Algorithms and Complexity Results for Pursuit-Evasion Problems[*]

Richard Borie
Computer Science
University of Alabama
borie@cs.ua.edu

Craig Tovey
Industrial and Systems Engineering
Georgia Institute of Technology
craig.tovey@isye.gatech.edu

Sven Koenig
Computer Science
University of Southern California
skoenig@usc.edu

Abstract

We study pursuit-evasion problems where a number of pursuers have to clear a given graph. We study when polynomial-time algorithms exist to determine how many pursuers are needed to clear a given graph and how a given number of pursuers should move on the graph to clear it with either a minimum sum of their travel distances or minimum task-completion time. We generalize prior work to both unit-width arbitrary-length and unit-length arbitrary-width graphs and derive both algorithms and complexity results for a variety of graph topologies. In this context, we describe a polynomial-time algorithm, called CLEARTHETREE, that is much shorter and algorithmically simpler than the state-of-the-art algorithm for the minimum pursuer problem on trees. Our theoretical research lays a firm theoretical foundation for pursuit evasion on graphs and informs practitioners about which problems are easy and which ones are hard.

1 Introduction

We study a standard formulation of pursuit-evasion problems from the literature where the pursuers and evaders move on the edges of a given graph and are able to stop or change directions anywhere on edges [Parsons, 1976]. The pursuers know the graph but not how many evaders are present. They must catch all evaders despite zero visibility. Evaders can move infinitely fast but cannot pass by a sufficiently large group of pursuers in the same location. The necessary group size is called the width of a vertex or edge. Prior work has focused on graphs where all vertex and edge widths and all edge lengths are one. We generalize these results to unit-width arbitrary-length and unit-length arbitrary-width graphs. The rules of the game are as follows:

- Initially the entire graph G is contaminated (evaders may reside anywhere within G).
- Each pursuer may start and finish at any location in G.
- Each pursuer may travel along edges of G, but may not travel outside G.
- The width w(e) of edge e is the least number of pursuers that can clear (decontaminate) e by traversing e simultaneously.
- The width w(v) of vertex v is the least number of pursuers needed to guard v when v is incident upon both cleared and contaminated edges.

Our paper addresses open questions of when polynomial-time algorithms exist to determine how many pursuers are needed to clear a given graph (Minimum Pursuer Problem) and how a given number of pursuers should move on the graph to clear it with either a minimum sum of travel distances or a minimum task-completion time (Minimum Distance and Time Problems). We do this by studying simple graph topologies, including paths, cycles, stars, trees, two-vertex multigraphs, series-parallel graphs and cliques. Our results are negative in many cases, which suggests that it is important to develop heuristic approaches for these cases. In the other cases, we outline polynomial-time algorithms. In particular, we describe CLEARTHETREE, a linear-time algorithm that is algorithmically much simpler than the state-of-the-art algorithm for the minimum pursuer problem on trees [Megiddo *et al.*, 1988]. Our new algorithm provides an important stepping stone towards polynomial-time algorithms for more general graph topologies.

Previously, it has been shown that polynomial-time algorithms for many combinatorial graph problems on trees (that is, treewidth-1 graphs) and series-parallel graphs (treewidth-2 graphs) can be generalized to polynomial-time algorithms on treewidth-k graphs for every k [Arnborg *et al.*, 1991; Borie *et al.*, 1992; Courcelle and Mosbah, 1993]. Real-world graphs often have small treewidth; for example, one author's office building is treewidth-2 and another's is treewidth-3, and Mammoth Cave is treewidth-4. So treewidth-k graphs for $k > 1$ are realistic models for pursuit-evasion problems in buildings and cave systems, also streets and computer networks. It remains future work to determine whether our algorithms can indeed be generalized to treewidth-k graphs for arbitrary k.

1.1 Related Work

Previous work on this problem has focused on the minimum pursuer problem for unit-width graphs. The decision version of the minimum pursuer problem is in NP for general graphs [LaPaugh, 1993], and this result can be extended to show that the decision versions of all problems in this paper are

[*]This material is based upon work 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 and by NSF under contract 0413196.

in NP. The minimum pursuer problem is NP-hard on general graphs but can be solved in linear time on trees [Megiddo *et al.*, 1988], interval graphs [Kirousis and Papadimitriou, 1985] and grid graphs [Ellis and Warren, 2008]. For example, an m-by-n grid graph can be cleared by $\min(m, n) + 1$ pursuers.

Many other variations of the pursuit-evasion problem have been considered in the literature. Some focus on the questions of minimizing the number of pursuers, time, or distance, as we do, but make different assumptions about movement or capture. In *node searching* problems, pursuers and evaders jump from vertex to adjacent vertex in a fraction of or a single time step, and an evader is captured if at some time it is at the same vertex as a pursuer [Kirousis and Papadimitriou, 1986; Bienstock and Seymour, 1991]. Our *edge searching* problem is more suited to robot, vehicle, or human movement for which an edge traversal has a physical meaning. In the robotics and AI literature, progress has been made on questions of perimeter guarding [Agmon *et al.*, 2008a; 2008b]. In this line of research, guards move along edges, as in our problem, but intruder speeds are bounded and intruder movement may not be restricted to the graph. Capture is different since guards detect intruders within some given radius of capture. In these problems, it is usually impossible to guarantee that the perimeter will not be breached. Instead, the goal is to minimize the probability or expected time to success for the intruders, usually by means of a randomized strategy that is not predictable and has desirable game-theoretic properties. See [Pita *et al.*, 2008] for an application to airport security, and references therein for underlying theory. Other research includes both land and aerial pursuers, for example [Vidal *et al.*, 2002].

1.2 Preliminaries

Figure 1 summarizes our results. The class of two-vertex graphs are multigraphs; that is, multiple edges exist between the two vertices. The two starred entries are found in [Megiddo *et al.*, 1988], although we do give an alternate proof for one of these in Subsection 3.2. Each other result is either derived directly in Section 2 or 3, or can be inferred from a derived result in conjunction with the following relations.

Proposition 1 *If algorithm A solves problem Q on graph class C, then algorithm A also solves problem Q for any subclass of C, aside from the issue of recognizing membership in the subclass. Conversely, if problem Q is NP-hard for polynomial-time recognizable graph class C, then problem Q remains NP-hard for any superclass of C.*

Paths and stars are subclasses of trees; trees, cycles, and two-vertex graphs are subclasses of series-parallel graphs; trees, cycles, and cliques are subclasses of general graphs; and series-parallel graphs are subclasses of multigraphs. Membership in each of these classes can be determined in linear time.

Proposition 2 *Suppose the minimum pursuer 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 pursuers is sufficient to clear the graph. So, if any such approximation algorithm A exists, then the minimum pursuer problem could be solved in polynomial time by checking whether A returns a finite value. □

Section 2 describes our algorithms and complexity results for unit-length arbitrary-width graphs, and Section 3 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 number of pursuers needed to clear it, and each vertex v has a width $w(v)$, that denotes the number of pursuers needed to guard it. We consider both decision and optimization versions of the minimum pursuer, distance and time problems. For example, the decision version of the minimum pursuer problem is to determine whether a given number of pursuers can clear a given graph, and the optimization version is to determine the minimum number of pursuers.

2 Unit-Length Arbitrary-Width Graphs

This section considers only graphs for which all $L(e) = 1$. Note that each $w(v) \geq 1$ and $w(e) \geq 1$.

2.1 Complexity Results

In the following theorems, we will reduce from the NP-complete *partition problem* [Garey and Johnson, 1979]. 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 3 *The minimum pursuer problem is NP-hard for 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 pursuers 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 pursuers start at v_i, clear edge (v_i, z), and simultaneously arrive at z. Then, for all $i \notin X$, a_i pursuers 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 $w(z) = k$ pursuers, no edge can be cleared while z is guarded. Let $X = \{a_i : (v_i, z)$ is cleared before k pursuers reach $z\}$. Then $\Sigma_{i \in X} a_i = k = \Sigma_{i \notin X} a_i$, and the partition instance must have a solution. □

Theorem 4 *The minimum pursuer problem is NP-hard for 2-vertex multigraphs.*

Proof: Let $(a_1, \ldots, a_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} a_i/2$. Construct a 2-vertex 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

	Unit-Length Arbitrary-Width Graphs			Unit-Width Arbitrary-Length Graphs		
	Minimum Pursuer	Minimum Distance	Minimum Time	Minimum Pursuer	Minimum Distance	Minimum Time
Paths	P	Pseudo-P	Pseudo-P	P	P	P
Cycles	P	Pseudo-P	Pseudo-P	P	P	P
Stars	NP-Complete, Pseudo-P	NP-Complete	Strongly NP-Complete	P	P	Strongly NP-Complete
Trees	NP-Complete	NP-Complete	Strongly NP-Complete	P*	(Open)	Strongly NP-Complete
Two-Vertex Graphs	NP-Complete, Pseudo-P	NP-Complete	Strongly NP-Complete	P	P	Strongly NP-Complete
Series-Parallel Graphs	NP-Complete	NP-Complete	Strongly NP-Complete	(Open)	(Open)	Strongly NP-Complete
Cliques	NP-Complete	NP-Complete	NP-Complete	P	P	Strongly NP-Complete
General Graphs	NP-Complete	NP-Complete	Strongly NP-Complete	NP-Complete*	NP-Complete	Strongly NP-Complete

Figure 1: Summary of Results

$1 \leq i \leq n$. We claim that G can be cleared using $k+1$ pursuers iff the partition instance has a solution.

"If" direction: Suppose the partition problem has a solution $X \subseteq \{1,\ldots,n\}$. Start all $k+1$ pursuers at vertex y; one pursuer will remain at y throughout. For all $i \in X$, a_i pursuers clear edge e_i and simultaneously arrive at z. Then, for all $i \notin X$, a_i pursuers 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 $k+1$ pursuers, the edges cleared before z is cleared and after z is cleared must form a solution to the partition problem. □

Theorem 5 *The minimum pursuer problem is NP-hard for 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 pursuers 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 pursuers at v_i for each $i \in X$. Also place $2a_i a_j$ pursuers midway along the edge (v_i,v_j) for all $i,j \in X$. Note that this is a total of k^2 pursuers. Along each of these edges, $a_i a_j$ pursuers move in each direction, all reaching the endpoints simultaneously. Now there are $a_i k$ pursuers at each vertex v_i such that $i \in X$. Next $a_i a_j$ pursuers simultaneously traverse each edge (v_i,v_j) such that $i \in X$, $j \notin X$. Now there are $a_j k$ pursuers at each vertex v_j such that $j \notin X$. Finally, $a_i a_j$ pursuers 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. □

In subsequent theorems, we will reduce from the strongly NP-complete *3-partition problem* [Garey and Johnson, 1979]. 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 6 *The minimum time problem is strongly NP-hard for 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 $k+1$ pursuers 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 $k+1$ pursuers. Leave one pursuer at the central vertex. Clear three edges in parallel using exactly k pursuers, 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 $k+1$ pursuers. □

In both the previous and following proofs, the two "extra" edges must be the first and last edge cleared, which then forces all other edges to be traversed twice.

Theorem 7 *The minimum time problem is strongly NP-hard for 2-vertex 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 $k+2$ pursuers 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 pursuer at vertex y. Then clear edge e_{3n+1} using $k+1$ pursuers. Leave another pursuer at vertex z. Clear three edges in parallel using exactly k pursuers. Repeat the previous step n times, until all edges $e_1,\ldots,e_{3n}$ are cleared. Finally clear edge e_{3n+2} using $k+1$ pursuers. □

2.2 Algorithmic Results

Theorem 8 *The minimum pursuer problem can be solved in linear time for paths.*

Proof: Let $G = (V, E)$ be a path, and let $r = \max(\{w(v) : v \in V\} \cup \{w(e) : e \in E\})$. It is not difficult to see that r pursuers is both necessary and sufficient. All r pursuers start at one endpoint of the path, and move together until they reach the other endpoint. □

Theorem 9 *The minimum pursuer problem can be solved in polynomial time for 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 digraph $G' = (V', E')$ 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 pursuers, 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 pursuers needed to guard the border, that is, the width of the border.

Finally, the minimum number of pursuers 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 2 for an example of Theorem 9. The cycle G_1 with widths w as given reduces to the digraph 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 3 for another example of Theorem 9. 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 pursuer will travel from d to b and then back to d. So using 3 pursuers, the minimum distance is 10 and the minimum time is 6.

Theorem 10 *The minimum distance problem can be solved in pseudo-polynomial time for paths.*

Proof: Denote the path by $(v_1, e_1, v_2, \ldots, e_{n-1}, v_n) = (x_1, x_2, x_3, \ldots, x_{2n-2}, x_{2n-1})$. Minimum distance can be obtained by marching along the path from v_1 to v_n, with possibly some pursuers halting or new pursuers starting at each point. For $0 \leq i \leq 2n-1$ and $0 \leq j \leq k \leq r$, define $D(i, j, k)$ as the minimum distance needed to clear the subpath $(x_1, \ldots, x_i)$ such that j pursuers have halted and k total pursuers have been used. We give a recursive definition of $D(i, j, k)$ such that the solution $D(2n-1, r, r)$ can be computed in $O(nr^2)$ time using dynamic programming. For all i, j, k, let $D(i, j, k)$ be the minimum of these four values:

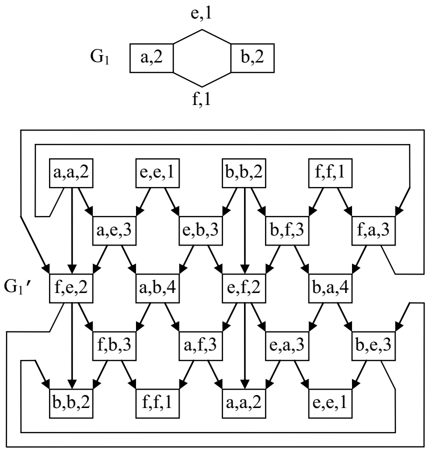

Figure 2: First Example for Theorem 9

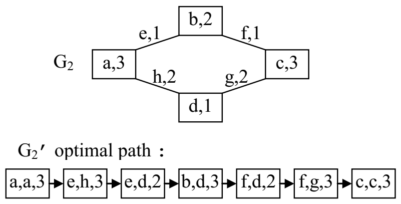

Figure 3: Second Example for Theorem 9

- If $i = 0$ then 0 else ∞. (Initialization.)
- If $i > 0$ and $k - j \geq w(i)$ then $D(i-1, j, k) + (k-j) \cdot L(x_i)$ else ∞. (This corresponds to clearing the vertex or edge x_i. For convenience, say $L(x_i) = 0$ when $x_i \in V$.)
- If $j > 0$ then $D(i, j-1, k)$ else ∞. (This corresponds to halting a pursuer at x_i.)
- If $k > 0$ then $D(i, j, k-1)$ else ∞. (This corresponds to starting a new pursuer at x_i.) □

Theorem 11 *The minimum distance problem can be solved in pseudo-polynomial time for cycles.*

Proof sketch: Combine the ideas in the proofs of Theorems 9 and 10. Given cycle G, construct G' to have nodes of the form (x_i, x_j, k, l, m) where x_i and x_j are as in Theorem 9, k is the number of halted pursuers, l is the number of pursuers at x_i, and m is the number of pursuers at x_j. Each node must satisfy $w(x_i) \leq l$, $w(x_j) \leq m$, and $k + l + m \leq r$. As a special case, when $i = j$, there are two nodes (a source and a sink) each having the form (x_i, x_i, k, l).

Each edge (y', z') in G' corresponds to a valid pursuer move, and is assigned a weight $W(y', z')$ equal to the shortest distance that pursuers must travel in G to transition from state y' to state z'. 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'. The running time is $O(n^2r^3)$. □

Theorem 12 *For any fixed number r of pursuers, the minimum time problem can be solved in pseudo-polynomial time for both paths and cycles.*

Proof sketch: Similar to Theorem 11, but more complicated. At most r different border vertices and/or edges can be guarded, so at most r disjoint 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 pursuers guarding each border location, and the number of halted pursuers. Assign to each edge (y', z') in G' a weight $W(y', z')$ equal to the shortest time needed in G to transition from state y' to state 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 $O(n^r r^{r+1})$. □

Theorem 13 *The minimum pursuer problem can be solved in pseudo-polynomial-time for 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. Phases 1 and/or 3 might be empty.

Here then is the pseudo-polynomial-time algorithm:

- Sort edges $e_1, \dots, e_m$ by descending $w(e)$ values.
- For $k = 1$ to m do
  - Suppose k edges $e_1, \dots, e_k$ will be cleared during phases 1 and 3.
  - Then the number of pursuers used during phase 2 is $w(x) + w(e_{k+1})$.
  - Let $S = \sum_{i=1}^{k} w(e_i)$.
  - We want to balance $e_1, \dots, e_k$ between phases 1 and 3.
  - Run a pseudo-polynomial-time *subset sum* algorithm on $w(e_1), \dots, w(e_k)$ to determine the largest possible sum $j \leq S/2$. [Garey and Johnson, 1979]
  - Split $e_1, \dots, e_k$ into two subsets that use j and $S - j$ pursuers 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 14 *The minimum pursuer problem can be solved in pseudo-polynomial-time for 2-vertex 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 2-vertex 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 pursuers to the minimum value that is computed. □

3 Unit-Width Arbitrary-Length Graphs

This section considers only graphs for which all $w(v) = 1$ and all $w(e) = 1$. Also $L(e) > 0$ is arbitrary, but this is only relevant when minimizing distance or time.

3.1 Complexity Results

Theorem 15 *The minimum time problem is NP-hard for stars, even for fixed $r = 3$ pursuers.*

Proof: Let $(x_1, \dots, x_n)$ be an instance of partition, and let $k = \sum_{i=1}^{n} x_i/2$. Construct a star G with edges $e_1, \dots, e_{n+6}$. 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 pursuers in time $4k$, as follows: First traverse three of the six length-k edges heading toward the center vertex in time k. While one pursuer guards the center vertex, the other two pursuers 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. □

Theorem 16 *The minimum time problem is NP-hard for 2-vertex graphs, even for fixed $r = 4$ pursuers.*

Proof: Let $(x_1, \dots, x_n)$ 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 $e_1, \dots, e_{n+7}$. 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 pursuers in time $6k + 1$, as follows: First, while one pursuer guards vertex a, the other pursuers traverse three of the six length-$2k$ edges from a to b in time $2k$. Next, while two pursuers 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 pursuers traverse edge e_{n+7} in unit time so that afterward there are three pursuers at one vertex (say x) and one pursuer at the other vertex (say y). Finally, while one pursuer guards vertex y, the other pursuers traverse the remaining three length-$2k$ edges from x to y in time $2k$. □

Theorem 17 *The minimum time problem is strongly NP-hard for stars.*

Proof: Let $(x_1, \dots, x_{3n})$ be an instance of 3-partition, and let $k = \sum_{i=1}^{3n} x_i/n$. Construct a star G with edges $e_1, \dots, e_{5n+2}$. 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 $n+1$ pursuers 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 pursuer guards the center vertex, the other n pursuers 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. □

Theorem 18 *The minimum time problem is strongly NP-hard for 2-vertex 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 $n+2$ pursuers in time $3k$, as follows: First, while one pursuer guards vertex a, the other pursuers traverse $n+1$ of the $2n+2$ length-k edges from a to b in time k. Next, while two pursuers guard a and b, the other n pursuers 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 pursuers must traverse exactly three such edges, from b to a to b to a. Finally, while one pursuer guards vertex b, the other pursuers traverse the remaining $n+1$ length-k edges from a to b in time k. □

Theorem 19 *The minimum time problem is strongly NP-hard for 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 $(3n+1)(6n+1)$ pursuers in time $3k$, as follows:

($\Rightarrow$) Begin with $6n+1$ pursuers at each c_i vertex, and let these $6n+1$ pursuers 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 $(3n+1)/2$ pursuers 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 $n+1$ pursuers will arrive at a at time $k-1$. One pursuer remains at a, and the other n pursuers traverse each of the (a, b_i) edges twice (once in each direction) in time $2k$, which is only possible if the 3-partition instance has a solution. At time $3k-1$ there will again be $n+1$ pursuers at a. One remains at a, and the other n pursuers head toward $c_{n+1}, \ldots, c_{2n}$. Hence all the remaining (a, c_i) edges will be cleared at time $3k$.

($\Leftarrow$) The opposite direction of this proof is a simple but huge case analysis (omitted due to space limitations). □

3.2 Algorithmic Results

Proposition 20 *We begin with some simple cases: (i) The minimum number of pursuers needed to clear a path is 1. (ii) The minimum number of pursuers needed to clear a cycle is 2. (iii) The minimum number of pursuers needed to clear a star with 3 or more edges is 2. (iv) The minimum number of pursuers needed to clear a 2-vertex graph with 3 or more edges is 3. (v) The minimum number of pursuers needed to clear a clique with $n \geq 4$ vertices is n.*

Theorem 21 *The minimum time and distance problems can be solved in polynomial time for paths.*

Proof: Let L denote the length of the path, and let r denote the number of pursuers. Divide the path into r segments $s_1, \ldots, s_r$ of length L/r each. Note: each endpoint between two consecutive segments does not necessarily coincide with a vertex. For $1 \leq i \leq r$, place pursuer i at the left endpoint of its segment if i is odd, and otherwise place pursuer i at the right endpoint of its segment. Now all pursuers 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 22 *The minimum time and distance problems can be solved in polynomial time for cycles.*

Proof: Let L denote the total length of the cycle, and let $r \geq 2$ denote the number of pursuers. 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 pursuer i at the left endpoint of its segment if i is odd, and otherwise place pursuer i at the right endpoint of its segment. Now all pursuers 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 23 *The minimum distance problem can be solved in polynomial time for 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 24 *The minimum distance problem can be solved in polynomial time for 2-vertex 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 pursuer resides at each of the two vertices, and a third pursuer traverses each edge. The minimum distance is $\Sigma_{e \in E} L(e)$. □

Theorem 25 *The minimum distance problem can be solved in polynomial time for cliques.*

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

- If v is a leaf then $S(v) = \{1\}$. Also $L(v, \mathit{parent}(v)) = 1$ 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 $L(v, \mathit{parent}(v)) = y$.
  - 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 $L(v, \mathit{parent}(v)) = z$.
  - 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 $L(v, \mathit{parent}(v)) = z$.

Figure 4: Algorithm CLEARTHETREE

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 $r-1$ pursuers start at some vertex z and traverse the edges to the other $r-1$ vertices. If r is even, the r^{th} pursuer 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} pursuer now traverses an Eulerian circuit of $G - z - M$, and then it is easy for the remaining $r-1$ pursuers to clear the edges of M. In any case, the minimum distance is $\Sigma_{e \in E} L(e)$. □

Minimum Pursuers on Trees

Recall that the minimum pursuer problem can be solved in linear-time on trees [Megiddo *et al.*, 1988]. Let T_r denote the smallest tree that requires r pursuers to clear. So T_1 has only one edge, and T_2 is a star with three edges. The following results were obtained in [Megiddo *et al.*, 1988]:

- For $r \geq 2$, the smallest tree T_r that requires r pursuers 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 pursuers, then T can be cleared by r pursuers in such a way that, at any instant, all pursuers lie along a common path.
- A tree T can be cleared by r pursuers 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 $r-1$ pursuers.

We now present a new linear-time algorithm for the minimum pursuer problem on trees T, called CLEARTHETREE, that is much shorter and algorithmically simpler than the existing algorithm because it uses a postorder traversal, see Figure 4. We label each vertex v with a subset $S(v) \subseteq \{1, 2, \ldots, r\}$ where r is the minimum number of pursuers. The intuition behind CLEARTHETREE is as follows: First, when pursuer number $\min(S(v))$ is at vertex v, the pursuers numbered in the range $\{\min(S(v)), \ldots, \max(S(v))\}$ will be located within the subtree rooted at v. Also, any pursuers numbered above $\max(S(v))$ will be located at ancestors of v. Second, a pursuer that clears exactly one child of v should continue up the path to also clear v. Third, a pursuer 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 pursuer, and among such pursuers that are available, we choose a pursuer with the lowest number. Finally, a pursuer 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 pursuer to clear the edge leading upward from v, unless v is the root of the tree. However, in certain circumstances given in Figure 4, 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 pursuer clears three or more children of v).

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

After CLEARTHETREE runs, T can be cleared using $r = \max(S(\mathit{root}(T)))$ pursuers. The edge labels L denote which pursuer should clear each edge. (Note that sometimes a pursuer is listed in a vertex label S but not in any incident edge labels.)

Theorem 26 *CLEARTHETREE is optimal.*

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

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

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

CLEARTHETREE runs in $O(n \cdot r)$ time, where the number of pursuers 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. A more

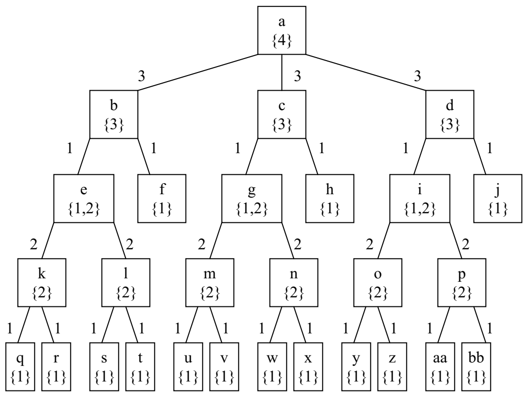

Figure 5: First Example for CLEARTHETREE

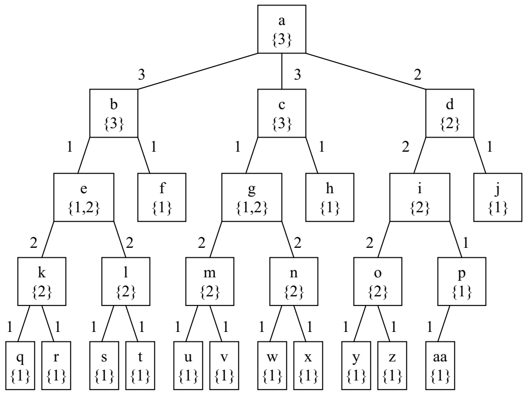

Figure 6: Second Example for CLEARTHETREE

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

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

4 Conclusions

We studied pursuit-evasion problems where a number of pursuers have to clear a given graph. Overall, it appears that pursuit-evasion problems on unit-width arbitrary-length graphs tend to be easier than pursuit-evasion problems on unit-length arbitrary-width graphs, where the minimum pursuer, distance and time problems are already NP-complete for stars, trees, two-vertex graphs, series-parallel graphs and cliques. Future work includes generalizing our results to treewidth-k graphs for arbitrary k, extending our results to different formulations of pursuit-evasion problems [Kirousis and Papadimitriou, 1986; Bienstock and Seymour, 1991], and resolving the three open slots in Figure 1. To show that any of these three open problems are in P, we would need an algorithm more general than CLEARTHETREE which solves the minimum pursuer problem on trees.

References

[Agmon *et al.*, 2008a] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. *Proceedings of IEEE International Conference on Robotics and Automation*, 2008.

[Agmon *et al.*, 2008b] N. Agmon, V. Sadov, S. Kraus, and G. A. Kaminka. The impact of adversarial knowledge on adversarial planning in perimeter patrol. *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, 2008.

[Arnborg *et al.*, 1991] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

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

[Borie *et al.*, 1992] R.B. Borie, R.G. Parker, and C.A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(1–6):555–581, 1992.

[Courcelle and Mosbah, 1993] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1–2):49–82, 1993.

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

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, New York, 1979.

[Kirousis and Papadimitriou, 1985] L.M. Kirousis and C.H. Papadimitriou. Interval graphs and searching. *Discrete Mathematics*, 55(2):181–184, 1985.

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

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

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

[Parsons, 1976] T.D. 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.

[Pita *et al.*, 2008] 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. *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, 2008.

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