ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1370

Hierarchical Strategy Synthesis for Pursuit-Evasion Problems

Rattanachai Ramaithitima[1] and **Siddharth Srivastava**[2] and **Subhrajit Bhattacharya**[1] and **Alberto Speranzon**[2] and **Vijay Kumar**[1]

Abstract. We present a novel approach for solving pursuit-evasion problems where multiple pursuers with limited sensing capabilities are used to detect all possible mobile evaders in a given environment. We make no assumptions about the number, the speed, or the maneuverability of evaders. Our algorithm takes as input a map of the environment and sensor models for the pursuers. We then obtain a graph representation of an environment using a Čech Complex. Even with such a representation, the configuration space grows exponentially with the number of pursuers. In order to address this challenge, we propose an abstraction framework to partition the configuration space into sets of topologically similar configurations that preserve the space of possible evader locations. We validate our approach on several simulated environments with varying topologies and numbers of pursuers.

1 Introduction

With the advent of technology, robotics has come to play a significant role in many applications, including autonomous search and pursuit-evasion. In this work, we address the problem of pursuit-evasion where a team of coordinated pursuers needs to search a given environment for an unknown number of evaders or targets. Pursuit-evasion formulations can be used to represent many useful applications such as surveillance or search and rescue. For instance, consider the problem of deploying a team of mobile sensing robots to patrol a military base in order to detect any intruders breaking into the base, or to search for survivors after a disaster. In such scenarios, pursuers must take into account the fact that evaders are mobile and may avoid being detected; they may also know the location of all pursuers at all times and may move faster than the pursuers. As a result, simply checking all areas is not sufficient and one needs to generate sophisticated pursuit strategies.

The pursuit-evasion problem has been studied extensively from many perspectives including differential game theory [8], graph-based search [14, 10, 7] , visibility-based search[12, 5, 11, 4], probabilistic search [2, 3, 6, 9, 13, 15], and sensor placement[1]. Isaacs [8] determined sufficient and necessary conditions for a pursuer to capture an evader in the scenario where a pursuer and an evader alternatively take turn moving in finite space.

Parsons [14] pioneered the graph theory aspect of pursuit-evasion problem in 1976 by solving the problem of searching for a lost man in the known structure cave, which he represents as searching on a discrete graph. In this framework, evaders reside on edges and can be adversarial. To detect the evaders, the pursuer must move along the edge occupied by the evaders and touch the evader. Initially all edges are contaminated, and become cleared if they do not contain any evaders. The edges can also be recontaminated if an evader moves back to the cleared edge without being detected. The pursuers' goal is to find a trajectory that clears all edges.

Later in 2007, Kolling and Carpin [10] proposed an algorithm to solve a similar problem called GRAPH-CLEAR, where the goal is to compute an optimal solution in clearing the graph under special circumstances. Nevertheless, this form of edge-search, where evaders reside on the edges, is not directly applicable to most robotics applications, especially in unstructured environments where construction of the graph is non-trivial. In more recent work, Hollinger et al. [7] discussed an algorithm called GSST for adversarial search where evaders reside on the node. GSST still suffers from reliance on graph construction in unstructured environments and the solution is limited to tree structures, which may extend to non-tree structures by placing some stationary pursuers to break up the cycles.

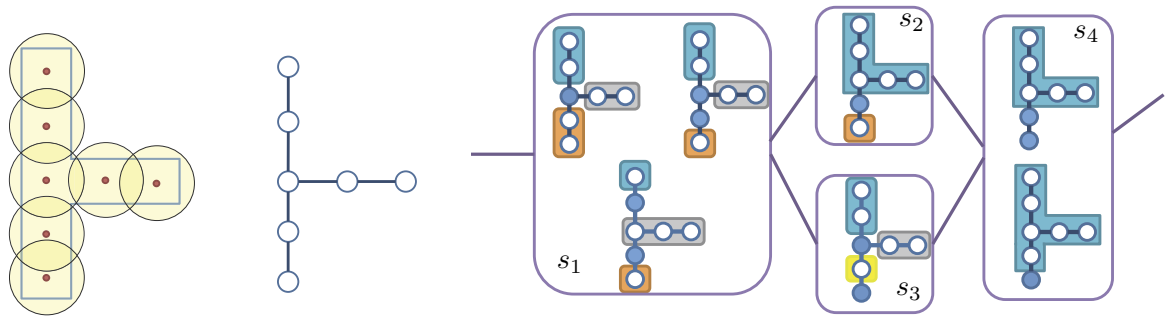
On the other hand, LaValle et al. [12, 11], Guibas et al. [5] and Gerkey et al. [4] formulated the pursuit-evasion problem as searching in a continuous space where each pursuers has infinite line of sight and the goal of the pursuers is to see all possible evaders. Their method partitions free space based on the critical points, which are vertices of the polygonal obstacles, and then performs a graph-based searching technique. Nevertheless, their approach only works with very few pursuers and the critical point technique is limited to two dimensional environment.

Probabilistic formulations of pursuit-evasion relax the worst-case scenario with the models of evaders or some uncertainty. Hespanha et al. [6] pioneered the probabilistic framework by using a greedy policy to control swarms of robots. Ong et al. [13] proposed an approximate sampling-based algorithm to solve pursuit-evasion as a POMDP problem, while Isler [9] et al. proposed a randomized strategy that allows one to two pursuers to capture any evader in simply connected polygonal environment. On the other hand, Bourgault et al. [2, 3] applied Bayesian filtering approaches to model the motion of a non-adversarial target which were then extended to multiple targets by Wong et al [15].

In sensor networks, Adams and Carlsson [1] use tools from algebraic topology to determine sufficient and necessary conditions to identify the existence of an evasion path given the positions of each mobile sensor in the space-time dimension of two dimensional environments.

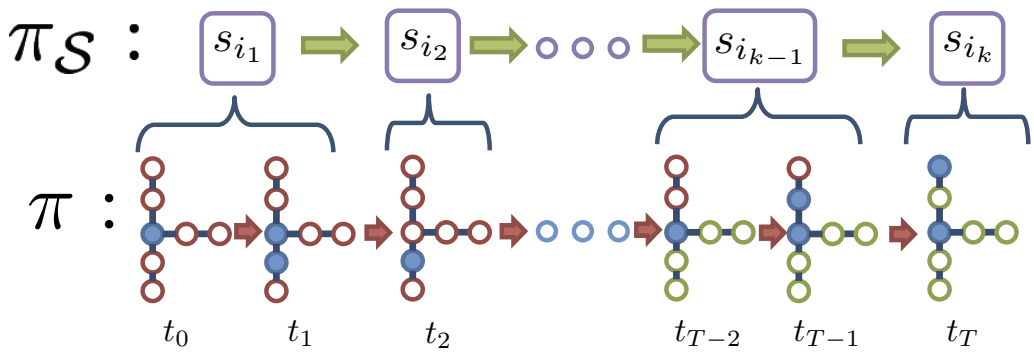
In this paper, we propose an alternative approach for solving the worst-case adversarial pursuit-evasion problems where multiple pursuers equipped with limited-range sensors are used to detect and capture all possible mobile evaders in a given environment. Our main ob-

[1] University of Pennsylvania, USA, email: ramar@seas.upenn.edu
[2] United Technologies Research Center.



(a) A graph representation is constructed for the given environment and the pursuers sensor model.

(b) The entire configuration space of $N = 2$ pursuers is partitioned into a set of abstract states (purple boxes), where a subset of them are being shown, $\{s_1, s_2, s_3, s_4\}$. This abstraction is described in section 3. For each concrete member state, the pursuers' positions are depicted by blue shaded vertices. An edge occurs between two abstract states s_1 and s_2 when it is possible to go in one move of a pursuer, from a concrete member state of s_1 to a concrete member state of s_2.



(c) The hierchical planner synthesizes a strategy as a sequence of abstract actions (π_S) and then refines π_S into a sequence of actions (π) that can be executed by the pursuers for $N = 2$. In π, the positions of the pursuers are depicted by blue shaded vertices, while the evader may reside in the unshaded vertices with red boundary.

Figure 1: **Overview:** Illustration of the main steps for synthesize the solution strategy for pursuit-evasion problem with our framework.

jective is to design a general algorithm for automatically computing the strategy that pursuers should follow for detecting all evaders.

The essence of our approach is illustrated in Figure 1. Our algorithm takes as input a map representing the environment, a sensor model of the pursuers, and the number of pursuers. Using the sensor model of the pursuers, we first construct a graph representation of the environment as shown in Figure 1(a). Next, we formulate the configuration space of the pursuers using the graph representation and the number of pursuers N and partition it into the set of abstract states (Figure 1(b)). Finally, we synthesize the strategy as a sequence of abstract actions and then perform the refinement step to map the abstract strategy into the solution strategy in the configuration space of the pursuers as illustrated in Figure 1(c).

We begin with the presentation of the preliminaries including problem description and formulation of pursuit-evasion as a partially observable planning problem in section 2. In section 3, we introduce the abstraction framework. Section 4 presents strategy synthesis over the abstraction framework and our algorithm for retrieving the solution strategy in the original graph representation. We validate our approach with simulations and analysis in section 5.

2 Preliminaries

2.1 Problem Description

We consider the problem of pursuit-evasion (PE) for worst-case adversarial targets with N pursuers, where the number of evaders is unknown and the evader is capable of moving arbitrary fast.

A map is defined as a free space, $\mathcal{F}$, in an n-dimensional Euclidean space, where n is typically 2 or 3. The position of the ith pursuer is specified by $p_i \in \mathcal{F}$, which can be applied to the sensor model O to get the sensor footprint, a set of points in $\mathcal{F}$ that can be observed from position $p_i, O(p_i) \subseteq \mathcal{F}$. An *evader space* $\mathcal{E}$ is defined as a set of points not being observed by any pursuers,

$$\mathcal{E} = \mathcal{F} \setminus \bigcup_i O(p_i).$$

Each point in the map can be *clear* or *contaminated*. The point is contaminated if an evader could be present in it, otherwise it is clear. The map is said to be clear when all points in $\mathcal{F}$ are clear. A point can be made clear by being observed by any pursuer. However, a clear point $q \in \mathcal{E}$ can become contaminated again if there exists a *path* in the evader space from another contaminated point $p \in \mathcal{E}$ to q, where the path from p to q is defined as a continuous function $\tau_{pq} : [0, T] \to \mathcal{E}$ such that $\tau_{pq}(0) = p, \tau_{pq}(T) = q$.

The process of clearing and contaminating the map as the pursuers move around is illustrated in Figure 2. Initially, evader can be at any unobserved points, so they are all contaminated. The pursuer then moves forward and clears the points along the path. However, as the pursuer moves further and clear points are exposed to the contaminated ones, they become contaminated again. The objective of PE is then to compute trajectories for each pursuer for clearing all regions in the evader space, where a trajectory of ith pursuer is defined as a continuous function of time, $p_i(t)$ for $t \in [0, T]$.

Definition 1. ***(Strategy on map)*** *Let $\mathcal{F}$ be a free space representing*

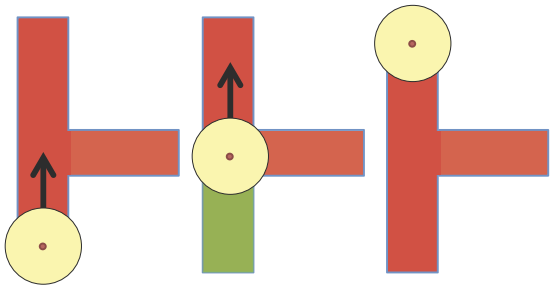


Figure 2: Illustration of the contaminated regions, shaded in red, and cleared regions, shaded in green, as the pursuer moves around in the free space. Initially, evader can be in any regions, so they are all contaminated. The pursuer then moves forward and clear the regions along the path. However, as the pursuer moves further and the cleared regions are connected to the contaminated ones, they become contaminated again.

a map. A strategy (π) *is a collection of trajectories for all pursuers,* $\pi_{\mathcal{F}} : [0, T] \to \mathcal{F}^N$, *i.e.* $\pi_{\mathcal{F}}(t) = [p_1(t), p_2(t), ..., p_N(t)]$.

Definition 2. ***(PE problem on a map)*** *Given the map* $\mathcal{F}$ *with* N *pursuers with sensor model* O*, determine a strategy* π *that clears the map* $\mathcal{F}$.

Synthesizing a solution in continuous space can quickly become intractable, especially when multiple pursuers are required. As a result, we choose to reduce PE problem on a map to *PE problem on a graph* using Čech complex construction. We will first describe how to construct a graph and then formally introduce PE problem on a graph.

One of the main step in graph construction is choosing a set of representative points such that every point in $\mathcal{F}$ can be observed from at least one of the samples. Ideally, we also want to minimize the size of the representative set. However, this is essentially a minimum set cover problem, one of the well-known NP-complete problems and hence we use a sampling-based method. First, we uniformly distribute the points to cover the convex hull of $\mathcal{F}$ based on the sensor model O. We then keep the sampling points that lie within $\mathcal{F}$ and set aside the rest. Next, we iterate through the points in $\mathcal{F}$ that are not within the sensor footprints of any chosen positions and choose the point in $\mathcal{F}$ closest to the nearest samples from the discarded points.

Assuming that O is convex and the pursuer can move holonomically, we then construct the Čech complex over the sampling points. For Čech complex, a 0-simplex exists for each sampling point; a 1-simplex exists between two 0-simplices whose their corresponding points have a non-empty intersected sensor footprints; and a 2-simplex exists for every 3-tuple of points whose sensor footprints have a non-empty intersection. To assert that we attain the hole-less coverage of the free space, we want the 2-simplices to cover all the points that are *sufficiently* far away from the obstacle. The points are sufficiently far away if it cannot be observed from the closest boundary of the obstacles.

Definition 3. ***(Graph representation)*** $G = (V, E)$*, where* V *is the set of 0-simplices and* E *is the set of 1-simplices.*

Similar to the map, each vertex on G can be either clear or contaminated. The vertex v is clear when pursuer visits. However, v can be recontaminated at any time step if there exists a sequence of unobserved vertices to another point u, i.e. $(w_1, ..., w_k)$, where $w_1 = v, w_k = u, (w_i, w_{i+1}) \in E$ and all w_i's are unobserved. G is clear when all vertices are clear.

The trajectory on a graph is then defined as a sequence of vertices, $(v^0, v^1, ..., v^T), v^i \in V$ such that $(v^i, v^{i+1}) \in E$. For simplicity, we will discretize the movement of pursuer into time step of 1. In addition, we assume that multiple pursuers can occupied same vertex.

Definition 4. ***(Strategy on a graph)*** *Let* $G = (V, E)$ *be a graph. A strategy on graph* $(\pi_G$ *or* $\pi)$ *is a collection of trajectories on graph,* $\pi : \{0, 1, ..., T\} \to V^N$, *i.e.* $\pi(t) = [v_1^t, v_2^t, ..., v_N^t]$.

Definition 5. ***(PE on a graph)*** *Let* $G = (V, E)$ *be a graph. Determine a strategy* π *that clears* G.

Furthermore, the strategies computed on the graph can be translated back into executable trajectories on the map. For any $(u, v) \in E$, a path between u, v is defined as a continuous function $\tau_{uv} : [0, 1] \to \mathcal{F}$ such that $\tau(0) = u$ and $\tau(1) = v$. Additionally, to prevent v from immediately contaminate u during execution, τ_{uv} must satisfy the following property:

$$\bigcap_{t\in[0,1]} O(\tau_{uv}(t)) = O(u) \cap O(v),$$

which ensures that the clearing process on the discrete graph would entail the clearing process on the continuous map.

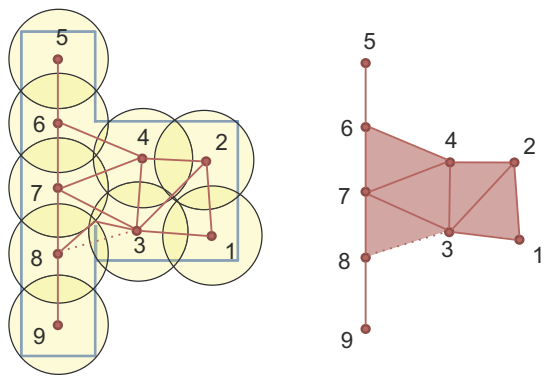


Figure 3: Example of graph representation in 2D environment with holomomic pursuer equipped with circular sensor footprint (left) and its Čech Complex (right). The path between vertices are denoted by solid lines, which are either a straight line or a pair of lines through an intermediate point in the presence of obstacles.

In this paper, we will focus on the holonomic pursuer with sensor model of a ball with radius r. We demonstrate the construction of graph representation using Čech complex on a map with circular sensor model in Figure 3. The path between any vertices will either be a straight line or a pair of straight lines through the point inside the intersection of their sensor footprints due to the presence of obstacles. In both cases, these paths satisfy the property for τ that prevents the immediate contamination during execution. For instance, $\tau_{3,7}$, a straight line from vertex 3 to 7, and $\tau_{3,8}$, a pair of straight lines from vertex 3 to 8, always cover their corresponding intersection as shown in Figure 4.

2.2 Solving PE as a Partially Observable Planning Problem

Since the positions of the evaders are unknown, we cannot fully observe the state during planning. Hence, we introduce the notion of *belief state*. The belief state at any time step, denoted by $x(t)$, consists of the configuration/position of all pursuers, denoted by $\mathbf{p}$, and

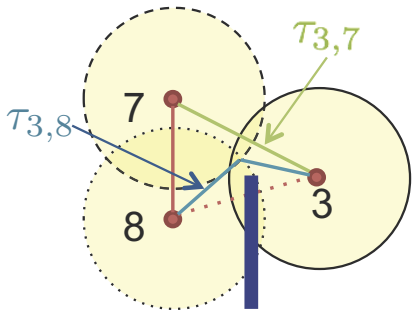


Figure 4: A valid path exist for any edges in G. For instance, any points along $\tau_{3,7}$ and $\tau_{3,8}$ remain observing $O(3) \cap O(7)$ and $O(3) \cap O(8)$ respectively.

the possible positions of the evaders, which will be referred as the *contaminated regions* denoted by $\mathbf{c}$. We omit the argument in $x(t)$ when it is clear from the context. The collection of all belief states is referred as the *belief space,* X , while the *configuration space,* P, is spanned by the position of the pursuers. The span of contaminated regions will be referred as the *contamination space,* C, so that

$$x = (\mathbf{p}, \mathbf{c})$$

On the graph representation G, the configuration space is spanned by the pursuer positions, $\mathbf{p} = \{p^1, ..., p^N\}$, where $p^i \in V$. On the other hand, the contamination space can be defined as a set of vertices that the evaders could be present in. Hence, it is a subset of a power of set of V, $\mathbf{c} \in C \subseteq \mathbb{P}(V)$.

The update step occurs when the pursuers take action, i.e. move along an edge in G. With the time discretization on graph, the action can simply be written as the next configuration of the pursuers, $\mathbf{p}'$, and hence the update function can be defined as

$$Update(x_t, \mathbf{p}') :$$
$$x_{t+1} = (\mathbf{p}', UpdateContaminate(\mathbf{c}_t, \mathbf{p}'))$$

$UpdateContaminate$ updates the contaminated vertices based on the current contamination status and next configuration of the pursuers by computing a set of reachable vertices on $G' = (V \setminus \mathbf{p}', E \setminus \mathbf{p}')$ beginning at $\mathbf{c}_t \setminus \mathbf{p}'$. In addition to all edges that contain occupied vertices, the edge subtraction may require removing some additional edges. The additional edge removal will be explained in section 3.1.

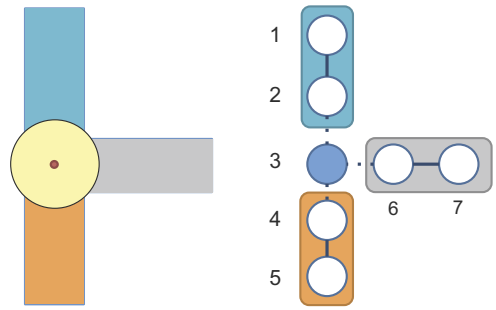
The solution strategy is then a sequence of actions in the belief space such that the contaminated regions become an empty set, i.e. $\pi = \{(\mathbf{p}_0, \mathbf{c}_0), (\mathbf{p}_1, \mathbf{c}_1), ..., (\mathbf{p}_T, \emptyset)\}$, where $(\mathbf{p}_0, \mathbf{c}_0)$ is the initial state. Solving this as a partially observable planning problem requires search in an intractably large space of belief states, which is exponential in the number of joint pursuer-evader configurations. To address this challenge, we will use a novel abstraction technique that is described in the next section.

3 Abstraction Framework

3.1 Abstract State Space

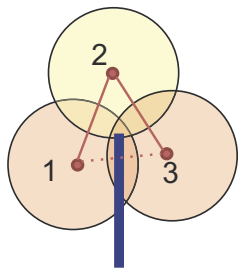
To cope with the exponential growth of the belief space, we propose a novel method to partition the configuration space into *abstract state space*, denoted by $\mathcal{S}$, by utilizing the topological invariants of the evader space.

Although the contamination space might appear to be exponential in $|V|$, not all combinations of the contaminated regions are reachable. Since the evader can move arbitrary fast, any adjacent regions in the evader space will both be either contaminated or cleared.

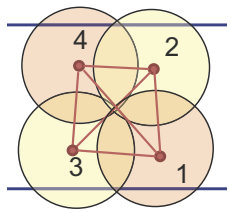


(a) (b)

Figure 5: Illustration of an connected component function on G with pursuers at $\mathbf{p} = \langle 3, 3 \rangle$. (a) The sensor footprint is projected onto $\mathcal{F}$ which then separates the evader space into multiple connected components (b) The graph is reconstructed on the evader space where the position of pursuers are depicted by blue-shaded vertices, while the evader space are grouped by shaded boxes for each connected component. $CC(\langle 3, 3 \rangle, G) = \{\{1, 2\}, \{4, 5\}, \{6, 7\}\}$



(a) With pursuer at $\langle 2 \rangle$, 1 and 3 remain connected with removal of 2 in G but actually get disconnected in $\mathcal{F}$.



(b) Similarly, with pursuers at $\langle 2, 3 \rangle$, 1 and 4 remain connected with removal of 2 and 3 in G but actually get disconnected in $\mathcal{F}$.

Figure 6: Additional edges removal is required when computing connected component on G on these cases.

Utilizing this fact, we define the *connected component (CC)* function which returns the sets of adjacent vertices in the evader space of G based on the assignment of pursuers, $\mathbf{p}$, denoted by $CC(\mathbf{p}, G)$ or simply $CC(\mathbf{p})$ when G is obvious. The connected component function can be computed by projecting the sensor footprints of the pursuers onto the free space, and then reconstructing the graph representation of the evader space, $\mathcal{E}$, as illustrated in Figure 5.

The connected component function can also be computed by subtracting the vertices (and their associated edges) occupied by the pursuers from G. Nevertheless, in the present of obstacles, the evader space might remain connected in G while becoming disconnected in $\mathcal{F}$ as illustrated in Figure 6. These exceptions lead to additional edges being removed from G. For 2D environment, there are only two possible scenarios. The first scenario occurs when the intersection of two sensor footprints is completely contained inside the sensor footprint of another vertex (Figure 6(a)). The other scenario occurs when there is a 4-way intersection of sensor footprints, resulting in edge intersection in G (Figure 6(b)).

During graph construction, we can keep track of the intersection between sensor footprints to handle the first scenario, while edge intersection can be easily computed. Hence, the CC function can be computed on G for 2D environment. For higher dimension, the CC function can be computed on G only if all exceptions are tractable. Otherwise, completeness is not guaranteed.

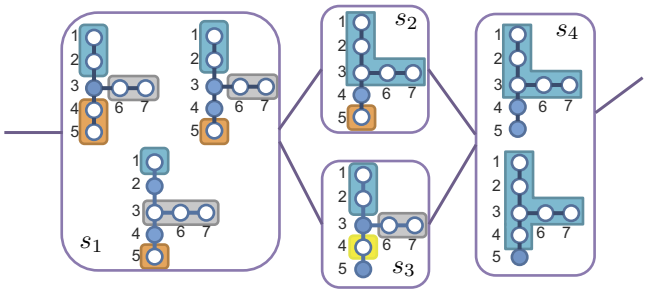


Figure 7: Simple configuration space of two pursuers is partitioned into abstract state space using equivalence relation.

Using the results of the CC function, we want to partition the configuration space into *abstract states* in a way that preserves the topology of the evader space, which is equivalent to the contamination status of each connected component remains unchanged. Using abstract state $\mathcal{S}_1$ in Figure 7 as an example, $\langle 3,3\rangle \sim \langle 3,4\rangle$ and there exists a one-to-one mapping between $CC(\langle 3,3\rangle)$ and $CC(\langle 3,4\rangle)$ which preserves the contamination status of each connected component. On the other hand, the edge between two abstract states denotes the transition that does not preserve the topology of an evader space, which could then lead to the changes in contamination status of the evader space. For instance, the edge between $\mathcal{S}_1$ and $\mathcal{S}_2$ represents the transition between $\langle 3,4\rangle$ and $\langle 4,4\rangle$ which is resulted in two connected components of $CC(\langle 3,4\rangle)$ merging and could potentially changes their contamination statuses. We first introduce a relation between two adjacent configurations and then formally define an equivalence relation for partitioning the configuration space as follow.

Let $\mathbf{p} \rightarrow \mathbf{q}$ denotes two adjacent configurations $\mathbf{p},\mathbf{q} \in P$ s.t. $(p^k, q^k) \in E, \forall k \in \{1,...,N\}$. Given two adjacent configurations we can define a relation, which we call *transition relation*, as follows.

Definition 6. ***(Transition Relation)*** *Let $\boldsymbol{p},\boldsymbol{q} \in P$ s.t. $\boldsymbol{p} \rightarrow \boldsymbol{q}$. The transition relation $\rho_{\boldsymbol{p},\boldsymbol{q}}$ between connected components of $\boldsymbol{p}$ and $\boldsymbol{q}$ is defined as*

$$(cc_i^{\boldsymbol{p}}, cc_j^{\boldsymbol{q}}) \in \rho_{\boldsymbol{p},\boldsymbol{q}} \quad \Leftrightarrow \quad cc_i^{\boldsymbol{p}} \cap cc_j^{\boldsymbol{q}} \neq \emptyset\,,$$

where $cc_i^{\boldsymbol{p}} \in CC(\boldsymbol{p}), cc_j^{\boldsymbol{q}} \in CC(\boldsymbol{q})$.

Given the previous definition we can now formally introduce an equivalence relation between states, which we will use to define a state abstraction.

Definition 7. ***(Equivalence Relation)*** *For all $\boldsymbol{r},\boldsymbol{s} \in P$ we say that $\boldsymbol{r}$ is equivalent to $\boldsymbol{s}$, or $\boldsymbol{r} \sim \boldsymbol{s}$, if and only if there exists a finite sequence $\{\boldsymbol{z}^i\}_0^T \in P^{T+1}$ with ρ such that*

1. *$\boldsymbol{z}^0 = \boldsymbol{r}$, and $\boldsymbol{z}^T = \boldsymbol{s}$;*
2. *$\boldsymbol{z}^i \rightarrow \boldsymbol{z}^{i+1}$, with $i \in \{0,...,T-1\}$;*
3. *$\rho_{\boldsymbol{z}^i,\boldsymbol{z}^{i+1}}$ is a bijection.*

Hence, the abstract state space can be defined as $\mathcal{S} = P/\!\sim$, where each abstract state $s_i \in \mathcal{S}$ is a collection of equivalent configurations.

As a result, the contamination status of each connected component could only be changed upon transition between abstract states. Hence, we can synthesize a solution strategy on the abstract state space instead of synthesizing the strategy directly in the configuration space.

In next section, we will describe the algorithm to incrementally construct the abstract state space $\mathcal{S}$, the function mapping P to $\mathcal{S}$, denoted by (γ), and the adjacency matrix of abstract states, denoted by $\mathcal{M}$.

3.2 Partition Algorithm

Algorithm 1 Partition algorithm

```
1:  S ← ∅, M ← ∅
2:  Q.Insert(p_I) for some arbitrary p_I
3:  while Q ≠ ∅ do
4:      p ← Q.GetFirst(),    mark p as visited
5:      γ(p) ← null,    Adjacent_S ← ∅
6:      for p' ∈ Adjacent(p) do
7:          if p' is visited then
8:              if CC(p) ∼ CC(p') then
9:                  if γ(p) is null then
10:                     γ(p) ← γ(p')
11:                 else
12:                     Resolve conflict if needed
13:                 end if
14:             else
15:                 Adjacent_S.Insert(γ(p'))
16:             end if
17:         else if p' is unvisited then
18:             Q.Insert(p'),    mark p' as alive
19:         end if
20:     end for
21:     if γ(p) is null then
22:         S.Insert(Abstract(p)),    γ(p) ← Abstract(p)
23:     end if
24:     for a ∈ Adjacent_S do
25:         Update M(γ(p), a)
26:     end for
27: end while
```

Algorithm 1 outlines an incrementally construction of the abstract state space and other components required for synthesizing a strategy. The concept is to perform a forward search over the configuration space P beginning at an arbitrary state $\mathbf{p}_I$ and partition the configuration states into the abstract state, $a \in \mathcal{S}$, based on the output of $CC(\mathbf{p})$.

Following standard forward search algorithm (line 2-6, 17-18), each state in P begins as *unvisited* and will be marked *alive* or *visited* upon inserting to or removing from Q respectively. The set of alive states is stored in the list Q and the search is completed when the list Q is empty. The function $Adjacent(\cdot)$ in line 6 returns the set of adjacent states by moving the pursuers along graph G. In this step, we will restrict the adjacent states to one pursuer movement only.

The partitioning occurs between line 7-15 and 21-25, where we compare the connected components of the current state to the visited adjacent states and either assign the current state to the new abstract state or append it to the existing one.

In general, comparing the connected component between two arbitrary configurations is nontrivial. Comparing those of the adjacent configurations is much simpler. In line 8, we compute the transition relation, $\rho_{\mathbf{p},\mathbf{p}'}$, as defined in Definition 6 and check whether it is bijective. This transition relation is also used for updating the contaminated regions when transiting between abstract states.

If the graph consists of *cycles*, a conflict might occur when two similar configurations get assigned into two different abstract states. This will be resolved in line 12 where two abstract states will be combined.

Furthermore, the adjacency matrix, $\mathcal{M}$, is updated based on the connectivity of the corresponding configuration states. In line 15,

$Adjacent_A$ keeps track of the adjacent abstract states which will then update $\mathcal{M}$ in line 25. The adjacency matrix also store the transition relation(s) between two abstract states and the corresponding configurations. The transition relation might not be unique if G consists of cycles.

As a result, the computational complexity of the partition algorithm is approximately $O(dN|V|^{N+1})$, which consists of $O(dN|V|^N)$ from the forward search algorithm over the configuration space of size $|V|^N$ where each configuration has $O(dN)$ adjacent states, d denotes the average degrees of the vertices, and $O(|V|)$ from comparison of evader space in line 8. Although the number might seem large, it is much smaller comparing to searching over original belief space because the partition algorithm is only exponential with respect to the number of pursuers, which is commonly known as *curse of dimensionality* in multi-robot motion planning problem. In the next section, we will explain how to use the output of the partition algorithm to synthesize the solution strategy.

4 Hierarchical algorithm

To synthesize the strategy using the abstraction framework, we first search for the strategy in the abstract state space and then refine the strategy into the configuration space. If the number of pursuers, N is given, planning in abstraction framework would either return the strategy or indicate that no solution exists for the given N. The search for strategy in the abstract state space can be done using existing techniques for graph-based searching such as Dijkstra's algorithm. We will describe the abstract belief space for planning in the abstract state space in section 4.1, and then discuss the refinement step in section 4.2.

4.1 Planning in the abstract state space

The abstract belief state for the abstract state space, denoted by $x_{\mathcal{S}}$, becomes a pair of the abstract state (s) and the list of contaminated regions (L), where each region represents a set of adjacent vertices of the evader space.

$$x_{\mathcal{S}} = (s, L), \quad L = \{s^j\}.$$

The update step during planning will keep track of the contaminated regions using the information stored in the adjacency matrix $\mathcal{M}$. Since the transition relation ρ may not be unique, the update step with input s_k has to be called for each ρ stored in $\mathcal{M}(s_t, s_k)$.

$$Update(x_{\mathcal{S},t}, s_k, \rho):$$
$$L_{t+1} = \{s_k^j \mid \exists s_t^i \in L_t, (s_t^i, s_k^j) \in \rho\}$$
$$x_{\mathcal{S},t+1} = (s_k, L_{t+1})$$

The solution strategy is a sequence of abstract belief states such that the list of contaminated regions eventually becomes empty, denoted by $\pi_{\mathcal{S}} = (a_{i_1}, \{a_{i_1}^j\}), (a_{i_2}, \{a_{i_2}^j\}), ..., (a_{i_k}, \emptyset)$. We will then describe how to map the solution strategy into the strategy on a graph with the refinement process on the following section.

4.2 Refinement

The information stored in $\mathcal{M}$ provides the boundary configurations representing the transition between abstract states. Given the sequence of abstract states, the entering configuration might not be adjacent to the leaving one. For instance, in Figure 8, the incoming and outgoing configuration of abstract state s_j are not adjacent. Thus, the refinement step is required to find the trajectory from the incoming to outgoing configuration such that all intermediate configurations are the member of s_j.

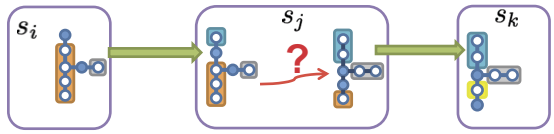


Figure 8: The transition between abstract states represents the action between boundary configurations, however, the incoming configuration need not be adjacent to the outgoing one. Hence, the refinement step is necessary to find the trajectories between them.

Using mapping function γ, one method for refinement is to perform the forward search inside each abstract state to find the trajectory from the incoming to outgoing configurations. However, this method might be inefficient since we already expand the full configuration space while executing partition algorithm.

An alternative method is to store information of the spanning trees of each abstract state during the partition algorithm and then search for the trajectory on the spanning trees during refinement. The downside of this method is that the result is usually suboptimal compared with one from forward search method.

Given the strategy $\pi_{\mathcal{A}} = \{s_0, s_1, s_2, ..., s_k\}$ where $\gamma(\mathbf{p}_I) = s_0$, the refinement then first searches for a trajectory from $\mathbf{p}_I$ to the boundary configuration connecting to s_1 while remaining within s_0. Then, we continue refine the solution inside s_1 from the incoming configuration from s_0 to the outgoing configuration connecting to s_2 while remaining within s_1. The same process continues until we reach the final abstract state s_k.

4.3 Minimizing the number of pursuers

The full algorithm for synthesizing the solution strategy is given in algorithm 2 where the number of pursuers, N, required is unknown. Note that incrementing N by one at each iteration is more efficient that doing binary search because the computational complexity is exponential with respect to N.

Algorithm 2 Hierarchical Strategy Synthesis

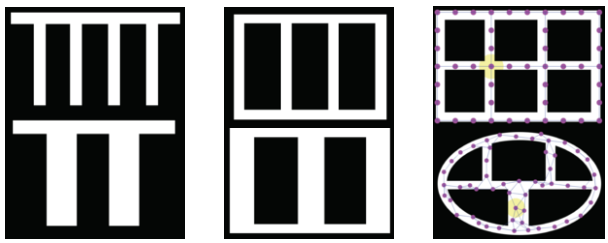
1: Construct G of free space $\mathcal{F}$ with sensor model $O(\cdot)$
2: $N \leftarrow 1, \pi_{\mathcal{S}} \leftarrow [\,]$
3: **while** $\pi_{\mathcal{S}}$ is empty **do**
4: Partition G^N into abstract state space $\mathcal{S}$
5: $\pi_{\mathcal{S}} \leftarrow Planning(\mathcal{S}, \mathbf{p}_I)$
6: Increment N
7: **end while**
8: $\pi \leftarrow Refine(\pi_{\mathcal{S}})$

5 Results

The construction of graph representation is implemented in MATLAB, whereas the remaining components are implemented in C++. We validate the proposed method in simulations with environments of varying topologies and using different number of pursuers as discussed in section 5.1. Then, we compare our results with the graph-based searching over the full belief space on simple environments in section 5.2.

5.1 Simulation Results

We evaluated the performance in simulation on environments with three different topologies as show in Figure 9. Each pursuer has a disc sensor footprint with a radius of one meter. Due to various structures of the environments, we represent their dimensions with the number of vertices in the graph representation. Figure 9(c) illustrates the graph representation of the testing environments and the sensor footprint of the pursuer. The number of pursuers required in each environment is computed by iterating from $N = 1$.



(a) (b) (c)

Figure 9: Testing Environments: (a) Tree Structures; (b) Ladder Structures; (c) Random Loops Structures with graph representation

5.1.1 Tree Structures

We evaluated on the tree structures with varying number k and width w of branches (vertical corridors). The graph representations contain $8-48$ vertices. It requires 2 pursuers to clear the map for $w = 1$ and 3 pursuers to clear the map for $w = 2$. The execution times of each component are illustrated in Figure 10.

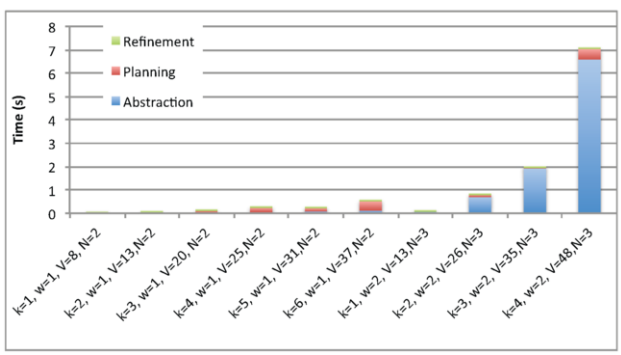


Figure 10: The execution time of the proposed method on tree structure with k branches of width w using N pursuers on graph with V vertices.

5.1.2 Ladder Structures

We evaluated on the ladder structures with varying number k and width, w, of steps (vertical corridors). The graph representations contain $30-52$ vertices. For ladder with single loop $k = 2$, it requires 2 pursuers to clear the map with $w = 1$ and 3 pursuers to clear the map with $w = 2$. If the ladder with multiple loops $k > 2$, it requires 3 pursuers to clear the map with $w = 1$ and 4 pursuers to clear the map with $w = 2$. The execution times of each component are illustrated in Figure 11.

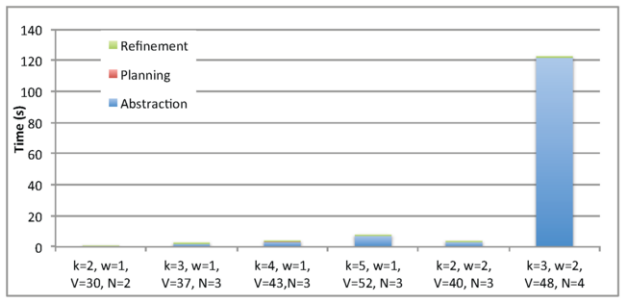
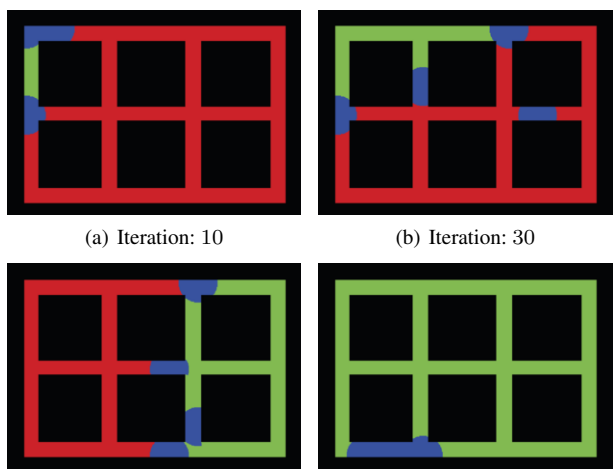


Figure 11: The execution time of proposed method on ladder structure with k steps of width w using N pursuers on graph with V vertices.

5.1.3 Random Loop Structures

We evaluated the proposed methods on two maps shown in 9(c) using 4 pursuers for the top one, which consists of 46 vertices, and 5 pursuers for the bottom map, which consists of 53 vertices. The total execution times are 105.59 and 4076.32 seconds, in which abstraction framework are accounted for 103.25 and 4074.44 seconds respectively. The intermediate steps during the clearing process are shown in Figure 12 and Figure 13.



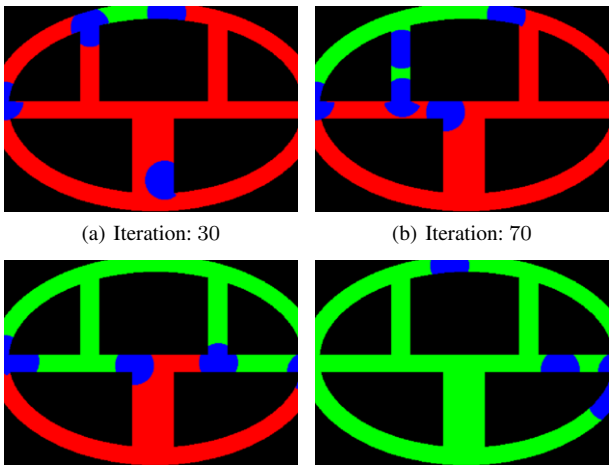
(a) Iteration: 10 (b) Iteration: 30

(c) Iteration: 100 (d) Final Iteration (147)

Figure 12: Snapshots of clearing process on 3×4 grid map with all narrow passages using 4 pursuers.

As explained in section 4.2, one the main disadvantages of refinement using spanning tree is the lengthy strategy as indicated by a high number of iteration in both examples. This strategy can be further improved; however, this is out the scope of this paper.

The simulation results show that the abstraction framework is responsible for the majority of computation time as N increases, which conforms with our analysis on computational complexity of the partition algorithm. Nevertheless, the abstraction framework only needs to be executed once for each given map with the same number of pursuers. It is invariant of the initial position and thus we can quickly synthesize the strategy again for different initial position. This can be useful if an error occurs while executing the solution strategy and the pursuers are deviated from the planned strategy.



(a) Iteration: 30 (b) Iteration: 70

(c) Iteration: 250 (d) Final iteration (309)

Figure 13: Snapshots of clearing process on curved hallway with vary passages size using 5 pursuers.

5.2 Comparison

We compare the results of our proposed algorithm with a baseline (brute force) planner which searches for a strategy on the full belief space, described in section 2.2. We used maps with tree and ladder structures. The baseline planner can only solve the maps containing up to 2 branches (13 vertices) for tree structure and the maps containing one loop (14 vertices).

Map	Our framework	Baseline planner
Tree with 1 branch, V= 8	0.014	0.04
Tree with 2 branches, V=13	0.034	49.95
Ladder with 2 steps, V=14	0.018	1082.77

Table 1: Comparison of execution time (sec) between our framework and baseline (brute force) planner.

The execution time of the baseline planner grows exponentially as V increases. Additionally, the baseline planner suffers greatly from the topological invariant of the loop structure (such as a ladder) due to a large reachable states of the contamination space. On the other hand, our framework reduces the configuration space of 2 pursuers with one loop into only two abstract states; one for two pursuers being adjacent and other when they are separated.

6 Conclusion and Future Work

In this paper, we proposed an abstraction framework to solve a worst-case adversarial pursuit-evasion problem where multiple pursuers with limited-range sensor coverage are used to detect all possible mobile evaders. This method involves constructing the graph representation of an environment using the sensor model equipped on the pursuers, partitioning the configuration space of the pursuers over graph into an abstract state space, searching for a strategy in the abstract state space, and finally refine the strategy into the configuration space. We validate our proposed method by simulating environments with different topologies and compare the result with a brute force searching over the full belief space. Since the full belief space grows exponentially with N and the number of vertices in V, the brute force search can only solve PE problems on small maps ($|V| < 20$) for $N = 2$. In contrast, our approach reduces the complexity into exponential of N with base $|V|$ and can solve the PE problems with a few hundred vertices for $N = 2$ and up to 50 vertices for $N = 5$.

Although the abstraction framework requires inspecting the full configuration space of N pursuers, this step needs to be done only once for a given map with the same number of pursuers. The output can be reused for different initial configurations. Furthermore, we observe that many maps with the similar structure yield the same abstraction. This opens up an interesting problem of how can we apply the result of one abstraction framework to the new maps with similar structure without recomputing it. Additionally, we are interested in converting an abstraction framework into an on-line algorithms so that we can concurrently synthesize for the solution strategy, which may avoid exploring the entire configuration space.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of ONR grant N00014-14-1-0510 and the United Technologies Research Center.

REFERENCES

[1] Henry Adams and Gunnar Carlsson, 'Evasion paths in mobile sensor networks', *The International Journal of Robotics Research*, **34**(1), 90–104, (2015).

[2] Frederic Bourgault, Tomonari Furukawa, and Hugh F Durrant-Whyte, 'Coordinated decentralized search for a lost target in a bayesian world', in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pp. 48–53. IEEE, (2003).

[3] Frédéric Bourgault, Tomonari Furukawa, and Hugh F Durrant-Whyte, 'Optimal search for a lost target in a bayesian world', in *Field and service robotics*, pp. 209–222. Springer, (2003).

[4] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon, 'Visibility-based pursuit-evasion with limited field of view', *The International Journal of Robotics Research*, **25**(4), 299–315, (2006).

[5] Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani, 'A visibility-based pursuit-evasion problem', *International Journal of Computational Geometry & Applications*, **9**(04n05), 471–493.

[6] Joao P Hespanha, Hyoun Jin Kim, and Shankar Sastry, 'Multiple-agent probabilistic pursuit-evasion games', in *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, volume 3, pp. 2432–2437. IEEE, (1999).

[7] Geoffrey Hollinger, Athanasios Kehagias, and Sanjiv Singh, 'Gsst: anytime guaranteed search', *Autonomous Robots*, **29**(1), 99–118, (2010).

[8] Rufus Isaacs, *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*, Courier Corporation, 1999.

[9] Volkan Isler, Sampath Kannan, and Sanjeev Khanna, 'Randomized pursuit-evasion in a polygonal environment', *Robotics, IEEE Transactions on*, **21**(5), 875–884, (2005).

[10] Andreas Kolling and Stefano Carpin, 'The graph-clear problem: definition, theoretical properties and its connections to multirobot aided surveillance', in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 1003–1008. IEEE, (2007).

[11] Steven M LaValle and John E Hinrichsen, 'Visibility-based pursuit-evasion: The case of curved environments', *Robotics and Automation, IEEE Transactions on*, **17**(2), 196–202, (2001).

[12] Steven M LaValle, David Lin, Leonidaa J Guibas, Jean-Claude Latombe, and Rajeev Motwani, 'Finding an unpredictable target in a workspace with obstacles', in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pp. 737–742. IEEE, (1997).

[13] Sylvie CW Ong, Shao Wei Png, David Hsu, and Wee Sun Lee, 'Planning under uncertainty for robotic tasks with mixed observability', *The International Journal of Robotics Research*, **29**(8), 1053–1068, (2010).

[14] Torrence D Parsons, 'Pursuit-evasion in a graph', in *Theory and applications of graphs*, 426–441, Springer, (1978).

[15] El-Mane Wong, Frédéric Bourgault, and Tomonari Furukawa, 'Multi-vehicle bayesian search for multiple lost targets', in *Proceedings of the 2005 ieee international conference on robotics and automation*, pp. 3169–3174. IEEE, (2005).