a_1, \ldots, a_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} a_i/2$. Construct a clique $G = (V, E)$ as follows. Let $V = \{v_i : 1 \leq i \leq n\}$ and $E = \{(v_i, v_j) : 1 \leq i < j \leq n\}$. Let each $w(v_i) = a_i k$, and let each $w(v_i, v_j) = a_i a_j$. We claim that G can be cleared using k^2 robots iff the partition instance has a solution.

"If" direction: Suppose the partition problem has a solution $X \subseteq \{1, \ldots, n\}$. Initially place a_i^2 robots at v_i for each $i \in X$. Also place $2a_i a_j$ robots midway along the edge (v_i, v_j) for all $i, j \in X$. Note that this is a total of k^2 robots. Along each of these edges, $a_i a_j$ robots move in each direction, all reaching the endpoints simultaneously. Now there are $a_i k$ robots at each vertex v_i such that $i \in X$. Next $a_i a_j$ robots simultaneously traverse each edge (v_i, v_j) such that $i \in X$, $j \notin X$. Now there are $a_j k$ robots at each vertex v_j such that $j \notin X$. Finally, $a_i a_j$ robots traverse from each endpoint of edge (v_i, v_j) for $i, j \notin X$, until they meet somewhere in the middle of each edge.

"Only if" direction: Suppose the pursuit-evasion problem has a solution. Consider any moment of time at which some set Y of vertices simultaneously becomes cleared (possibly $|Y| = 1$). Let X denote the vertices cleared before Y and let Z denote the vertices cleared after Y. Let $p(X) = \Sigma\{a_i : v_i \in X\}$, and similarly for Y and Z. During any solution, there must exist some such sets X, Y, and Z for which $p(X) \leq k$ and $p(X \cup Y) > k$. But $p(X \cup Y \cup Z) = 2k$, so $p(Z) < k$. At this particular moment of time, for each $x \in X$, either x is guarded or for all $z \in Z$ the edge (x, z) is guarded. Also, by assumption, each $y \in Y$ becomes guarded. Therefore $p(X)p(Z) + p(Y)k \leq k^2$, or equivalently $p(X)p(Z) + [2k - p(X) - p(Z)]k \leq k^2$. Hence $[p(X) - k][p(Z) - k] \leq 0$, from whence we must have $p(X) = k$, and X is a solution to the partition instance. □

Corollary 9 *The minimum distance and minimum time problems are NP-hard for AWUL cliques.*

In subsequent theorems, we will reduce from the strongly NP-hard *3-partition problem* [13]. An instance of 3-partition is defined by positive integers $(x_1, \ldots, x_{3n})$. Let $k = \sum_{i=1}^{3n} x_i/n$. The 3-partition problem asks whether there

exists any partition of $(x_1, \ldots, x_{3n})$ into n groups of 3 elements each, such that each group has sum exactly k.

Theorem 10 *The minimum time problem is strongly NP-hard for AWUL stars, even when all* $w(v) = 1$.

Proof: Let $(x_1, \ldots, x_{3n})$ be an instance of 3-partition, and let $k = \sum_{i=1}^{3n} x_i/n$. Construct a star G with $3n + 2$ edges $e_1, \ldots, e_{3n+2}$. The first $3n$ edges have $w(e_i) = x_i$. The remaining two edges have $w(e_{3n+1}) = w(e_{3n+2}) = k + 1$. All the vertices have $w(v) = 1$.

We claim that G can be cleared with $k+1$ robots in time $2n+2$ iff the given 3-partition instance has a solution. Essentially, subject to the given constraints, G can only be cleared as follows: Clear edge e_{3n+1} using all $k+1$ robots. Leave one robot at the central vertex. Clear three edges in parallel using exactly k robots, which then return to the central vertex. Repeat the previous step n times, until all edges $e_1, \ldots, e_{3n}$ are cleared. Finally clear edge e_{3n+2} using all $k + 1$ robots. Note that the two "extra" edges must be the first and last edge cleared, which then forces all other edges to be traversed twice. □

Corollary 11 *The minimum time problem is strongly NP-hard for AWUL trees, AWUL series-parallel graphs, and AWUL general graphs, even when all* $w(v) = 1$.

Theorem 12 *The minimum time problem is strongly NP-hard for AWUL two-vertex multi-graphs, even when all* $w(v) = 1$.

Proof: Let $(x_1, \ldots, x_{3n})$ be an instance of 3-partition, and let $k = \sum_{i=1}^{3n} x_i/n$. Construct a graph G with 2 vertices y, z and $3n + 2$ edges $e_1, \ldots, e_{3n+2}$. The first $3n$ edges have $w(e_i) = x_i$. The remaining two edges have $w(e_{3n+1}) = w(e_{3n+2}) = k + 1$. Both the vertices have $w(y) = w(z) = 1$.

We claim that G can be cleared with $k+2$ robots in time $n+2$ iff the given 3-partition instance has a solution. The justification is similar to that given above for stars: First place one robot at vertex y. Then clear edge e_{3n+1} using $k+1$ robots. Leave another robot at vertex z. Clear three edges in parallel using exactly k robots. Repeat the previous step n times, until all edges $e_1, \ldots, e_{3n}$ are cleared. Finally clear edge e_{3n+2} using $k + 1$ robots. Note that the two "extra" edges must be the first and last edge cleared, which then forces all other edges to be traversed twice. □

4.1.2 Algorithmic Results

Theorem 13 *The minimum robot problem can be solved in linear time for AWUL paths.*

Proof: If $G = (V, E)$ is a path, then obviously $\max(\{w(v) : v \in V\} \cup \{w(e) : e \in E\})$ robots is optimal. □

Theorem 14 *The minimum robot problem can be solved in polynomial time for AWUL cycles.*

Proof: Denote the cycle G by $(v_1, e_1, \ldots, v_n, e_n) = (x_1, x_2, \ldots, x_{2n-1}, x_{2n})$. First construct a directed graph $G' = (V', E')$ as follows. Let $X = V \cup E$, and initially define $V' = X \times X$. So V' corresponds to the possible borders between cleared and contaminated portions of the cycle G, that is, vertex (x_i, x_j) in G' means that the clockwise path from x_i to x_j is clear in G. E' contains edges from (x_i, x_j) to (x_{i-1}, x_j) and to (x_i, x_{j+1}), and these edges correspond to advancing the border.

For each $x_i \in V$, E' also contains edges from (x_i, x_i) to (x_{i-1}, x_{i+1}) and from (x_{i+1}, x_{i-1}) to (x_i, x_i). Note that we do *not* add similar edges to E' when $x_i \in E$.

Split each vertex (x_i, x_i) into a source and sink, so that $|V'| = 4n^2 + 2n$. Each source represents a possible starting location for the robots, and each sink represents a final location after the cycle has been cleared.

Now define a function w' on V' as follows: $w'(x_i, x_i) = w(x_i)$, and $w'(x_i, x_j) = w(x_i) + w(x_j)$ when $i \neq j$. Observe that w' represents the number of robots needed to guard the border, that is, the width of the border.

Finally, the minimum number of robots needed to clear the cycle equals the minimum possible maximum value of w' encountered along any source-to-sink path in G'. This bottleneck value of w', and also the optimal path that yields such w', can be obtained via dynamic programming. □

See Figure 11 for an example of Theorem 14. The cycle G_1 with widths w as given reduces to the directed graph G'_1, with values for function w' also given. The bottleneck value of w' is 2, which can be obtained along two source-to-sink paths: $(a, a) \to (f, e) \to (b, b)$, and $(b, b) \to (e, f) \to (a, a)$.

Also see Figure 12 for another example of Theorem 14. Cycle G_2 has widths w as given. The bottleneck value of w' is 3, which can be obtained along two source-to-sink paths in G'_2. One such optimal path is shown, and the other is its opposite. Note that in the middle of the solution sequence, one robot will travel from d to b and then back to d. So using 3 robots, the minimum distance is 10 and the minimum time is 6.

Theorem 15 *The minimum distance problem can be solved in polynomial time for AWUL paths. In fact, this result continues to hold even in the more general case when edge lengths are also arbitrary.*

Proof: First, we claim that without loss of generality all robots move from left to right on the path in an optimal solution. Suppose some solution has some pursuers moving right-to-left from vertex k to vertex j, where $j < k$. Then this solution can be transformed to another solution where these same

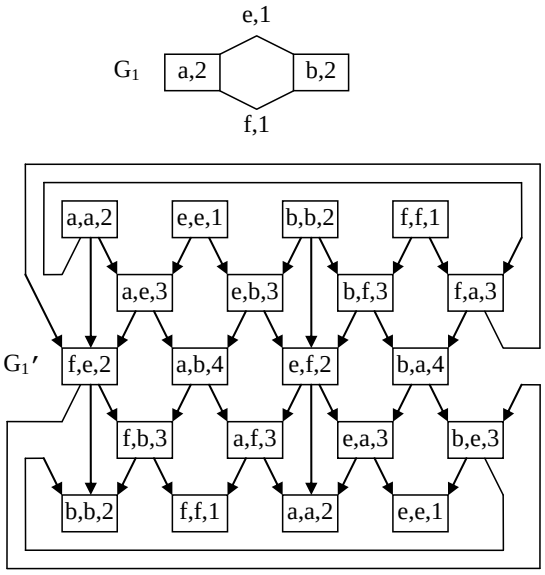

Figure 11: First Example for Theorem 14

pursuers move left-to-right from j to k. If j is the leftmost vertex they can proceed immediately, otherwise they must wait until other pursuers arrive at j from the left. Similarly if any other pursuers originally start at k and move right, these must now wait until our pursuers arrive at k from j. This transformation can increase the elapsed time, but it cannot increase the total distance, and also the number of pursuers can only decrease. Now there is no reason to ever move left, because at any given moment, all edges to the left of the current location have been cleared.

Second, given a path (V, E) create a minimum cost flow instance with nodes s, t and v_{in}, v_{out} $\forall v \in V$, and directed edges $(s, v_{in}), (v_{out}, t)$ and (v_{in}, v_{out}) $\forall v \in V$, and also edges (v_{out}, v'_{in}) $\forall (v, v') \in E$. Set the supply at s to r and the demand at t to r. All other nodes have supply zero. Put lower bound $w(v)$ on the arc (v_{in}, v_{out}), lower bound $w(e)$ on arc (v_{out}, v'_{in}) where $e = (v, v')$, and cost $L(e)$ on arc (v_{out}, v'_{in}) where $e = (v, v')$. All other lower bounds and costs equal zero.

A robot starting at node i and stopping at node j is represented by a flow $s, v_{i_{in}}, v_{i_{out}}, \ldots v_{j_{out}}, t$. Conversely, by the flow decomposition principle, any feasible flow decomposes into r such paths. The flow cost equals the total distance traveled.

Third, apply a strongly polynomial algorithm for minimum cost flows such as [38]. □

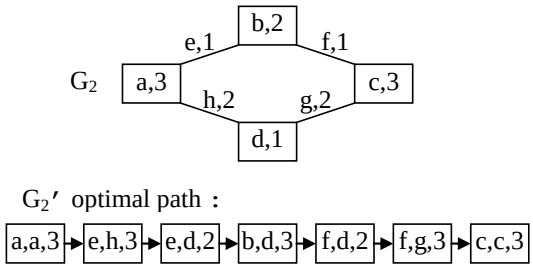

Figure 12: Second Example for Theorem 14

Theorem 16 *The minimum distance problem can be solved in pseudo-polynomial time for AWUL cycles.*

Proof: We extend the ideas in the proof of Theorem 14. Given cycle G, construct a directed graph G' to have nodes of the form (x, y, i, j, k) where x and y denote locations along the cycle, i is the number of halted robots, j is the number of robots at x, and k is the number of robots at y. The possible locations for x, y are the vertices of G and the two ends of each edge of G. If edge $e = (a, b)$, then denote the ends of edge e by e_a and e_b, which are arbitrarily close to vertices a and b respectively. Each node in G' must satisfy $w(x) \leq j$, $w(y) \leq k$, and $i + j + k \leq r$. As a special case, when $x = y$ we discard k, and let there be two nodes (a source and a sink) each having the form (x, x, i, j). The source corresponds to the starting position, and the sink corresponds to the final position after G has been cleared.

Each edge $z \rightarrow z'$ in G' corresponds to one or more valid robot moves, and is assigned a weight $W(z, z')$ equal to the shortest distance that robots must travel in G to transition from vertex z to vertex z'. All-pairs shortest distances can be precomputed for the cycle G in $O(n^2)$ time. The minimum distance solution for the pursuit-evasion problem in G will be the minimum total weight along any source-to-sink path in G'. Note that G' is a directed acyclic graph with $O(n^2r^3)$ vertices and $O(n^2r^4)$ edges, so the running time is $O(n^2r^4)$. □

For example, again consider the cycle G_2 given in Figure 12, and let $r = 3$ robots. Figure 13 shows an optimal path in the directed graph that is described in the proof of Theorem 16. The minimum total distance, computed by summing the weights W shown above the edges, is 10.

Theorem 17 *For any fixed number r of robots, the minimum time problem can be solved in pseudo-polynomial time for both AWUL paths and AWUL cycles.*

Proof sketch: Similar to Theorem 16, but more complicated. At most r different border vertices and/or edges can be guarded, so at most r disjoint

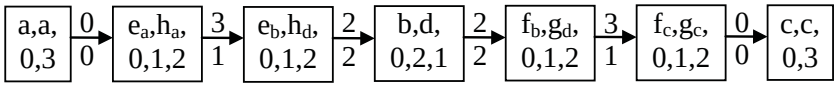

Figure 13: Example for Theorems 16 and 17

segments of the cycle can be cleared. Construct G' so that each node corresponds to a set of at most r disjoint segments, the number of robots guarding each border position, and the number of halted robots. Assign to each edge $z \to z'$ in G' a weight $W(z, z')$ equal to the shortest time needed in G to transition from vertex z to vertex z'. The minimum time solution for the pursuit-evasion problem in G has the minimum total weight along any source-to-sink path in G'. The running time is $O(n^r r^{r+1})$. □

As an example, again consider the cycle G_2 given in Figure 12, with $r = 3$ robots. Figure 13 shows an optimal path in the directed graph that is described in the proof of Theorem 17. The minimum time, obtained by summing the weights W shown below the edges, is 6.

Theorem 18 *The minimum robot problem can be solved in pseudo-polynomial-time for AWUL stars.*

Proof: Let x denote the central vertex of the star. Then any procedure for clearing the star must work in three phases as follows:

- Phase 1: Clear some edges before arriving at x. These edges must all be cleared concurrently.
- Phase 2: Clear some edges while guarding x. These edges can be cleared sequentially. If edge e is cleared in phase 2, and $w(e) \geq w(e')$, then we should also clear edge e' during phase 2.
- Phase 3: Clear some edges after departing from x. These edges must all be cleared concurrently.

All the edges of a star meet at the central vertex, so this three-phase approach is the only way to clear a star without recontamination. Any of the phases 1, 2, 3 might be empty. Here then is the pseudo-polynomial-time algorithm:

- Let m denote the number of edges. For $1 \leq i \leq m$ let each edge $e_i = (x, v_i)$. Because $\max(w(e_i), w(v_i))$ robots are needed to clear the edge e_i and its incident non-central vertex v_i, assign $w(e_i) = \max(w(e_i), w(v_i))$.
- Sort edges $e_1, \ldots, e_m$ by descending $w(e_i)$ values. Also, to properly handle the case when phase 2 might be empty, let $w(e_{m+1}) = 0$.

- For $k = 1$ to m do
  - Suppose k edges $e_1, \ldots, e_k$ will be cleared during phases 1 and 3.
  - The number of robots used during phase 2 is $w(x) + w(e_{k+1})$.
  - Let $S = \sum_{i=1}^{k} w(e_i)$.
  - To balance edges $e_1, \ldots, e_k$ between phases 1 and 3, run a pseudo-polynomial-time subset sum algorithm on $w(e_1), \ldots, w(e_k)$ that determines the largest possible sum $j \leq S/2$. [13]
  - Split $e_1, \ldots, e_k$ into two subsets that use j and $S - j$ robots respectively during phases 1 and 3.
  - Let $r_k = \max\{w(x) + w(e_{k+1}), S - j\}$.
- Choose a value k such that r_k is minimized. □

Theorem 19 *The minimum robot problem can be solved in pseudo-polynomial-time for AWUL two-vertex multi-graphs.*

Proof: Let y, z denote the two vertices, such that $w(y) \leq w(z)$. Vertex y will remain guarded during the entire procedure. Then any procedure for clearing the graph must work in three phases similar to the algorithm for stars described above, with central vertex x replaced by vertex z.

The pseudo-polynomial-time algorithm for two-vertex multi-graphs is essentially the same as the algorithm given above for stars, with just two minor changes: Replace $w(x)$ in the star algorithm by $w(z)$, and add $w(y)$ extra robots to the minimum value that is computed. □

4.2 Unit-Width Arbitrary-Length Graphs

This section considers only graphs for which all widths $w(v) = 1$ and all $w(e) = 1$. Also each length $L(e) > 0$ is arbitrary, but this is only relevant for the minimum distance and time problems. For conciseness, we denote such graphs as UWAL (unit-width arbitrary-length). The time complexity of the minimum robot problem does not depend on the edge lengths and is thus the same as for unit-width unit-length graphs. For example, we have seen already that the minimum robot problem can be solved in linear and thus polynomial time on unit-width unit-length trees.

4.2.1 Time Complexity Results

Theorem 20 *The minimum time problem is NP-hard for UWAL stars, even for fixed $r = 3$ robots.*

Proof: Let $(x_1, \ldots, x_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} x_i/2$. Construct a star G with edges $e_1, \ldots, e_{n+6}$. The first n edges have $L(e_i) = x_i$, and the other six edges have $L(e_i) = k$.

We claim that the given partition instance has a solution iff G can be cleared with three robots in time $4k$, as follows: Initially place three robots at the outer endpoints of three of the six length-k edges (that is, not at the center vertex). First traverse these three edges to reach the center vertex in time k. While one robot guards the center vertex, the other two robots traverse each of the first n edges twice (once in each direction) in time $2k$, which is only possible if the partition instance has a solution. Finally traverse the remaining three length-k edges heading away from the center vertex in time k. □

Corollary 21 *The minimum time problem is NP-hard for UWAL trees, UWAL series-parallel graphs, and UWAL general graphs, even for fixed $r = 3$ robots.*

Theorem 22 *The minimum time problem is NP-hard for UWAL two-vertex multi-graphs, even for fixed $r = 4$ robots.*

Proof: Let $(x_1, \ldots, x_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} x_i/2$. Construct a graph G with vertices a, b and edges $e_1, \ldots, e_{n+7}$. The first n edges have $L(e_i) = 2x_i$, the next six edges have $L(e_i) = 2k$, and the remaining edge has $L(e_{n+7}) = 1$.

We claim that the given partition instance has a solution iff G can be cleared with four robots in time $6k + 1$, as follows: First, while one robot guards vertex a, the other robots traverse three of the six length-$2k$ edges from a to b in time $2k$. Next, while two robots guard a and b, the other two traverse each of the first n edges in time $2k$, which is only possible if the partition instance has a solution. Then, either one or both of those two robots traverse edge e_{n+7} in unit time so that afterward there are three robots at one vertex (say x) and one robot at the other vertex (say y). Finally, while one robot guards vertex y, the other robots traverse the remaining three length-$2k$ edges from x to y in time $2k$. □

Theorem 23 *The minimum time problem is strongly NP-hard for UWAL stars.*

Proof: Let $(x_1, \ldots, x_{3n})$ be an instance of 3-partition, and let $k = \sum_{i=1}^{3n} x_i/n$. Construct a star G with edges $e_1, \ldots, e_{5n+2}$. The first $3n$ edges have $L(e_i) = x_i$, and the other $2n + 2$ edges have $L(e_i) = k$.

We claim that the given 3-partition instance has a solution iff G can be cleared with $n + 1$ robots in time $4k$, as follows: First traverse $n + 1$ of the $2n + 2$ length-k edges heading toward the center vertex in time k. While one robot guards the center vertex, the other n robots traverse each of the first $3n$ edges twice (once in each direction) in time $2k$, which is only possible if the 3-partition instance has a solution. Finally traverse the remaining $n + 1$ length-k edges heading away from the center vertex in time k. □

Corollary 24 *The minimum time problem is strongly NP-hard for UWAL trees, UWAL series-parallel graphs, and UWAL general graphs.*

Theorem 25 *The minimum time problem is strongly NP-hard for UWAL two-vertex multi-graphs.*

Proof: Let $(x_1, \ldots, x_{3n})$ be an instance of 3-partition, and let $k = \sum_{i=1}^{3n} x_i/n$. Construct a graph G with vertices a, b and edges $e_1, \ldots, e_{5n+2}$. The first $3n$ edges have $L(e_i) = x_i$, and the other $2n+2$ edges have $L(e_i) = k$.

We claim that the given 3-partition instance has a solution iff G can be cleared with $n+2$ robots in time $3k$, as follows: First, while one robot guards vertex a, the other robots traverse $n+1$ of the $2n+2$ length-k edges from a to b in time k. Next, while two robots guard a and b, the other n robots traverse each of the first $3n$ edges in time k, which is only possible if the 3-partition instance has a solution. Observe that each of these n robots must traverse exactly three such edges, from b to a to b to a. Finally, while one robot guards vertex b, the other robots traverse the remaining $n+1$ length-k edges from a to b in time k. □

Theorem 26 *The minimum time problem is strongly NP-hard for UWAL cliques.*

Proof: Let $(x_1, \ldots, x_{3n})$ be an instance of 3-partition, and let $k = \sum_{i=1}^{3n} x_i/n$. Note that if n is even, we can reduce it to the case when n is odd by defining $x_{3n+1} = 1$, $x_{3n+2} = 1$, $x_{3n+3} = k-2$, and $n' = n+1$. [For our reduction, it won't matter if these values are not between $k/4$ and $k/2$.] So without loss of generality, we can assume that n is odd.

Construct a clique G with $6n+2$ vertices $\{a\} \cup \{b_1, \ldots, b_{3n}\} \cup \{c_0, \ldots, c_{3n}\}$. Let each $L(a, b_i) = x_i$, each $L(b_i, b_j) = 2k+1$, and each $L(c_i, c_j) = 6k$. Also let $L(a, c_i) = k-1$ when $i \leq n$, $L(a, c_i) = 3k+1$ when $n < i \leq 2n$, and $L(a, c_i) = 3k$ when $i > 2n$. Finally let each $L(c_i, b_j) = k-1$ when $i < 3n/2$, and $L(c_i, b_j) = 3k$ when $i > 3n/2$.

We claim that the given 3-partition instance has a solution iff G can be cleared with $(3n+1)(6n+1)$ robots in time $3k$, as follows:

"If" direction: Begin with $6n+1$ robots at each c_i vertex, and let these $6n+1$ robots traverse the $6n+1$ edges outward from c_i. Note that each edge (c_i, c_j) with length $6k$ will be cleared in exactly $3k$ time, so we can now consider only the remaining edges. Of the $(3n+1)/2$ robots that arrive at each b_j at time $k-1$, one remains at b_j and the others head toward $b_{j+1}, \ldots, b_{j+(3n-1)/2}$. [Arithmetic in preceding subscripts is modulo $3n$.] Hence all the (b_i, b_j) edges and remaining (c_i, b_j) edges will be cleared at time $3k$.

The only edges yet to be considered are the edges incident to vertex a. Note that $n+1$ robots will arrive at a at time $k-1$. One robot remains at a, and the other n robots traverse each of the (a, b_i) edges twice (once in each direction)

Graph $G = (V, E)$	Minimum robots
Path	1
Cycle	2
Star with $\|E\| \geq 3$	2
Two-vertex multi-graph with $\|E\| \geq 3$	3
Clique with $\|V\| = n \geq 4$	n

Figure 14: Simple Cases

in time $2k$, which is only possible if the 3-partition instance has a solution. At time $3k - 1$ there will again be $n + 1$ robots at a. One remains at a, and the other n robots head toward $c_{n+1}, \ldots, c_{2n}$. Hence all the remaining (a, c_i) edges will be cleared at time $3k$.

"Only if" direction: The opposite direction of this proof is a simple but huge case analysis, and is omitted. □

4.2.2 Algorithmic Results

Proposition 27 *We begin with some simple cases as shown in Figure 14. For each kind of UWAL graph specified in the left column, the minimum number of robots needed to clear such a graph is given in the right column.*

Proof: All cases except possibly the clique should be obvious. For the clique, start $n - 1$ robots at any $n - 1$ distinct vertices, and use the n^{th} robot to clear all the edges between these $n - 1$ vertices. Finally the original $n - 1$ robots each move to the one remaining vertex. □

Theorem 28 *The minimum distance and minimum time problems can be solved in polynomial time for UWAL paths.*

Proof: Let L denote the length of the path, and let r denote the number of robots. Divide the path into r segments $s_1, \ldots, s_r$ of length L/r each. Note that each endpoint between two consecutive segments does not necessarily coincide with a vertex. For $1 \leq i \leq r$, place robot i at the left endpoint of its segment if i is odd, and otherwise place robot i at the right endpoint of its segment. Now all robots simultaneously move exactly distance L/r each to clear their respective segments. So the minimum possible time is L/r, and the minimum possible total distance is L. □

Theorem 29 *The minimum distance and minimum time problems can be solved in polynomial time for UWAL cycles.*

Proof: Let L denote the total length of the cycle, and let $r \geq 2$ denote the number of robots. Define $r' = r$ if r is even, and $r' = r - 1$ if r is odd, so r' is even in either case. Beginning at any point, divide the cycle into r' segments $s_1, \ldots, s_{r'}$ of length L/r' each. Arbitrarily use "left" for clockwise, and "right" for counterclockwise around the cycle. For $1 \leq i \leq r'$, place robot i at the left endpoint of its segment if i is odd, and otherwise place robot i at the right endpoint of its segment. Now all robots simultaneously move exactly distance L/r' each to clear their respective segments. So the minimum possible time is L/r', and the minimum possible total distance is L. □

Theorem 30 *The minimum distance problem can be solved in polynomial time for UWAL stars.*

Proof: If $|E| \leq 2$ then solve using the path algorithm. Now suppose $|E| \geq 3$, and hence $r \geq 2$. First consider when the number of edges $|E| \geq 2r$. Find $A \subseteq E$ which consists of the longest $2r$ edges. The shortest distance is obtained by first traversing r of the edges in A heading toward the center vertex, then traversing each edge in $E - A$ twice (once in each direction) while the center vertex remains guarded, and finally traversing the remaining r edges of A heading away from the center. Alternatively, if $|E| < 2r$, then the minimum distance is trivially $\Sigma_{e \in E}\ L(e)$. □

Theorem 31 *The minimum distance problem can be solved in polynomial time for UWAL two-vertex multi-graphs.*

Proof: If $|E| = 1$ then $r \geq 1$, and solve using the path algorithm. If $|E| = 2$ then $r \geq 2$, and solve using the cycle algorithm. Now suppose $|E| \geq 3$, and hence $r \geq 3$. One robot resides at each of the two vertices, and a third robot traverses each edge. The minimum distance is $\Sigma_{e \in E}\ L(e)$. □

Theorem 32 *The minimum distance problem can be solved in polynomial time for UWAL cliques.*

Proof: If $|V| \leq 3$, then solve using the path or cycle algorithm. Now suppose $|V| \geq 4$, and hence $r \geq |V|$, so without loss of generality assume $r = |V|$. First let $r - 1$ robots start at some vertex z and traverse the edges to the other $r - 1$ vertices. If r is even, the r^{th} robot now traverses an Eulerian circuit of $G - z$. Alternatively, if r is odd, let M be any perfect matching in $G - z$. The r^{th} robot now traverses an Eulerian circuit of $G - z - M$, and then it is easy for the remaining $r - 1$ robots to clear the edges of M. In any case, the minimum distance is $\Sigma_{e \in E}\ L(e)$. □

5 Conclusion

We studied pursuit-evasion problems where a number of robots have to clear a given graph, and we provided a catalog of algorithmic and complexity results for several kinds of pursuit-evasion problems on several classes of graphs. First, we described a linear-time algorithm, called CLEARTHETREE, that solves the minimum robot problem on unit-width unit-length trees. Second, we either developed optimal polynomial-time algorithms or proved the NP-hardness of the minimum robot, distance and time problems on both unit-width arbitrary-length and arbitrary-width unit-length graphs of a variety of simple topologies. It is future research to resolve the open cases in Figure 9, and to extend our results to other formulations of pursuit-evasion problems [22] [4].

Acknowledgments

This article is based on a conference paper [9]. It adds new material to these conference papers, namely the minimum cost flow method to minimize distance on paths, and a different description of the algorithm to minimize distance on cycles. This article is based upon research supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number W911NF-08-1-0468, by ONR in form of a MURI under contract/grant number N00014-09-1-1031 and by NSF under contract 0413196 (while serving at NSF). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

References

[1] N. Agmon, S. Kraus, and G. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2339–2345, 2008.

[2] N. Agmon, V. Sadov, S. Kraus, and G. Kaminka. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 55–62, 2008.

[3] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, 2002.

[4] D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.

[5] H. Bodlaender and T. Kolks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal on Algorithms*, 21:358–402, 1996.

[6] H. Bodlaender, T. Kolks, and D. Kratsch. Treewidth and pathwidth of permutation graphs. *SIAM Journal on Discrete Mathematics*, 8:606–616, 1995.

[7] H. Bodlaender and R. Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6:181–188, 1993.

[8] H. Bodlaender and D. Thilikos. Computing small search numbers in linear time. In *Proceedings of the International Workshop on Parameterized and Exact Computation*, pages 37–48, 2004.

[9] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for pursuit-evasion problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 59–66, 2009.

[10] J. Ellis, I. Sudborough, and J. Turner. The vertex separation and search number of a graph. *Information and Computation*, 113(1):50–74, 1994.

[11] J. Ellis and R. Warren. Lower bounds on the pathwidth of some grid-like graphs. *Discrete Applied Mathematics*, 156(5):545–555, 2008.

[12] R. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

[13] M. Garey and D. Johnson. *Computers and Intractability*. Freeman and Company, New York, 1979.

[14] B. Gerkey. http://ai.stanford.edu/∼gerkey/research/pe/. Retrieved in 2011.

[15] G. Gordon, S. Thrun, and B. Gerkey. Visibility-based pursuit-evasion with limited field of view. In *Proceedings of the National Conference on Artificial Intelligence*, pages 20–27, 2004.

[16] J. Gustedt. On the path width of chordal graphs. *Discrete Applied Mathematics*, 45:233–248, 1993.

[17] G. Hollinger. Search in the physical world. Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh PA, 2010.

[18] G. Hollinger, A. Kehagias, and S. Singh. Improving the efficiency of clearing with multi-agent teams. *International Journal of Robotics Research*, 29(8):1088–1105, 2010.

[19] A. Kehagias, G. Hollinger, and S. Singh. A graph search algorithm for indoor pursuit/evasion. *Mathematical and Computer Modeling*, 50(9–10):1305-1317, 2009.

[20] N. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42:345–350, 1992.

[21] L. Kirousis and C.H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55:181–184, 1985.

[22] M. Kirousis and C. Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47(2):205–218, 1986.

[23] A. Kolling. Multi-robot pursuit-evasion. Ph.D. dissertation, School of Electrical Engineering and Computer Science, University of California at Merced, Merced CA, 2009.

[24] A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. *Proceedings of the IEEE/RSI International Conference on Intelligent Robots and Systems*, pages 2323–2328, 2008.

[25] A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 26(1):32–47, 2010.

[26] A. Kolling and S. Carpin. Multi-robot surveillance: an improved algorithm for the GRAPH-CLEAR problem. *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pages 2360–2365, 2008.

[27] A. Kolling, A. Kleiner, M. Lewis, and K. Sycara. Pursuit-evasion in 2.5d based on team-visibility. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4610–4616, 2010.

[28] A. Kolling, A. Kleiner, M. Lewis, and K. Sycara. Computing and executing strategies for moving target search. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2011 (in print).

[29] A. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40(2):224–245, 1993.

[30] F. Makedon and I. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243–265, 1989.

[31] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou. The complexity of searching a graph. *Journal of the ACM*, 35(1):18–44, 1988.

[32] F. Monien and I. Sudborough. Min cut is NP-complete for edge weighted trees. *Theoretical Computer Science*, 58:209–229, 1988.

[33] T. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. Lick, editors, *Theory and Applications of Graphs*, Lecture Notes in Mathematics, pages 426–441. Springer Verlag, 1976.

[34] D. Pellier and H. Fiorino. Coordinated exploration of unknown labyrinthine environments applied to the pursuit evasion problem. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 895–902, 2005.

[35] J. Pita, M. Jain, J. Marecki, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed armor protection: the application of a game theoretic model for security at the Los Angeles international airport. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 125–132, 2008.

[36] B. Simov, G. Slutzki, and S. LaValle. Pursuit-evasion using beam detection. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1657–1662, 2000.

[37] K. Skodinis. Computing optimal linear layouts of trees in linear time. In *Proceedings of the Annual European Symposium on Algorithms*, pages 403–414, 2000.

[38] Eva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–255, 1985.

[39] D. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Applied Mathematics*, 105:239–271, 2000.

[40] R. Vidal, O. Shakernia, H. Kim, D. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662–669, 2002.