Journal of Artificial Intelligence Research 69 (2020) 297-342 Submitted 04/2020; published 09/2020

Planning High-Level Paths in Hostile, Dynamic, and Uncertain Environments

Jacopo Banfi JB2639@CORNELL.EDU
Vikram Shree VS476@CORNELL.EDU
Mark Campbell MC288@CORNELL.EDU
Cornell University,
Ithaca, NY 14853 USA

Abstract

This paper introduces and studies a graph-based variant of the path planning problem arising in hostile environments. We consider a setting where an agent (e.g. a robot) must reach a given destination while avoiding being intercepted by probabilistic entities which exist in the graph with a given probability and move according to a probabilistic motion pattern known a priori. Given a goal vertex and a deadline to reach it, the agent must compute the path to the goal that maximizes its chances of survival. We study the computational complexity of the problem, and present two algorithms for computing high quality solutions in the general case: an exact algorithm based on Mixed-Integer Nonlinear Programming, working well in instances of moderate size, and a pseudo-polynomial time heuristic algorithm allowing to solve large scale problems in reasonable time. We also consider the two limit cases where the agent can survive with probability 0 or 1, and provide specialized algorithms to detect these kinds of situations more efficiently.

1. Introduction

Motivated by the recent interest of the AI community in the navigation of hostile environments (Agmon, 2017), in this paper we study the problem of planning high-level paths in a graph-represented environment containing probabilistic static and/or dynamic threats. Specifically, we consider an agent (e.g. a robot) moving in a graph that must reach a given destination while avoiding being intercepted by some entities —generically called "obstacles"— existing with a given probability, moving according to a probabilistic motion pattern, and capable of "intercepting" the agent along its path (e.g., within a given range). Given a goal vertex to reach within a given deadline, the objective of the agent is simple: computing the path to the goal that maximizes its chances of survival.

Possible applications of this work include, for instance, those related to intrusion settings. Here, obstacles may represent enemy guards deployed to protect the entrances buildings (Portugal & Rocha, 2011), while the agent can be thought as a strong adversary endowed with a full knowledge (although probabilistic) of the patrolling entities (Agmon, Kaminka, & Kraus, 2011). Another example is given by those robotic scenarios where the robot's sensors have been compromised by an attacker, who is trying to make the robot crash by injecting false dynamic obstacles into the robot's perception pipeline: although the robot might be able to recognize these kinds of attacks by leveraging probabilistic techniques –as shown very recently by Liu et al. (2018)– and abort the mission, there are situations where reaching the goal within a given deadline could be of vital importance.

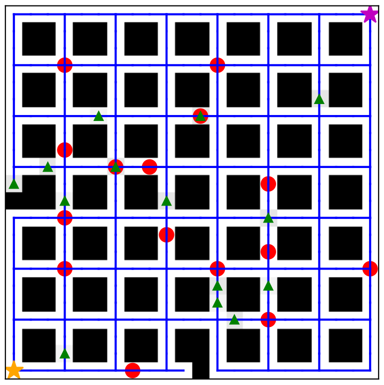


Figure 1: An example problem instance. The environment is a 22x22 grid graph containing some “classical” obstacles (the black squares). The agent must move from the bottom-left corner (orange star) to the upper-right (purple star). Static probabilistic obstacles (green triangles) and dynamic probabilistic obstacles (red circles) can intercept the agent along its path. The agent must reach the goal within a given deadline by traveling along the path that maximizes its chances of survival.

The paper is structured as follows. We start by introducing a graph-based formalization of the problem (see Figure 1 for an example instance) in Section 2, and present some computational complexity results, including some non-trivial NP-hardness proofs sheding some light on the intractability of seemingly simple instances (rectangular grid graphs, obstacles moving at at most 1 cell/step, very small interception range, etc.). We then propose two algorithms for computing high quality solutions in the general case: an exact algorithm based on Mixed-Integer Nonlinear Programming, presented in Section 3, working well in instances of moderate size, and a pseudo-polynomial time heuristic algorithm allowing to solve large scale problems in reasonable time, described in Section 4. We also consider the two limit cases where the agent can survive with probability 0 or 1 in Section 5, and provide specialized algorithms to detect these kinds of situations more efficiently. Finally, in Section 6, we show the results of an extensive validation campaign in a simulated intrusion scenario: for the general problem, we show the limits of the MINLP approach in dealing with large scale problem instances, and evaluate the optimality losses introduced by the heuristic —also comparing it against two baselines (a Monte Carlo Tree Search algorithm and a sampling-based approach); for the two limit cases, instead, we show the computational advantages offered by more specialized algorithms over slight variations of the general MINLP approach. We conclude the paper by showing how this new problem can be framed in the wide literature landscape of adversarial navigation problems, in Section 7, and by providing some interesting future research directions in Section 8.

2. Problem Setting

This section provides a formal definition of the problem considered in this work. Section 2.1 introduces the environment and the agent's motion model. Sections 2.2 and 2.3 introduce, respectively, the models of static and dynamic obstacles. Section 2.4 formalizes the problems we aim to solve, and studies their computational complexity. Finally, Section 2.5 contains some additional comments about the proposed model.

2.1 Environment and Agent's Motion Model

Let $G = (V, E)$ be a connected, undirected, simple graph with unitary edge lengths representing the environment, and let $V = \{1, \ldots, n\}$. Let also $\mathbf{d}(u, v)$ be the shortest path distance between two vertices $u, v \in V$. An agent must plan a path on G in the form of an ordered sequence of vertices $\pi = (v_s, v_1, v_2, \ldots, v_g)$ from a start vertex v_s to a goal vertex v_g. The agent moves deterministically on G. In particular, we assume that time evolves in discrete steps $t \in \mathbb{N}_0$. At each step, the agent can either stay still at the current vertex, or move along a graph edge: formally, $\forall v_i, v_{i+1} \in \pi$, either $v_i = v_{i+1}$ or $\{v_i, v_{i+1}\} \in E$. We use $\pi[i]$ to denote the position of the agent at the i-th time step when executing π. Also, with a slight abuse of notation, we use $|\pi|$ to denote the number of time steps required to reach the goal in π. Finally, we use $\mathcal{P}$ to denote the set of all the possible paths according to the above definition.

We assume that the environment may contain *probabilistic obstacles* able to *intercept* the agent along its path. These obstacles are either *static* or *dynamic*. Although the former is a particular case of the latter, we will make use of this distinction in the following sections. We further assume that the existence and the interception events related to a given obstacle in G are *independent* of those of all the others. In this work, we make the standard assumption that all the probabilities contained in the problem instance are rational numbers (see, e.g., Madani, Hanks, and Condon, 2003). In the following, the terms "probability of survival" and "survivability" will be used interchangeably.

2.2 Static Probabilistic Obstacles

The set of static probabilistic obstacles is denoted by S. Each $s \in S$ is completely described by a probability of existence p_s and by a set of vertices $V(s) \subseteq V \setminus \{v_s, v_g\}$ inducing a connected subgraph on G. We assume $p_s < 1$ since, otherwise, the problem instance can be easily reformulated by directly embedding this information into the graph G. The semantic associated with a static probabilistic obstacle s is as follows: if the agent executes a path π traversing *any* of the vertices in $V(s)$, it is *intercepted* by s with probability p_s and it *survives* s with probability $1 - p_s$. We use the function $\sigma : \mathcal{P} \times S \to \{0, 1\}$ to associate each $\langle \pi, s \rangle$ pair with the presence ($\sigma(\pi, s) = 1$) or absence ($\sigma(\pi, s) = 0$) of at least one passage of π through a vertex belonging to $V(s)$.

2.3 Dynamic Probabilistic Obstacles

The set of dynamic probabilistic obstacles is denoted by D. Each $d \in D$ is associated with a probability of existence p_d and, if it exists, moves in the graph according to a probabilistic motion model described by a Markov chain. In particular, the Markov chain's $|V|$ states

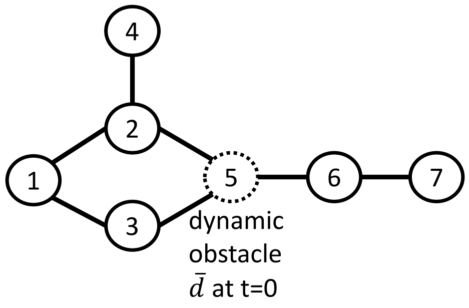


Figure 2: A simple graph with a probabilistic dynamic obstacle $\bar{d}$ having $p_{\bar{d}} = 0.5$ and located in vertex 5 at step 0.

denote the presence of d in one of the vertices at a given time step, while its transitions represent the probability that d moves between two given vertices in two subsequent time steps. The motion model of d can hence be encoded in a stochastic matrix M^d whose generic entry M^d_{uv} represents the probability that d will move from u to v between t and $t+1$. (If we had assumed to have a motion model depending on time, we could have defined multiple stochastic matrices $M^d(t)$ with the same interpretation.)

The probability distribution describing the initial location of dynamic obstacle $d \in D$ in G at step 0 is given by the belief vector

$$\mathbf{b}^d(0) = \begin{bmatrix} b^d_1(0) & \dots & b^d_n(0) \end{bmatrix}. \tag{1}$$

We use the notation $\widehat{\mathbf{b}}^d(t)$ to denote the application of $(M^d)^t$ to the initial belief vector $\mathbf{b}^d(0)$, and use $\hat{b}^d_v(t)$ to denote its element corresponding to vertex v; formally, $\widehat{\mathbf{b}}^d(t) = \mathbf{b}^d(0)(M^d)^t$. Intuitively, $\widehat{\mathbf{b}}^d(t)$ encodes where d might be located at step t, assuming no interactions with the agent.

We also introduce the notation $\eta_d = (v_0, v_1, v_2, \dots, v_{\bar{t}})$ to denote a possible sequence of vertices occupied by dynamic obstacle $d \in D$ between $t = 0$ and $t = \bar{t}$ that is coherent with the initial belief vector $\mathbf{b}^d(0)$ and the corresponding motion model M^d. We call η_d a *path of dynamic obstacle* d, and $\bar{t}$ its *length*. For each vertex v_t belonging to a given η_d, note that it must hold that $\hat{b}^d_{v_t}(t) > 0$. Finally, we use $\mathcal{H}^d$ to denote all the possible sequences of vertices associated with dynamic obstacle d respecting the above definition, regardless of their length.

Example 1. *A simple graph is shown in Figure 2. It contains a single probabilistic dynamic obstacle $\bar{d}$ with $p_{\bar{d}} = 0.5$ whose starting position is precisely known at $t = 0$ (assuming its existence), i.e. $b^{\bar{d}}_5(0) = 1.0$. The motion of $\bar{d}$ is described by the following stochastic matrix:*

$$M^{\bar{d}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This implies that the obstacle will move to either vertex 2 or 3 between step 0 and 1, and then it will stay there forever. Formally,

$$\mathbf{b}^{\bar{d}}(0) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$\widehat{\mathbf{b}}^{\bar{d}}(t) = \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \forall t \geq 1.$$

Dynamic obstacle $\bar{d}$ can hence follow only two types of paths: $\eta^1_{\bar{d}} = (5, 2, 2, \ldots)$ or $\eta^2_{\bar{d}} = (5, 3, 3, \ldots)$.

Let us now consider the interactions between dynamic obstacles and the agent. Each $d \in D$ has the ability to *intercept* the agent which is moving in the graph G. This ability is specified by an interception function $N^d : V \to 2^V$ which associates each vertex $v \in V$ in which the agent can be located with those vertices from which d can intercept it when the former is located in v.

Example 2 (Continued from Example 1). *Following the previous example, let us assume that $\bar{d}$ can only intercept the agent when located in the same vertex, with two exceptions: when $\bar{d}$ is located in vertex 2 or 3, it can intercept the agent even when the latter is located in vertex 1. This can be expressed formally as*

$$N^{\bar{d}}(1) = \{1, 2, 3\}$$
$$N^{\bar{d}}(v) = \{v\} \qquad \forall v \in \{2, 3, 4, 5, 6, 7\}.$$

As the agent moves towards the goal by following a given path π, we can express the probability that each $d \in D$ has intercepted it by time step t —assuming its existence in the environment— as follows. For each step t, we introduce the vector

$$\mathbf{b}^{d;\pi}(t) = [b^{d;\pi}_1(t), \ldots, b^{d;\pi}_n(t)]$$

to keep track of the probability that, at time t, d is in vertex v and has not yet intercepted the agent when the latter has followed π up to t. Element-wise, this probability is denoted by $b^{d;\pi}_v(t)$. We introduce a separate variable $b^{d;\pi}_0(t)$, which we call *interception state*, to represent the probability that the agent, while executing path π, has already been intercepted by d by time t. Clearly, it must hold that $b^{d;\pi}_0(t) = 1 - \sum_{v \in V} b^{d;\pi}_v(t)$. Together, $b^{d;\pi}_0(t)$ and $\mathbf{b}^{d;\pi}(t)$ represent the *belief* about the result of the interactions between the agent executing path π and d up to time step t in all the possible realizations of the world. For time step 0, $b_0^{d;\pi}(0) = 0$ and $\mathbf{b}^{d;\pi}(0)$ encodes the initial probability distribution of d on V, so $\mathbf{b}^{d;\pi}(0) = \mathbf{b}^d(0)$.

For each t and π, $b_0^{d;\pi}(t)$ and $\mathbf{b}^{d;\pi}(t)$ are updated as follows. First, the motion model is applied to $\mathbf{b}^{d;\pi}(t)$:

$$\widehat{\mathbf{b}}^{d;\pi}(t+1) = \mathbf{b}^{d;\pi}(t)M^d. \tag{2}$$

We call $\widehat{\mathbf{b}}^{d;\pi}(t+1)$ the *uncorrected belief vector* at $t+1$, which assumes no interception events happening at $t+1$. Denoting now by $\hat{b}_v^{d;\pi}(t+1)$ a generic element of the uncorrected belief vector $\widehat{\mathbf{b}}^{d;\pi}(t+1)$, each element $b_v^{d;\pi}(t+1)$ of the final vector $\mathbf{b}^{d;\pi}(t+1)$ is then computed as:

$$b_v^{d;\pi}(t+1) = \begin{cases} 0 & \text{if } v \in N^d(\pi[t+1]), \\ \hat{b}_v^{d;\pi}(t+1) & \text{otherwise.} \end{cases} \tag{3}$$

The interception state can finally be updated simply as $b_0^{d;\pi}(t+1) = 1 - \sum_{v \in V} b_v^{d;\pi}(t+1)$. Intuitively, the effect of the application of Eqs. (2)-(3) on $\widehat{\mathbf{b}}^{d;\pi}(t+1)$ is to "move" some probabilities related to obstacle d being in such vertices to the agent's interception state. Hence, the semantic associated with a dynamic probabilistic obstacle d is as follows: if the agent executes path π, it is *intercepted* by d with probability $p_d \cdot b_0^{d;\pi}(|\pi|)$ and it *survives* d with probability $1 - p_d \cdot b_0^{d;\pi}(|\pi|)$.

Example 3 (Continued from Example 2). *Let us now introduce start and goal vertices for our agent: $v_s = 1$, $v_g = 7$. Suppose that the agent follows path $\pi = (1, 2, 5, 6, 7)$. At step 1, $\bar{d}$ will be in vertex 2 with probability 0.5, hence we have*

$$\begin{aligned} \widehat{\mathbf{b}}^{\bar{d};\pi}(1) &= \begin{bmatrix} 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \end{bmatrix} \\ \mathbf{b}^{\bar{d};\pi}(1) &= \begin{bmatrix} 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \end{bmatrix}. \end{aligned}$$

This implies that $b_0^{\bar{d};\pi}(1) = 0.5$. Also, note that the belief vector will not change in subsequent steps. Recalling that $p_{\bar{d}} = 0.5$, once the agent reaches v_g it will have been intercepted with probability 0.25.

2.4 Optimization Problem, Decision Problems, and Complexity Results

The main optimization problem considered in this work can be concisely stated as follows.

Problem 1 (Path Planning in Hostile Environments — PPHE). *Given $\langle G, S, D, v_s, v_g, T\rangle$, where $T \in \mathbb{N}$ is a deadline, compute the path π^* (and the corresponding probability of survival) defined as*

$$\pi^* = \arg\max_{\pi \in \mathcal{P}} \prod_{s \in S} (1 - p_s)^{\sigma(\pi,s)} \prod_{d \in D} (1 - p_d \cdot b_0^{d;\pi}(|\pi|)) \tag{4}$$

s.t.

$$\text{Belief Update Equations (1)} - (3) \quad \forall d \in D, t \in \{1, \ldots, |\pi|\} \tag{5}$$

$$|\pi| \leq T \tag{6}$$

Note, in Eq. (4), the effect of the independency assumption that allows the survivability of the agent to be expressed as the product of the survivabilities relative to each obstacle.

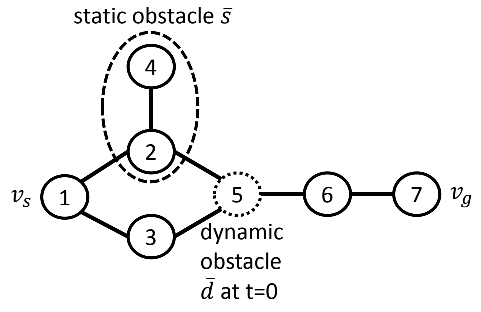


Figure 3: PPHE instance obtained from Example 3 by adding a single probabilistic static obstacle $\bar{s}$ with $p_{\bar{s}} = 0.2$, which covers two vertices.

Example 4 (**Continued from Example 3**). *Let us add a single static probabilistic obstacle to the graph of the previous example, and set a deadline* $T = 5$ *steps. The complete PPHE instance is shown in Figure 3. Static obstacle* $\bar{s}$ *has* $V(\bar{s}) = \{2, 4\}$ *and* $p_{\bar{s}} = 0.2$. *Now, note that all the paths of the form* $(1, 1, \ldots, 7)$ *have survivability not greater than* 0.5*: if the agent does not move from* v_s *at the beginning, it will surely be intercepted by* $\bar{d}$ *regardless of its non-deterministic motion if* $\bar{d}$ *is actually present in the graph (recall that* $p_{\bar{d}} = 0.5$*). It is easy to see that an optimal solution is* $\pi^* = (1, 3, 5, 6, 7)$*, having probability of survival equal to* 0.75*: being on vertex 3 at step 1 allows the agent to completely avoid* $\bar{s}$.

In order to characterize the complexity profile of the PPHE problem, we introduce its decision version.

Problem 2 (Path Planning in Hostile Environments (Decision Version) — PPHE-D). *Given* $\langle G, S, D, v_s, v_g, T, \overline{p} \rangle$*, where* $\overline{p} \in [0, 1]$*, does there exist a path* π^* *reaching* v_g *in at most* T *steps having a probability of survival not less than* $\overline{p}$*?*

The following theorem summarizes a tight NP-hardness result for PPHE-D, which implies the (likely) intractability of PPHE even on seemingly simple special cases.

Theorem 1. *PPHE-D is NP-hard, even when the following conditions hold simultaneously:*

(1) rectangularity: G is a rectangular grid graph;

(2) **no overlap between static obstacles:** for any $a \neq b \in S$, $V(a) \cap V(b) = \emptyset$;

(3) **coherence between dynamic obstacles' and agent's motion model:** all dynamic obstacles can only move to a neighbor vertex between two subsequent steps;

(4) **limited dynamic obstacles' motion:** for any $d \in D$, the corresponding motion model M^d describes a Markov chain such that, for any possible starting vertex v such that $b_v^d(0) > 0$, an absorbing state is reached within $|V|$ steps with probability 1;

(5) **no overlap between static and dynamic obstacles:** for any $s \in S$ and $d \in D$, there does not exist a vertex $v \in V(s)$ and a time step $0 \leq t \leq T$ s.t. $\hat{b}_v^d(t) > 0$;

(6) **dynamic obstacles' maximum interception range limited to 1:** for any $d \in D, v \in V$, $u \in N^d(v)$ implies $\mathbf{d}(u, v) \leq 1$;

(7) **dynamic obstacles' constant interception range:** for any $d \in D$ and $u, v \in V$, $u \in N^d(v)$ with $\mathbf{d}(u, v) = c$ implies $k \in N^d(z)$ for each $k, z \in V$ s.t. $\mathbf{d}(k, z) = c$;

(8) **no uncertainty in dynamic obstacles' initial conditions:** for any $d \in D$, $p_d = 1$ and $b_v^d(0) = 1$ for a single $v \in V$;

(9) **deadline bounded by number of vertices:** $T \leq |V|$;

(10) **no overlap between dynamic obstacles:** for any $d \neq e \in D$, there does not exist a vertex $v \in V$ and a time step $0 \leq t \leq T$ s.t. $\hat{b}_v^d(t) > 0$ and $\hat{b}_v^e(t) > 0$.

The proof is quite long and hence reported in Appendix A. Note that Theorem 1 implies also the intractability of more general classes of problems (e.g. those defined on planar graphs).

In general, it can also be convenient to know beforehand whether a problem instance only admits solutions with zero chances of survival. Similarly, it would be nice to have at hand a fast procedure able to retrieve a solution with survivability 1, whenever present. The two decision problems stated below capture these insights.

Problem 3 (PPHE with Non-Zero Survivability — PPHE-D0)**.** *Given $\langle G, S, D, v_s, v_g, T \rangle$, does there exist a path π^* reaching v_g in at most T steps having a probability of survival greater than 0?*

Problem 4 (PPHE with Survivability 1 — PPHE-D1)**.** *Given $\langle G, S, D, v_s, v_g, T \rangle$, does there exist a path π^* reaching v_g in at most T steps having a probability of survival equal to 1?*

Specialized algorithms for these last two problems are given in Section 5. For PPHE-D0, we have a result similar to Theorem 1:

Theorem 2. *PPHE-D0 is NP-hard, even when Conditions (1)-(9) of Theorem 1 hold simultaneously.*

The proof is again reported in Appendix A. For what concerns PPHE-D1, we have the following result:

Theorem 3. *PPHE-D1 can be decided in pseudo-polynomial time* $O\big(|V||S| + |V|^2T|D|\big)$.

Proof. See Algorithm 3 (Section 5.1). □

Now, note that we cannot easily conclude that PPHE-D, PPHE-D0, and PPHE-D1 belong to the NP complexity class. This is because a feasible solution π of a PPHE instance represented as a time-stamped sequence of vertices $\pi = (v_s, v_1, v_2, \ldots, v_g)$ on G does not have polynomial size w.r.t. the input, but only pseudo-polynomial. However, there are two special classes of problem instances that may be of practical interest for which the NP membership of PPHE-D and PPHE-D0, as well as the membership of PPHE-D1 in P, are easy to prove.

Theorem 4. *PPHE-D and PPHE-D0 are in NP, while PPHE-D1 is in P, when Conditions (4) or (9) of Theorem 1 hold.*

Proof. Condition (9) is immediate. For what concerns (4), note that this condition implies that by time step $|V|$ a generic obstacle d will have reached a vertex v with $M^d_{vv} = 1$. This implies that any solution π where the agent visits more than once a given vertex after step $|V|$ can be turned into another solution π' , equivalent in terms of survivability, where a single visit to that vertex is made. This implies that an instance with $T > 2|V|$ is equivalent to one having $T = 2|V|$. □

Theorem 4 is useful in those situations where the autonomy of the agent and/or the dynamic obstacles only matches the size of the environment. As a corollary of the above theorem, we have that PPHE-D and PPHE-D0 are NP-complete when restricted to the special instances specified by Theorem 1 and 2.

Finally, we look at the membership in PSPACE of PPHE-D, PPHE-D0, and PPHE-D1. For what concerns PPHE-D, as in the NP case, it is not easy to come up with an algorithm –possibly non-deterministic in virtue of Savitch's theorem (1970)– running in polynomial space. This is due to the fact that encoding a generic element of a vector $\mathbf{b}^{d;\pi}(t)$ might require not less than t bits. However, this is not the case for PPHE-D0 and PPHE-D1. The intuitive reason is that, for these two problems, we do not need to keep track of the exact probability of survival along a given agent's path.

We start with two observations regarding PPHE-D0, which will also be useful in Section 5.2 to devise an efficient solution method.

Observation 1. *A PPHE instance where* $p_d < 1$ *for each* $d \in D$ *always admits a solution with survivability greater than* 0.

Observation 1 holds regardless of the presence of some probabilistic static obstacles. We can therefore focus our attention on dynamic obstacles having $p_d = 1$. The following observation provides a key insight.

Observation 2. *A PPHE instance admits a solution with survivability* > 0 *iff, for each* $d \in D$ *having* $p_d = 1$*, there exists a path* $\eta_d \in \mathcal{H}^d$ *with length* T *and an agent's path* $\pi \in \mathcal{P}$ *such that, for each* $t \in \{1, \ldots, |\pi|\}$ *and* $d \in D$*,* $\eta[t] \notin N^d(\pi[t])$.

The above condition on the N^d functions is simply expressing the fact that, for that particular fixed set of joint dynamic obstacles' paths, the agent is able to reach the goal without being intercepted. We can now state the theorem.

Theorem 5. *PPHE-D0 is in PSPACE.*

Proof. The theorem follows from the existence of a non-deterministic Turing Machine (NDTM) deciding PPHE-D0 in polynomial space. If $p_d < 1$ for each $d \in D$, the NDTM accepts (Observation 1). Otherwise, the algorithm starts a non-deterministic search of an agent's path π and a set of dynamic obstacle paths $\eta_d \in \mathcal{H}^d$ for each $d \in D$ having $p_d = 1$ respecting the property described in Observation 2. This search can be carried out step-by-step by storing only a polynomial amount of bits: for each pair of subsequent time steps, the next vertex that can be occupied by the agent only depends on its current position on G; the same holds for each dynamic obstacle d, which can move from vertex v in any vertex $u \in V$ such that $M^d_{vu} > 0$. The current step t of that particular evolution of the world can also be stored in $O(\log(T))$ bits. The Turing Machine enters the accepting state iff it is able to find one of such paths. PSPACE membership follows from the fact that PSPACE=NPSPACE (Savitch, 1970). $\square$

The same result can be proved for PPHE-D1 by leveraging the following observation:

Observation 3. *An agent's path $\pi = (v_s, v_1, v_2, \ldots, v_g)$ has survivability 1 iff, for each v_i:*

- *$v_i \notin V(s)$ for all $s \in S$, and*
- *there do not exist $d \in D$ and $u \in V$ such that $\hat{b}^d_u(i) > 0$ and $u \in N^d(v_i)$.*

Theorem 6. *PPHE-D1 is in PSPACE.*

Proof. The proof is similar to that of Theorem 5. This time, however, the NDTM does not have to search for a particular set of dynamic obstacles' path. Instead, it is sufficient to run a non-deterministic search of an agent's path π respecting the properties outlined in Observation 3. Note that we do not need to compute the exact belief to check the second condition: at each step, it is sufficient to set a boolean flag to "true" on all the vertices having non-zero probability of being occupied by a dynamic obstacle, given the flags set to "true" at the previous step. $\square$

2.5 Remarks

Some observations about Problem 1 are in order before proceeding further with its analysis.

2.5.1 Duality with some robotic search problems

First, we point out that the belief update equations (2)-(3) are often used in (robotic) moving target search problems; see, for example, Hollinger et al., (2009); Hollinger et al., (2015); Banfi et al., (2018a). In those settings, the objective is to maximize the chances that the targets will be found by the robots. In this problem, on the contrary, we want to keep the interception states as low as possible while trying to reach a given destination. Anticipating some results presented in Section 3, one of our contributions is to show that the belief update equations (2)-(3) admit a formalization based on Mixed-Integer Linear Programming. This allows to derive new MILP-based planners for the above search problems, as well as for some of their variants considering connectivity constraints (Hollinger & Singh, 2012) by leveraging some recent works in multirobot connected path planning; see Banfi et al., (2018a); Banfi et al., (2018b). The investigation of the performance of our MILP-based (re)formulation of Eqs. (2)-(3) in the context of a search problem is left for future works.

2.5.2 A note on the independency assumption

Second, the independency assumption might be violated in some settings. For instance, in an intrusion scenario, the agent might have received conflicting information from two different sources regarding the presence/absence of some enemies in a given region of the environment. Another example may be found in robotic path planning in presence of potentially compromised sensors, where the robot is relying on two sensors momentarily providing diametrically opposed readings (e.g., the camera is seeing another vehicle coming towards the robot, but the lidar does not agree). More generally, suppose that some possible scenarios $\theta \in \Theta$ are defined to describe the actual state of the world, along with a probability distribution associated with them. Each scenario describes the existence of some obstacles (both static and dynamic): either all the obstacles exist, or not. Informally, the optimization problem can be written in this case as

$$\pi^* = \arg\max_{\pi \in \mathcal{P}} \sum_{\theta \in \Theta} p(\text{agent survives executing } \pi \text{ in } \theta | \theta) \cdot p(\theta).$$

Assuming dynamic obstacles' interception events still being independent in each scenario, the first term of the above summation must take value 0 in case *any vertex* of *any static obstacle* is traversed in path π at *any step*; otherwise, such term be computed in a way similar to the second product in Eq. (4). The methods proposed in Sections 3 and 4 could be exteded to handle these cases but, since this problem would be more complex, we leave its complete analysis for future works.

2.5.3 The best possible problem formulation?

Finally, we remark that the problem we consider admits an alternative formulation in terms of a Finite-Horizon Partially Observable Markov Decision Process (POMDP). However, note that our formulation is implicitly assuming that the agent will not gain new knowledge about the true positions of the obstacles in the environment unless it is intercepted along its path. This allows to greatly simplify the problem by directly focusing on the maximization of Eq. (4), avoiding the need of optimizing a reward defined over an exponential number of states as would be required in the POMDP case. The well-known "high" intractability of POMDPs –due to the PSPACE-completeness of the policy existence problem even when the horizon is polynomially bounded (Littman, 1996)– would make such optimization infeasible even for instances of moderate size. Nevertheless, if the agent were allowed to update its belief in accordance with some additional observation events (possibly including false positives and/or negatives), a formulation based on Finite-Horizon POMDPs would be more appropriate.

3. A Mixed-Integer Nonlinear Program

This section presents a MINLP for optimally solving the PPHE problem. The MINLP we propose is based on the idea that the agent's path can be planned on a time-stamped

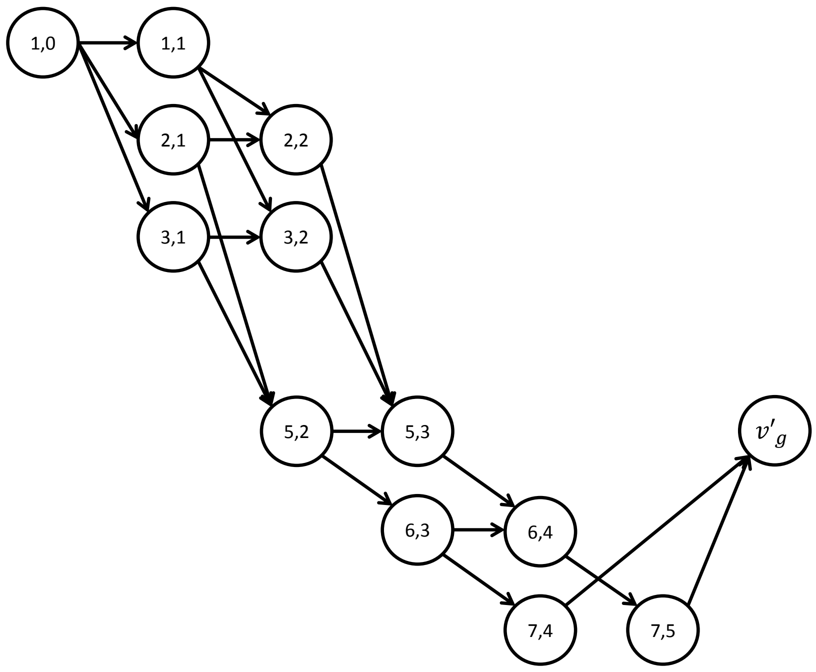


Figure 4: The time-stamped graph G^{TS} corresponding to Example 4.

version of the graph G, coupled with the fact that interception events involving both static and dynamic obstacles can be represented by simple linear constraints. The model's only nonlinearity appears in the objective function, but this is an immediate consequence of the problem definition, as can be seen in Eq. (4).

We start with some notation and concepts that will also be useful in the subsequent sections. In the following, to make the notation more compact, we will use $\mathcal{T}$ to denote set $\{1, \ldots, T\}$. Let V^{TS} be the set of vertex-time step pairs $\langle v, t\rangle$ such that:

(a) the agent can reach v by time t and

(b) it can also reach the goal within the deadline when in v at time t.

Formally,

$$V^{\text{TS}} = \{\langle v, t\rangle \in V \times \{0\} \cup \mathcal{T} | \mathbf{d}(v_s, v) \leq t \wedge t + \mathbf{d}(v, v_g) \leq T\}.$$

With a slight abuse of notation, it is also convenient to define $V^{\text{TS}}(t) = \{v \in V | \langle v, t\rangle \in V^{\text{TS}}\}$ and $V^{\text{TS}}(v) = \{t \in \mathcal{T} | \langle v, t\rangle \in V^{\text{TS}}\}$. Intuitively, $V^{\text{TS}}(t)$ is the set of vertices in which the agent can reside at time t, while $V^{\text{TS}}(v)$ is the set of time steps compatible with the agent residing in vertex v (in both cases, respecting the above conditions (a) and (b)).

Now, note that any feasible solution of a PPHE instance defines a path on a time-stamped version of the graph G with vertex set $\widehat{V}^{\text{TS}} = V^{\text{TS}} \cup \{v'_g\}$, where v'_g denotes a dummy goal vertex (in general, the agent might reach the goal at any step between $\mathbf{d}(v_s, v_g)$ and T), and (directed) arc set

$$A^{\mathrm{TS}} = \{(\langle u,t\rangle, \langle v,t+1\rangle) | \langle u \neq v_g, t\rangle \in V^{\mathrm{TS}} \wedge \langle v, t+1\rangle \in V^{\mathrm{TS}} \wedge [(u,v) \in E \vee u = v]\} \bigcup \{(\langle v_g, t\rangle, v'_g) | \langle v_g, t\rangle \in V^{\mathrm{TS}}\}.$$

Intuitively, A^{TS} simply encodes the agent's legal movements between two subsequent steps in terms of transitions between two time-stamped vertices, or between a time-stamped vertex and the dummy goal vertex v'_g. Note that v'_g only has incoming arcs originating from time-stamped vertices describing the agent being at the goal location v_g, regardless of the particular step in which v_g is reached.

We use $G^{\mathrm{TS}} = (\widehat{V}^{\mathrm{TS}}, A^{\mathrm{TS}})$ to denote this time-stamped graph, and introduce the notation $\mathcal{N}^-(i)$ to denote the set of vertices of G^{TS} having an arc pointing toward $i \in G^{\mathrm{TS}}$; similarly, we use $\mathcal{N}^+(i)$ do denote the set of vertices of G^{TS} for which there exists an arc from $i \in G^{\mathrm{TS}}$ to them.

Figure 4 shows the particular G^{TS} associated with Example 4. Note the absence of the time-stamped versions of vertex 4, due to the fact that the agent cannot make a detour to that vertex when it must reach the goal within 5 steps.

The MINLP is now introduced gradually. Legal paths are modeled in Section 3.1; static obstacles in Section 3.2; dynamic obstacles in Section 3.3; finally, the complete model is shown in Section 3.4.

3.1 Legal Paths

To model legal paths, we start by introducing binary variables x_v^t for each $\langle v,t\rangle \in V^{\mathrm{TS}}$ to express the fact that the agent is located in vertex v at time t in the optimal solution. We also define binary variables y_{uv}^t to express the fact that the agent will move from vertex u to v between t and $t+1$ (with $(\langle u,t\rangle, \langle v,t+1\rangle) \in A^{\mathrm{TS}}$), and additional $y_{v_g v'_g}^t$ variables for each $t \in V^{\mathrm{TS}}(v_g)$. The legality of the paths is enforced through the following constraints:

$$\sum_{\langle j,1\rangle \in \mathcal{N}^+(\langle v_s,0\rangle)} y_{v_s j}^0 = x_{v_s}^0 = 1 \tag{7}$$

$$\sum_{t \in V^{\mathrm{TS}}(v_g)} y_{v_g v'_g}^t = 1 \tag{8}$$

$$\sum_{\langle i,t-1\rangle \in \mathcal{N}^-(\langle v,t\rangle)} y_{iv}^{t-1} = \sum_{\langle j,t+1\rangle \in \mathcal{N}^+(\langle v,t\rangle)} y_{vj}^t = x_v^t \qquad \forall \langle v,t\rangle \in V^{\mathrm{TS}} \text{ s. t. } v \neq v_g, t > 0 \tag{9}$$

$$\sum_{\langle i,t-1\rangle \in \mathcal{N}^-(\langle v_g,t\rangle)} y_{iv_g}^{t-1} = y_{v_g v'_g}^t = x_{v_g}^t \qquad \forall t \in V^{\mathrm{TS}}(v_g) \tag{10}$$

$$x_v^t \in \{0,1\} \qquad \forall \langle v,t\rangle \in V^{\mathrm{TS}} \tag{11}$$

$$y_{uv}^t \in \{0,1\} \qquad \forall (\langle u,t\rangle, \langle v,t+1\rangle) \in A^{\mathrm{TS}} \tag{12}$$

$$y_{v_g v'_g}^t \in \{0,1\} \qquad \forall t \in V^{\mathrm{TS}}(v_g) \tag{13}$$

Constraint (7) set the agent's start vertex, and immediately enforces a decision for what concerns its location at step 1 –note that at most one $y_{v_s j}^0$ can be set to one. Similarly, Constraint (8) sets the agent's arrival at the dummy goal vertex; again, at most one $y^t_{v_g v'_g}$ can be set to one: this means that the agent can be located in v_g at exactly one time step. Constraints (9)-(10) enforce path consistency as the agent travels through the time-stamped graph G^t. The idea is the following: if $\sum_{\langle i,t-1\rangle \in \mathcal{N}^-(\langle v,t\rangle)} y^{t-1}_{iv} = 0$, the agent will not be present in v at time t ($x^t_v = 0$), and hence will not be able to "depart" from v at the subsequent step ($\sum_{\langle j,t+1\rangle \in \mathcal{N}^+(\langle v,t\rangle)} y^t_{vj} = 0$); conversely, if the agent travels from a vertex i to v between $t-1$ and t ($\sum_{\langle i,t-1\rangle \in \mathcal{N}^-(\langle v,t\rangle)} y^{t-1}_{iv} = 1$), the agent is in v at time t ($x^t_v = 1$) and must hence "depart" from v at the subsequent step ($\sum_{\langle j,t+1\rangle \in \mathcal{N}^+(\langle v,t\rangle)} y^t_{vj} = 1$). All together, these constraints ensure the possibility of exploring the space of all the possible $(\langle v_s, 0\rangle, v'_g)$-paths in the corresponding graph G^{TS}. Note that, if the agent reaches the goal before the deadline T, say at step $\hat{t}$, no x^t_v variable with $\hat{t} < t \leq T$ can be active.

We remark that path constraints could alternatively be enforced without resorting to binary edge variables, as done by Morin et al. (2009) for a robotic search problem. However, for the instances used in our experiments (Section 6), the above formulation has proved to be superior.

3.2 Static Obstacles

Interception events related to static probabilistic obstacles are easy to model. We introduce binary variables z_s for each $s \in S$ to express the fact that the agent can traverse any of the vertices of static obstacle s at any time step. This is enforced by means of these constraints:

$$\sum_{v \in V(s)} \sum_{t \in V^{\mathrm{TS}}(v)} x^t_v \leq T z_s \qquad \forall s \in S \tag{14}$$

$$z_s \leq \sum_{v \in V(s)} \sum_{t \in V^{\mathrm{TS}}(v)} x^t_v \qquad \forall s \in S \tag{15}$$

$$z_s \in \{0,1\} \qquad \forall s \in S \tag{16}$$

Constraints (14) activate variable z_s in case *at least* one x^t_v variable on the left-hand side is 1 (note that we multiply z_s by T as, in general, multiple passages through a static probabilistic ostacle can be made). Constraints (15) express the dual condition: if z_s is active, then at least one passage through s must be made.

3.3 Dynamic Obstacles

Interception events related to dynamic probabilistic obstacles are more complicated. We introduce continuous variables $\beta^d_{i;t}$ for each $d \in D$, $t \in \{0\} \cup \mathcal{T}, i \in \{0\} \cup V$ to represent the evolution of the belief according to Eqs. (2)-(3). We also define two additional sets of variables: $\alpha^d_{v;t}$ for each $d \in D$, $t \in \mathcal{T}, v \in V$ (continuous), used to represent uncorrected belief vector elements, and $\gamma^d_{v,t}$ for each $d \in D$, $t \in \mathcal{T}, v \in V$ (binary), used to represent the interception events. These two last sets of variables are actually redundant, but their usage provides a better understanding of the intuition underlying this formulation. The constraints are the following:

$$\beta^d_{i;0} = b^d_i(0) \qquad \forall d \in D, i \in \{0\} \cup V \tag{17}$$

$$\alpha^d_{v;t} = \sum_{u \in V} M^d_{uv} \beta^d_{u;t-1} \qquad \forall d \in D, v \in V, t \in \mathcal{T} \tag{18}$$

$$\gamma^d_{v;t} = 1 - \sum_{u \in V^{\mathrm{TS}}(t) \text{ s.t. } v \in N^d(u)} x^t_u \qquad \forall d \in D, v \in V, t \in \mathcal{T} \tag{19}$$

$$\beta^d_{v;t} \leq \alpha^d_{v;t} \qquad \forall d \in D, v \in V, t \in \mathcal{T} \tag{20}$$

$$\beta^d_{v;t} \leq \gamma^d_{v;t} \qquad \forall d \in D, v \in V, t \in \mathcal{T} \tag{21}$$

$$\beta^d_{v;t} \geq \alpha^d_{v;t} - 1 + \gamma^d_{v;t} \qquad \forall d \in D, v \in V, t \in \mathcal{T} \tag{22}$$

$$\beta^d_{0;t} = 1 - \sum_{v \in V} \beta^d_{v;t} \qquad \forall d \in D, t \in \mathcal{T} \tag{23}$$

$$\alpha^d_{v;t} \in [0,1], \gamma^d_{v;t} \in \{0,1\} \qquad \forall d \in D, v \in V, t \in \mathcal{T} \tag{24}$$

$$\beta^d_{i;t} \in [0,1] \qquad \forall d \in D, i \in \{0\} \cup V, \; t \in \{0\} \cup \mathcal{T} \tag{25}$$

Constraints (17) simply define the initial belief vector according to the provided input. Constraints (18) make the belief evolve according to the corresponding Markov motion model, "temporarily" storing the result in the $\alpha^d_{v;t}$ variables, which can be thought as the MINLP equivalent of the uncorrected belief vector elements $\hat{b}^{d;\pi}_v(t)$. Constraints (19) set a variable $\gamma^d_{v;t}$ to 0 whenever the agent traverses, at step t, a vertex where it can be intercepted when d is placed in v; in the opposite case, the variable takes value 1. Now, note that a variable $\beta^d_{v;t}, v \in V$ must take value 0 whenever $\gamma^d_{v;t}$ is also equal 0, in which case the corresponding "temporary" probability $\alpha^d_{v;t}$ will be moved to the interception state $\beta^d_{0;t}$ (recall the effect of the interception functions N^d); otherwise, $\beta^d_{v;t}$ must be equal to $\alpha^d_{v;t}$. In practice, $\beta^d_{v;t}, v \in V$ evolves as $\beta^d_{v;t} = \gamma^d_{v;t} \cdot \alpha^d_{v;t}$, mimicking the application of the update rule specified by Eq. (3). Constraints (20)-(22) express this product in a linear form by leveraging a Mixed Integer Linear Programming modeling trick. Finally, Constraints (23) update the interception state by exploiting the fact that the $\beta^d_{i;t}$ variables represent, at each step, the values of a probability distribution.

We conclude this subsection with two observations. The first is that, according to Section 3.1, it is possible that the agent reaches the goal before T, say at $\hat{t}$, hence "disappearing" from the world since all the x^t_v variables with $t > \hat{t}$ take value 0. Constraints (18)-(23) will still compute the belief evolution between $\hat{t}$ and T, but the interception state $\beta^d_{0;t}$ remains the same for all $\hat{t} \leq t \leq T$. The second observation is related to the fact that, in practice, one can facilitate the solver to compute an optimal solution by simply avoiding defining some useless variables. These are the ones corresponding to the (d, i, t) triples for which it is known a priori that $b^d_{i;t} = 0$ due to the structure of the underlying motion model M^d.

3.4 The Complete MINLP Model

Having introduced all the constraints, we can finally state the complete MINLP model as

$$\text{maximize} \sum_{s\in S} z_s \log(1-p_s) + \sum_{d\in D} \log(1-p_d\cdot\beta^d_{0;T}) \quad (26)$$
$$\text{s.t.}$$
$$\text{Constraints } (7)-(25)$$

Note, in Eq. (26), the application of the logarithm which allows the problem to be expressed as a *convex* MINLP.[1] However, the optimal solutions of instances only admitting paths with survivability 0 can no longer be found by the solver, which could however still detect model infeasibility (a more efficient approach will be presented in Section 5). In this case, one of such optimal solutions can be retrieved by simply computing the shortest (v_s, v_g)-path in G. We remark that the size of this MINLP is pseudo-polynomial in that of the input. This is due to the fact that we are operating on a time-stamped version of the graph G.

4. Heuristic

This section presents a rather effective heuristic for solving the PPHE problem. The idea of the proposed algorithm is to iteratively refine an initial path by driving it towards space-temporal regions of the time-stamped planning graph G^t (introduced at the beginning of the previous section) displaying low probability of being intercepted by an obstacle, taking into account the dynamic obstacles' belief evolution up to the time step in which the refinement is taking place.

4.1 A First Attempt

To elucidate this concept, let us begin by considering a simpler heuristic. The idea here is to compute a shortest $(\langle v_s, 0\rangle, v'_g)$-path on G^{TS}, where arcs are associated with weights corresponding to the negative log probability of not being intercepted by one or more (static or dynamic) obstacles when located in the corresponding time-stamped target vertices, but *assuming no other prior interaction with those obstacles*. The pseudocode of this heuristic is reported in Algorithm 1.

After having built the time-stamped graph G^{TS}, Lines 3-5 compute the evolution of the dynamic obstacles' beliefs assuming no interaction with the agent. Lines 9-13 associate each arc with a logarithmic weight representing the "history-independent" probability of not being intercepted by an obstacle in the corresponding target vertex (the weight is set to $+\infty$ in case the argument of the logarithm is 0). Finally, the shortest $(\langle v_s, 0\rangle, v'_g)$-path is computed and returned as a legal solution of PPHE in Lines 14-16.

The intuition underlying this heuristic is that of simplifying the problem by considering its history-independent version. Indeed, the algorithm does not embed the notion of interception state relative to a dynamic obstacle, nor it takes into account the actual probability of being intercepted by a static obstacle (i.e. the fact that the conditional probability of

1. Rigorously speaking, a MINLP is always non-convex due to the integrality requirements imposed on some of the variables. However, this expression has become customary in denoting those MINLPs whose relaxation is a convex program.

Algorithm 1: A history-independent heuristic for solving PPHE.

```
   Input: A PPHE instance
   Output: A path $\pi$ s.t. $|\pi| \leq T$
1  function historyIndependent($G, S, D, v_s, v_g, T$)
2      compute the time-stamped graph $G^{\mathrm{TS}}$
3      foreach $d \in D$ do
4          foreach $t \in \{1, \dots, T\}$ do
5              compute $\widehat{\mathbf{b}}^d(t)$;
6      let $w : A^t \to \mathbb{R}_0^+$
7      foreach $(\langle v_g, t \rangle, v'_g) \in A^t$ do
8          $w(a) \leftarrow 0$
9      foreach $\langle v, t \rangle \in V^t$ do
10         foreach $d \in D$ do
11             $\hat{p}_d \leftarrow \sum_{u \in N^d(v)} \hat{b}_u^d(t)$
12         foreach $\langle u, t-1 \rangle \in \mathcal{N}^-(\langle v, t \rangle)$ do
13             $w((\langle u, t-1 \rangle, \langle v, t \rangle)) \leftarrow -\log\left(\prod_{\substack{s \in S:\\ v \in V(s)}} (1 - p_s) \prod_{d \in D} (1 - p_d \cdot \hat{p}_d)\right)$
14     $\pi \leftarrow$ shortest $(\langle v_s, 0 \rangle, v'_g)$-path in $G^{\mathrm{TS}}$ with weights $w$
15     remove $v'_g$ from $\pi$
16     return $\pi$
```

being intercepted a second time after having crossed one of its vertices is always 1). In other words, the agent is planning like it could always be captured again by an obstacle placed in a vertex which should be clear, given the history of the path up to that point. It can be easily seen that, prior to the application of the logarithm, the arc weights represent lower bounds on the actual probability of survival between steps t and $t+1$. The application of the logarithm then allows to compute the path on G^{TS} that maximizes the product of these lower bounds. Although it is easy to provide examples of suboptimality, this algorithm is in general able to ensure that the agent will at least avoid space-temporal regions associated with a high probability of interception.

The algorithm has a pseudo-polynomial run time, due to the exponential dependency on the size of the input parameter T. The most time-consuming parts of the algorithm are Lines 9-13, taking $O\big(|V|^2 T |D|\big)$ when the static obstacles' probability values needed in the product of Line 13 are pre-computed outside the main for loop. The overall worst-case run time can therefore be expressed as $O\big(|V||S| + |V|^2 T |D|\big)$. Clearly, this quantity becomes polynomial if we only consider classes of problem instances where the deadline T is somehow bounded by a polynomial in the size of the remaining part of the input.

4.2 Taking into Account the Belief Evolution

Algorithm 1 is easy to understand and simple to implement, but is not able to leverage well the problem structure. We fix this drawback by presenting a new algorithm which

Algorithm 2: A heuristic for solving PPHE able to take into account the agent's belief about possible interception events.

```
Input: A PPHE instance, a simulation horizon $\Delta \leq T$
Output: A path $\pi$ s.t. $|\pi| \leq T$
1  function main($G, S, D, v_s, v_g, T, \Delta$)
2      $t \leftarrow 0$
3      $v_{\text{curr}} \leftarrow v_s$
4      $\pi \leftarrow (v_s)$
5      $\pi_{\text{prev}} \leftarrow$ null
6      bestObj $\leftarrow$ null
7      while $t < T$ do
8          $\pi_{\text{temp}} \leftarrow$ historyIndependent($G, S, D, v_{\text{curr}}, v_g, T - t$)
9          $\pi_{\text{tempFull}} \leftarrow \pi \oplus \pi_{\text{temp}}[1 :]$
10         if bestObj = null or bestObj < obj($\pi_{\text{tempFull}}$) then
11             bestObj $\leftarrow$ obj($\pi_{\text{tempFull}}$)
12             $\hat{\pi} \leftarrow \pi_{\text{temp}}$
13         else
14             $\hat{\pi} \leftarrow \pi_{\text{prev}}$
15         if $t + \Delta > T$ then
16             $\Delta \leftarrow T - t$
17         $\pi \leftarrow \pi \oplus \hat{\pi}[1 : \min(\Delta, |\hat{\pi}| - 1)]$
18         if $|\hat{\pi}| \leq \Delta$ then
19             return $\pi$
20         $v_{\text{curr}} \leftarrow \hat{\pi}[\Delta + 1]$
21         $\pi_{\text{prev}} \leftarrow \hat{\pi}[\Delta + 1 :]$
22         update $S$ and $D$, simulating the next $\Delta$ steps
23         $t \leftarrow t + \Delta$
24     return $\pi$
```

uses Algorithm 1 as a subroutine in the following way. First, we plan an initial path using Algorithm 1. Then, we simulate the first Δ steps along the computed path, moving the agent and updating the corresponding obstacles' beliefs. Finally, we consider this state as the input of a new, smaller problem instance, and repeat from the first step by keeping either this newly computed portion of path, or that belonging to the best path found so far. This planning scheme allows to plan far ahead in time –biasing the search towards space-temporal regions of the graph with low interception probability– while keeping into account the actual (conditional) probability of interception. The pseudocode of this enhanced version of the heuristic is reported in Algorithm 2. We use the following notation: $\pi \oplus \pi'$ denotes the concatenation of two (portions of) paths; $\pi[i : j]$ denotes the portion of path π from step i to step j (included); $\pi[i :]$ denotes the portion of π from step i until the end.

Lines 2-6 initialize some variables: t, the step corresponding to the current replanning time; v_{curr}, the agent's location at step t; π, the partial path built so far; π_{prev}, the best path from v_{curr} to v_g computed in the previous iteration of the algorithm; bestObj, the objective function corresponding to the best path found so far. The algorithm then enters a while loop in Line 7, remaining therein until the t variable, incremented at each iteration by Δ (Line 23), is less than the deadline T. Line 8 computes a heuristic path from v_{curr} to the goal using Algorithm 1, storing the result in the variable π_{temp}. Line 9 builds a "full" temporary path by concatenating the partial path built so far (π) with π_{temp}, after having removed its first vertex (v_{curr}, which already appears at the end of π). Lines 10-14 determine which path to the goal is better between π_{temp} and π_{prev} (π_{temp} is always used at the first iteration), storing the result in the variable $\hat{\pi}$. Lines 19-20 are needed to take into account the fact that T might not be, in general, a multiple of Δ. Line 17 appends to the path built so far the vertices corresponding to the first Δ steps after t (or just the $|\hat{\pi}|$ next steps in case $|\hat{\pi}| < \Delta$). Line 19 interrupts the while loop in case $\hat{\pi}$ leads to the goal in the next Δ steps. Finally, Lines 20-22 simulate the execution of the subsequent Δ steps assuming that the agent will move as prescribed by the newly computed portion of path, updating v_{curr}, π_{prev}, and the beliefs associated with S and D. In particular, to ensure that the agent will consider the static obstacles in a way consistent with π during the next call to Algorithm 1, it is sufficient to set $p_s = 0.0$ for each $s \in S$ associated with the existence of at least a vertex $v \in V(s)$ in π. For what concerns the dynamic obstacles, instead, it is sufficient to repeatedly apply the belief update equations (2)-(3) for Δ steps with the newly computed portion of π to obtain the new vectors $\mathbf{b}^{d;\pi}(t+\Delta)$, which will be treated as the new initial vectors $\mathbf{b}^d(0)$ in the subsequent call to Algorithm 1.

The introduction of the simulation horizon Δ in Algorithm 2 allows to trade-off a lower computational time ($\Delta \to T$) against more precise arc weights in G^t during subsequent calls to Algorithm 1 ($\Delta \to 1$). However, in general, there is no guarantee that setting a lower value of Δ will always provide a better solution. This is due to the fact that two different choices of Δ could lead the algorithm to explore completely different regions of the search space.

To conclude, we remark that Algorithms 1 and 2 can be integrated more efficiently than as shown in the above pseudocodes (for example, there is no need to rebuild G^{TS} from scratch each time Algorithm 1 is invoked). Regardless of such optimizations, the worst-case runtime of Algorithm 2 is $O\big(\lceil T/\Delta \rceil (|V||S| + |V|^2 T|D|)\big)$, since it performs at most $\lceil T/\Delta \rceil$ calls to Algorithm 1.

5. Limit Cases

This section presents specialized algorithms to solve PPHE-D0 and PPHE-D1.

5.1 Detecting Instances with Survivability 1

We start from the simplest problem, PPHE-D1. This asks, given a problem instance, whether it admits a solution with survivability 1 or not. To answer this question, it is sufficient to run Algorithm 1 on the given problem instance and check the survivability of the corresponding output path. This is because the following proposition holds:

Proposition 1. *A problem instance admits at least a solution with survivability 1 iff Algorithm 1 outputs a path π representing one of such solutions.*

Proof. The result immediately follows from the the fact that an arc $(\langle u, t\rangle, \langle v, t+1\rangle)$ of the time-stamped graph G^{TS} has weight 0 iff the robot can never by intercepted by an obstacle (static or dynamic) when located in v at time $t+1$. This implies that the shortest path computed on G^{TS} has total weight 0 iff the two conditions outlined in Observations 3 are respected. □

Note that this useful feature of Algorithm 1 is preserved by Algorithm 2. However, if the only objective is to detect instances with survivability 1, we can do better by avoiding to compute the graph weights as required by Algorithm 1. In particular, leveraging the idea used in the proof of Proposition 1, we can build a graph $\overline{G}^{\mathrm{TS}} = (\overline{V}^{\mathrm{TS}}, \overline{A}^{\mathrm{TS}})$ with $\overline{V}^{\mathrm{TS}} \subseteq \widehat{V}^{\mathrm{TS}}$ and $\overline{A}^{\mathrm{TS}} \subseteq A^{\mathrm{TS}}$, such that the simple existence of a directed $(\langle v_s, 0\rangle, v_g')$-path in $\overline{G}^{\mathrm{TS}}$ implies the fact that such path has survivability one. To achieve the desired property, it is sufficient to avoid creating all the vertices (along with the associated arcs) whose traversal leads to a violation of any of the two conditions outlined in Observation 3. Algorithm 3 summarizes the procedure just described.

Algorithm 3: A pseudo-polynomial time algorithm deciding PPHE-D1.

```
   Input: A PPHE-D1 instance
   Output: True if the PPHE-D1 instance has answer "yes", False otherwise
1  function survivabilityOneTest($G, S, D, v_s, v_g, T$)
2      compute the time-stamped graph $\overline{G}^{\mathrm{TS}}$
3      if there exists a directed $(\langle v_s, 0\rangle, v_g')$-path in $\overline{G}^{\mathrm{TS}}$ then
4          return True
5      else
6          return False
```

The time needed by Algorithm 3 to build the graph $\overline{G}^{\mathrm{TS}}$ is still $O\big(|V||S| + |V|^2T|D|\big)$. Our simulations, however, have shown that this algorithm can offer a significant speed-up with respect to Algorithm 1.

An alternative approach to detect these kinds of instances –although without any worst-case pseudo-polynomial run time– is to check the feasibility of a Mixed-Integer Linear Program (MILP) containing the same Constraints (7)-(25) of the MINLP presented in Section 3 and the additional constraints

$$\beta^d_{0;T} = 0 \quad \forall d \in D \tag{27}$$

$$z_s = 0 \quad \forall s \in S. \tag{28}$$

We refer to this MILP as MILP-D1. The two approaches are compared in Section 6.

5.2 Detecting Instances with Non-Zero Survivability

In order to detect problem instances guaranteeing a probability of survival greater than 0 without the need of solving the complex MINLP of Section 3, a first idea could be to check the feasibility of a suitable MILP, as done above for the PPHE-D1 case. Recall that we do not need to consider static obstacles here, in virtue of Observation 1. This allows us to restrict our attention to those instances having at least one dynamic obstacle that surely exists. We can therefore try to devise a MILP assuming that an instance will only contain dynamic obstacles respecting such property. However, it is not immediate to come up with such a MILP. For example, consider checking the feasibility of the program obtained by considering again Constraints (7)-(25) plus

$$\beta^d_{0;T} \leq 1 - \epsilon \quad \forall d \in D, \tag{29}$$

where ϵ is a small constant. We refer to this MILP as MILP-D0-APPROX. The name comes from the fact that this program is not actually able to decide PPHE-D0 –although in practice a very small survivability could be enough to justify a withdrawal from a mission– due to the need of introducing a small ϵ to approximate the proper inequality constraints, which would have the form $\beta^d_{0;T} < 1$ and hence could not be used in a MILP. Moreover, it seems a useless effort to use Constraints (17)-(25) to compute precisely the interception states relative to dynamic obstacles: after all, we do not care about the values of the $\beta^d_{0;T}$ variables, as long as they are less than 1.

To overcome these issues we now present a much simpler MILP to decide PPHE-D0, where the existence of a feasible solution on a given instance will be associated with the presence of at least one path guaranteeing some chances of survival. For the reminder of this section, we will assume that $p_d = 1$ for each $d \in D$ in virtue of Observation 1. Recall from Section 2 that we can use $\eta_d = (v_0, v_1, v_2, \ldots, v_T) \in \mathcal{H}^d$ to denote a possible path of dynamic obstacle d with length T. The model is based on the idea of searching an agent's path and a set of joint dynamic obstacles' paths with length T respecting the conditions outlined in Observation 2.

To this aim, we start by introducing auxiliary time-stamped graphs $G^{\mathrm{TS}}_d = (V^{\mathrm{TS}}_d, A^{\mathrm{TS}}_d)$ to represent all the possible paths that might be chosen by dynamic obstacle $d \in D$. To characterize the vertex set V^{TS}_d, we start by introducing start and dummy goal vertices ϕ^d and χ^d. We then define

$$V^{\mathrm{TS}}_d = \{\phi^d, \chi^d\} \cup \{\langle i, t\rangle \in V \times \{0\} \cup \mathcal{T} | \hat{b}^d_i(t) > 0\}.$$

The arc set A^{TS}_d is defined as

$$\begin{aligned} A^{\mathrm{TS}}_d = \{(\langle i, t\rangle, \langle j, t+1\rangle) | \langle i, t\rangle, \langle j, t+1\rangle \in V^{\mathrm{TS}}_d \wedge M^d_{ij} > 0\} \cup \\ \{(\phi^d, \langle i, 0\rangle) | i \in V \wedge \hat{b}^d_i(0) > 0\} \cup \\ \{(\langle i, T\rangle, \chi^d) | i \in V \wedge \hat{b}^d_i(T) > 0\}. \end{aligned}$$

As we did for the agent's planning graph G^{TS}, we use $\mathcal{N}^-_d(i)$ to denote the set of vertices of G^{TS}_d having an arc pointing toward $i \in G^{\mathrm{TS}}_d$, and $\mathcal{N}^+_d(i)$ do denote the set of vertices of G^{TS}_d for which there exists an arc from $i \in G^{\mathrm{TS}}_d$ to them.

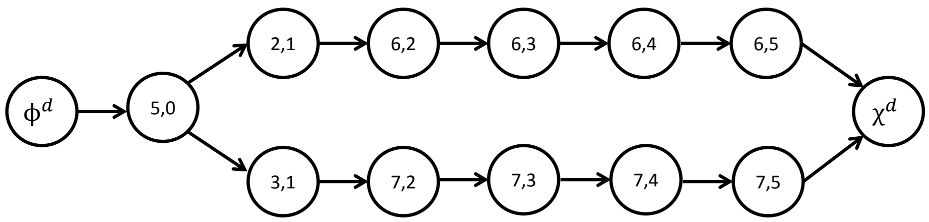


Figure 5: The graph $G_{\bar{d}}^{\mathrm{TS}}$ of Example 5.

Example 5 (Continued from Example 4). *To understand the idea underlying the graphs introduced above, consider again Example 4, but with a slightly different motion model matrix for obstacle d:*

$$M^d = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{30}$$

Moreover, let us now assume that $p_{\bar{d}} = 1.0$. The new instance does not admit any path with survivability greater than 0: at step 2, the obstacle will either be in vertex 6 or 7, and both need to be traversed to reach the goal. The graph $G_{\bar{d}}^{TS}$ associated with this example is shown in Figure 5.

The MILP model we propose uses the same variables and constraints introduced in Section 3.1 to model the agent's path. We further define continuous variables $\delta_{uv}^{d;t}$ to represent the existence of a path segment for dynamic obstacle d where it moves from vertex u to v between steps t and $t+1$ (with $(\langle u,t\rangle, \langle v,t+1\rangle) \in A_{\bar{d}}^{\mathrm{TS}}$). Similarly, we also define continuous variables $\delta_{\phi^d v}$ for each $\langle v,0\rangle \in V_{\bar{d}}^{\mathrm{TS}}$ and $\delta_{v\chi^d}$ for each $\langle v,T\rangle \in V_{\bar{d}}^{\mathrm{TS}}$. The following feasibility program, which we dub MILP-D0-EXACT, can be used to decide PPHE-D0.

$$\text{Constraints (7)-(13)}$$

$$\sum_{\langle v,0\rangle \in \mathcal{N}_d^+(\phi^d)} \delta_{\phi^d v} = \sum_{\langle v,T\rangle \in \mathcal{N}_d^-(\chi^d)} \delta_{v\chi^d} = 1 \qquad \forall d \in D \tag{31}$$

$$\delta_{\phi^d v} = \sum_{\langle u,1\rangle \in \mathcal{N}_d^+(\langle v,0\rangle)} \delta_{vu}^{0;d} \qquad \forall d \in D, \langle v,0\rangle \in V_d^{\text{TS}} \tag{32}$$

$$\delta_{v\chi^d} = \sum_{\langle u,T-1\rangle \in \mathcal{N}_d^-(\langle v,T\rangle)} \delta_{uv}^{T-1;d} \qquad \forall d \in D, \langle v,T\rangle \in V_d^{\text{TS}} \tag{33}$$

$$\sum_{\langle i,t-1\rangle \in \mathcal{N}_d^-(\langle v,t\rangle)} \delta_{iv}^{d;t-1} = \sum_{\langle j,t+1\rangle \in \mathcal{N}_d^+(\langle v,t\rangle)} \delta_{vj}^{d;t} \qquad \forall d \in D, \langle v,t\rangle \in V_d^{\text{TS}} \text{ s.t. } 0 < t < T \tag{34}$$

$$\sum_{\langle j,t+1\rangle \in \mathcal{N}_d^+(\langle v,t\rangle)} \delta_{vj}^{d;t} \leq 1 - \sum_{u \in V^{\text{TS}} \text{ s.t. } v \in N^d(u)} x_u^t \qquad \forall d \in D, \langle v,t\rangle \in V_d^{\text{TS}} \text{ s.t. } 0 < t < T \tag{35}$$

$$\delta_{v\chi^d} \leq 1 - x_{v_g}^T \qquad \forall d \in D, \langle v,T\rangle \in V_d^{\text{TS}} \text{ s.t. } v \in N^d(v_g) \tag{36}$$

$$\delta_{uv}^{d;t} \geq 0 \qquad \forall d \in D, (\langle u,t\rangle, \langle v,t+1\rangle) \in A_d^{\text{TS}} \tag{37}$$

$$\delta_{\phi^d v} \geq 0 \qquad \forall d \in D, \langle v,0\rangle \in V_d^{\text{TS}} \tag{38}$$

$$\delta_{v\chi^d} \geq 0 \qquad \forall d \in D, \langle v,T\rangle \in V_d^{\text{TS}} \tag{39}$$

We now show that MILP-D0-EXACT admits a feasible solution if and only if there exist an agent's path $\pi \in \mathcal{P}$ and a set of dynamic obstacles' paths $\eta_d \in \mathcal{H}^d$ for each $d \in D$ having length T and respecting the condition expressed in Observation 2. To see why this holds, let us examine our feasibility program, momentarily leaving aside Constraints (35)-(36). Constraints (7)-(13) define legal agent's paths in G^{TS}, as in the original MINLP. Constraints (31)-(34) and (37)-(39) do the same for what concerns the obstacles' paths in the corresponding G_d^{TS}, but with one fundamental difference. Specifically, Constraints (31) set the obstacles' dummy start and goal vertices, while Constraints (32)-(34) enforce the legality of the paths. However, note that the corresponding variables are no longer required to take binary values. Leveraging the well-known connection between graph-based path planning and network flow problems,[2] these constraints can be thought as encoding the search for one or more directed paths in G_d^{TS} allowing to send one unit of "commodity" between the obstacles' dummy start vertices ϕ^d, the flow sources, and the dummy goal vertices χ^d, the flow sinks. The commodity (i.e. the obstacle) can be thought as flowing through the arcs of G_d^{TS}, each having unlimited "capacity". This can be seen by noting that here is no upper bound associated with the δ variables: these represent, in a given solution, the amount of commodity sent through the corresponding arc. The commodity can arrive at destination iff there exists at least a *directed path* in G_d^{TS} between ϕ^d and χ^d. Note that this commodity is allowed to split, just like dynamic obstacles are allowed to make different choices when moving according to the corresponding motion model M^d. By construction, such a path would always exist if Constraints (35) and (36) were not present (in fact, it can be easily proven that there always exists a solution where all the δ variables take only

2. See, for example, the first chapter of the book by Ahuja et al., (1993).

values 0 or 1, representing one of the many obstacle's paths). However, the introduction of such constraints blocks all the paths associated with the interception events corresponding to a fixed agent's path: leveraging again the connection with network flow problems, the capacity of an arc $a \in A_d^{\mathrm{TS}}$ is set to 0 whenever, for a given agent's integer solution, an agent located in $u \in V$ at time t can be intercepted by d when the latter is located in a vertex $v \in V$ at time t (the source vertex $\langle v, t\rangle \in V_d^{\mathrm{TS}}$ of a) such that $v \in N^d(u)$. This implies that the model defined by Constraints (7)-(13) and (31)-(39) is unfeasible if and only if there exists an obstacle for which all the possible agent's paths are associated with interception events.

To get another insight on the idea underlying MILP-D0-EXACT, consider again Example 5. In a MILP solution where the agent remains at vertex 1 at step 1 (with $x_1^1 = 1$), the capacity of the arcs $(\langle 2, 1\rangle, \langle 6, 2\rangle)$ and $(\langle 3, 1\rangle, \langle 7, 2\rangle)$ is set to 0 ($\delta_{26}^{d;1} \leq 0$, $\delta_{37}^{d;1} \leq 0$). A solution the agent moves to vertex 2, instead, sets only the capacity of $(\langle 2, 1\rangle, \langle 6, 2\rangle)$ to 0; however, this solution must then necessarily reach vertex 7 at some step $\hat{t}$, hence setting to zero the capacity of the arc having source $\langle 7, \hat{t}\rangle$. A similar argument holds if the agent moves to vertex 3 between step 0 and 1.

To conclude, we remark that the size of MILP-D0-EXACT is pseudo-polynomial in that of the input, as for the original MINLP.

6. Evaluation

We run an extensive set of simulations to assess the performance of the proposed algorithms. All MI(N)LPs are solved with the SCIP solver (vers. 6.0.0) (Gleixner et al., 2018) with a 30 minutes timeout on a machine equipped with two Intel Xeon processors and 64 GB RAM. The heuristic is implemented in Python, using NumPy and the Python interface of the freely available igraph C library (Csardi & Nepusz, 2006). Preliminary experiments show that, when the default SCIP presolver setting is active, there are a few cases where a suboptimal solution is claimed optimal. This strange behavior, probably due to numerical issues, seems to disappear when setting the presolver to "fast".[3] Hence, we use this setting for our validation campaign. The remaining SCIP parameters are kept at their default values.

6.1 Evaluation of PPHE Algorithms

The heuristic developed for PPHE, Algorithm 2, is evaluated against the optimal solutions provided by the MINLP and against two baseline algorithms briefly described below: a Monte Carlo Tree Search approach (MCTS Baseline) and a sampling-based heuristic (SB Baseline). The two baselines are also implemented in Python, with the help of the NumPy and igraph libraries.

3. For all the instances SCIP claimed to be solved to optimality, the heuristics were not able to provide better solutions.

6.1.1 MCTS Baseline

MCTS is a method widely used in the AI community for finding heuristically good plans by randomly sampling the search space and building a search tree according to the obtained results. MCTS is mainly used for attacking complex two-player games, but can be adapted to solve general optimization problems as well. A thorough description of MCTS is out of the scope of this paper; the reader is referred to the survey by Browne et al. (2012) for all the details. In the following, we present a brief overview of the general MCTS approach, and discuss how its key building blocks have been implemented in the MCTS Baseline.

The general MCTS approach prescribes to iteratively build a search tree where nodes represent as usual domain states, and (directed) arcs to children nodes represent choices of actions leading to a subsequent state. Following the survey by Browne et al., a single iteration of the algorithm involves the following four steps:

1) **Selection**: starting from the root node, the search tree built so far is navigated by applying a *child selection policy* to reach the most urgent leaf node to expand.

2) **Expansion**: the node identified at step (1) is expanded by choosing an applicable action.

3) **Simulation**: a "simulation" is run from the newly created node by applying a *default policy* until a terminal (goal) state is reached.

4) **Backpropagation**: the simulation result is backpropagated from the expanded node up to the root to update their statistics.

In the MCTS Baseline, the generic MCTS approach is implemented as follows. *States* are completely specified by partial agent's paths, from which it is possible to recover the value of all the dynamic obstacles' belief vector and of the $\sigma(\cdot,\cdot)$ function values of all the static obstacles. These allow to compute the agent's survivability up to the last step specified in the partial path. *Actions* correspond, intuitively, to the possibility of moving the agent to a neighbor vertex in the subsequent step. The *child selection policy* we employ is quite standard: the tree is always descended through the child node j that maximizes

$$\overline{p}_j + C_p\sqrt{\frac{2\log n}{n_j}},$$

where $\overline{p}_j$ denotes the value of the best "simulation" path that has been obtained from state j, and n, n_j denote the number of visists to current (parent) node and child node, respectively. Note that the above formula is the one used in a particular MCTS implementation called UCT and encodes the usual exploration-exploitation trade-off.

As *default policy* for the simulation step, we consider a very common heuristic-biased stochastic sampling strategy (Bresina, 1996) that chooses the next action a with a (normalized) weight $w(a) = r(a)^{-\tau}$, where $r(a)$ is the rank order of action a (obtained by looking at the corresponding survivability at the next step) and τ is a parameter. Preliminary experiments show that the combination $C_p = 0.5$, $\tau = 4$ is the one that works best.

Finally, in the backpropagation step, the survivability at v_g, $\overline{p}$, is computed, and all the $\overline{p}_j$ values from the expanded node up to the root are replaced with $\overline{p}$ whenever $\overline{p}_j < \overline{p}$. Note that this is different from what usually done in a standard two-player games MCTS algorithm, where the node values are always updated by averaging the results of the different simulations passing through that node. Indeed, in our case, it would make little sense to lower down a node value if we already know the best possible outcome that we can "exploit" from there.

Note that the MTCS baseline allows to compute the optimal solution given a sufficient number of iteration; in practice, however, it runs until a given computational budged is met. In our experiments, we run it for a number of iterations linear in the deadline, precisely $25T$. This value is chosen in order to provide roughly the same computation time of Algorithm 2 to output the final solution (several minutes) when tested for the largest deadline in our simulations.[4] For lower deadline values, however, Algorithm 2 is much faster, making MTCS a very though baseline overall given the extra computational budget allowed in the vast majority of the cases.

6.1.2 SB Baseline

Sampling-based heuristic startegies have been successfully applied in motion planning domain and are known to provide computational advantages. Similar to Algorithm 1 and 2, the heuristic starts by computing the time-stamped graph G^{TS}. This is followed by sampling the existence of each static and dynamic obstacle (according to their corresponding probability of existence), and a further sampling of the trajectories of the existing dynamic obstacles (according to their motion model). The arc weights w in the time-stamped graph G^{TS} are set as follows: apply a fixed cumulative penalty whenever the agent's path intersects with a sampled obstacle, and set the weight to 0 otherwise. Finally, the best path on G^{TS} is computed for the agent by solving the resulting shortest path problem. Such path minimizes the the total number of interacteractions with the obstacles. As in Algorithm 2, in order to reduce the bias in the samples, we allow the agent to take Δ steps along the planned path and then replan a new partial path based on new samples. This process is repeated iteratively, until the goal is reached.

6.1.3 Grid Instances

We start by running simulations on randomly generated grid instances similar to that shown in Figure 1, which simulate an intrusion scenario. The agent must move from the bottom-left to the upper-right corner. Each dynamic obstacle is associated with a different 4x4 black square and moves around it at 1 cell/step in clockwise or counterclockwise direction. In order to make the instances more challenging, we block the two paths on the first column and on the last row. The following parameters are also kept fixed in all the simulations, unless explicitly stated otherwise:

- the 4x4 black squares are separated by a corridor whose width is a single grid cell;

4. See fourth set of simulations below, $T = 66$; avg. run time MCTS: 23 minutes, avg. run time Algorithm 2: 26 minutes.

- for any $s \in S$, $|V(s)| = 1$ and p_s is randomly chosen in $\{0.05, 0.1\}$;
- all dynamic obstacles have the same probability of existence p_d, and can intercept the agent within (Manhattan) distance r;
- 5% of the empty grid cells are populated with a static probabilistic obstacle, and 30% of the 4x4 black squares are associated with dynamic ones. The latter have initial belief centered in a single cell, and move around the corresponding black square according to the following motion pattern (representing a probabilistic delay): move in the next cell with probability $1 - \theta$, and stay at the current cell with probability θ;
- for each combination of parameters, tests are performed on 20 random instances.

Moreover, except when stated otherwise, all the MINLPs are solved by feeding the model with a good heuristic solution obtained by running Algorithm 1. Also, note that one can completely avoid to solve the model if the presence of an instance with optimal solution value 0 is detected (this can be done by running the feasibility program (7)-(13), (31)-(39)); however, this never happened for the instances we generated.

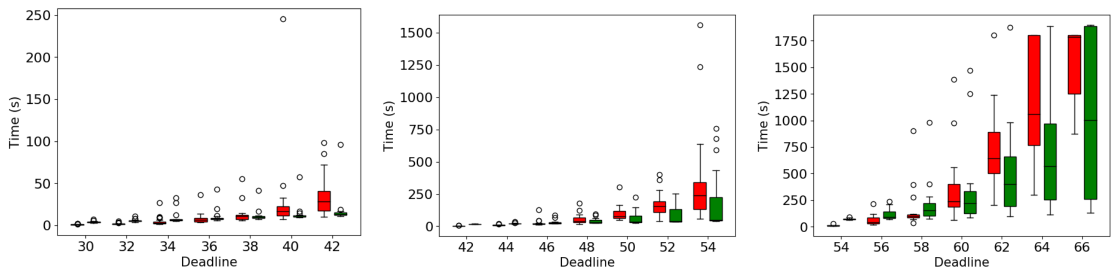
For data visualization, we will use box plots having whiskers with maximum length 1.5 times the interquartile range.

In the first set of simulations, we examine the scalability of the MINLP and, at the same time, evaluate the optimality losses introduced by the basic version of our heuristic (Algorithm 1). For this set of simulations, grids are generated with size 16x16, 22x22, and 28x28. For each $d \in D$ we set $p_d = 1$. For these simulations, we fix $\theta = 0.05$ and $r = 1$.

Figure 6 shows the difference in the MINLP computation times that results from feeding the solver with a good initial feasible solution. We can clearly see the advantage provided by such initialization procedure, especially when dealing with the 28x28 grids. The MINLP gaps[5] of those instances not solved to optimality within 30 minutes for the 28x28 grids are not shown in the figure. However, just to give an idea, we observed that they can remain quite large even when the solver is provided with an initial solution (between 13% and 150%, with a peak of 700% for one instance with $T = 66$).

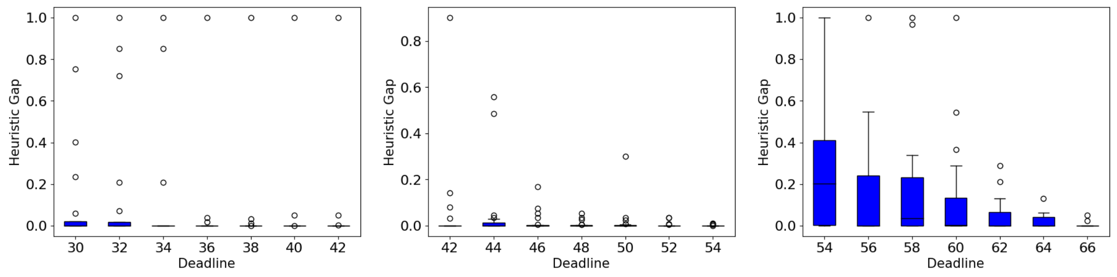
Figure 7 shows the optimality gaps introduced by Algorithm 1 on the instances that are solved to proven optimality by the solver. Heuristic gaps are computed as $(z^* - z)/z^*$, where z^* and z denote the optimal and the heuristic solution, respectively. The box plots show that the Algorithm 1 behaves generally very well in the 16x16 and 22x22 settings as, in most cases, the heuristic gap is 0 (or very close to 0). Note that there are sporadic cases where the heuristic is not able to find any path with survivability greater than 0, in spite of its existence (see the outliers with value 1.0). The heuristic performance decreases in the 28x28 instances, but with a fairly satisfactory median value never going above 25%. We also notice an interesting fact: for a fixed grid size, the heuristic performance increases with the deadline T. This is due to the fact that, when the deadline increases, the survivability of a given instance is likely to increase as well (of course, it cannot decrease). As the survivability approaches 1, the history-independent weights used in Algorithm 1 provide a better approximation of the "right way" of updating the beliefs. We do not show the heuristic computation times, but we point out that all the 16x16, 22x22, and 28x28 instances

5. $(|\text{primal bound} - \text{dual bound}|)/\min(|\text{primal bound}|, |\text{dual bound}|)$.



Grid size: 16x16 Grid size: 22x22 Grid size: 28x28

Figure 6: Grids, first set of simulations – MINLP solution times for different grid sizes and deadlines T, when a heuristic solution is not (red, left) and is (green, right) used to initialize the computation.



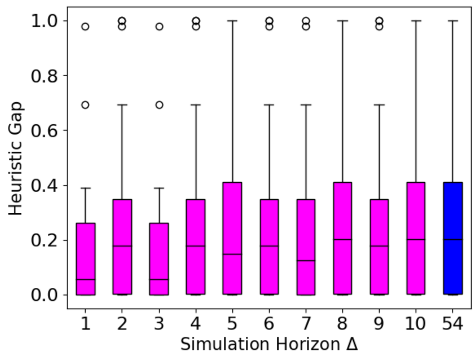
Grid size: 16x16 Grid size: 22x22 Grid size: 28x28

Figure 7: Grids, first set of simulations – Heuristic gaps of Algorithm 1 for different grid sizes and deadlines T.

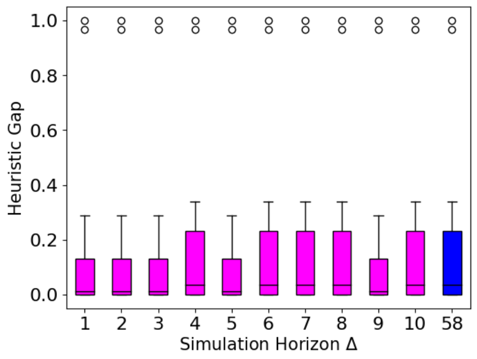
are solved in about, respectively, 3, 10, and 30 seconds (corresponding to the instances with the largest T). Clearly, these times would greatly improve by resorting to an implementation fully written in C.

In the second set of simulations, we examine the effect of the simulation horizon Δ in Algorithm 2. We focus on the previous 28x28 grid instances with low T, as they proved to be the most difficult to handle for Algorithm 1. Specifically, we run experiments with $\Delta \in \{1, \ldots, 10\}$ for $T = 54, 56, 58$. With $T = 56$ no choice of Δ in Algorithm 2 was able to improve the solution given by Algorithm 1, hence the corresponding heuristic gaps are not shown. Looking at those obtained for $T = 54, 58$, we notice that a low Δ is able to provide better results, especially in the $T = 54$ case where a value of $\Delta \in \{1, 3\}$ allows to decrease the median gap from ≈ 0.2 to ≈ 0.05. It can also be seen that there are situations where a higher value of Δ provides better results (see, for example $\Delta = 2, 3$ for $T = 54$), as anticipated in Section 4. Focusing now on the runtimes, we note that they scale coherently with our analysis.

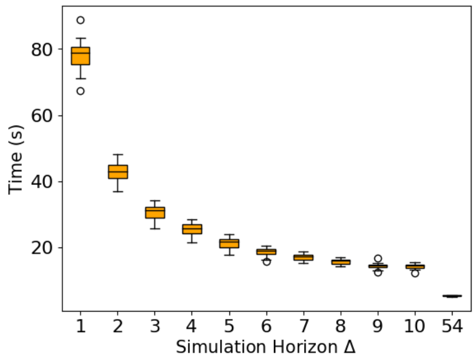
In the third set of simulations, we study how the MINLP solution times and the performance of Algorithm 2 vary as we change the simulation parameters p_d, r, and θ, also comparing Algorithm 2 against the two baselines. For this set of simulations, we keep fixed



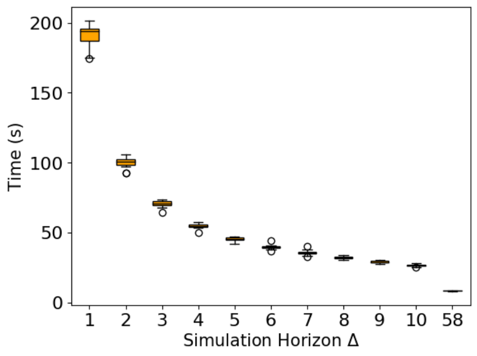
Deadline $T = 54$



Deadline $T = 58$

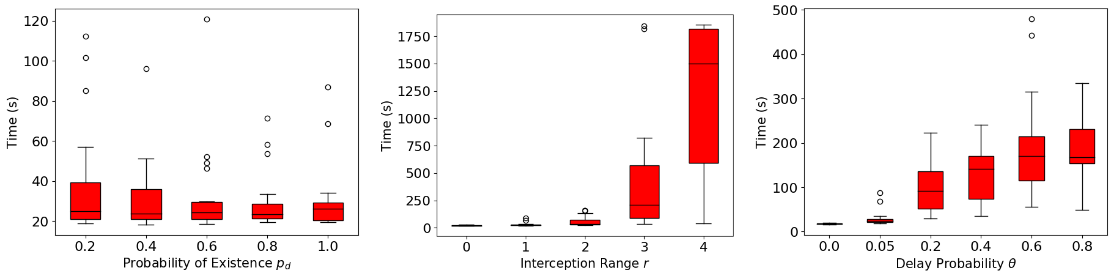


Deadline $T = 54$



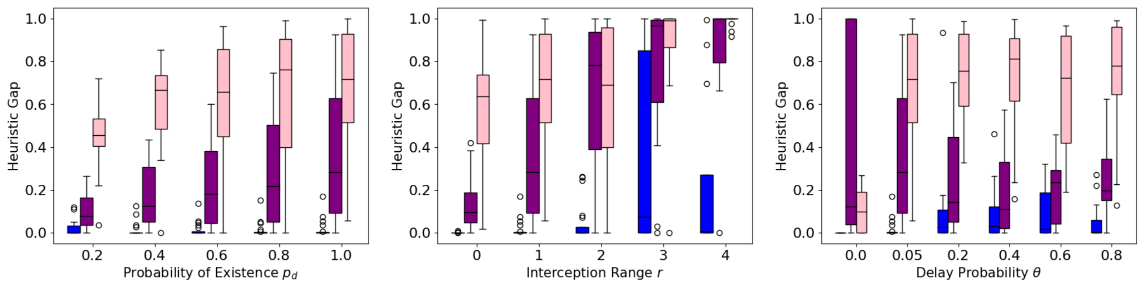
Deadline $T = 58$

Figure 8: Grids, second set of simulations – Top: heuristic gaps of Algorithm 2 for different values of the simulation horizon Δ; bottom: runtime of Algorithm 2 for different values of Δ.



Varying p_d ($r = 1$, $\theta = 0.05$) Varying r ($p_d = 1$, $\theta = 0.05$) Varying θ ($p_d = 1$, $r = 1$)

Figure 9: Grids, third set of simulations – MINLP solution times for different values of p_d, r and θ. Grid size: 22x22, deadline $T = 46$.



Varying p_d ($r = 1$, $\theta = 0.05$) Varying r ($p_d = 1$, $\theta = 0.05$) Varying θ ($p_d = 1$, $r = 1$)

Figure 10: Grids, third set of simulations – Heuristic gaps of Algorithm 2 (blue, left), MCTS (purple, center) and SB (pink, right) baselines for different values of p_d, r and θ. Grid size: 22x22, deadline $T = 46$.

$\Delta = 1$ for both Algorithm 2 and the SB baseline (corresponding to the best value obtained for Algorithm 2 in the previous set of experiments). We also keep fixed the grid size to 22x22 and the deadline T to 46. The results are shown in Figures 9-10, along with the choice of the remaining fixed parameters. Focusing first on the MINLP solution times (Figure 9), we can notice that the impact of a varying p_d is small compared to that of an increase in the motion delay and, above all, a larger interception range. The larger runtimes obtained with $r \in \{3, 4\}$ are easily explained by the presence of a high density of dynamic obstacles in all the randomly generated instances. Examining now the heuristic gaps (Figure 10), we note that Algorithm 2 is quite robust to changes in the simulation parameters. The large heuristic gaps obtained in the instances with $r \in \{3, 4\}$ confirm the insight that such instances are indeed the most challenging. Comparing Algorithm 2 against the MCTS and SB baseline, the superiority of the former is clear. The SB baseline, in spite of taking only $\Delta = 1$ steps along each computed path, is not able to handle effectively the probabilistic nature of the problem and, for this reason, is the algorithm performing worst. The MCTS baseline performs better than the SB baseline, but is clearly still inferior to Algorithm 2.

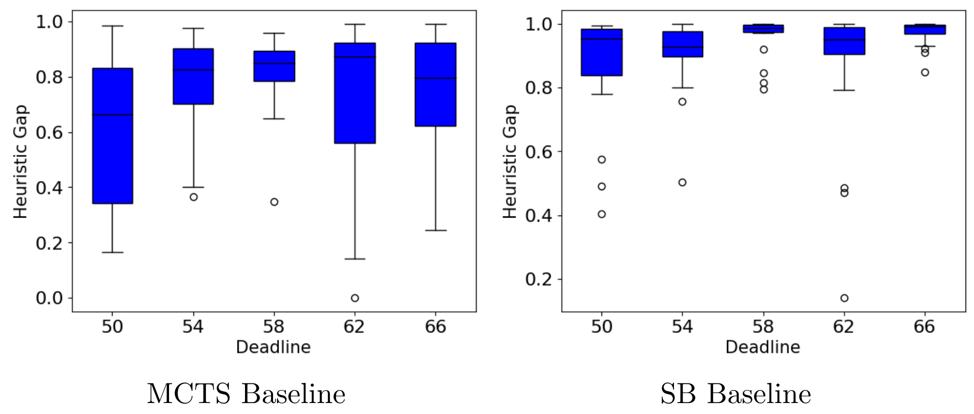


Figure 11: Grids, fourth set of simulations – Heuristic gaps for large deadline values computed w.r.t. the best heuristic solution. Grid size: 22x22, $p_d = 0.8$, $r = 2$.

In the fourth set of simulations, the last set for the grid environments, we examine the performace of Algorithm 2 in instances with large deadlines for which the MINLP approach is no longer a viable option. Algorithm 2 is hence compared only against the two baselines, with $\Delta = 1$. For this set of simulations, we use 22x22 grids where 40% of the black squares are associated with a dynamic obstacles moving around it, $p_d = 0.8$, and interception range $r = 2$. The deadline T varies in $\{50, 54, 58, 62, 66\}$. Note that, compared to the previous 22x22 grid problems, these deadline values result in many more "waiting" options for the agent (e.g. with $T = 66$ there are up to 24 steps in which the agent can decide between moving and staying still). Not having certified optimal MINLP solution at our disposal, we compute the heuristic gap w.r.t. the best solution obtained by the three algorithms. The results show that Algorithm 2 is not able to obtain the best solution in only one instance with $T = 62$ (heuristic gap of 0.015), and that both baselines obtain quite large gaps; see Figure 11.

6.1.4 Real Environments

We also test our algorithms on two real indoor environments/graphs, dubbed Office and Museum, first introduced by Hollinger et al. (2009) in the evaluation of strategies for a related multirobot search problem (which can be thought as the "dual" of the problem considered in this work; see Remark 2.5.1). The two environments are shown in Figure 12, along with the corresponding discretization. In particular, contrarily to the grid environments, a coarser discretization is used in which each room is associated with a graph vertex (edges connect adjacent rooms). This discretization method provides a good trade-off between model accuracy and the ability to solve problems of realistic size. As before, we consider 20 randomly generated instances for each environment. The parameters are the following:

- v_s, v_g Office: 14, 31; v_s, v_g Museum: 4, 65;

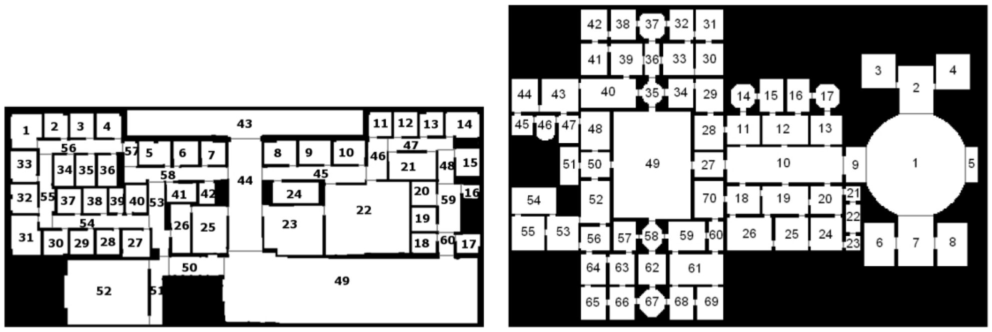


Figure 12: Two real environments used by Hollinger et al. to simulate multirobot search scenarios (2009). Each room is annotated with the index of the corresponding graph vertex. Left: Office environment (60 vertices). Right: Museum environment (70 vertices).

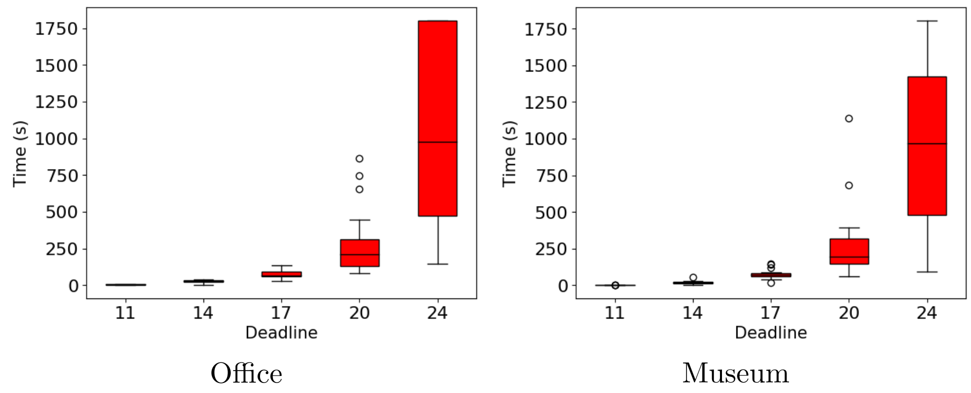
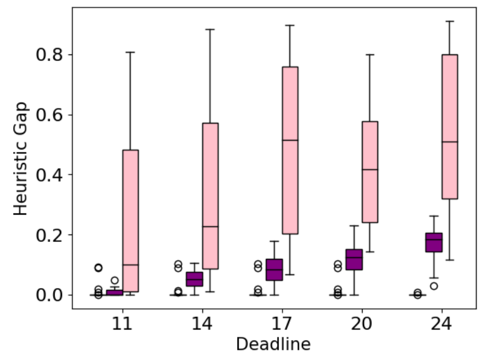


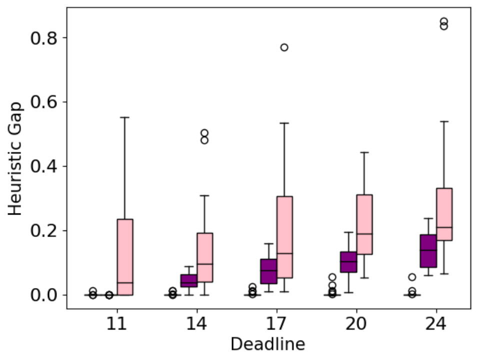
Figure 13: Real environments – MINLP solution times for different grid sizes.

- static obstacles: 6 for Office, 7 for Museum; p_s randomly chosen in $\{0.05, 0.1\}$; each obstacle occupies one room;
- dynamic obstacles: 2 for Office, 3 for Museum; $p_d = 0.8$; motion model: stay in current room with probability 0.5, and move to a neighbor room with uniform probability; interception model: the agent can be intercepted when located in the same room or in a neighboring room (in other words, the obstacle can inspect its current room and all neighboring ones in 1 step), with the exception of "big rooms" (Office: 43, 44, 22, 49, 52; Museum: 1, 10, 49). Also, big rooms do not allow intercepting the agent in a neighbor room.

Figure 13 shows the MINLP solution times for deadlines T varying in $\{11, 14, 17, 20, 24\}$. All the 20 instances are solved until $T = 20$. With $T = 24$, the MINLP approach is still able to provide an optimal solution in the majority of the cases, with MINLP gaps varying between 10% and 60% for those instances not solved to optimality within 30 minutes. With



Office



Museum

Figure 14: Real environments – Algorithm 2 (blue, left), MCTS (purple, center) and SB (pink, right) baselines for a varying deadline.

larger deadlines (28 and onward), we noticed that the MINLP approach starts to become no longer a viable option. Figure 14 shows the heuristic gaps of Algorithm 2, MCTS baseline, and SB baseline ($\Delta = 1$) for a varying deadline, computed on those instances solved to optimality by the MINLP approach. The superiority of Algorithm 2 against the two baselines is again clear.

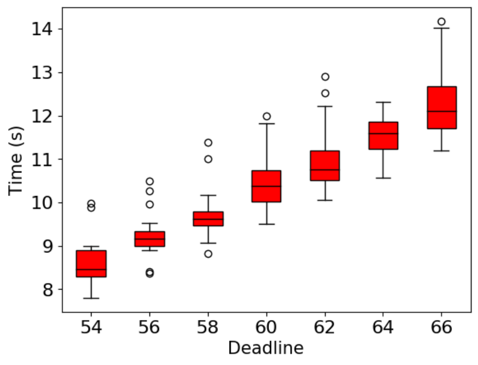
We also consider larger deadline values, for which the MINLP is typically not able to provide an optimal solution: 28, 32, and 36. Comparing Algorithm 2 against the two baselines, we obtain the following results:

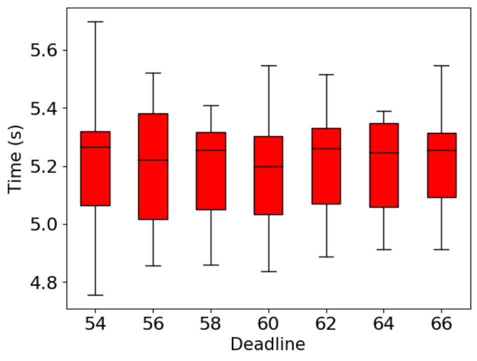
- Office environment: Algorithm 2 does not provide the best solution in a single instace with $T = 36$ (heuristic gap ≈ 0.005);

- Museum environment: Algorithm 2 does not provide the best solution in three instances: one with $T = 28$ (heuristic gap ≈ 0.005), one with $T = 32$ (heuristic gap ≈ 0.008), and one with $T = 36$ (heuristic gap ≈ 0.008).

6.2 Evaluation of PPHE-D1 and PPHE-D0 Algorithms

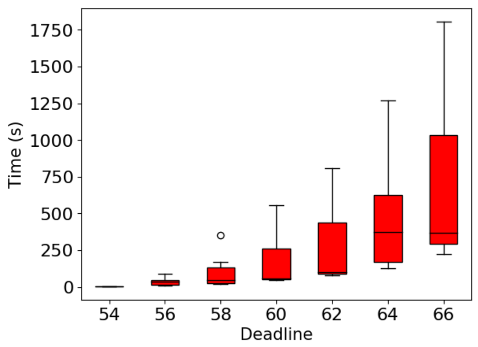
To conclude the evaluation, we compare the algorithms proposed in Section 5 to decide PPHE-D1 and PPHE-D0. For this set of experiments, we use the 28x28 grid instances used in the corresponding first set of simulations, and examine how the runtime of the algorithms varies as we increase the deadline T. Figure 15 shows the results relative to PPHE-D1. We can notice that the advantage introduced by Algorithm 3 is small but consistent across all the values of T. This fact, together with its pseudo-polynomial runtime guarantee, suggests that it should be the preferred choice for solving PPHE-D1 instances. Figure 16 shows the results relative to PPHE-D0. In this case, the computational advantage offered by a more specialized algorithm over the (inexact) variant of the general MINLP approach is huge.

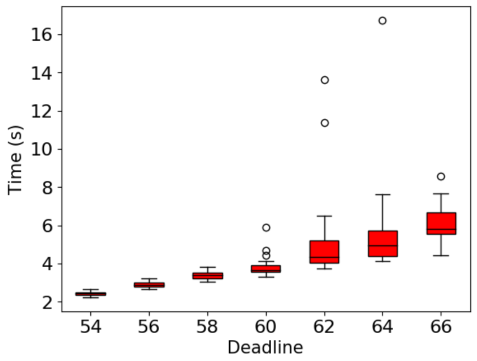




MILP-D1 Algorithm 3

Figure 15: Deciding PPHE-D1. Runtimes of MILP-D1 and Algorithm 3 in the 28x28 instances of the first set of grid simulations.





MILP-D0-APPROX MILP-D0-EXACT

Figure 16: Deciding PPHE-D0. Runtimes of MILP-D0-APPROX and MILP-D0-EXACT in the 28x28 instances of the first set of grid simulations.

7. Related Work

Given the significant body of literature related to navigation in hostile and/or uncertain environments, in this section we only provide an overview of the works that share some of our modeling assumptions. The reader is referred to the recent survey by Agmon (2017) and to the references of the papers mentioned in this section for a more comprehensive overview.

The class of works mostly related to the problem we consider is perhaps the one investigating *patrolling problems*. As the name suggests, these problems are concerned with the employment of some patrolling entities (humans or robots) to monitor a given environment with the aim of detecting the presence of intruders. Initially, the intruder was not directly modeled: the patrollers' objective was simply to maximize the frequency of two subsequent visits to the same location (Chevaleyre, 2004). However, these kinds of (deterministic) patrolling schemes can give rise to fully predictable strategies that may easily be avoided by a rational intruder. Therefore, more recent works explicitly model the intruder's behavior in a strategic adversarial framework. Basilico et al. (2012) present a graph-based game-theoretical patrolling framework where an intruder is able to observe the patroller as it moves on the graph, and to instantaneously place an "attack" on one of the graph vertices. This intrusion scenario is modeled as a leader-follower game, for which the optimal strategy for both the patroller and the intruder can be derived. This model has recently been extended by Basilico et al. (2017), where it is assumed that an alarm system is in place which is able to detect the presence of an attack, but with some uncertainty on the position where the attack is taking place. These two models are elegant, but work under the assumption that the intruder is extremely powerful (infinite sensing range, possibility of instantaneously placing attacks). Basilico et al. (2009) propose some ways of relaxing these assumptions, but these refined models might still fail to capture many real situations (for example, they assume that the intruder can move infinitely fast along a given graph path). Delle Fave et al. (2014) also present a model based on a leader-follower game, which considers the possibility that unexpected events might delay the patrollers (whose actions are modeled by means of Markov chains, as we do in this work). However, the intruder's model is still simple, in that attacks are still modeled as single actions.

The two classes of patrolling models that are most similar to the one considered in this work have been studied by Agmon et al. (2011) and Vaněk et al. (2010, 2011). Agmon et al. (2011) assume that the intruder is a strong adversary endowed with a full knowledge of the patrolling entities (their number and randomized patrolling strategy), but restrict the patrollers to move along perimeters or fences. Therefore, the intruder can succeed by simply penetrating a single "defense line". For this reason, the focus of the study of Agmon et al. (2011) is on the patrollers' strategy. In contrast, our work lies on a complementary direction: we make similar assumptions about the intruder's knowledge (a bit more general, as we assume that a patroller may exist with a given probability), but assume that the patrollers' strategy is given and focus on deriving a path for the intruder guaranteeing maximum chances of survival. Vaněk et al. (2010, 2011) consider a graph-based game-theoretical formalization of a problem similar to the one we investigate: an intruder must reach a goal vertex in a graph patrolled by some adversarial entities. However, they limit the "interception events" to those happening when the intruder meets a patroller on the same graph vertex. In contrast, our model admits situations where the intruder can be intercepted within a given "sensing range". We deem that our study, focused on the intruder's point of view, could provide a solid first step in the direction of extending the game-theoretical framework introduced by Vaněk et al. (2010, 2011) —and the corresponding solution methods, based on a double-oracle algorithm— to handle different kinds of patrolling settings, under less restrictive assumptions concerning the interception events. Specifically, we deem that our model can be particularly useful in dealing with those situations where the patrolling task is not persistent, but triggered by some events (e.g. a mulfunction of the camera monitoring system).

The problem formalized in this work bears also strong resemblances to discrete pursuit-evasion problems, such as those studied by Parsons (1978), Kolling and Carpin (2007), and more recently by Ramaithitima et al. (2016). Chung et al. (2011) provide a good survey of pursuit-evasion problems, classified from a mobile robotics perspective. Classically, work in pursuit-evasion problems focuses on devising worst-case guarantees about the capture of the evaders, which do not necessarily have a fixed goal to reach. Our approach is instead suitable for situations where the evader must reach a predefined goal, and is endowed with an evader-agnostic probabilistic model of the pursuers.

Another class of works related to the model we study considers hostile environments in the context of *coverage problems*, where some agents are employed to visit all the locations of a given enemy area at least once. Yehoshua et al. (2016a, 2016b) assume the presence of probabilistic "threats" that may stop the agent with a given probability while performing the coverage task on a graph-represented environment. These works consider a problem that is more complex than path planning, as we do, but assuming that all the probabilistic threats are static. In contrast, our model makes more general assumptions about the threats, which can move on the graph according to motion patterns describable in terms of Markov chains.

A similar notion of static threats is employed by Likhachev and Stentz (2007) in a *path planning problem* similar to ours. Contrarily to this work, they assume that the agent is always able to sense the actual presence of an adversary in a particular location, and to take appropriate countermeasures. Another notable class of problems similar to our work for what concerns the notion of static threats is that of the Canadian Traveller Problem (CTP) and its variants (Bar-Noy & Schieber, 1991). Here, an agent must reach a goal vertex in a graph where edges are associated with a probability of existence, whose actual presence is only revealed once the agent is about to make the traversal. This problem is similar to ours when all the probabilistic obstacles are assumed to be static. However, the CTP is an online problem where the objective is to compute an optimal policy, while in our case we are dealing with an offline problem where obstacles are in general dynamic.

Adversarial versions of the graph-based path planning problem are introduced by Keidar and Agmon (2017), where the objective of the agent is to reach one of several "safehouse" locations while minimizing the probability of meeting an adversarial agent. This setting can be thought as a particular case of the one considered in this work (a single dynamic obstacle which is surely present in the environment), and facilitates the usage of game theory to derive the optimal strategies of both the intruder and its adversary.

The problem we consider is also similar to the "classical" path planning problem in presence of dynamic obstacles with an uncertain, yet somehow predictable, motion pattern.

We refer the reader to the works by Kruse et al. (1996), Cannon et al. (2012), Aoude et al. (2013), Hardy and Campbell (2013), Saval-Calvo et al. (2017), and Steyer et al. (2018) for an overview of different classes of approaches and to the two recent surveys by Katrakazas et al. (2015) and Paden et al. (2016) focused on autonomous cars. To the best of our knowledge, however, no model presented in the literature allows to account for the actual existence of a dynamic obstacle in a principled probabilistic way.

For what concerns the computational complexity of this latter class of problems, to the best of our knowledge only Sutner and Maass (1988) studied a graph-based discrete setting. In particular, they showed the PSPACE-hardness of the motion planning problem in presence of time-dependent obstacles in the particular case where the obstacles' motion patterns are described by a deterministic periodic function. The period of this function, however, can be arbitrarily large. This fact, together with some additional different modeling assumptions (e.g. weighted graph edges), makes devising an extension of that hardness proof to our framework extremely difficult, if not impossible.

8. Conclusions

In this paper we have introduced and studied a graph-based version of a challenging path planning problem arising in hostile environments that are both dynamic and uncertain, which we dubbed PPHE (Path Planning in Hostile Environments) problem. We have presented some complexity results, a Mixed-Integer Nonlinear Program to compute optimal solution, and a fast –although suboptimal in general– pseudo-polynomial time heuristic algorithm. We have also considered the two limit cases related to the detection of instances with survivability 1 and greater than 0, providing more efficient specialized algorithms. The results of our simulation campaign show that, in spite of the general intractability of this problem, it is possible to solve instances of moderate size to optimality, and obtain good heuristic results in presence of reasonable assumptions about the characteristics of the obstacles.

We envision several directions for future research. First, it would be interesting to derive additional complexity results. Our intuition suggests that the decision version of PPHE might be extremely hard to solve. The reason is that any algorithm leveraging a time-stamped version of the graph for planning is doomed to display a runtime that is at least exponential in the input size. For example, even our heuristics, not offering any optimality guarantee, run in exponential time (due to the $O(T)$ depedency). For this reason, we conjecture that this problem is (at least) PSPACE-hard and leave proving this result as an open problem. Moreover, it would also be interesting to derive additional hardness results for the two limit cases we have examined.

Second, it would be nice to come up with ad hoc heuristics for special classes of problem instances. For example, it would be interesting to study how the problem structure varies when focusing on a particular class of graph topologies, like those where G is a circle or line graph as studied in Agmon et al. (2011), or those where G is a tree. Other ad hoc algorithms could be developed by imposing restrictions on the obstacles, like assuming that their maximum speed is restricted to that of the planning agent, or that their interception range is somehow limited. Recall, however, that our hardness result would likely preclude the existence of polynomial time algorithms even in some of the simplest of such scenarios.

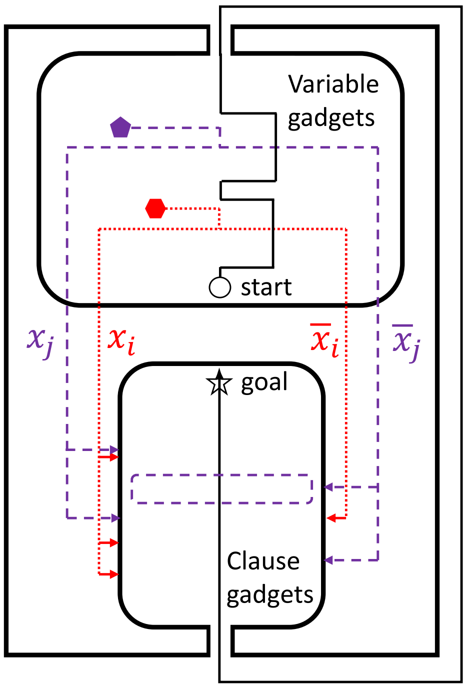


Figure 17: A high-level scheme of the reduction.

Finally, we deem that our results could be useful for the study of more realistic game-theoretical formalization of some patrolling scenarios. In this work, we have assumed the obstacles' motion patterns to be given a priori, and proposed methods to derive the best response of the agent to those patterns, which can be thought as non-deterministic patrolling strategies. However, the MINLP model we presented could also be adapted for usage in the "opposite direction", namely, determining the best non-deterministic patrollers' strategy given a fixed agent path. This model could be leveraged to develop a double-oracle algorithm, in the spirit of the works by Vaněk et al. (2010, 2011).

Acknowledgments

This paper extends preliminary results published in the proceedings of the 18th AAMAS conference as an extended abstract (Banfi & Campbell, 2019). This research has been supported by NASA (grant NNX16AB09G) and ONR (MURI grant N00014-17-1-2699). Any opinions, findings, conclusions, or recommendations here are those of the authors and do not necessarily reflect the views of the funding agencies.

Appendix A. Proofs of the NP-hardness Results

Proof of Theorem 1

In order to prove Theorem 1, we start by providing a sketch of the reduction. This is from the NP-complete problem 3-Satisfiability (3-SAT) (Garey & Johnson, 1979).

3-SAT
INSTANCE: a set U of Boolean variables, and a set C of disjunctive clauses defined on them with exactly three literals each.
QUESTION: does there exist a truth assignment for U that satisfies all the clauses in C?

From a generic instance of 3-SAT we construct, in polynomial time, a particular instance of PPHE-D respecting Conditions (1)-(10) such that the former is satisfiable iff the latter has "yes" answer. The high-level scheme of the reduction is depicted in Figure 17. Here, thick black lines represent probabilistic obstacles with a high probability of existence that the robot is not allowed to traverse. Starting from the *start* vertex, the robot is initially forced to move up in order to exit the *variable gadgets* block. While moving up, it meets $|U|$ *variable (dynamic) obstacles* (the pentagon and the hexagon) –representing a fundamental element of our variable gadgets– which are moving in the opposite direction to reach the *clause gadgets* block. In order to reach it, they have only two paths at their disposal, making them exiting the variable gadgets block either on the left or on the right; see the red (thick dashes) and purple (light dashes) paths in Figure 17. These paths will be executed with probability 0.5 each. The construction makes sure that, as the robot moves up, it cannot avoid being intercepted by each variable obstacle after this probabilistic decision has been made (see where the black and colored lines cross). In the example, the robot is implicitly setting those two variable values to "false".

While the robot is traveling through the external part of the grid region containing variable and clause gadgets —whose border is delimited by static obstacles— to reach the entry at the bottom, the variable obstacles probabilistically spread in that same block according to their appearance in the 3-SAT clauses. In particular, they create $|C|$ "probabilistic barriers", one for each clause, that must be traversed in sequence by the robot in order to reach the goal. These represent our clause gadgets. Figure 17 shows one of such barriers corresponding to a clause containing x_j negated. The idea of the reduction is that the 3-SAT instance admits a satisfiable assignment if and only if the robot is able to reach the goal without needing to pass –in the clause gadgets block– through some cells displaying some probability of being intercepted a second time by any variable obstacle. These cells correspond to those reachable by the variable obstacle when traveling along the opposite path w.r.t. the one where the robot has already been intercepted with probability 0.5, i.e. in the variable gadgets block.

In the following, it is immediate to verify that, in our construction, Conditions (1)-(10) hold. For the sake of brevity, we will not explicitly discuss them.

Variable Gadgets Block

Figure 18 shows the initial portion of the variable gadgets block. Here, grey and green (with a light gradient) squares denote probabilistic static obstacles with $p_s = 1 - 0.5^{|U|+1}$ (the reason for this value will become clear later). Triangles, penthagon and hexagon denote instead dynamic obstacles with $p_d = 1$. The triangles can only intercept the robot from their current vertex (distance 0), while the penthagon and the hexagon can intercept the robot within Manhattan distance 1. Let us examine how the robot, starting from the yellow circle, can exit this block without passing through the static obstacles.

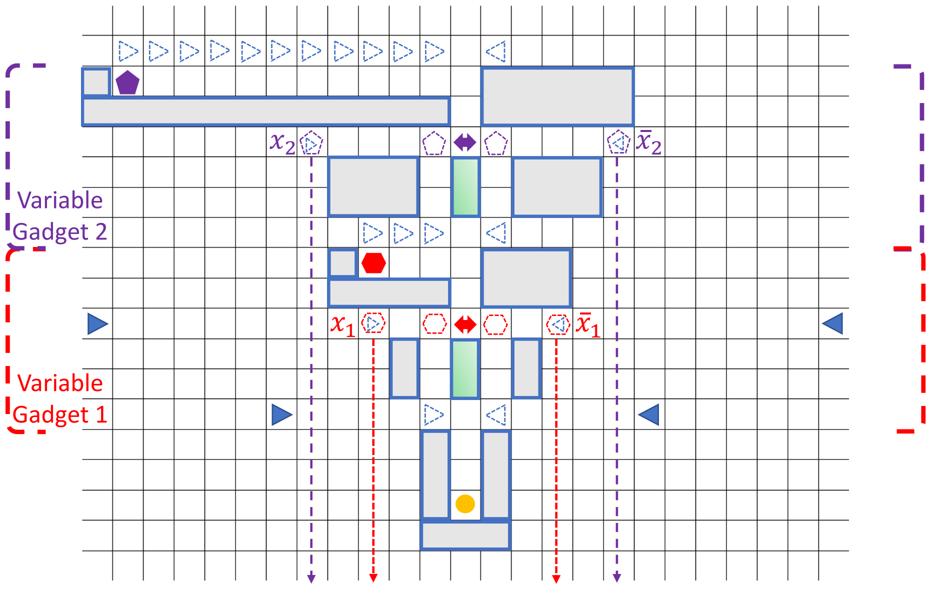


Figure 18: The initial portion of the variable gadgets block. The gadgets corresponding to the first two variables are shown.

The robot must move up for the first three steps. Then, it has to decide whether to pass the green obstacle on the left or on the right: this is associated with setting the value of variable x_1 to true or false, respectively. If the robot does not make a decision in the next step, it will remain blocked due to the two triangular dynamic obstacles which move in the corresponding pointing direction at 1 cell/step until the corresponding dashed position is reached. Therefore, at step 5, the robot can start traversing one of the two passages next to the green obstacle. Meanwhile, also the red (hexagonal) dynamic obstacle, associated with variable x_1, has started to move deterministically, reaching the cell denoted by two red arrows. According to the corresponding motion model matrix, at step 6 the red variable obstacle will be in one of the two adjacent cells with probability 0.5. In the subsequent steps, it will deterministically move (after that probabilistic decision) as shown, again at 1 cell/step towards the clause gadgets block. At step 6, the robot is still forced to move up due to the fact that the next two paths next to the second green obstacle will be closed by step 13. Again, this is enforced by means of triangular obstacles (only the last occupied positions are shown). This implies that, by step 6, the robot will have been intercepted with probability 0.5. Also, note that the robot cannot follow the variable obstacle in reaching the clause gadgets block because those paths will again be blocked thanks to additional triangular dynamic obstacles, again moving at 1 step/cell until the dashed position is reached. What we have just described is a variable gadget. The same pattern is then repeated for the remaining $|U|-1$ variables by properly displacing the dynamic obstacles' starting positions, and by adding additional triangular obstacles to forbid the passages right above the variable obstacles' starting positions. This construction allows the robot to exit the variable gadgets block from above not before time step $8|U|+4$.

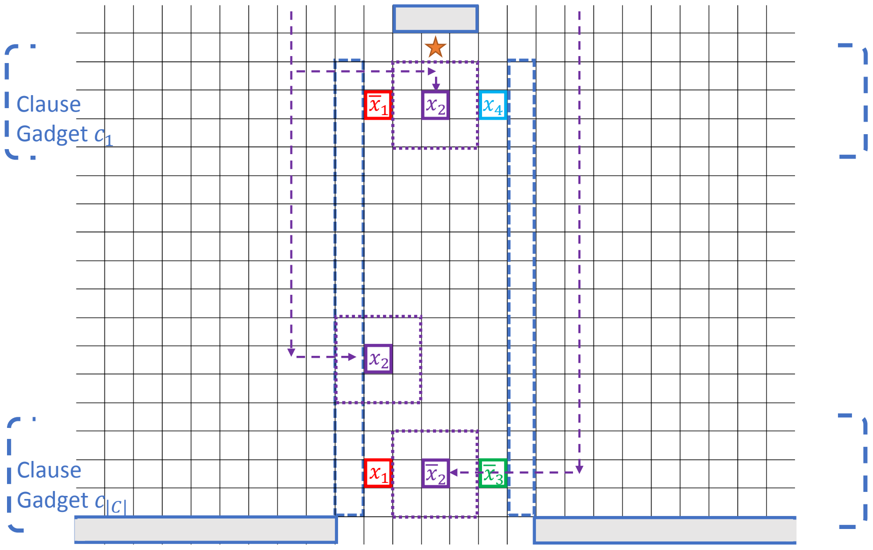


Figure 19: A portion of the clause gadgets block associated with a possible 3-SAT instance. Clause gadgets associated with clauses c_1 and $c_{|C|}$ are fully shown.

Clause Gadgets Block

This block contains the (probabilistic) goal locations of variable obstacles, which are organized into a set of $|C|$ "probabilistic barriers" that the robot needs to traverse, starting from the bottom, to reach the goal (the star). Figure 19 shows a portion of the construction associated with a possible instance having $c_1 = \bar{x}_1 \vee x_2 \vee x_4$ and $c_{|C|} = x_1 \vee \bar{x}_2 \vee \bar{x}_3$, and where x_2 only appears three times (two times as a positive literal and one time as a negative literal). Now, let $|U(l_i)|$ be the number of appearances of literal l_i in the clauses. We show how to construct the motion model matrix of variable obstacle x_2. Looking at the rightmost path (negated variable), note that it describes a deterministic travel having as destination the cell denoted by $\bar{x}_2$. This means that, if the robot were not present, that cell would eventually be occupied by the corresponding variable obstacle with probability $0.5/|U(\bar{x}_2)| = 0.5$. The leftmost path, instead, describes a travel eventually leading the obstacle to be in the corresponding two cells with probability $0.5/|U(x_2)| = 0.25$ each. This is achieved by deviating the obstacle's downward travel at the same height of the c_1 barrier with probability 0.5 (by adding a horizontal travel segment), and go straight with the same probability; the remaining part of the travel is then deterministic. As a general rule, the j-th clause containing literal l_i encountered while moving downwards corresponds to a turn with $1/(|U(l_i)| - j + 1)$ probability. Note, in Figure 18, that these barriers have a height of 3 cells (resulting from the obstacles' interception ranges) and are separated by an empty row. These allow to make all the variable obstacles headed to the same probabilistic barrier turning at a slightly different height (± 1 w.r.t. their possible goal cell), hence easily avoiding the possibility that two variable obstacles may be located in the same cell at the same step (the precise turning scheme is not shown in Figure 19 to keep the figure simple).

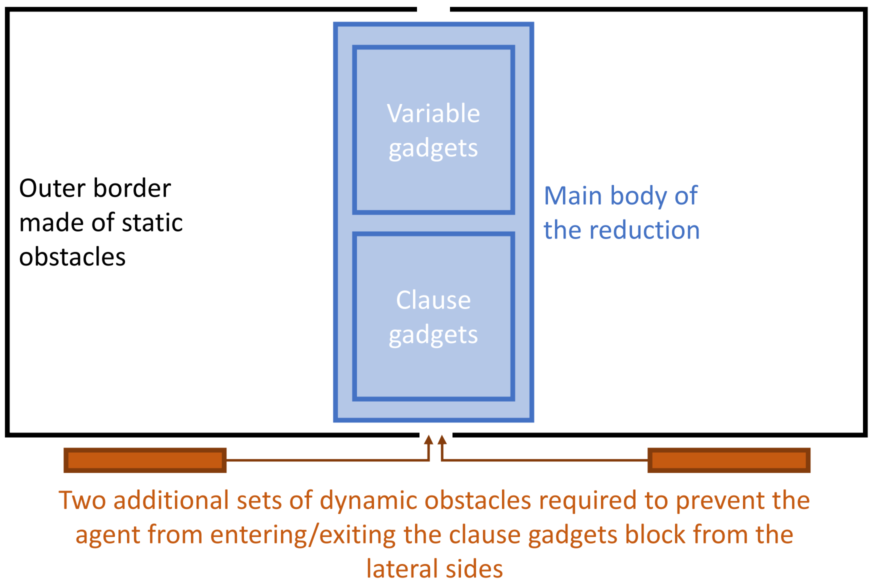

Variable gadgets
Clause gadgets
Outer border made of static obstacles
Main body of the reduction
Two additional sets of dynamic obstacles required to prevent the agent from entering/exiting the clause gadgets block from the lateral sides

Figure 20: Another view of the reduction.

The last ingredient we need is a way to block this passage on the two sides of the probabilistic barriers by using two additional sets of $4|C|-1$ dynamic obstacles each, whose destinations are the dashed blue cells. Their initial positions are set below the clause gadgets block, as qualitatively shown in Figure 20. We just need to make sure that the first obstacle of each set enters the clause gadgets block as soon as the last variable obstacle has reached its destination in any possible realization of the world. For example, it is safe to set the starting position of the rightmost (leftmost) obstacles in the left (right) set by considering a displacement of $4(8|U|+4)+4|C|+9$ cells w.r.t. the one placed in the middle of the clause gadgets block entrance.

Finally, note that the robot must reach the entrance of the this block only when all the dynamic obstacles (variables and the two additional sets just defined) have reached their destinations. To ensure this condition, it is sufficient to appropriately set the horizontal distance between the entrance of the clause gadgets block and the left/right borders of the construction (made of static obstacles; see again Figure 20). Given the structure of the two blocks discussed above, this can safely be set to, for example, $4(8|U|+4)+8|C|+9$.

Putting blocks together

The reduction is almost complete: we just need to specify the deadline T and the survivability threshold of the decision problem $\bar{p}$. We set the latter to $0.5^{|U|}$. The deadline T, instead, can be set to *any value* (bounded by $|V|$) allowing the robot to reach the goal after having exited the variable gadgets block, traveled along the outside border of the construction (either on the left or on the right), and executed an arbitrary snake-like path in terms of traversed "probabilistic barrier cells" from the entrance of the clause gadgets block to the goal (this implies that also a deadline-free version of the problem would be at least NP-hard).

Now, given the above construction, it is immediate to verify that a 3-SAT instance admitting a "yes" answer is associated with a path in the constructed grid world with survivability exactly equal to $0.5^{|U|}$. On the contrary, suppose that there exists a path with survivability greater or equal than $0.5^{|U|}$ in our particular grid instance. Then, it must be the case that such path never traverses neither a static probabilistic obstacle (this would immediately make the survivability drop to $0.5^{|U|+1}$), nor a dynamic probabilistic obstacle d following a deterministic path (the triangles of the variable gadgets block and the two additional sets heading to the clause gadgets block from below). Moreover, a path where the robot sets a variable, say x_i, to a given value (as discussed in Section 8) is incompatible with a passage –in the clause gadgets block– through a cell lying in the interception range of a possible final position of the corresponding variable obstacle associated with the opposite value. Indeed, this would immediately imply a survivability w.r.t. that variable obstacle < 0.5 (and, hence, a total survivability $< 0.5^{|U|}$). This means that we are able to reconstruct a coherent truth assignment for the variables of the original 3-SAT instance able to satisfy all of its clauses. □

Proof of Theorem 2

The proof of Theorem 2 is built on a reduction very similar to that used for Theorem 1. However, instead of reducing from plain 3-SAT, we reduce from the following variant which restricts each variable to appear the exact same number of times as a positive and negative literal across the whole set of clauses.

POSITIVE-EQUAL-NEGATIVE 3-SAT
INSTANCE: a set U of Boolean variables, and a set C of disjunctive clauses defined on them with exactly three literals each. Each variable appears in C the same number of times as a positive and as a negative literal.
QUESTION: does there exist a truth assignment for U that satisfies all the clauses in C?

Lemma 1. *POSITIVE-EQUAL-NEGATIVE 3-SAT is NP-complete.*

Proof. The problem is clearly in NP. To prove its NP-hardness, we reduce from 3-SAT. From a general instance of 3-SAT, we construct a particular instance of POSITIVE-EQUAL-NEGATIVE 3-SAT as follows. First, we copy the original 3-SAT instance. Then, for each variable x_i, let $d_i \leq C$ be the difference in the number of positive and negative literals associated with x_i appearing in C. If $d_i \neq 0$, we create d_i new clauses of the form $l_i \vee a_i^j \vee \bar{a}_i^j$ for each $j \in \{1, \dots, d_i\}$, where a_i^j is an auxiliary variable and l_i is positive if, in the original 3-SAT instance, there are more negative than positive literals associated with x_i, and vice versa. Since these new clauses always evaluate to *true*, this particular instance of POSITIVE-EQUAL-NEGATIVE 3-SAT has answer "yes" iff the original 3-SAT instance has answer "yes". □

Note that the above reduction is implicitly assuming that we are considering "legal" all the instances containing clauses where the same variable appears both negated and un-negated.

Now, to prove Theorem 2, we reduce from POSITIVE-EQUAL-NEGATIVE 3-SAT. We use the same construction of the previous proof, but with a small modification concerning the variable obstacles. Suppose that variable x_i appears in C m_i times negated and m_i times un-negated in the POSITIVE-EQUAL-NEGATIVE 3-SAT instance. Instead of spawning a single variable obstacle in each variable gadget, we spawn m_i variable obstacles at the same grid cell, each one associated with a different pair of clauses in C whenever possible (namely, unless x_i only appears in a single clause, in which case both obstacles can be associated with it). The motion model of these obstacles is then set exactly as before, implying that each variable obstacle will move to the associated clause gadgets with probability 0.5 each. The theorem then follows from the exact same argument used above. $\square$

To conclude, we also point out that our reductions can also be easily modified to prove the NP-hardness of problem instances defined on 2D grid graphs with holes not containing any static probabilistic obstacle, but only dynamic obstacles whose only uncertainty lies in the motion pattern.

References

Agmon, N. (2017). Robotic strategic behavior in adversarial environments. In *Proc. IJCAI*, pp. 5106–5110.

Agmon, N., Kaminka, G. A., & Kraus, S. (2011). Multi-robot adversarial patrolling: facing a full-knowledge opponent. *J Artif Intell Res*, *42*, 887–916.

Ahuja, R., Magnanti, T., & Orlin, J. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall.

Aoude, G. S., Luders, B. D., Joseph, J. M., Roy, N., & How, J. P. (2013). Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns. *Auton Robot*, *35*(1), 51–76.

Banfi, J., & Campbell, M. (2019). High-level path planning in hostile dynamic environments. In *Proc. AAMAS*, pp. 1799–1801.

Banfi, J., Basilico, N., & Amigoni, F. (2018a). Multirobot reconnection on graphs: Problem, complexity, and algorithms. *IEEE Trans Robot*, *34*(5), 1299–1314.

Banfi, J., Basilico, N., & Carpin, S. (2018b). Optimal redeployment of multirobot teams for communication maintenance. In *Proc. IROS*, pp. 3757–3764.

Bar-Noy, A., & Schieber, B. (1991). The canadian traveller problem. In *Proc. SODA*, pp. 261–270.

Basilico, N., De Nittis, G., & Gatti, N. (2017). Adversarial patrolling with spatially uncertain alarm signals. *Artif Intell*, *246*, 220–257.

Basilico, N., Gatti, N., & Amigoni, F. (2012). Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artif Intell*, *184-185*, 78 – 123.

Basilico, N., Gatti, N., Rossi, T., Ceppi, S., & Amigoni, F. (2009). Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings. In *Proc. WI-IAT*, Vol. 2, pp. 557–564.

Bresina, J. (1996). Heuristic-biased stochastic sampling. In *Proc. AAAI*, p. 271–278.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE T Comp Intel AI*, *4*(1), 1–43.

Cannon, J., Rose, K., & Ruml, W. (2012). Real-time motion planning with dynamic obstacles. In *Proc. SOCS*, pp. 33–40.

Chevaleyre, Y. (2004). Theoretical analysis of the multi-agent patrolling problem. In *Proc. IAT*, pp. 302–308.

Chung, T. H., Hollinger, G. A., & Isler, V. (2011). Search and pursuit-evasion in mobile robotics. *Auton Robot*, *31*(4), 299.

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, *Complex Systems*, 1695.

Delle Fave, F. M., Jiang, A. X., Yin, Z., Zhang, C., Tambe, M., Kraus, S., & Sullivan, J. P. (2014). Game-theoretic patrolling with dynamic execution uncertainty and a case study on a real transit system. *J Artif Intell Res*, *50*, 321–367.

Garey, M., & Johnson, D. (1979). *Computers and intractability: A guide to the theory of NP-completeness.* W.H. Freeman.

Gleixner, A., et al. (2018). The SCIP Optimization Suite 6.0. Technical report, Optimization Online.

Hardy, J., & Campbell, M. (2013). Contingency planning over probabilistic obstacle predictions for autonomous road vehicles. *IEEE Trans Robot*, *29*(4), 913–929.

Hollinger, G., & Singh, S. (2012). Multirobot coordination with periodic connectivity: Theory and experiments. *IEEE Trans Robot*, *28*(4), 967–973.

Hollinger, G., Singh, S., Djugash, J., & Kehagias, A. (2009). Efficient multi-robot search for a moving target. *Int J Robot Res*, *28*(2), 201–219.

Hollinger, G., Yerramalli, S., Singh, S., Mitra, U., & Sukhatme, G. (2015). Distributed data fusion for multirobot search. *IEEE Trans Robot*, *31*(1), 55–66.

Katrakazas, C., Quddus, M., Chen, W., & Deka, L. (2015). Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transport Res C-Emer*, *60*, 416–442.

Keidar, O., & Agmon, N. (2017). Safety first: Strategic navigation in adversarial environments. In *Proc. AAMAS*, pp. 1581–1583.

Kolling, A., & Carpin, S. (2007). The graph-clear problem: definition, theoretical properties and its connections to multirobot aided surveillance. In *Proc. IROS*, pp. 1003–1008.

Kruse, E., Gutsche, R., & Wahl, F. (1996). Estimation of collision probabilities in dynamic environments for path planning with minimum collision probability. In *Proc. IROS*, Vol. 3, pp. 1288–1295.

Likhachev, M., & Stentz, A. (2007). Goal directed navigation with uncertainty in adversary locations. In *Proc. IROS*, pp. 4127–4134.

Littman, M. (1996). *Algorithms for sequential decision making.* Ph.D. thesis, Brown University.

Liu, J., Corbett-Davies, J., Ferraiuolo, A., Ivanov, A., Luo, M., Suh, G. E., Myers, A. C., & Campbell, M. (2018). Secure autonomous cyber-physical systems through verifiable information flow control. In *Proc. CPS-SPC*.

Madani, O., Hanks, S., & Condon, A. (2003). On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif Intell*, *147*(1-2), 5–34.

Morin, M., Abi-Zeid, I., Lang, P., Lamontagne, L., & Maupin, P. (2009). The optimal searcher path problem with a visibility criterion in discrete time and space. In *Proc. FUSION*, pp. 2217–2224.

Paden, B., Čáp, M., Yong, S. Z., Yershov, D., & Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans Intell Veh*, *1*(1), 33–55.

Parsons, T. D. (1978). Pursuit-evasion in a graph. In *Theory and applications of graphs*, pp. 426–441.

Portugal, D., & Rocha, R. (2011). A survey on multi-robot patrolling algorithms. In Camarinha-Matos, L. M. (Ed.), *Technological Innovation for Sustainability*, pp. 139–146. Springer Berlin Heidelberg.

Ramaithitima, R., Srivastava, S., Bhattacharya, S., Speranzon, A., & Kumar, V. (2016). Hierarchical strategy synthesis for pursuit-evasion problems. In *Proc. ECAI*, pp. 1370–1378.

Saval-Calvo, M., Medina-Valdés, L., Castillo-Secilla, J., Cuenca-Asensi, S., Martínez-Álvarez, A., & Villagrá, J. (2017). A review of the bayesian occupancy filter. *Sensors*, *17*(2), 344.

Savitch, W. (1970). Relationships between nondeterministic and deterministic tape complexities. *J Comput Syst Sci*, *4*(2), 177–192.

Steyer, S., Tanzmeister, G., & Wollherr, D. (2018). Grid-based environment estimation using evidential mapping and particle tracking. *IEEE Trans Intell Veh*, *3*(3), 384–396.

Sutner, K., & Maass, W. (1988). Motion planning among time dependent obstacles. *Acta Inform*, *26*(1-2), 93–122.

Vaněk, O., Bošanskỳ, B., Jakob, M., & Pěchouček, M. (2010). Transiting areas patrolled by a mobile adversary. In *Proc. CIG*, pp. 9–16.

Vaněk, O., Jakob, M., Lisỳ, V., Bošanskỳ, B., & Pěchouček, M. (2011). Iterative game-theoretic route selection for hostile area transit and patrolling. In *Proc. AAMAS*, pp. 1273–1274.

Yehoshua, R., Agmon, N., & Kaminka, G. (2016a). Multi-robot adversarial coverage. In *Proc. ECAI*, pp. 1493–1501.

Yehoshua, R., Agmon, N., & Kaminka, G. (2016b). Robotic adversarial coverage of known environments. *Inter J Robot Res*, *35*(12), 1419–1444.