Annals of Math and AI manuscript No.
(will be inserted by the editor)

Safe Navigation in Adversarial Environments

Ofri Keidar · Noa Agmon

the date of receipt and acceptance should be inserted later

Abstract

This work deals with the problem of navigation while avoiding detection by a mobile adversary, featuring adversarial modeling. In this problem, an evading agent is placed on a graph, where one or more nodes are defined as *safehouses*. The agent's goal is to find a path from its current location to a safehouse, while minimizing the probability of meeting a mobile adversarial agent at a node along its path (i.e., being captured). We examine several models of this problem, where each one has different assumptions on what the agents know about their opponent, all using a framework for computing node utility, introduced herein. Using risk attitudes for computing the utility values, their impact on the constructed strategies is analyzed both theoretically and empirically. Furthermore, we allow the agents to use information gained along their movement, in order to efficiently update their motion strategies on-the-fly. Theoretical and empirical analysis shows the importance of using this information and these on-the-fly strategy updates.

1 Introduction

The problem of path planning is one of the fundamental problems in the field of agents and robotics [29,36,39,43]. The goal in path planning is to find a sequence of world locations which allows the agent to arrive at its destination while optimizing some criteria, usually minimizing travel cost while avoiding obstacles.

In this work we examine the problem of navigating to a location while avoiding an adversary, that wishes to intercept the agent's movement towards its goal.

Ofri Keidar
Department of Computer Science
Bar-Ilan University, Israel.
E-mail: ofri.keidar@biu.ac.il

Noa Agmon
Department of Computer Science
Bar-Ilan University, Israel.
E-mail: agmon@cs.biu.ac.il

The adversarial presence, as well as topological characteristics of the environment, for example obscured or visible points, result in the fact that simply following the shortest path might not guarantee a safe journey because once the adversary detects the agent, it might infer its planned path and intercept it. Moreover, committing to a deterministic path (shortest path or other) may result in an increasing chance of interception by a knowledgeable adversary. Hence, in such case, stochastic movement could be a better option. Therefore, we introduce a new variant of traditional path planning problem: **SA**fe **N**avigation in Adversarial **E**nvironments (or SANE, in short). In this problem, we aim at planning a path for our agent (denoted as R), while avoiding being captured by a mobile adversarial agent (denoted as C). The agents travel along a graph, representing a map of the environment, where some nodes in this graph are defined as *safehouses*. The goal of R is to arrive at one of the safehouses without being intercepted by C on its way there (being *captured*).

The problem of traveling in an environment while avoiding threats has been studied from different perspectives [2, 12, 20, 32, 33, 47]. In the problem of Pursuit-Evasion [5, 15, 37, 41], two or more rival agents move around the environment (e.g., along the edges of a graph), until the pursuer moves to the evader's location. Most research in pursuit evasion focuses on aspects concerning topology of the graph, for example, on defining properties related to graph theory of the given graph, in order to characterize graphs where the pursuer is guaranteed to capture the evader or finding minimal number of pursuers required. In our problem, however, the evader R moves to a certain destination, and does not try only evade its pursuer indefinitely. Furthermore, R's path is planned based on the topology of the graph, its risk attitude, and the knowledge it has on C (and its understanding of C's knowledge).

We formally introduce the SANE problem and examine different variants of it, which differ in the level of knowledge the agents have on their opponents, and on the risk attitude they adopt. In our proposed solution to the SANE problem, we refer to two layers of knowledge. The first layer is an a-priori evaluation of the map nodes, expressed as the *utility values*, representing the safety of traveling through the node based on the topology of the graph. If the agents have no knowledge about their opponent's location, they base their movement decision only on these utility values. The second layer of knowledge is obtained by combining information related to the opponent's location. This information is received either as an input (the initial location) or when the agent moves along the map graph (the opponent's current location in the online model). The agents choose their next location throughout the game based on both initial utility of the nodes, as well as on the information they have (or deduce) on their opponent's location.

The framework, combined with the constructed strategies, is used to allow updates by the agents during execution, i.e., on-the-fly updates. Assuming the agents may gain information about their opponent while they are on their way to their target—either a safehouse or interception, by R and C (respectively)—this information is used to adjust the strategy efficiently.

Theoretical guarantees are proven for the proposed solutions for each variant, e.g., expected utility maximization, or equilibrium. We have fully implemented the algorithms, and show, by theoretical analysis and extensive experiments, that these updates significantly increase the chances of R to reach its destination safely.

We also examine the impact of the risk attitude of the agents on their chances of successfully achieving their goals, both theoretically and empirically.

This paper is organized as follows. In Section 2 we discuss research literature which addresses problems of capturing an evading mobile agent, and relate to known solution concepts and their incompatibility to SANE. Section 3 provides a formal definition to our problem. Section 4 presents the framework for computing utility values for the nodes of the map graph, values which will be used for constructing the motion strategies for both agents. Section 5 addresses various scenarios in the offline modeling of SANE, namely constructing strategies for each agent and providing guarantees. In Section 6 we allow on-the-fly strategy updates, hence, modify the motion strategies accordingly and analyze the contribution of the online strategy updates. Section 7 discusses additional aspects for SANE: termination, handling hyperactive agent behavior and referring to zero-knowledge players. In Section 8 we empirically evaluate our strategies for each problem modeling and conclude in Section 9.

2 Related Work

The problem of safe navigation is tied to various fundamental problems in the literature: path planning [29,36,39,43], pursuit-evasion [4,5,15,37,41], path planning with threats [38,47] and covert path planning [21,32,33,45,46]. In this section, we present research literature addressing similar problems and discuss the differences in the problem modeling.

Many authors have been dealing with problems where an agent has to evade another pursuing agent, e.g., [2–4, 9, 18, 20, 25]. *Pursuit-Evasion* is a game where one player, called the *evader*, must avoid *capture* by the other players, called *pursuers*, while all players move in some environment [31]. The term *capture* might mean that the evader is caught in a pursuer's line of sight [25], relate to a scenario where both the evader and a pursuer reside at the same location [9] or surround the evader [15]. Existing literature mainly focus on characterizing graphs with some graph-theoretic properties, such as ensuring that the evader will be captured eventually, determining the minimal number of pursuers required to capture the evader, or examining the class of graphs where a given number is the minimal number of pursuers required to capture the evader [15]. Furthermore, in these problems the evader usually moves randomly until caught. In our problem, the evader has a destination location (one or more) and acts with response to the pursuer's knowledge (strategically). Moreover, we incorporate adaptive strategies with the concept of pursuit-evasion (Section 6). In addition, our game is simultaneous with incomplete information (e.g., initial and current location of the opponent might be unknown).

Adler et al. [1] present a randomized pursuit-evasion game on a graph. While the pursuer (*hunter*) is restricted to moving on the edges, two models are examined for the evader (*rabbit*): one where it can move on the edges, while in the other it is allowed to jump to any node unrestrictedly. In all models, the players cannot see each other unless occupy the same node. In our problem, though, the evader is also restricted to the map graph. In addition, we examine several models of knowledge, e.g., knowing the initial location of the opponent (Section 5.3) or allowing partial observation throughout the game (Section 6).

Barret et al. [6] present an empirical study of ad-hoc teamwork in the pursuit-evasion domain. They evaluate a selection of algorithms used for generating on-line behaviors for a single ad-hoc team agent which collaborates with various types of teammates in the pursuit-evasion domain. The point of view of Barret et al. is different from ours in several aspects. First, they construct the behavior for the pursuers and address cooperation. However, we model the behavior of the evader[1]. Second, although they relate to strategies, their evader is not strategic (moves randomly), while in SANE both agents behave strategically. Moreover, they assume perfect sensing, while we address scenarios with either no visibility (Section 5) or with viewpoints (Section 6).

In the problem of covert path planning (or stealth-based path planning) a robot must navigate efficiently from one point to another without being seen by hostile entities [21,32,45]. While both covert path planning and our problem share the aspect of evading another entity, in the former static threats are assumed, and in this work the adversary poses dynamic threats. Örgen et al. [38] present a problem of path planning for a UAV in an environment with threats, seeking to minimize distance cost and risks. The UAV follows a deterministic path planned offline. Having stochastic motion for at least one agent can be beneficial, since it increases the opponent's uncertainty regarding the agent's location, therefore we explore such stochastic solutions, considering also time-dependent (not only constant) threats.

Foderaro et al. [19] address a problem where an agent needs to visit some points of interest in a known environment (these locations are known a-priori), while evading multiple adversarial agents. In their settings, each agent knows the location of each other. Therefore, the adversaries actually chase the evading agent. In contrary, we address the uncertainty arising when the agents are not aware of the exact location of each other.

Boidot et al. [11] consider a problem of an autonomous agent traveling through an environment where an adversary tries to set an ambush. Modeled as a flow problem, the traveler starts from a source area and wins if reaches the target area. However, the traveler loses if passes close enough to the location in which the adversary has set its ambush. Although this problem shares common properties with SANE, these problems relate to vastly different information models: Boidot et al. assume the locations of the source and sink are known to both agents but no sensing capabilities. We, however, tackle different assumptions on the knowledge of the starting locations of the agents (Section 5) and further examine the influence on the motion strategies when the agents incorporate information, gained at runtime, regarding their opponent's current location.

The SANE problem addresses scenarios where the agents choose their next move stochastically, based on current world state only. Uncertainty concerning opponent's location implies not knowing the exact world state. Hence, it seems as if our problem could have been modeled by a Partially Observable Markov Decision Process (POMDP) [44]. The variant where both agents use information gained along their movement (Section 6) associates with Interactive POMDP (I-POMDP) [23]. At a map graph node, an agent chooses its move according to a probability distribution over its neighbors. If such distributions would have been

[1] In order to prove theoretical guarantees for the evader's strategies, in each modeling of the SANE problem, we present the optimal strategy for the pursuer, as well.

considered as actions available at a world state, an action would have been choosing a neighboring node at a certain probability. Since the probability variable is continuous, it implies infinite number of possible actions at each state, making modeling our problem using any variant of MDP simply impossible.

Huang et al. [26] present a framework for solving a two-player capture-the-flag games, modeled as zero-sum differential games. Even though the authors consider capture-the-flag games as being able to examine automated solutions for adversarial games, there are some major differences between their problem modeling and solution approach to ours. In their solution to the two-player game, agents are allowed to know the past and present choices. In contrary, we cope with uncertainty regarding the opponent's initial or current locations (and sometimes both of them). Furthermore, they constructed deterministic strategies. However, we consider an adversary with equal computational abilities to our evading agent, thus the adversary can perform the same computations as the evader agent. Therefore, if the evader follows a deterministic strategy, once the adversary knows the location of the evader, it can intercept the evader easily. Another difference relies in the objectives of the agents. In capture-the-flag games, the agents need to attack and defend at the same time, while our agents need to attack only.

In reachability games, one player has to reach a target set of world states, while the other seeks to prevent the first one from succeeding [17]. In this modeling, a strategy returns a player's next move given a world state. Namely, a player makes a move given the locations of both players. However, we handle uncertainty regarding the opponent's location and if such information is acquired at runtime, the motion strategies are updated accordingly on-the-fly (Section 6).

Due to the strategic aspect of our problem, a Nash equilibrium could be used for constructing a strategy for each agent. Yet, finding a Nash equilibrium for mixed strategies is complex [16]. Uncertainty regarding adversary's current location implies a large number of possible actions, which increases problem size. Moreover, this game is repeated, but there is a different game at each time step (due to different neighboring nodes to which each agent can move). We will need to compute it to a certain horizon (i.e., game duration in time steps), which could be very large. Therefore, computing the stochastic motion strategies using mixed-strategy Nash equilibrium requires solving an unbounded sequence of complex sub-problems, which becomes infeasible.

In imperfect-information extensive-form games [42], the players might not be aware of the choices of their opponent, thus these games are capable of modeling simultaneous games, as well. The game is represented as a game tree, where nodes match choices of the players, edges are possible actions and the terminal game states are in the leaves. Computing equilibrium strategies requires solving a linear-program, which can be solved in time polynomial in the size of the game tree. However, as our game is of unbounded horizon (see Section 7.1), the maximum depth of corresponding game tree would be unbounded. A known solution is to set a user-defined bound on the game tree's depth and treat the lowest nodes as leaves, whose values are approximated by some heuristic function. Yet, we compute the motion strategies in time polynomial – specifically, quadratic – in the map graph size (Section 4.4), while the size of the bounded game tree might not guarantee a linear-program solved in time polynomial of the map graph size. In addition, when the agents receive information at runtime regarding their opponent's whereabouts (Section 6), modeling SANE as an imperfect-information extensive-form game

would require on-the-fly changes of the information sets along the game tree. In contrary, in our solution, node utility values are computed beforehand and are used for constructing the motion strategies. These utility values are used whether an agent is aware of its opponent's initial location or not, and enable efficient on-the-fly strategy updates in case new information is received at runtime.

A Stochastic game [30] is a repeated game where the players choose their action and the next game state is drawn from a probability distribution over the game states. In our problem, though, at least one player chooses its actions stochastically and given the chosen actions, the next game state is set deterministically. Even though the probability in which an agent chooses actions can be used to model the probability in which the next game state is set, stochastic games cannot model our problem: the probability distribution over the next game state is part of a stochastic game's definition, while in our problem, the probability distribution for choosing an action (which implies the next game state) is what we seek to compute. Therefore, for each model of the SANE problem presented herein, R's strategies are computed based on utility of the related game states or *configurations* (Section 4). These utility values are propagated throughout a graph (referred to as the *configuration graph*), from configurations matching terminal game states (either win or lose) to all other game states. Belief propagation (BP) is an algorithm for computing marginal probabilities in a graphic model [10]. Even though BP seems like a method for propagating values through a graph, for several reasons it cannot be applied in order to compute the configurations utility values. First, our utility values are not probabilities. Hence, the configuration graph is not a probabilistic graphic model, and its nodes are not variables of some distribution, rather possible game states. Second, in BP, values (*messages*) are passed around the graph between neighboring nodes by conditioning probabilities on all values that a node can receive from its neighbors. Thus, applying this method for propagating our utility values would enable using risk neutral attitude only (namely, taking an average on values received from the neighbors). Another BP version, called the max-sum algorithm, maximizes the marginals of every variable. However, this is not equivalent to taking the maximal (or symmetrically, the minimal) value among a node's neighbors in the configuration graph, as our risk-seeking function (or risk-averse, for minimal value). In addition, BP converges to the exact values in a graphic model with no cycles. Loopy BP is used when cycles are present, but its convergence is not always guaranteed. In contrary, our method is much simpler – we simply propagate values from the terminal configurations. In conclusion, BP is an algorithm for computing marginals, it is not designed for a mere propagation of values through some general graph while applying functions that are not specifically sums or products.

Security games address problems where a mobile agent (usually a patroller) tries to detect an adversarial intruder. The intruding agent is assumed to be able to observe the patrolling agent in order to devise its optimal strategy. This leads to a leader-follower solution concept [8]. In SANE, though, we consider a different modeling, which allows knowledge of the initial location only (in the offline model, Section 5) or observation at runtime, used for on-the-fly strategy updates (in the online model, Section 6). Basilico et al. [7] present a security game with the requirement of responding to signals raised by an alarm system. As in SANE, their work focuses on finding an optimal strategy for a patrolling agent, combined with information provided at runtime (by the alarm system). Despite the resemblance

to our problem, SANE cannot be modeled by this work. In this work of Basilico et al., the alarm is triggered if a target is under attack, thus, information is gained about an agent regardless of the patrolling agent's location. However, in SANE, an agent observes its opponent only if its current location is visible to the agent. That is, observation depends on the locations of both agents and not only on the penetrator's location. Moreover, we consider on-the-fly strategy updates (which are proven to improve the strategies).

Network security games are a variant of security games variants, which take place on graphs. Jain et al. [27] present a network-based security game, where a defender places security resources on the edges of the graph in order to protect against an adversary, which moves along the edges. The authors consider multiple static security resources, while we consider a single mobile resource (being our agent R). McCarthy et al. [35] address a model which differs than SANE in similar manners to [27]. In addition, we consider strategy adaptations at runtime, in contrary to these works. Zhang et al. [**?**] address the problem of scheduling defender resources in order to capture an evading attacker. Even though both SANE and their model consider dynamic threats, we handle a single defender resource (being the agent C) and allow the agents utilize their knowledge of their opponent's location in order to produce better motion strategies.

3 Problem Definition

The SANE (**SA**fe **N**avigation in Adversarial **E**nvironments) problem is formally defined as follows:
Given a graph $G = (V, E)$, representing a map of the environment (referred to as *map graph*), $V_G \subseteq V$ a set of goal nodes (*safehouses*) and two distinct initial positions of an agent R and an adversarial agent C, find a strategy that will maximize R's chances of reaching some node $v_g \in V_G$ without being captured by C. R is captured by C if both agents reside the same node (any node, including a safehouse $v_g \in V_G$). R wins if it reached a goal node $v_g \in V_G$ without being captured, while C wins if it captures R.

Note that the strategy may be deterministic or stochastic, and changes based on the knowledge the agents have on their opponent's strategy and location, and on the risk attitude adopted by the agents (discussed in details in Sections 5,6). We assume that C and R act simultaneously, and that both are capable of performing the same computations (i.e., each agent knows its opponent's strategy).

4 Framework for Computing Node Utilities

In this section we establish a framework for computing utility values for the graph nodes based on the graph topology, that will be used for the various models of the SANE problem (Sections 5,6). The utility of $v \in V$ expresses how *safe* it is for R if moves to v, i.e., how probable it is to evade capture and reach a goal node (i.e., win), as demonstrated in Figure 1. This value is derived by evaluating the game configurations (i.e., game states) where R resides at v. When there is no additional information about the location of the agents, their decisions are made based on this basic utility. We shall begin with introducing preliminary concepts.

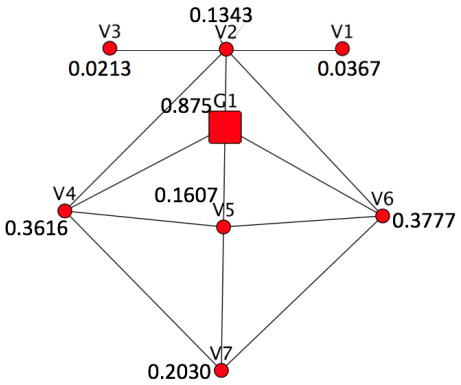


Fig. 1: Node utility values for each graph node (G_1 is a goal). Greater utility means node is safer *for* R

At time t, let $v_{\mathrm{R}}^t, v_{\mathrm{C}}^t \in V$ be the nodes where R and C are located in, respectively. We will omit t and refer to v_{R},v_{C} wherever possible.

Our goal is to find a sequence of nodes (or a policy) for arriving safely at a goal node. As a result, this path should avoid reaching certain nodes according to our position and adversary's estimated position at a given time. Since we consider also C's position, we refer to the *game configurations* rather than map locations: A *game configuration* $\langle v_{\mathrm{R}}, v_{\mathrm{C}} \rangle$ holds the locations of both R and C. A configuration is a *win configuration* if it is of the form $\langle v_g, v \rangle$, such that $v_g \in V_G$ and $v_g \neq v$. A configuration of the form $\langle v, v \rangle$, $v \in V$, is a *lose configuration*.

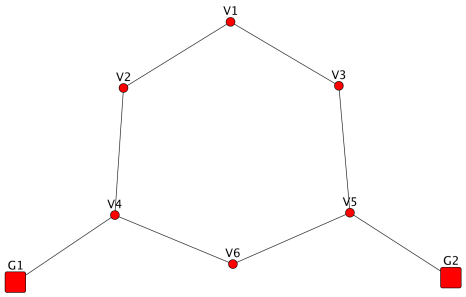
The *configuration graph* $G_{conf} = (V_{conf}, E_{conf})$ is an undirected graph where each node is a game configuration. Therefore, $|V_{conf}| = |V|^2$. For each $v_1, v_2 \in V$, E_{conf} contains an edge[2] $(\langle v_1, u_1 \rangle, \langle v_2, u_2 \rangle)$ if at least $(v_1, v_2) \in E$ or $(u_1, u_2) \in E$. A configuration is *terminal* if locations of R and C are the same (C has won), or if only R's location is a goal (R has won) or if it has no neighbors in the configuration graph G_{conf}[3]. Figure 2 demonstrates a map graph G and a portion of its configuration graph G_{conf} (entire graph not shown for clarity).

4.1 Estimating Safety of Game Configurations

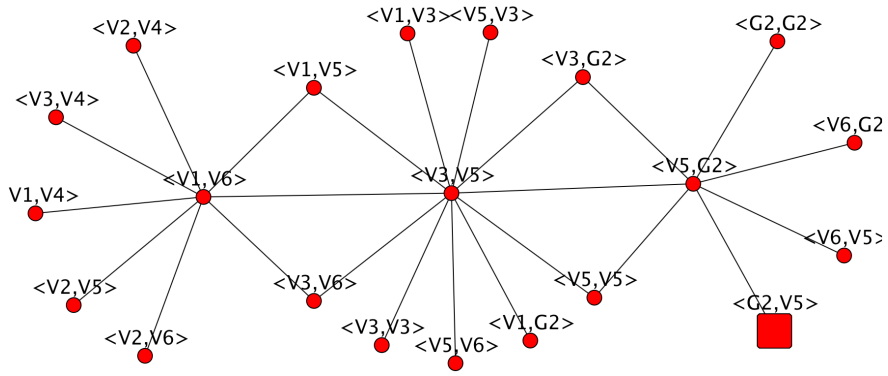
We define as follows a *utility* value for each game configuration $\mathcal{V} \in V_{conf}$, which expresses how probable it is for R to win if moves from $\mathcal{V}$. Terminal configurations are given a utility of 1 for a win configuration and 0 for a lose one. The utility of a configuration depends on the utility of its neighbors. As a consequence, in an undirected graph, it is necessary to determine the order of traversing through the configuration graph in order to avoid mutual dependencies. Since utility values are propagated from the terminal configurations, the non-terminal configurations can be ordered in ascending order of distance (in terms of number of edges, i.e., hops) from terminal configurations. Utility of a configuration is computed based

[2] Our motion strategies do not allow the agents to maintain their location in the next time step (Sections 5,6), while the configuration graph does. Section 7.2 explains why.

[3] Although it is impossible in connected undirected graphs, we would like to generalize our definitions to match directed graphs, as well



(a) *Graph of a world map, where* R *starts at* V_1 *and* C *at* V_6. G_1, G_2 *are goal locations (box-shaped nodes)*



(b) *A portion of the matching configuration graph (entire graph not displayed for clarity). Note that although* G_2 *is a goal location,* $\langle G_2, G_2 \rangle$ *is not a goal configuration because* C *captures* R*, while* $\langle G_2, V_5 \rangle$ *is a goal configuration (box-shaped node)*

Fig. 2: An example of a graph of a world map and its generated (portion of) configuration graph

only on its neighbors whose utility value had already been computed, therefore, such ordering ensures that the utility values are propagated in correct order from the terminal configurations to the other ones.

Hence, a prior step is to calculate for each configuration its distance from the terminal configurations. This distance is defined as minimal number of edges required to reach a terminal configuration (i.e., hops). Algorithm msBFS (Multiple Source BFS) calculates this distance by applying BFS Algorithm for a set of source nodes, instead of the common version with a single source node.

Algorithm CalcConfigsUtil receives the configuration graph G_{conf} and uses the result of Algorithm msBFS to traverse along the configuration graph G_{conf} in a breadth-first manner. Moving along G_{conf} in such a way finds which neighboring configurations had already been processed. Therefore, utility of a configuration $\mathcal{V} \in V_{conf}$ can be easily computed based on neighboring configurations that had been visited before. Various risk attitudes can be used: risk averse (utility of $\mathcal{V}$ is minimal utility among visited neighbors), risk neutral (average utility among neighbors). The risk attitude is an intrinsic factor of the agent, which defines the agent's behavior (or so to say, "personality"). In contrary, prior knowledge about the environment is an extrinsic factor. We have addressed these risk attitudes in

order to examine the influence of the agent's behavior on its strategy. However, in case prior knowledge about the environment is available, then it could definitely be used (e.g., a non-uniform probability distribution over the opponent's initial location), regardless of the risk attitude type.

Formally, the function $f_{cost}^{conf} : 2^{V_{conf}} \rightarrow [0,1]$ gets a set of visited neighboring configurations and returns the matching utility value. The utility value is computed according to the risk attitude being applied, as shown in Algorithm CalcConfigsUtil Line 13. Note that Algorithm CalcConfigsUtil does not construct a strategy, rather is the first step towards obtaining the utility values of the map graph nodes.

Algorithm 1 msBFS(G, V_{source})

Input:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$ input graph
$\mathbf{V_{source}}$ set of source nodes
1: initialize mapping d //$d[v]$ *maps* $v \in V$ *to its distance*
2: initialize mapping Π //$\Pi[v]$ *maps* $v \in V$ *to parent node on shortest path*
3: **for all** $s \in V_{source}$ **do**
4: $d[s] \leftarrow 0$
5: $\Pi[s] \leftarrow NULL$
6: **end for**
7: **for all** $v \in V \setminus V_{source}$ **do**
8: $d[v] \leftarrow \infty$
9: $\Pi[v] \leftarrow NULL$
10: **end for**
11: $V_0 \leftarrow V_{source}$ //V_i *holds all nodes* v *s.t.* $d[v] = i$
12: $V_\infty \leftarrow V \setminus V_{source}$ //V_∞ *holds all non-source nodes*
13: **for all** $i \leftarrow 0$ to $|V|$ **do**
14: **for all** $v \in V_i$ **do**
15: **for all** $u \in$ Neighbors(v) **do**
16: **if** $d[u] > d[v] + 1$ **then**
17: $V_{d[u]}$.Remove(u)
18: $V_{d[v]+1}$.Append(u)
19: $d[u] \leftarrow d[v] + 1$
20: $\Pi[u] \leftarrow v$
21: **end if**
22: **end for**
23: **end for**
24: **end for**
25: return d,Π

4.2 Estimating Safety of Nodes Using the Configuration Graph

Utility for a map graph node $v \in V$ is computed based on the utility of configurations of the form $\langle v, u \rangle$ ($u \in V$), i.e., configurations where R resides at v. Once all configurations have been given a utility value (applying Algorithm CalcConfigsUtil), the utility of each map graph node $v \in V$ is the average utility of the configurations where v is the location of R. In other words, the utility for a map graph node $v \in V$ is the average utility of all configurations $\mathcal{V} \in V_{conf}$, such that $\mathcal{V} = \langle v, u \rangle$, $u \in V$. Algorithm CalcNodesUtil computes utility values for the map graph nodes, given the utility values of the configurations.

Algorithm 2 CalcConfigsUtil($G_{conf}, T_{conf}, f_{cost}^{conf}$)

Input:
$\mathbf{G_{conf}} = (\mathbf{V_{conf}}, \mathbf{E_{conf}})$ configuration graph
$\mathbf{T_{conf}} \subseteq \mathbf{V_{conf}}$ set of the terminal configurations
$\mathbf{f_{cost}^{conf}}$ function to calculate configuration utility according to utility values of given configurations
$\mathbf{U}_{\mathrm{R}}$ maps each configuration to its utility value

1: for each terminal configuration $\mathcal{V} \in T_{conf}$, set $U_{\mathrm{R}}(\mathcal{V})$ as 1 if a winning configuration, or 0 otherwise
2: for each non terminal configuration $\mathcal{V} \in V_{conf} \backslash T_{conf}$, set $U_{\mathrm{R}}(\mathcal{V})$ as $NULL$
3: run msBFS(G_{conf}, T_{conf}), denote by d the mapping from configurations to distances
4: for each distance value i in d, initialize a set V_i of configurations with distance i $//V_0$ *is* T_{conf}
5: **for all** distance values $i \leftarrow 1$ to Max(d.Values) **do**
6: **for all** configurations $\mathcal{V} \in V_i$ at distance i from some terminal configuration **do**
7: initialize an empty set S
8: **for all** configuration $\mathcal{U}$ such that $(\mathcal{V},\mathcal{U}) \in E_{conf}$ **do**
9: **if** $U_{\mathrm{R}}(\mathcal{U})$ is not $NULL$ **then**
10: add $\mathcal{U}$ to S
11: **end if**
12: **end for**
13: $U_{\mathrm{R}}(\mathcal{V}) \leftarrow f_{cost}^{conf}(S)$
14: **end for**
15: **end for**

Computing node utility in this manner relates to a risk neutral type of player. Other risk attitudes can be applied (by modifying Algorithm CalcNodesUtil Line 3), e.g., risk averse (utility of a node v is the minimal utility of a configuration $\mathcal{V} = \langle v, u \rangle$). Formally, the utility value of a map node $v \in V$ is obtained by applying $f_{cost} : 2^{V_{conf}} \rightarrow [0, 1]$:

$$U_{\mathrm{R}}(v) = f_{cost}\left(\{\langle v, u\rangle | u \in V\}\right) \tag{1}$$

For graphs without uniform travel cost, we could use this framework while applying cost functions which take into account both the node utility and travel cost. That is, utilize both the *safety* term and *travel* term. Section 7.4 explains how.

Although we assume capture is possible only when R and C occupy the same node, our framework can easily support capture when the agents cross each other when moving towards their next node (i.e., capture on the edges). In that case, we shall assume the existence of a function $f_{cross} : V_{conf} \times V_{conf} \rightarrow 0, 1$. f_{cross} receives configurations $\mathcal{V}$, $\mathcal{U} \in V_{conf}$ and returns 1 if R is captured during the transition from $\mathcal{V}$ to $\mathcal{U}$, and 0 otherwise. Note that f_{cross} depends on the map graph layout, thus, can be considered as some *metadata* associated with the map graph and returns an answer in $O(1)$ time. Algorithm CalcConfigsUtil can be modified as follows. When iterating over the neighbors of a configuration $\mathcal{V}$ (Line 6), denote $\mathcal{V}$'s current neighbor as $\mathcal{U}$. If $f_{cross}(\mathcal{V},\mathcal{U}) = 1$, consider $\mathcal{U}$'s utility as 0 (whether $\mathcal{U}$ had already been visited or not) when computing $\mathcal{V}$'s utility. Namely, $\mathcal{U}$ is considered as a lose configuration only in the context of $\mathcal{V}$, i.e., indicating that the current game step leads to a loss.

Algorithm 3 CalcNodesUtil(G, G_{conf})

Input:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$ map graph
$\mathbf{G_{conf}} = (\mathbf{V_{conf}}, \mathbf{E_{conf}})$ configuration graph
1: **for all** $v \in V$ **do** *//for each map node*
2: $S_v \leftarrow \{\langle v, u\rangle \in V_{conf} | u \in V\}$
//initialize a set S_v of all configurations where v denotes location of R
3: set $v.utility$ as average utility of configurations in S_v
4: **end for**

4.3 Utility Values for C

C's objective is opposed to R's, i.e., a win configuration for R is a lose one for C and vice versa – as expected in a zero-sum game. Therefore, if a terminal configuration where R is captured is given a value of 1 and a terminal configuration where R resides at a goal node (and C at another node) is given a value 0, applying Algorithms CalcConfigsUtil,CalcNodesUtil yields an evaluation of the map graph nodes. In that case, greater utility values relate to nodes where it is more probable for C to capture R. Therefore, for any risk attitude of R – averse (take minimal value), neutral (take average value) and seeking (take maximal value) – C's utility values are defined to be opposed to those of R:

$$\begin{aligned} \forall \mathcal{V} \in V_{conf} : U_{\mathrm{C}}(\mathcal{V}) = 1 - U_{\mathrm{R}}(\mathcal{V}) \\ \forall v \in V : U_{\mathrm{C}}(v) = 1 - U_{\mathrm{R}}(v) \end{aligned} \tag{2}$$

Where U_{R},U_{C} are the utility values for R,C, respectively. This observation optimizes performance when computing strategies which require utility values of both agents (Sections 5.3, 5.4, 6). Unless stated otherwise, the term *utility values* refers to R's utilities.

4.4 Analysis of Utilities Computation

We shall first prove that Algorithm msBFS computes the correct distance between nodes of the input graph and the source nodes. The conclusion would be that Algorithm CalcConfigsUtil propagates the utility values from the terminal configurations in ascending order of distances from the terminal configurations. Then, running times of the algorithms involved in the computation of utility values will be analyzed.

Lemma 1 *Let $G = (V, E)$ be a graph. For each $u \in V$, if Algorithm* msBFS *sets $d[u]$ as $d[v] + 1$, for some $v \in V$ such that $(v, u) \in E$, then $d[u]$ will not change.*

Proof Let $u, v, v' \in V$ be nodes, such that $(u, v), (u, v') \in E$ and $d[v'] < d[v]$. Let us assume that $d[u]$ was set as $d[v] + 1$ at the i'th iteration and was then set as $d[v'] + 1$ at the j'th iteration. It must exist that $j > i$, otherwise it contradicts that $V_{d[v]}$ was accessed before $V_{d[v']}$ (Algorithm msBFS Line 14). Therefore, it follows that $d[v'] > d[v]$, which contradicts the assumption that $d[v'] < d[v]$.

Time complexity of Algorithm msBFS is linear in the input graph. The initialization steps (Lines 3-7) require iterating over all nodes, therefore take $O(|V|)$ time. Following V_i's definition, for each i, j such that $i \neq j$, $V_i \cap V_j = \emptyset$. Therefore, in Line 13 we iterate over all nodes, but in Line 14 each node is iterated once. For each node, all neighbors are inspected in Line 15, that is, all edges are iterated over. Hence, running time of the entire loop of Line 13 is $O(|V|+|E|)$. In conclusion, the total time complexity of Algorithm msBFS is $O(|V|+|E|)$.

Time complexity of Algorithm CalcConfigsUtil is linear in the size of the configuration graph. The initialization step requires iterating over all configurations (Lines 1-2), which takes $O(|V_{conf}|)$ time. Executing Algorithm msBFS (Line 3) takes $O(|V_{conf}|+|E_{conf}|)$. In Line 5, $|V_{conf}|-|T_{conf}| \leq |V_{conf}|$ configurations are iterated over. For each configuration, all its neighbors are inspected, therefore all edges are iterated over. Cost of applying $f_{cost}^{conf}(S)$ (Line 13) is $O(|S|)$ and for each $\mathcal{V} \in V_{conf}$, $|S| \leq |Neighbors(\mathcal{V})|$. Hence, the combined cost of all the times where $f_{cost}^{conf}(S)$ is applied is $O(|E_{conf}|)$. Therefore, the cost of the loop of Line 6 is $O(|V_{conf}|+|E_{conf}|)$. In conclusion, the total time complexity of Algorithm CalcConfigsUtil is $O(|V_{conf}|+|E_{conf}|)$, which is polynomial in the size of the input graph.

Time complexity of Algorithm CalcNodesUtil is linear in the configuration graph. Let $v \in V$ be a map graph node. The algorithm generates all configurations where R resides at v: $\langle v, u \rangle$, such that $u \in V$ (Line 2). Hence, $O(|V|)$ elements. Then, the algorithm computes the average utility value of these configurations, or takes either the minimal or maximal value (depends on the risk attitude applied). These computations are linear in the number of elements in S_v (Line 3). Therefore, $O(|V|)$ work is performed for v. This process is repeated for each map graph node (Line 1). Therefore, the total time complexity of Algorithm CalcNodesUtil is $O(|V|^2) = O(|V_{conf}|)$.

In conclusion, given a map graph $G = (V, E)$, the node and configuration utility values are obtained by applying Algorithm CalcConfigsUtil and then Algorithm CalcNodesUtil. This takes polynomial time in the input graph, specifically, $O(|V_{conf}|+|E_{conf}|)$.

We shall now discuss several models of our problem and construct a strategy for both agents, using the node utility values.

5 Offline Strategies

In the *offline* model the agents are not visible during the game, unless they occupy the same map node. An exception is made for the initial location: four cases are addressed herein, differ in which agent knows where its opponent starts. For each of these scenarios, we analyze the strategy adopted by each agent. Table 1 summarizes the results for each of the examined scenarios where both agents plan their motion strategy offline. These scenarios differ in the knowledge of each agent regarding the initial location of its opponent.

5.1 Unknown Opponent's Initial Location

This scenario assumes that no agent knows its opponent's initial location.

Does R know where C starts?			
		No	Yes
Does C know where R starts?	No	R moves deterministically, C stochastically. Both agents maximize their probability to win (Section 5.1)	R moves deterministically, ignoring the information regarding C's start. C moves stochastically (Section 5.2)
	Yes	Stochastic strategies. Expected payoff is maximized with risk-neutral for both node and configuration utilities (Section 5.4)	Stochastic strategies. Expected payoff is maximized with risk-neutral for both node and configuration utilities (Section 5.3)

Table 1: Summarized results for each of the examined scenarios in the Offline Strategies model

R***'s Strategy.*** Let $p = \langle v_1, \ldots, v_k \rangle$ be a path. Define *path utility* $U(p)$ as follows:

$$U(p) = \sum_{i=1}^{k} U(v_i) \tag{3}$$

Where $U(v)$ is the utility value of $v \in V$. R plans a path which maximizes the path utility to a goal node $v_g \in V_G$. Such path is obtained as follows:

1. Convert the undirected map graph $G=(V,E)$ to a directed graph $G'=(V',E')$: $V'=V$, $E'=\{(v_1,v_2),(v_2,v_1)|(v_1,v_2) \in E\}$
2. For each $(v_1,v_2) \in E'$, define the weight $U_{max} - U(v_2)$ (U_{max} is maximum utility value among the map nodes)
3. Find the shortest path (in terms of edge weights) from R's initial location to any goal node $v_g \in V_G$, using a shortest-path algorithm (e.g., Dijkstra's)

C***'s Strategy.*** C is assumed to be capable of computing node utilities as R. Thus, given R's initial location, C can compute the path R would have taken. Since C has no indication of where R starts, it is preferable for C to move stochastically while maximizing the probability of intercepting R (presented formally in Algorithms InterceptDetOpp, PlanInterceptingPath):

1. For each possible starting point for R $v \in V \setminus (V_G \cup \{v_C^0\})$ (v_C^0 is C's initial location), plan R's path (according to R's strategy)
2. For each of these paths, plan a path for intercepting R
3. For each map node, count number of intercepting paths that pass through it
4. At each node, choose a neighbor randomly with bias towards neighbors with a greater counter value (i.e., towards nodes associated with more intercepting paths, Algorithm InterceptDetOpp Line 10)

While an intercepting path relies on a specific time and location for the interception to take place, C's stochastic movement does not guarantee that C will follow a certain interception path from start to end. However, C might have to wait at the interception point $k>0$ time steps. For an intercepting path p, denote C's waiting time at p's end as $t_{wait}(p)$. Let v be the i'th node along p ($1 \leq i \leq |p|$). Therefore, $t_{wait}(p)+i+1$ is the latest time step that C can arrive at v and intercept R if C follows p (Algorithm InterceptDetOpp Lines 7,9).

The following definition formalizes our criteria for an optimized strategy for R.

Definition 1 Let SecureTravel be the following prioritized optimization criteria regarding a path p in the map graph G:

Algorithm 4 PLANINTERCEPTINGPATH$(G, p_{\mathrm{R}}, d, \Pi)$

Input:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$ map graph
$\mathbf{p_R}$ R's optimal path from its assumed initial location
$\mathbf{d}$ $d[v]$ maps $v \in V$ to its distance from C's initial location
$\mathbf{\Pi}$ $\Pi[v]$ maps $v \in V$ to parent node on shortest path from C's initial location
for all $i \leftarrow |p_{\mathrm{R}}|$ to 1 **do** *//find latest node* C *can reach before* R
  if $d[p_{\mathrm{R}}[i]]$ is smaller than the number of steps required for R to reach $p_{\mathrm{R}}[i]$ **then** *//this number of steps is i*
    use Π for backtracking the path from $d[p_{\mathrm{R}}[i]]$, denote this path as p_{intcpt} *//the interception point was found, reconstruct the path*
    reverse p_{intcpt}
    $t_{wait} \leftarrow d[p_{\mathrm{R}}[i]] - i$ *//compute* C*'s waiting time at the interception point*
    return t_{wait}, p_{intcpt} *//return the waiting time and intercepting path*
  end if
end for
return $NULL$, $NULL$ *//could not find an interception point*

Algorithm 5 INTERCEPTDETOPP$(G, V_G, v_{\mathrm{C}}^0)$

Input:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$ map graph
$\mathbf{V_G}$ set of goal nodes
$\mathbf{v_C^0}$ C's initial location
initialize a mapping Arr_{times} from each node to its list of arrival times
run MSBFS$(G, \{v_{\mathrm{C}}^0\})$, denote by d the mapping from nodes to distances
for all $v \in V \setminus (V_G \cup \{v_{\mathrm{C}}^0\})$ **do** *//for each possible* R*'s start*
  plan R's optimal path p_{R}
  $t_{wait}(p_{intcpt})$, $v_{intcpt} \leftarrow$ PLANINTERCEPTINGPATH(G, p_{R}, d) *//plan* C*'s intercepting path p_{intcpt} and compute* C*'s waiting time until interception: $t_{wait}(p_{intcpt})$*
  $t_{wait}(p_{intcpt})$ is $NULL$, continue to the next iteration
  for each node v along p_{intcpt}:
  $Arr_{times}.at(v).insert(t_{wait}(p_{intcpt}) + i_v - 1)$
  //set latest arrival time: $i_v - 1$ is v's index in p_{intcpt}
end for
At runtime:
for each neighbor u of v_{C}, remove all arrival times from $Arr_{times}.at(u)$ that are smaller than current time step
//discard intercepting paths that are no longer relevant
choose a neighbor u of v_{C} in transition probability:
$P_{v_{\mathrm{C}}}[u] = \frac{|Arr_{times}.at(u)|}{\sum_{v' \in N(v_{\mathrm{C}})} |Arr_{times}.at(v')|}$

1. Maximize the sum of the node utility values along p.
2. Minimize the number of hops along p.

We deal with an offline scenario where we know nothing about our opponent's location - neither its initial location nor its current location throughout the game (and so is the adversary). Therefore, any node we reach can be treated as flipping a coin - R is captured or not. Thus, R's objective should be reaching a goal node while minimizing the number of these "coin flips" (since any further flip raises the chance to be captured), and in each of the coins we have to flip, the probability of success should be maximized. Minimizing the number of the coin flips is having the fewest number of nodes along the path, as obtained from Corollary 1. As follows from Lemma 2, the utility of a map node grows with the probability that R

avoids capture if goes through that node. Hence, higher probability of success in each coin flip is tied to higher sum of utility values of the nodes along the path.

The following Lemma will be used for proving the likelihood of the safety of R's strategy in Theorem 1, based on the SecureTravel criteria.

Lemma 2 *Assume* R *is risk neutral. Let* $v_1, v_2 \in V$ *be nodes neighboring to* v_R *such that* $U(v_1) > U(v_2)$*. Then,* R*'s expected probability for not being captured at* v_1 *is greater than at* v_2*.*

Proof R is risk neutral[4]. Then $U(v_1) > U(v_2)$ implies:

$$\sum_{v_i \in V} \frac{U(\langle v_1, v_i \rangle)}{|V|} > \sum_{v_i \in V} \frac{U(\langle v_2, v_i \rangle)}{|V|} \tag{4}$$

That is, the average utility of the game states where R resides at v_1 is greater than the average utility where R resides at v_2. The better a game state is for R, the greater its utility value is. Thus, v_1 is associated with better game states than v_2. Therefore, R's expected chance for a safe game state is greater if resides at v_1, hence, R is more probable to avoid being captured at v_1 than at v_2.

Corollary 1 *Let* $v_1, v_2 \in V$ *be nodes such that* $U(v_1) = U(v_2)$*, thus* R *has the same probability to be captured at* v_1 *or at* v_2*. Therefore, if* p_1 *and* p_2 *are paths which lead* R *to a goal node, such that the utility along each node of* p_1 *is the same as along* p_2*,* R *should follow the path with the fewer hops.*

Theorem 1 *In the offline model where none of the agents know the initial location of its opponent,* R*'s proposed strategy based on the* SecureTravel *criteria is likely to guarantee safest navigation.*

Proof Let $G'=(V', E')$ be the weighted directed graph constructed as mentioned above and let $p = \langle v_1, \ldots, v_k \rangle$ be a path. Recall that $U(v) \in [0, 1]$ for each $v \in V$. Let $w'(p)$ be p's weight in G' and let $U(p)$ be p's path utility.

$$\begin{aligned} w'(p) &= \sum_{i=1}^{k} (1 - U(v_i)) = 1 - U(v_1) + 1 - U(v_2) + ... + 1 - U(v_k) = \\ &= k - U(p) \end{aligned}$$

Therefore, the shortest path in G' minimizes $k - U(p)$, where k is the number of nodes along the shortest path. First, since Dijkstra's algorithm returns the shortest path, this path will not be longer than necessary – satisfying requirement 1 of SecureTravel. Second, in order to minimize $k - U(p)$, $U(p)$ must be the maximal among the paths with k nodes (and that terminate at some goal node, of course), thus requirement 2 of SecureTravel is met.

In our settings, finding the shortest path in G' (i.e., after the transition from node utilities to directed edge weights) does not correlate to finding the longest path in the map graph G, rather relates to satisfying SecureTravel criteria. We

[4] If R would have been risk averse, then a map node's utility would not relate to the expected probability for winning, rather to the worst associated game state.

shall now emphasize the difference:
As explained in the proof of Theorem 1, R's path p minimizes $k - U(p)$, where k is the number of nodes along p. For example, consider a path with 3 nodes with path utility 2.9 and a path with 25 nodes with path utility 20. While the first path minimizes $k - U(p)$, the longer path is the second one. However, as claimed above, the first path is indeed preferable. Therefore R's strategy is not intended for finding the longest path in the map graph G.

Conjecture 1 Without any further knowledge, following Algorithm InterceptDetOpp will maximize C's probability to capture R.

We believe C's proposed strategy is optimal, yet its optimality proof is left for future work. The following is our suggested justification for this heuristic:
Suppose C's proposed strategy does not maximize C's probability to capture R (i.e., win). Therefore, there exists an optimal strategy P^* which maximizes C's win probability. Namely, when C resides at a node $v \in V$ at time t, $P_v^{*(t)}$ is a distribution over v's neighbors, such that if C chooses its next move according to $P_v^{*(t)}$, C maximizes its probability to capture R. Let $u \in V$ be v neighboring node such that $P_v^{*(t)}[u]$ is greater than any other neighbor of v. As a consequence, C's probability to capture R is greater at u than at any other neighbors of v. Thus, most of C's interception options go through u. This is how C's proposed strategy behaves: choose a neighboring node stochastically, biased towards nodes with more interception options. Hence, P^* and C's proposed strategy act the same.

Theorem 1 explains why R benefits the most if follows its proposed strategy, based on the knowledge available to the agents, although regardless of C's strategy. Conjecture 1 claims that if R follows a deterministic strategy which is known to C, then C should not deviate from its proposed strategy. Thus, the combination of Theorem 1 and Conjecture 1 yields that in the scenario of Section 5.1, R should follow its proposed strategy, and given this choice, C would not deviate from its proposed strategy. Hence, a Nash equilibrium is indeed obtained.

Corollary 2 *If both* R,C *follow their suggested strategies, a Nash equilibrium is obtained.*

Time complexity of computing R's strategy is $O(|E|\log|V|)$. Generating the directed graph G' requires a traversal over the entire map graph G and $O(1)$ for each edge (creating the two directed edges), thus $O(|E|+|V|)$. Constructing the weight function over the edge set E' of G' takes $O(|E'|)$, but since $|E'|= 2\cdot|E|$, then $O(|E|)$. Applying Dijkstra's algorithm on G' takes $O(|E'|\log|V'|)$ but since $V = V'$, then $O(|E|\log|V|)$. Therefore, total running time complexity is $O(|E|\log|V|)$.

Time complexity of the offline preprocessing computations of C's strategy is $O\left(|V|(|E|\log|V|)\right)$. In each iteration of the Algorithm InterceptDetOpp (Line 3) C simulates R's strategy (Line 4). Given a possible path for R, C plans a suitable intercepting path (Line 5). This is obtained by first computing the distance (in terms of hops) from each map graph node to C's initial location (applying BFS Algorithm once). Then, C finds the first node on R's assumed path that is closer to C than to R's assumed start (Line 5), thus traversing over a list of $O(|V|)$ nodes. Note that the distance from R's assumed start to any node on its path is the node's index on the path. Once such node is found, C's path is constructed by backtracking Π (the mapping from a node to its parent on the shortest path)

obtained from Algorithm msBFS. The construction of an intercepting path is followed by another traversal over this path with $O(1)$ work for each node (Line 7). Thus, each iteration takes $O(|E|\log|V|) + O(|V|) \in O(|E|\log)|V|$ time and the entire loop takes $O(|V|+|E|) + O\left(|V|(|E|\log|V|)\right) \in O\left(|V|(|E|\log|V|)\right)$ time (there are $O(|V|)$ possible start locations for R).

Time complexity of the runtime computations of C's strategy is $\mathbf{O}(|\mathbf{V}|+|\mathbf{E}|)$ for the entire execution. At each time C has to choose its next move, the arrival times which are smaller than the current time step are removed (Line 9). If the arrival times are stored in a sorted manner, then the arrival times list can be iterated from start until reaching the first element that is greater or equal to the current time step. That is, each time another chunk of the arrival times list is removed. The list is at most $|V|$ elements long, because R's longest simple path is of length $|V|-1$. Hence, if combining the number of removed elements in all iterations, at most $|V|$ elements will be removed. Once the irrelevant arrival times had been removed, the transition probability to each neighbor is computed. The denominator in the transition probability expression in Line 10 can be computed once, and be used for each neighbor. The denominator is obtained in $O(|N(v_{\mathrm{C}})|)$ work, where $N(v_{\mathrm{C}})$ is the set of C's neighbors. Therefore, throughout the entire execution, all edges will be processed. Hence, $O(|V|+|E|)$ work for the entire execution.

5.2 Evader Knows Where Pursuer Starts

This scenario assumes that R knows where C starts, but C has no knowledge regarding R's initial location.

R's Strategy. Similarly to Section 5.1, R plans a path which maximizes its utility. Since C's initial location, v_{C}^0, is now known, at time t R can ignore configurations $\langle v, u\rangle$, $v, u \in V$ where u is more than t hops away from v_{C}^0. Namely, R can ignore game states that are not reachable at the moment. Due to the dependency of the configuration space subset on the time step t, the edge weights of the directed graph G' (Section 5.1) are time-dependent as well, since node utility values are drawn from configuration utilities (Section 4.2). Therefore, planning the shortest path (i.e., utility-maximizing path) is now an instance of a path planning with dynamic weights problem, solved by Algorithm MaxTimeDependUtil. (Note that the table D can be replaced with 2 line vectors. D is used for simplicity). $U_t(i, v)$ (Line 6) is v's utility at time i, based on reachable configurations at this time only:

$$U_{\mathrm{R},t}(i, v) = f_{cost}(\{\langle v, u\rangle | d(v, u) \leq i\}) \tag{5}$$

Where $d(v, u)$ is the distance in terms of hops from v to u, and f_{cost} is the risk attitude applied in order to obtain the node utility value.

C's Strategy. C's strategy from Section 5.1 can cope with any deterministic motion strategy for R. Therefore, C can apply Algorithm InterceptDetOpp.

Empirical Evaluation. We have evaluated R's winning rate when both agents plan their strategy offline, yet only R knows its opponent's initial location (see results in Figure 3). Even though C's location is not known at runtime, R was expected to take advantage of knowing C's initial location while C does not know R's. However, Figure 3 shows otherwise. R's winning rate is much lower if it incorporates the knowledge of C's initial location into its strategy. Namely, R should have

Algorithm 6 MaxTimeDependUtil$(G, U_t, v_{\mathrm{R}}^0, v_{\mathrm{C}}^0)$

Input:
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$ map graph
$\mathbf{U_t(i, v)}$ node utility function, considers relevant configurations at time i
$\mathbf{v_R^0}$ R's initial location
$\mathbf{v_C^0}$ C's initial location
initialize a table D of size $|V|\times|V|$, default values: $-\infty$
//$D[i][v]$ is optimal cost to $v\in V$ at time i
initialize a table Π of size $|V|\times|V|$, default values: $NULL$
//$\Pi[i][v]$ is v's parent on shortest path from v_{R}^0 at time i
for each neighbor v of v_{R}^0: $D[0][v]\leftarrow U_t(0, v)$, $\Pi[0][v]\leftarrow v_{\mathrm{R}}^0$
for all $i \leftarrow 1$ to $|V|-1$ **do** *//possible time steps along a simple path*
  for all $v \in V$ **do**
    $D[i][v]\leftarrow \max_{u\in V\setminus V_G}\{D[i-1][u] + U_t(i, v)|(u, v) \in E\}$
    $\Pi[i][v]\leftarrow \arg\max_{u\in V\setminus V_G}\{D[i-1][u] + U_t(i, v)|(u, v) \in E\}$
    //set path maximal utility to v for current time obtained from some non-goal neighbor u
  end for
end for
backtrack to v_{R}^0 from $v_g \in V_G$ with maximal D value

a symmetrical knowledge to C's, regarding the initial location. We have further inspected the settings of this scenario, and concluded that R plans its path based on an unknowledgeable adversary C, while C plans its path trying to intercept the knowledgeable R. That is, R knows that its opponent C does not know R's initial location. Thus, R plans its path under the assumption that its opponent does not know R's initial location. However, C is aware that its initial location is widely known. Therefore, when C plans the path R would have taken given some possible initial location, this path is planned under the assumption that C's initial location is indeed known to R. Namely, while R assumes an opponent which lacks knowledge, C assumes an opponent which takes advantage of the available knowledge (i.e., C's initial location). Hence, while R plays against the same opponent, C takes advantage of R's knowledge. Thus, in this scenario, R follows the same strategy as in Section 5.1 and consequently, C follows its strategy of Section 5.1, as well.

5.3 Mutually Known Initial Locations

This scenario assumes that both agents know each other's initial locations. Since C is capable of performing the same computations as R, and it knows R's initial location, then if R moves deterministically, C can intercept R. Hence, R should move stochastically. Because R moves stochastically, C cannot apply Algorithm InterceptDetOpp and has no reason to plan a deterministic path either.

Denote the stochastic motion strategies of R, C as $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$, respectively. $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$ are defined as follows.

R***'s Strategy.*** Denote R's location at time t as v_{R}^t. At each time step t, R computes the *expected utility* value of its neighboring nodes, then chooses a neighbor stochastically, with transition probabilities biased towards greater expected node utilities. Let $v_i \in V$ be a neighbor of v_{R}^t. The transition probability from v_{R}^t to v_i, $\mathcal{S}_{\mathrm{R},v_{\mathrm{R}}}^{(t)}[v_i]$, is obtained as follows:

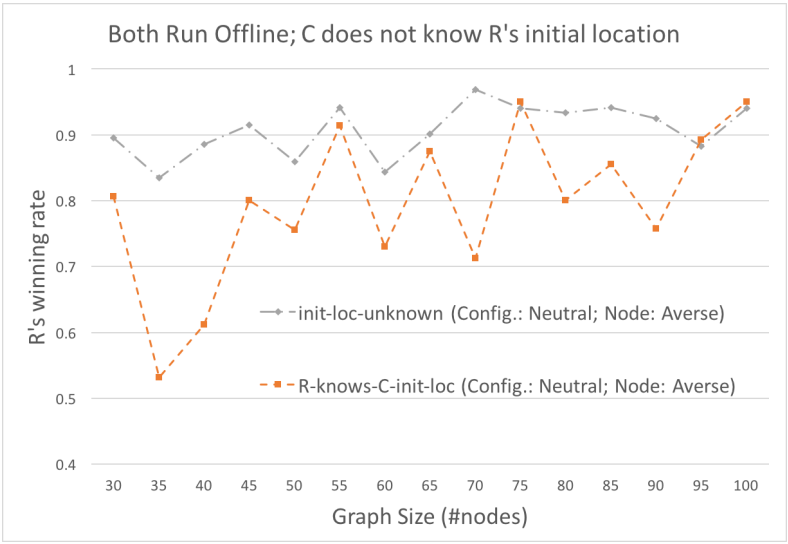


Fig. 3: R*'s winning rate when both* R,C *compute their strategies offline.* R*'s performance when knows* C*'s initial location is compared to when* R *does not know* C*'s initial location*

$$\mathcal{S}^{(t)}_{\mathrm{R},v_\mathrm{R}}[v_i] = \frac{EU^{(t+1)}_\mathrm{R}(v_i)}{\sum_{v_j:(v,v_j)\in E} EU^{(t+1)}_\mathrm{R}(v_j)} \tag{6}$$

Where $(v_\mathrm{R}, v_i)\in E$. The expected node utilities at time t are computed as follows:

$$EU^{(t)}_\mathrm{R}(v) = f_{cost}\left(\left\{\hat{P}^{(t)}_\mathrm{C}[v_i]\cdot U_\mathrm{R}(\langle v, v_i\rangle)|v_i \in V\right\}\right) \tag{7}$$

Where $\hat{P}^{(t)}_\mathrm{C}$ is a probability distribution over C's location at time t. As in Equation 1, f_{cost} denotes the risk attitude (e.g., risk neutral or risk averse). However, instead of taking the configuration utility values as is, the utility value of a configuration $\langle v, v_i\rangle$ is multiplied by the probability that C resides at v_i at the next time step. Namely, more influence is given to the more probable game states. Note that R's transition probabilities at time t are derived from its expected utility at time $t+1$, since R is biased towards the neighboring node which is expected to be safer at the next time step.

$\hat{P}^{(t)}_\mathrm{C}$ is initialized as $\hat{P}^{(0)}_\mathrm{C}[v^0_\mathrm{C}] = 1, \hat{P}^{(0)}_\mathrm{C}[v] = 0$ for $v \neq v^0_\mathrm{C}$. For the following time step, this distribution is computed by applying C's stochastic strategy $\mathcal{S}_\mathrm{C}$ at each node (i.e., forming a transition matrix for time t, which is multiplied by $\hat{P}^{(t)}_\mathrm{C}$). This strategy is obtained symmetrically to Equations 6,7, due to the common knowledge of initial locations. Note, however, that when R simulates $\mathcal{S}_\mathrm{C}$ for updating $\hat{P}^{(t)}_\mathrm{C}$, the computations of $\hat{P}^{(t)}_\mathrm{R}$ are based on R's node utilities (rather than the *expected* utilities). Otherwise, a mutual-dependency would occur.

C***'s Strategy.*** C follows a symmetric strategy to R's.

Time complexity of obtaining transition probabilities of an agent is $O(\max deg \cdot |V|)$**,** where $\max deg$ is maximal degree in the map graph. Computing the expected utility value of some node $v \in V$ at time t (Equation 7) requires

$O(1)$ work for each map node $v_i \in V$, hence a total of $O(|V|)$. At time t, if an agent resides at $v \in V$, when computing its transition probabilities to its neighbors as in Equation 6, the denominator can be computed once and be used for any v_i. Computing the denominator requires the expected utility of v's neighbors, thus $O(|V|)$ for any of v's neighbors. Once the denominator is obtained, the transition probability from v to some neighbor v_i is computed in $O(1)$. Therefore, obtaining the transition probabilities from v at time t takes $O(deg(v) \cdot |V|)$, where $deg(v)$ is v's degree. Hence, bounded by $O(\max deg \cdot |V|)$.
Note that when an agent simulates its adversary's stochastic strategy, no new information is required for obtaining the estimated probability distribution over its location (e.g., when R updates $\hat{P}_{\mathrm{C}}^{(t)}$). Therefore, these computations can be performed offline for future time steps, thus reducing the computations performed at runtime.

Theorem 2 *If the players apply risk neutral as configurations and map nodes utility function when computing utility values, and follow the stochastic strategies $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{R}}$ (as described in Equation 6), then at each time step* R,C *maximize their expected payoff (Equation 8).*

Proof At time t, let v_{R},$v_{\mathrm{C}} \in V$ be the locations of R,C, respectively. R's expected payoff for this sub-game at time t is as follows:

$$EP(\mathsf{R}) = \sum_{v_i:(v_{\mathrm{R}},v_i)\in E} \mathcal{S}^{(t)}_{\mathrm{R},v_{\mathrm{R}}}[v_i] \cdot \left(\sum_{v_j:(v_{\mathrm{C}},v_j)\in E} \mathcal{S}^{(t)}_{\mathrm{C},v_{\mathrm{C}}}[v_j] \cdot U_{\mathrm{R}}(\langle v_i, v_j \rangle) \right) \tag{8}$$

As a risk-neutral player, R is biased towards neighboring nodes with greater expected node utility values (Equations 6,7). Lemma 2 has shown that nodes with greater utility values have a greater chance to be associated with safer game states. Therefore, if $\mathcal{S}^{(t)}_{\mathrm{R},v_{\mathrm{R}}}[v_i] > \mathcal{S}^{(t)}_{\mathrm{R},v_{\mathrm{R}}}[v_{i'}]$ $((v_{\mathrm{R}}, v_i),(v_{\mathrm{R}}, v_{i'}) \in E)$, then for all $u \in V$:

$$\begin{aligned} &\Pr\left(U_{\mathrm{R}}\left(\langle v_i, u\rangle\right) \geq U_{\mathrm{R}}\left(\langle v_{i'}, u\rangle\right) \right) > \\ &\Pr\left(U_{\mathrm{R}}\left(\langle v_i, u\rangle\right) < U_{\mathrm{R}}\left(\langle v_{i'}, u\rangle\right) \right) \end{aligned} \tag{9}$$

Hence, if R follows the stochastic strategy $\mathcal{S}_{\mathrm{R}}$, then R maximizes its expected payoff for the sub-game at time t (Equation 8). Note that this applies even if at time t the game state is not completely visible (i.e., at least one agent might not observe its opponent), because, as claimed above, $\mathcal{S}_{\mathrm{R}}$ maximizes the probability for a better game state at the next time step regardless of the opponent's choice. Proving for C is symmetric.

Lemma 3 *Let $f_{cost} : 2^{V_{conf}} \rightarrow [0,1]$ be a utility function and let $\tilde{f}_{cost}$ be a utility function which multiplies each configuration utility by $\frac{1}{|V|}$.*
Denote: $U(v) = f_{cost}\left(\{\langle v, u\rangle | u \in V\}\right)$, $\tilde{U}(v) = \tilde{f}_{cost}\left(\{\langle v, u\rangle | u \in V\}\right)$.
Then, $\forall v \in V$, $U(v) = \frac{1}{|V|} \cdot \tilde{U}(v)$.

Proof Let us examine $U(v)$ for each risk attitude:
Averse: $U(v)$=$\min_{u\in V} \left\{U_{conf}(\langle v, u\rangle)\right\}$
Neutral: $U(v) = \sum_{u\in V} \frac{1}{|V|} U_{conf}(\langle v, u\rangle)$

Compare to $\tilde{U}(v)$:
Averse: $\tilde{U}(v)=\min_{u\in V}\left\{\frac{1}{|V|}U_{conf}(\langle v,u\rangle)\right\}=\frac{1}{|V|}\min_{u\in V}\left\{U_{conf}(\langle v,u\rangle)\right\}=\frac{1}{|V|}U(v)$.
Neutral: $\tilde{U}(v)=\sum_{u\in V}\left(\frac{1}{|V|}\right)^2 U_{conf}(\langle v,u\rangle)=\frac{1}{|V|}\sum_{u\in V}\frac{1}{|V|}U_{conf}(\langle v,u\rangle)=\frac{1}{|V|}U(v)$.

Theorem 3 *At time t, a node's safety evaluation based on its current expected utility value, is more accurate than its safety evaluation based on the time-independent utility value.*

Proof The entropy regarding the estimation of C's location is maximum when $\hat{P}_{\mathrm{C}}^{(t)}$ is uniform. That is, R has no indication of C's current location, hence, the expected node utility values evaluate the nodes' safety least accurately. According to Lemma 3, the time-independent utility values are equal up to a factor of $\frac{1}{|V|}$ to the expected utility values if $\hat{P}_{\mathrm{C}}^{(t)}$ is a uniform distribution. Since the transition probability to a neighboring node v is v's utility normalized by all other neighbors' utilities, R would be biased to the same nodes. That is, since the same factor is applied to all of the nodes, the scale is maintained. Thus, the stochastic strategies would be the same with the time-independent utility values or when $\hat{P}_{\mathrm{C}}^{(t)}$ is uniform. Hence, at time t, a node's safety is more accurate using the expected node utilities rather than the time-independent utility values.

5.4 Pursuer Knows Where The Evader Starts

This scenario assumes that C knows where R starts, but R has no knowledge regarding C's initial location. R and C move stochastically, for the same reasons of Section 5.3.

R***'s Strategy.*** R follows the same strategy $\mathcal{S}_{\mathrm{R}}$ proposed in Section 5.3, yet in this scenario, $\hat{P}_{\mathrm{C}}^{(0)}$ is initialized as: $\hat{P}_{\mathrm{C}}^{(0)}[v_{\mathrm{R}}^0]=0$, $\hat{P}_{\mathrm{C}}^{(0)}[v_g]=0$ for $v_g\in V_G$ and $\hat{P}_{\mathrm{C}}^{(0)}[v]=\frac{1}{|V|-|V_G\cup\{v_{\mathrm{R}}^0\}|}$ for $v\notin(V_G\cup\{v_{\mathrm{R}}^0\})$. That is, although R does not know C's initial location, R knows C does not start at a goal node nor at R's start. Such initialization of $\hat{P}_{\mathrm{C}}^{(0)}$ takes advantage of R's knowledge, as limited as might be. When simulating C's strategy, $\hat{P}_{\mathrm{R}}^{(0)}$ is initialized with 1 for v_{R}^0 and 0 otherwise, because R is aware that its initial location is known.

C***'s Strategy.*** C follows the same strategy $\mathcal{S}_{\mathrm{C}}$ proposed in Section 5.3. In order to match R's strategy simulation to R's actual strategy, $\hat{P}_{\mathrm{C}}^{(0)}$ is initialized as R does.

This scenario is modeled the same as in Section 5.3, except for a slight difference in the initialization. Note that the proofs for guarantees of Section 5.3 (Theorems 2, 3) do not rely on the manner in which the model is initialized. Therefore, these theoretical guarantees apply in this scenario, as well.

6 Strategy Updates On-the-Fly

In the offline problem variant (Section 5), no new information was gained while moving around the map graph. Strategies were computed offline, such that they aim to reduce the probability that C captures R, based on the map graph's topology. However, relying solely on graph topology, i.e., offline planning, means no reaction to new information gained while moving around the map graph. In this section,

we discuss a model where some nodes can be observed by the other nodes, which can be considered as *viewpoints*. Such nodes provide information (or, *knowledge*) regarding an agent, e.g., tracks that the agent has left behind, or perhaps whether the agent currently resides at the node. When the agents follow the stochastic strategies (based on node expected utilities) – $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$ – computed as described in Section 5.3, they can use these viewpoints in order to acquire information concerning their opponent's location or visited nodes, and update their strategies accordingly (each agent and its own objective).

For example, Figure 4 shows a scenario where R resides at node v_2. v_2 is a viewpoint which provides information associated with itself and nodes v_3,v_5 (dashed edges). If C resides at v_3, R can update the strategy at v_2 such that it reaches v_5 at the next turn, and increase its chance to reach any of the goal nodes (G_1,G_2) safely. Less trivially, if C resides at v_4 and had left a track at v_3 at previous turn. R does not know C's exact current location, but it is less likely to be v_5 (agents move a single hop each turn, and v_5 is 2 hops away from v_4). Therefore, R can update the strategy at v_2 such that it favors moving towards v_5.

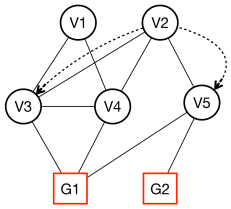


Fig. 4: Map graph edges are straight lines, while dashed lines denote nodes that are visible from other nodes

We shall now formally model the knowledge (i.e., information regarding an agent's whereabouts) and how it is acquired. Then, we present a binary knowledge model (where an agent is either currently observed or not), followed by a method to update the strategies efficiently online.

6.1 Knowledge Modeling and Definitions

Let $E_{visible} \subseteq V \times V$ be a set of directed *visibility edges*, i.e., $(v,u) \in E_{visible}$ means that u is visible from v. Let $d^{vis}_{out}(v)$ be the out-degree in terms of visibility edges of a node $v \in V$. This implies that a viewpoint is any node $v \in V$ with $d^{vis}_{out}(v) > 0$. Each time an agent visits a map graph node $v \in V$, it gains some *knowledge*. This information is associated with v and also with other nodes which are visible from v. Formally, we denote by $\mathbb{K}$ the set of possible knowledge values (e.g., time stamp of last visit). Therefore, $\mathcal{K} : V \to 2^{V \times \mathbb{K}}$ is defined as follows:

$$\mathcal{K}(v) = \{(u,k) | (v,u) \in E_{visible}, k \in \mathbb{K}\} \cup \left\{(v,k')\right\}, k' \in \mathbb{K} \tag{10}$$

The *information utility* associated with a node $v \in V$ expresses the extent of the map graph that v reveals. That is, the information utility of a node $v \in V$

matches the a-priori value of the information gained at v. Various risk attitudes can be applied:
Risk averse:

$$\mathcal{IU}(v) = 1 - \frac{d_{in}^{vis}(v)}{\max d_{in}^{vis}} \tag{11}$$

$d_{in}^{vis}(v)$ is v's in-degree and $\max d_{in}^{vis}$ is maximal in-degree in terms of visibility edges. This function favors nodes where an agent is less likely to be observed.
Risk seeking:

$$\mathcal{IU}(v) = \frac{d_{out}^{vis}(v)}{\max d_{out}^{vis}} \tag{12}$$

$\max d_{out}^{vis}$ is maximal out-degree in visibility edges. This function favors nodes where an agent observes a larger portion of the graph.
Risk neutral:

$$\mathcal{IU}(v) = 0 \tag{13}$$

An agent uses information gained at viewpoints, but does not seek new information nor avoids locations where it can be observed.

Our discussion will focus on a binary knowledge model, where $\mathbb{K} = \{0, 1^*\}$. 1^* means an agent is currently observed, 0 means agent is not observed.

6.2 On-the-Fly Strategy Update Scheme

Because both agents receive information from the visibility edges, the agents can be observed not only at their initial location, rather also along the game. Therefore, both agents obey the stochastic strategies of Section 5.3. If R observes C's initial location, then $\hat{P}_{\mathrm{C}}^{(0)}$ is 1 for C's initial location and 0 for all other nodes (as in Section 5.3). Otherwise, $\hat{P}_{\mathrm{C}}^{(0)}$ is initialized as in Section 5.4. C operates similarly. We shall now discuss the required modifications of the strategies $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$ from Section 5.3 for this online model.

$\hat{P}_{\mathrm{R}}^{(t)}$ for R's simulation of C's strategy is initialized according to R's knowledge of C's perspective. Namely, if C's initial location is known, R knows whether C observes R or not. Hence, $\hat{P}_{\mathrm{R}}^{(t)}$ would be updated accordingly. In case R does not initially observe C, the best R can do is initialize $\hat{P}_{\mathrm{R}}^{(t)}$ as if R is not observed by C (because there is no indication otherwise). Initializations performed by C are symmetric.

Assume R's location at time t, v_{R}^t, is a viewpoint. If C is observed at some node v, R sets $\hat{P}_{\mathrm{C}}^{(t)}$ with 1 for v and 0 otherwise. However, if C is not observed at any of the visible nodes, then C is known not to reside at these nodes. Hence, $\hat{P}_{\mathrm{C}}^{(t)}$ is set with 0 for each observed node and then the former probabilities associated with these nodes are split proportionally to the remaining nodes. Formally:
Denote the set of nodes being visible from v_{R}^t as $V_{visible}(v_{\mathrm{R}}^t)$:

$$V_{visible}(v_{\mathrm{R}}^t) = \left\{ u \in V | (v_{\mathrm{R}}^t, u) \in E_{visible} \right\} \tag{14}$$

Denote $\hat{P}_{\mathrm{C}}^{(t)}$ before being updated as $\tilde{P}_{\mathrm{C}}^{(t)}$, then:

$$\forall v \in V_{visible}(v_{\mathrm{R}}^t) : \hat{P}_{\mathrm{C}}^{(t)}[v] = 0$$
$$\forall v \in V \setminus V_{visible}(v_{\mathrm{R}}^t) : \hat{P}_{\mathrm{C}}^{(t)}[v] = \tilde{P}_{\mathrm{C}}^{(t)}[v] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} \quad (15)$$

Note that if $\sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] = 0$, then the distribution over C's location is not changed. Namely, if C was not expected to reside at any of $v \in V_{visible}(v_{\mathrm{R}}^t)$, then the visibility edges do not provide new information, thus $\hat{P}_{\mathrm{C}}^{(t)}[v]$ remains the same.

Once $\hat{P}_{\mathrm{C}}^{(t)}$ has been updated according to the visible nodes, $\hat{P}_{\mathrm{C}}^{(t)}$ is multiplied by C's transition matrix in order to obtain $\hat{P}_{\mathrm{C}}^{(t+1)}$. In addition, the transition probabilities are adapted:
If v_{R}^t is a viewpoint, then some information is received, i.e., C either resides at the observed nodes or is known not to reside there. Therefore, R's estimation of C's current location has not been proven wrong and can be used to re-weight R's transition probabilities:

$$\tilde{\mathcal{S}}_{\mathrm{R},v}^{(t)}[v_i] = \frac{\mathcal{S}_{\mathrm{R},v}^{(t)}[v_i] \cdot \left(1 - \hat{P}_{\mathrm{C}}^{(t+1)}[v_i]\right)}{\sum_{v_j:(v,v_j)\in E} \mathcal{S}_{\mathrm{R},v}^{(t)}[v_j] \cdot \left(1 - \hat{P}_{\mathrm{C}}^{(t+1)}[v_j]\right)} \quad (16)$$

Where $\mathcal{S}_{\mathrm{R},v}^{(t)}[v_i]$ is computed as Equation 6. That is, although $EU_{\mathrm{R}}^{(t)}(v_i)$ reflects v_i's utility while increasing influence of more probable game configurations, it is necessary to address the probability that C will move to v_i at the next turn: $\hat{P}_{\mathrm{C}}^{(t+1)}[v_i]$. C operates symmetrically at a viewpoint, except that in Equation 16, C multiplies the transition probabilities by $\hat{P}_{\mathrm{R}}^{(t+1)}[v_i]$. That is, C favors nodes where it is more likely to capture R, while R aims at avoiding C.

If v_{R}^t is not a viewpoint, then no new information regarding C is received. $\hat{P}_{\mathrm{C}}^{(t+1)}$ is obtained by multiplying $\hat{P}_{\mathrm{C}}^{(t)}$ by C's transition matrix. R's transition probabilities are re-weighted based on information utility of v_{R}^t's neighbors. That is, if R's estimation of C's current location cannot be supported, R reacts based on its attitude towards new information (if risk neutral $\mathcal{IU}$, no change is done):

$$\tilde{\mathcal{S}}_{\mathrm{R},v}^{(t)}[v_i] = \frac{\mathcal{S}_{\mathrm{R},v}^{(t)}[v_i] \cdot \mathcal{IU}(v_i)}{\sum_{v_j:(v,v_j)\in E} \mathcal{S}_{\mathrm{R},v}^{(t)}[v_j] \cdot \mathcal{IU}(v_j)} \quad (17)$$

C operates symmetrically at a node which is not a viewpoint.

We should note that although both agents update their common-knowledge baseline strategies, an agent is not aware of the current state of its opponent's motion strategies.

Updating the transition probability distribution of some node is linear in its degree. Let $v \in V$ be the node where some agent resides at time t. The update scheme, as given in Equations 16 and 17, is applied once the transition probabilities from v to any of its neighbors (Equation 6) had been computed. The denominator of Equation 16 is computed once and used for each of v's neighbors. Given the transition probabilities, the denominator is computed with $O(1)$ work

for each neighbor, thus $O(deg(v))$. Hence, applying the update scheme of Equation 16 is linear in v's degree. As for Equation 17, its denominator can also be computed once and be used for all of v's neighbors. In case of risk averse or risk seeking $\mathcal{IU}$ (Equations 11,12), the maximal visibility degree (either in-degree or out-degree) can be computed offline, as well as the visibility degree of any other node. Thus, the $\mathcal{IU}$ of any node can be obtained in $O(1)$ at runtime. Therefore, given the transition probabilities, the denominator of Equation 17 is computed with $O(1)$ for each of v's neighbors and applying this update scheme is linear in v's degree, as well.

6.3 Contribution of On-the-Fly Updates

We shall now prove that relying on information received online for updating the strategies $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$ at runtime, provides a more accurate estimation of C's current location, thus yields better strategies.

Definition 2 Let P,Q be two discrete probability distributions on the countable set Ω. Their Hellinger Distance [22] is defined as follows:

$$H(P,Q) = \sqrt{\sum_{\omega\in\Omega}\left(\sqrt{P(\omega)} - \sqrt{Q(\omega)}\right)^2} \tag{18}$$

Theorem 4 *Denote the actual probability distribution over C's location at time t as $P_{\mathrm{C}}^{(t)}$ (i.e., $P_{\mathrm{C}}^{(t)}$ is 1 for C's location and 0 for any other node). $\tilde{P}_{\mathrm{C}}^{(t)}$ denotes R's estimation for $P_{\mathrm{C}}^{(t)}$ before updating it with the information received from $V_{visible}(v_{\mathrm{R}}^t)$, while $\hat{P}_{\mathrm{C}}^{(t)}$ denotes $\tilde{P}_{\mathrm{C}}^{(t)}$ after being updated. Incorporating the information received from the visibility edges for updating $\hat{P}_{\mathrm{C}}^{(t)}$, improves the accuracy of this estimation. Namely, $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) \geq H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$.*

For an easier reading, the following proof does not specify all solution steps. The full proof can be found in Section 10 (appendix).

Proof As stated in Section 6.2, if $\sum_{v\in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] = 0$ then $\hat{P}_{\mathrm{C}}^{(t)} = \tilde{P}_{\mathrm{C}}^{(t)}$. Hence, $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) = H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$.

Now, assume $\sum_{v\in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] > 0$ (a sum of probabilities is either 0 or positive). If C has been observed ($v_{\mathrm{C}}^t \in V_{visible}(v_{\mathrm{R}}^t)$), then $\hat{P}_{\mathrm{C}}^{(t)} = P_{\mathrm{C}}^{(t)}$.

Otherwise, $v_{\mathrm{C}}^t \notin V_{visible}(v_{\mathrm{R}}^t)$. Let us compute the Hellinger distance between the actual distribution over C's location to R' estimated distribution prior being updated, $H\left(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right)$:

$$H\left(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) =$$

$$\sum_{v\in V_{visible}(v_{\mathrm{R}}^t)} \left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v]} - \sqrt{P_{\mathrm{C}}^{(t)}[v]}\right)^2 + \left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - \sqrt{P_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]}\right)^2 \Bigg)^{1/2}$$

C resides at v_{C}^t, hence $\forall v \in V \setminus \{v_{\mathrm{C}}^t\}$, $P_{\mathrm{C}}^t[v] = 0$, $P_{\mathrm{C}}^t[v_{\mathrm{C}}^t] = 1$. Therefore:

$$H\left(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) =$$
$$\left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \tilde{P}_{\mathrm{C}}^{(t)} - 2\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} + 1\right)^{1/2}$$

Now, we shall compute the Hellinger distance between the actual distribution over C's location to R' estimated distribution after being updated, $H\left(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right)$:

$$H\left(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) =$$
$$\left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)} \hat{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \hat{P}_{\mathrm{C}}^{(t)}[v] + \left(\sqrt{\hat{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - 1\right)^2\right)^{1/2}$$

C resides at v_{C}^t and $v_{\mathrm{C}}^t \notin V_{visible}(v_{\mathrm{R}}^t)$, hence:

$$\forall v \in V_{visible}(v_{\mathrm{R}}^t) : \hat{P}_{\mathrm{C}}^{(t)}[v] = 0$$
$$\forall v \in V \setminus V_{visible}(v_{\mathrm{R}}^t) : \hat{P}_{\mathrm{C}}^{(t)}[v] = \tilde{P}_{\mathrm{C}}^{(t)}[v] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \hat{P}_{\mathrm{C}}^{(t)}[v]}$$

Therefore, we obtain:

$$H\left(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) =$$
$$\left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]\right.$$
$$\left. -2\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} + 1\right)^{1/2}$$

If $\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] = 0$ then $\hat{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] = \tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]$ and $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) = H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$. Otherwise:

$$\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] > 0 \Rightarrow \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} > 0$$
$$\Rightarrow \sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} > \sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]}$$
$$\Rightarrow H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) > H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$$

In conclusion, we obtain that $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) \geq H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$, hence R's estimated probability distribution over C's location is closer to $P_{\mathrm{C}}^{(t)}$ after incorporating the information gained from the visibility edges.

Proving Theorem 4 for C is similar.

Now, we prove that if the agents apply risk averse as configuration and node utility functions, the game converges into a Markov perfect equilibrium [34]. This is proven in the following steps: the probability distribution over the adversary's location converges (Lemma 4), thus the expected node utility values converge (Lemma 5). As a consequence, R's transition probabilities converge, as well (Lemma 6). These claims can be proven symmetrically for C, so we obtain that if omitting the strategy update at the current time step, the motion strategies of both agents ($\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$) converge into time-independent strategies. Therefore, applying risk averse attitude yields a Nash Equilibrium at each time step (Lemma 7). If the agents seek to view their opponent, it follows that the game converges into a Markov perfect equilibrium (Theorem 5).

Lemma 4 *For all $\epsilon>0$, there exists a time step $t^*>0$, such that for each $t>t^*$:* $H\left(\hat{P}_{\mathrm{C}}^{(t)}, \hat{P}_{\mathrm{C}}^{(t^*)}\right)<\epsilon$.

Proof Figure 5 relates to the simulated models when R runs online (it would be symmetrical for the online model of C). When R runs online, R simulates C's strategy, denoted as C′. C′ is an offline stochastic model, based on C's expected utility values (i.e., C's strategy in Section 5.3). In such strategy, an agent computes its expected utilities based on a time-independent (i.e., basic) model of its opponent. That is, C′ simulates a time-independent offline stochastic model of R, denoted as R″.

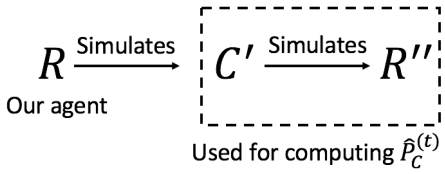


Fig. 5: R's online model simulates C – an offline stochastic model based on C's expected utility values – denoted as C′. C′ simulates R – an offline stochastic model based on time-independent utility values – denoted as R″.

R″, which is used for updating $\hat{P}_{\mathrm{C}}^{(t)}$, can be represented as a Markov chain, M_{R}, where each map node is a world state. Hence, M_{R}'s state space is finite. The undirected map graph is assumed to be connected, making M_{R} irreducible. An irreducible Markov chain with a finite state space has a stationary distribution [24].

Let $\hat{P}_{\mathrm{R}''}^{(t)}$ be the probability distribution over the location of R″ at time t. Let $\tilde{t}$ be a time step where $H(\hat{P}_{\mathrm{R}''}^{(\tilde{t})}, \hat{P}_{\mathrm{R}''}^{(\tilde{t})} \cdot M_{\mathrm{R}})$ is small enough, i.e., at time $\tilde{t}$, M_{R} is close enough to convergence. It follows that for $t>\tilde{t}$, the expected node utility values of C′ at time t are close enough to those at $\tilde{t}$, thus the transition probabilities of C′ converge. Therefore, as for R″, $\hat{P}_{\mathrm{C}}$ becomes stationary. Namely, there exists $t^*\geq\tilde{t}$ such that for each $t>t^*$: $H\left(\hat{P}_{\mathrm{C}}^{(t)}, \hat{P}_{\mathrm{C}}^{(t^*)}\right)< \epsilon$.

Lemma 5 *For all $\epsilon>0$, there exists a time step $t^*>0$, such that for each $t>t^*$: $\forall v\in V$,* $|EU^{(t)}(v)-EU^{(t^*)}(v)|<\epsilon$.

Proof At time t, for each node $v \in V$, R's expected utility value is obtained as in Equation 7. Following Lemma 4, let $\tilde{t}>0$ denote the time step where $\hat{P}_{\mathrm{C}}$'s change over time is negligible. Since a configuration utility is time-independent, there exists $t^* \geq \tilde{t}$ such that for each $t>t^*$: $\forall v \in V$, $|EU^{(t)}(v) - EU^{(t^*)}(v)| < \epsilon$.

Lemma 6 *At time t, let v denote* R*'s location and let $\mathcal{S}^{(t)}_{\mathrm{R},v}[v_i]$ denote* R*'s transition probability from v to v_i, $(v, v_i) \in E$. For all $\epsilon>0$, there exists a time step $t^*>0$, such that for each $t>t^*$: $|\mathcal{S}^{(t)}_{\mathrm{R},v}[v_i] - \mathcal{S}^{(t^*)}_{\mathrm{R},v}[v_i]| < \epsilon$.*

Proof At time t, for each node v_i ($(v, v_i) \in E$), recall from Equation 6 R's transition probability from v to v_i. Following Lemma 5, let $\tilde{t}>0$ denote the time step where EU's change over time is negligible. Therefore, there exists $t^* \geq \tilde{t}$ such that for each $t>t^*$: $\forall v \in V$, $|\mathcal{S}^{(t)}_{\mathrm{R},v}[v_i] - \mathcal{S}^{(t^*)}_{\mathrm{R},v}[v_i]| < \epsilon$.

Lemma 7 *If the players apply risk averse as configuration and map nodes utility function when computing utility values and follow their stochastic strategies $\mathcal{S}_{\mathrm{R}}$ and $\mathcal{S}_{\mathrm{C}}$, then at each time step, their choices are in Nash Equilibrium.*

Proof At time t, let $v_{\mathrm{R}}, v_{\mathrm{C}} \in V$ be the locations of R,C, respectively. R's expected payoff for this sub-game at time t is as in Equation 8. The expected payoff for C is obtained symmetrically. R is biased towards nodes with greater expected node utility. Therefore, if R applies risk averse as a configuration and node utility function, then at each time step R favors neighboring nodes whose worst game state is better than the other neighbors' worst state. That is, R follows a maxmin strategy, and symmetrically, C follows a minmax strategy. Given that our game is a zero-sum game with two players, it follows from the Minimax Theorem [40] that the strategies of R,C are in equilibrium at time t.

The following Theorem establishes that in a fully-visible map graph, under certain conditions the game converges to a stochastic game with a Markov perfect equilibrium. As shown in [34], in a stochastic game where the actions are observable to each player and the players' strategies depend on the game state only (i.e., Markovian), then if an equilibrium is reached in each sub-game, then the stochastic game is in Markov perfect equilibrium.

Theorem 5 *Assume a fully-visible map graph (i.e., a visibility edge between each pair of nodes) and that* R,C*'s utility values were obtained by applying risk averse as configuration and map nodes utility function. Then, if* R,C *follow their stochastic strategies $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$ at each time step, the game converges to a stochastic game with a Markov perfect equilibrium.*

Proof Due to the full visibility assumption, R,C move simultaneously and know the current game state (i.e., current configuration). The next game state is determined stochastically, according to the probabilistic transitions played by R,C. Therefore, the game becomes a stochastic game. According to Lemma 6, there exists a time step t^*_{R}, such that for each $t>t^*_{\mathrm{R}}$, R's stochastic strategy $\mathcal{S}_{\mathrm{R}}$ becomes time-independent[5]. Lemma 6 can be applied symmetrically for C, yielding the

[5] While an agent follows the strategy associated with its current node, it would be the same strategy in recurrent visits after time t^*.

existence of a time step t^*_{C} for which C's stochastic strategy $\mathcal{S}_{\mathrm{C}}$ becomes time-independent. Define t^*=max $\{t^*_{\mathrm{R}}, t^*_{\mathrm{C}}\}$. Therefore, for each time step $t>t^*$, both R,C follow time-independent stochastic strategies. Namely, the strategies depend only on the game state. According to Lemma 7, each sub-game is in equilibrium. Therefore, our game converges to a stochastic game which is in Markov perfect equilibrium.

Re-weighting transition probabilities based on risk seeking information utility means being biased towards neighboring nodes which provide visibility information regarding a larger portion of the map graph. That is, approximating full visibility for the next time step. As followed from Theorem 5, full visibility (with risk averse configuration/node utility functions) enables convergence towards a Markov perfect equilibrium. The following corollary summarizes this conclusion:

Corollary 3 *Assume $E_{visible} \subset V \times V$ and that* R,C*'s utility values were obtained by applying risk averse as configuration and map nodes utility function. Then, re-weighting transition probabilities $\mathcal{S}_{\mathrm{R}}$, $\mathcal{S}_{\mathrm{C}}$ based on risk seeking information utility, helps converging towards a Markov perfect equilibrium.*

7 Additional Results for SANE

In this section we examine further aspects of the SANE problem. Sections 7.1 and 7.2 provide more aspects to the analysis of the problem variants which were discussed above, while Section 7.3 addresses a new variant.

7.1 Game Termination

All of the experiments we have conducted – for both offline and online models – have reached a terminal state, i.e., either R or C have won within a finite number of moves (the empirical analysis is detailed in Section 8). In this section we present a *theoretical* analysis of whether the game is guaranteed to terminate.

Theorem 6 *Any instance of the offline strategies scenarios (Sections 5.1-5.4) is a game with a finite expected duration.*

Proof If R plans a deterministic path (Section 5.1, 5.2), the game terminates either after R has completed its finite path or if C has succeeded capturing R on its way. Hence, the game is over after a finite number of time steps.
If R follows a stochastic strategy based on the *expected* utility values, R's strategy is time-dependent (Section 5.3, 5.4). However, as stated in Lemma 6, there exists a time step t^*_{R}, such that for each $t>t^*_{\mathrm{R}}$, R's stochastic strategy $\mathcal{S}_{\mathrm{R}}$ becomes time-independent. In that case, R follows a fixed stochastic strategy, namely, R's strategy converges to a Markov chain in a finite number of time steps. In a finite Markov chain, the expected hitting time between any pair of nodes is finite [14]. Therefore, the expected hitting time between R's initial location and some goal node $v_g \in V_G$ is also finite. Hence, the expected number of time steps until R wins is finite. Moreover, the game might terminate earlier in case C captures R.

In the on-the-fly strategy updates model (Section 6), game termination analysis becomes much more difficult. The motion strategy can change constantly, based on the information gained at the viewpoints. Incorporating Lemma 6 yields that at each time step, the expected number of time steps until R wins is finite. However, as the strategy keeps changing at each turn, this number of time steps is associated with each time step separately. Namely, denote T_t as the expected number of time steps until R wins at time step t. At time $t+1$, the expected number of time steps until R wins would be T_{t+1}, but there is no guarantee that $T_{t+1} = T_t - 1$. Hence, at each time step, the expected number of time steps required until R wins does not necessarily decrease monotonically. However, so far we did not manage to formulate an example for a game which does not end. Therefore, proving whether the game in the online model is guaranteed to terminate or not, is left as an open research question.

7.2 Hyperactive Property

The strategies proposed in the various problem modelings choose the next move from the neighboring nodes only (Sections 5,6). We refer to such strategies as *hyperactive*, since they do not allow an agent to stay in its place at the next time step. Hyperactive strategies are preferred, because otherwise, the game could remain in the same configuration for an unbounded number of time steps, making the game degenerated.

Suppose the configuration graph G_{conf} would had been constructed as in Section 4, with the addition of a self loop for each node. As a consequence, the game is allowed to stay in the same configuration at the next time step. Algorithm CalcConfigsUtil computes the utility value for each configuration $\mathcal{V} \in V_{configig}$, based on the values of the visited neighbors of $\mathcal{V}$. If assuming self loops in G_{conf}, $\mathcal{V}$ is considered one of its neighbors. However, when Algorithm CalcConfigsUtil computes the utility value of $\mathcal{V}$, this configuration obviously does not have a utility value yet. Therefore, $\mathcal{V}$'s self loop is omitted when computing $\mathcal{V}$'s utility value. Hence, the outcome is computing the same utility values that Algorithm CalcConfigsUtil yields for G_{conf} without the self loops – which is the configuration graph that our framework uses. This way our framework for computing utility values retains its generality: the framework computes utility values for the most general case (where agents are allowed to stay in their place), while the motion strategies can be hyperactive or not.

7.3 Zero-Knowledge Agents

The major part of our discussion relates to agents with symmetric computational capabilities and have prior knowledge about the starting location or motion strategy of their opponent. Nonetheless, a comprehensive study of the SANE problem would include also zero knowledge agents. In this modeling, the agents are not aware of their opponent's starting location and do not use information gained at viewpoints. Two scenarios are addressed: one where C follows a deterministic motion pattern and another where C follows a probabilistic motion pattern. Namely,

C is a non-strategic agent. In both scenarios, R does not know C's actual motion pattern nor its type (deterministic or probabilistic).

As explained in section 5.1 (Theorem 1), R maximizes its winning probability if follows the path to a goal node with maximum *path utility* (Equation 3). However, the current model features a different adversary: while Section 5.1 assumes an adversary who is capable of computing node utility values and plan a path accordingly, we now face an adversary that does not maximize its probability of intercepting R, rather merely follows some unknowledgeable motion strategy. Thus, R might be able to follow the shortest path towards its closest goal node. In this section, we compare two behaviors for R when playing against the unknowledgeable C: a strategic and an opportunistic behavior.

Figure 6a compares R's winning rate if follows the path with maximal path utility or the shortest path, when C follows a random walk. Figure 6b compares the same strategies for R, when C follows a deterministic patrol strategy: a cyclic tour along the graph, generated by a depth-first traversal. If C is a zero-knowledge agent, i.e., follows either a random walk or a deterministic patrol strategy (Figures 6a,6b), T-Test does not show any significant advantage for maximizing the utility along R's path (p-value was 0.0533).

The following theorem provides a theoretical interpretation of the empirical results of Figures 6a,6b.

Theorem 7 *If* C *follows a random walk,* R *maximizes its probability to win by following the shortest path to some goal node.*

Proof The random walk of C can be viewed as a Markov chain, such that each node of the map graph is a state and the transition probabilities are given as follows:

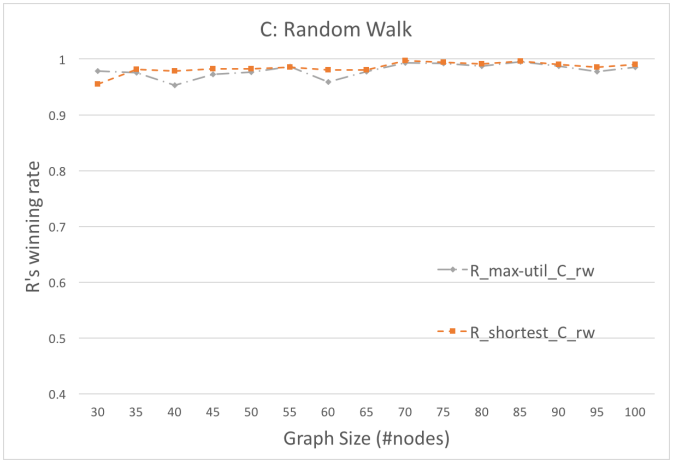
$$M(v_i, v_j) = \begin{cases} \frac{1}{deg(v_i)} & (v_i, v_j) \in E \\ 0 & otherwise \end{cases} \tag{19}$$

Where M is the transition matrix and $deg(v)$ is the degree of $v \in V$. Because the map graph is connected, this Markov chain is irreducible. Due to the finite number of nodes, the state space is finite. Therefore, M is recurrent [13]. In a recurrent Markov chain, if starting from state $v \in V$, the probability of returning to v after some finite number of moves is 1 [28]. Due to the Markov property, when a state v is visited, it is as if the Markov chain starts again from v. Hence, the probability to return to v once again remains 1. Thus, each state $v \in V$ will be visited an infinite number of times regardless of the initial state. Namely, if C follows a random walk, the number of C's visits in each node increases with time, no matter where it has started. Thus, if R minimizes the number of nodes it visits (i.e., follows the shortest path), R minimizes the probability to encounter C. Hence, R maximizes its probability to win.

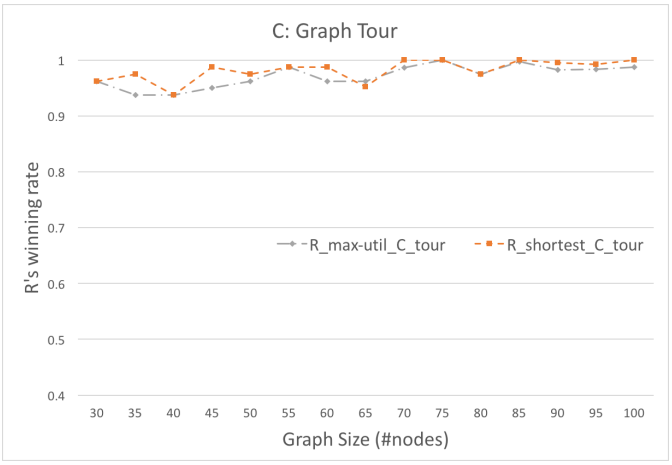
Note, though, that Theorem 7 applies only when C follows a random walk. In case C is strategic, R should also behave strategically, as stated in Theorem 1.

7.4 Non-Uniform Edge Costs

The various motion strategies of the agents choose the next move based on the utility values of the neighboring nodes. This could be generalized for non-uniform



(a) R*'s winning rate when* R *follows either the path with maximal utility or the shortest path.* C *follows a random walk*



(b) R*'s winning rate when* R *follows either the path with maximal utility or the shortest path.* C *a deterministic depth-first traversal along the graph*

Fig. 6: R*'s winning rate when* R,C *are zero-knowledge agents*

edge costs by having the motion strategies prioritize the neighboring nodes depending on the node where the agent currently resides. Namely, if an agent resides at $u \in V$ at time t, the *cost* f of some neighboring node $v \in V$ would be given as follows (without loss of generality, we demonstrate for R):

$$f(u,v) = c \cdot U_{\mathrm{R}}(v) + (1-c) \cdot w(u,v) \tag{20}$$

Where $U_{\mathrm{R}}(v)$ is v's utility value independently of R's current node, as computed in Section 4.2. $w(u,v)$ denotes the cost of the edge (u,v). That is, v's cost is composed of two factors: safety ($U_{\mathrm{R}}(v)$) and the travel cost. These factors are weighted by the constant $c \in [0,1]$, such that if $c = 1$ it is the same as uniform edge costs, and if $c = 0$ then the problem becomes an instance of the shortest path problem. Note,

however, that since the node utilities are values between 0 to 1, the travel costs should be normalized, as well.

8 Empirical Evaluation

In this section we evaluate our navigation strategies, as constructed by our utility-computation framework, using different risk attitudes for computing the utility values, and examine the effect of online strategies update. A collection of graphs with 30 to 100 (with jumps of 5) nodes was randomly generated. For each number of nodes, 10 graphs were generated with each of the following probabilities to create an edge: 0.05,0.1,0.15,0.2, making a total of 40 graphs per number of nodes (self loops were not allowed). For each number of nodes, 10% of the nodes were randomly set as goal nodes (i.e., safehouses). For each graph, a directed visibility edge (v, u) (v,$u \in V$) was created with probability $\frac{1}{2 \cdot d(v,u)}$, $d(v, u)$ is distance (in hops) between v,u. That is, the probability of a visibility edge decreases the further the nodes are away from each other. This corresponds with most real-life environments, where closer locations are more visible. For the offline model, an experiment was executed for each combination of node and configuration utility functions for R,C: risk averse, risk neutral (Section 4.2). Another risk attitude which had been examined was risk seeking: for a configuration $\mathcal{V} \in V_{conf}$, takes the maximum value among $\mathcal{V}$'s neighbors, and for a map node $v \in V$ takes the maximum value of a configuration where R resides at v. For the online model, various utility functions were also used to evaluate the information obtained at a node visited by an agent (Equations 11-13).

Each experiment was repeated 20 times for each graph within the graphs collection. Each time new starting locations for both agents were randomly chosen (not among the safehouses). This makes, for each settings of node, configuration and information utilities, a total of $40 \cdot 20$ experiments for each graph size, resulting in a total of $40 \cdot 20 \cdot 9 = 7200$ runs for each offline model scenario (Section 5) and $7200 \cdot 5 = 36000$ runs where agents update their strategies on-the-fly. For each graph size, combination of node, configuration and information utilities, the average winning rate of R was calculated and will be referred to as R*'s winning rate*.

8.1 Evaluating the Utility-Based Strategies

This section presents the experiments conducted for evaluating the framework for utility values computation and the generated navigation strategies for the various scenarios of the offline (Section 5) and online (Section 6) models. Figure 7 shows R's winning rate for the offline model (Section 5). Each curve matches a different setting of knowing the opponent's initial location (utility functions are indicated). Applying risk seeking (maximal utility among neighbors), for either configurations or nodes, decreased R's winning rate.

Figure 8a shows R's winning rate when both agents run offline but only C knows R's initial location, with risk averse for configurations. Risk neutral for map nodes was compared to risk seeking for map nodes. T-Test has confirmed that R's winning rate is significantly greater with risk neutral for map nodes (p-value <<

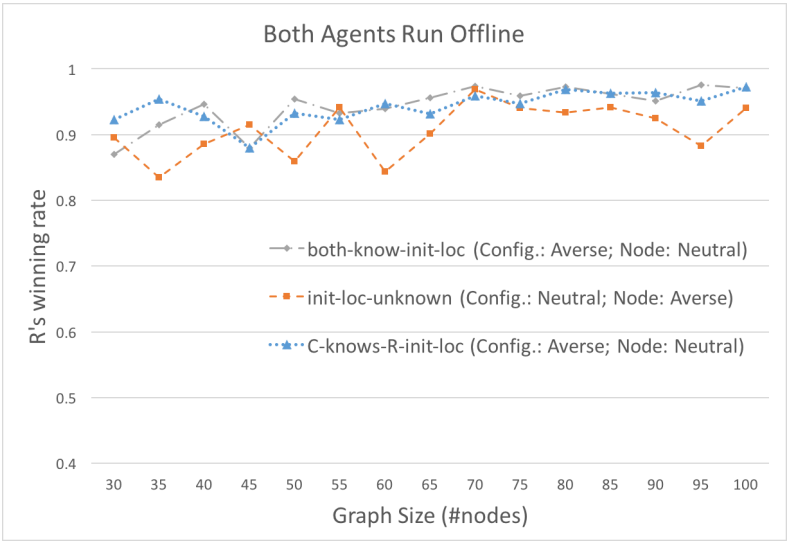


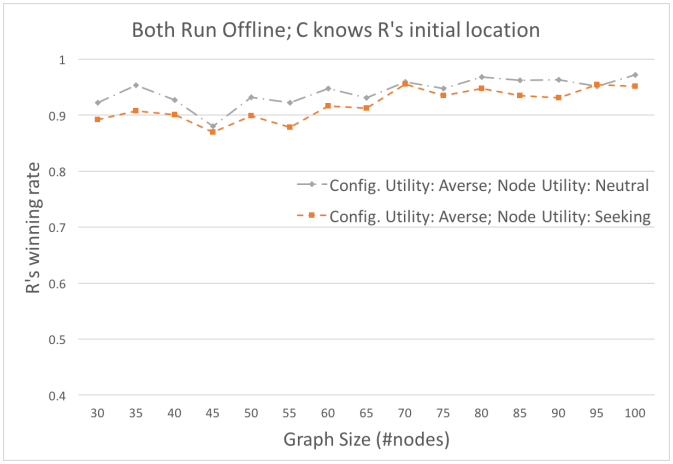
Fig. 7: R*'s winning rate when both* R,C *compute their strategies offline. Different setting of knowing the opponent's initial location are presented*

0.001). Figure 8b compares these settings of utility function when initial locations are mutually known, there was no statistically significant difference (T-Test gave a p-value of 0.2). This shows that information influences more than the risk attitude.

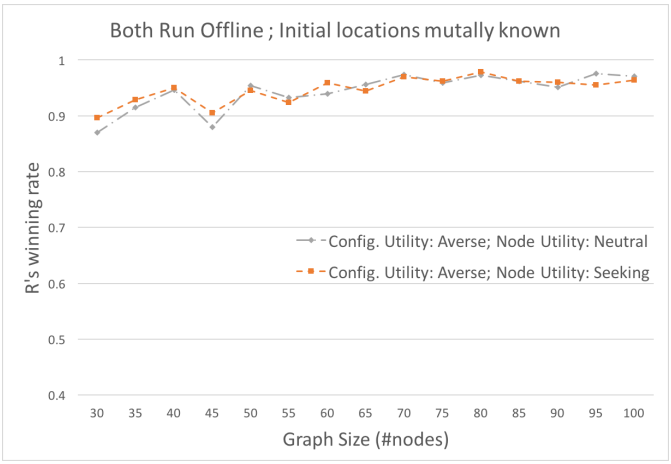
Figure 9a shows R's winning rate for the following settings: both agents perform on-the-fly strategy updates (i.e., run online), only C runs online and only R runs online. When C ran online, C's information utility (IU) was risk seeking, at a viewpoint updated strategies by increasing capture probability and at a non-viewpoint node, updated by IU of its neighbors (Section 6.2). When R ran online, R's IU was risk neutral and updated by decreasing capture probability at a viewpoint. In all experiments, R and C were set with risk averse utility for configurations and risk neutral for map nodes. In order to specifically examine the influence of on-the-fly updates on R's winning rate, when R ran online, executions where R did not observe C towards the last quarter of the game were discarded. Same for C when R ran offline. ANOVA Single-Factor test has confirmed that these differences of the winning rates are indeed statistically significant (p-value $<<$ 0.001), verified with Tukey's HSD post-hoc test.

The conclusion from Figure 9a is that on-the-fly strategy updates are the best response, as each agent benefits from using the information obtained at runtime.

We have repeated these experiments where C updated by IU of neighboring nodes also at a viewpoint. This time, however, the curve for an offline strategy for C is omitted, since it would not change when modifying C's strategy update policy. R's winning rate is shown in Figure 9b. R's winning rate has increased, which points out the importance of using information when having it: C should use the information it receives at a viewpoint (update by capture probability) instead of looking for new information (update by IU).



(a) R*'s winning rate when both* R,C *compute their strategies offline applying risk averse utility for the configurations.* C *knows* R*'s start. Risk neutral for map nodes is significantly better than applying risk seeking for the map nodes*

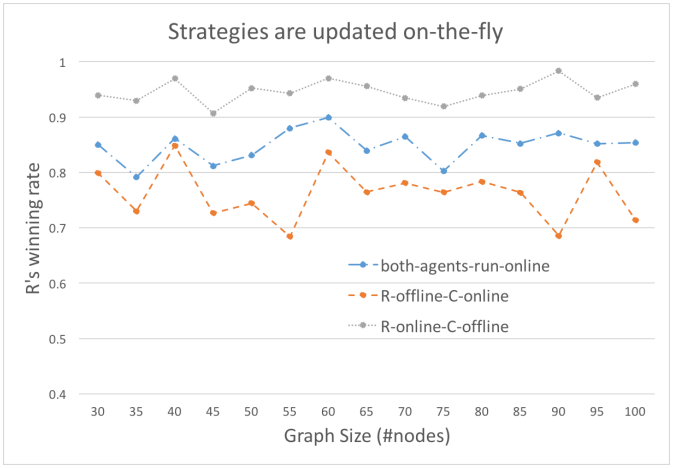


(b) R*'s winning rate when both* R,C *compute their strategies offline applying risk averse utility for the configurations. Each agent knows its opponent's start. No significant difference among risk attitudes for map nodes, implying information influences more than the risk attitude*

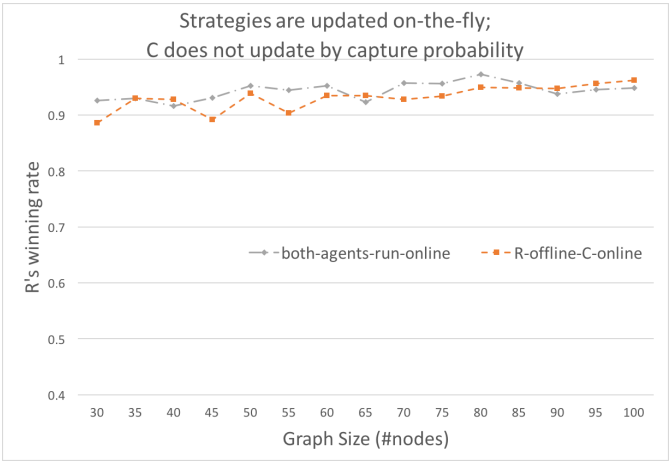
Fig. 8: Significance of knowing the opponent's initial location over the risk attitude

8.2 Utility-Based Strategies Compared to State-of-the-Art Strategies

The online model is indeed the most realistic and interesting world modeling, especially due to the visibility edges. Therefore, we have taken this model and compared our utility-based strategies (allowing on-the-fly updates) to other motion strategies which are not based on our framework for computing utility values. For clarity, in this section we refer to the stochastic strategy based on our utility values with on-the-fly updates (as in Section 6) as *online-utility-based* strategy. Since our stochastic strategy imposes bias when choosing the next move among the available neighbors at a time step, R was tested with random walk (i.e., next neighbor is



(a) *R's winning rate when* C *performs on-the-fly strategy updates.* C*'s information utility is risk seeking, at a viewpoint* C *updates by capture probability.* R*'s performance when* R *runs offline is compared to performing on-the-fly updates*



(b) *R's winning rate when* C *performs on-the-fly strategy updates.* C*'s information utility is risk seeking,* C *updates by information utility at all nodes. Comparing to* C *updating by capture probability at a viewpoint,* R*'s winning rate has increased and there is no significant difference between online and offline runs for* R

Fig. 9: R*'s winning rate when runs offline or online,* C *runs online*

chosen in uniform distribution). In response, C was tested with random walk and with our online-utility-based strategy.

Figure 10 shows R's winning rate for the following settings of stochastic strategies: R follows our online-utility-based strategy and C follows random walk; both agents follow online-utility-based ; R follows random walk and C follows online-utility-based. We should note that the node, configuration and information utility assigned for each agent are those reported in Section 8.1: C's information utility (IU) was risk seeking, at a viewpoint updated strategies by increasing capture probability and at a non-viewpoint node, updated by IU of its neighbors (Section 6.2). R's IU was risk neutral and updated by decreasing capture probability at

a viewpoint. In both experiments, R and C were set with risk averse utility for configurations and risk neutral for map nodes. As shown in Figure 10, when C follows online-utility-based, R's winning rate increases if R responds with online-utility-based. Symmetrically, when R follows online-utility-based, C's winning rate increases when responds with our online-utility-based strategy (being a zero-sum game, C's winning rate is opposed to R's, thus a lower curve means greater success for C). ANOVA Single-Factor test has confirmed that these differences of the winning rates are indeed statistically significant (p-value $<<$ 0.001), verified with Tukey's HSD post-hoc test. Hence, when comparing to stochastic strategies which are not based on our framework for computing utility values, our online-utility-based strategy is the best response for both agents.

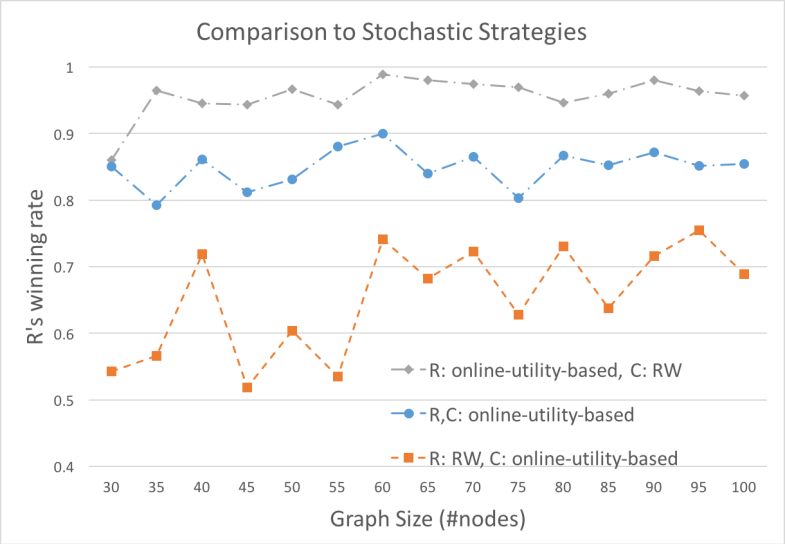


Fig. 10: R*'s winning rate for several settings of stochastic strategies: online-utility-based (Section 6) and random walk (RW). Our online-utility-based strategy is each agent's best response*

In addition, R was tested with alternative deterministic strategies:

1. Shortest path: R follows the shortest path (in terms of hops) towards a goal node $v_g \in V_G$.
2. Covert path: the map graph is converted into a directed graph, where the weight of the directed edge (v, u) is u's in-degree in terms of visibility edges. R follows the shortest path (in terms of these edge weights) to a goal node $v_g \in V_G$. Hence, R follows a path that obscures the most the nodes along it.

Since C is assumed to be able to perform the same computation as R, in case C observes R when R follows a deterministic strategy (i.e., shortest or covert path), C can infer R's destination and plan a path for intercepting R. Thus, taking advantage of R's deterministic movement, in case R follows a deterministic strategy, the strategy for C would be as follows:

1. If R is observed, plan an intercepting path and follow it.

2. Otherwise, follow a path to a goal node such that the sum of out-degree of visibility edges of the nodes along this path is maximized.

Namely, C moves along the graph in a manner that increases its chances to detect R. Once R is observed, C intercepts it (if possible).

Figure 11 shows R's winning rate when R follows the deterministic strategies of covert path and shortest path. Since C is able to perform the same computations as R, for each deterministic strategy R follows, C follows the above scheme: searches for R and intercepts it once observed. The curve matching the experiment where both agents follow online-utility-based (as shown first in Figure 10) is also shown in Figure 11. These results are not surprising, because as explained in Section 5.3, once C knows R's location and strategy, C can easily intercept R. As the curve of online-utility-based shows, interception becomes harder if R moves stochastically. ANOVA Single-Factor test has confirmed that these differences of the winning rates are indeed statistically significant (p-value $<< 0.001$), verified with Tukey's HSD post-hoc test. As obtained from Figure 11, online-utility-based for both agents is the best response for each of them, in case of deterministic strategies.

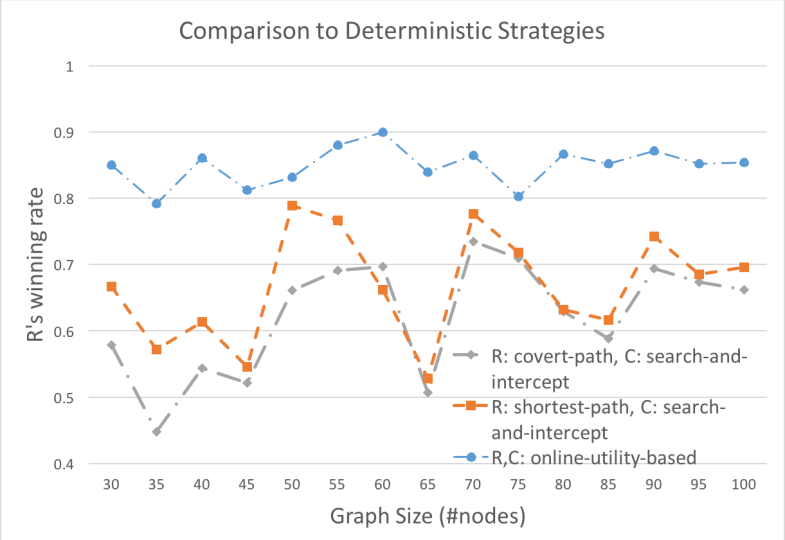


Fig. 11: R*'s winning rate when* R *follows deterministic strategies and* C *follows its best response (i.e., searches for* R *and then intercepts* R *if observes it). Comparison to* R*'s winning rate when both agents follow online-utility-based (best response when both agents follow a stochastic strategy) shows that* R*'s is greater*

9 Conclusions and Future Work

In this paper we address the problem of path planning in adversarial environments, where another adversary agent tries to intercept our agent along its way to its destination. A multi-purpose framework for computing a utility value for the

map graph nodes was presented, and was used for solving various problem modelings. Each model differed in what the agents know about their opponent and the type of their motion pattern. Theoretic and empiric analysis addressed the influence of various risk attitudes on the constructed strategies. For the first time, to our knowledge, updates are allowed during the execution in a pursuit-evasion problem variant. The updates where conducted in two aspects: one was using the information known so far (updating by capture probability at a viewpoint), while the other related to the information available in the environment (updating by information utility). Analytic analysis showed the contribution of on-the-fly updates, such that R's probability to reach a goal node safely is increased. This analysis was affirmed by experiments.

Much work is left for the future. Additional aspects of the SANE to be researched are, for example, other knowledge models and their influence on path safety, featuring observation errors to the visibility edges, deceiving the adversary by being observed deliberately, and playing against an adversary with more limited computational abilities than ours. Although we prioritize the safehouses uniformly, incorporating preference among the safehouses into the framework for computing node utility values might also be considered. Intuitively, these priorities could be expressed via the utility values. However, some challenges arise, e.g., a trade-off between following a risky path towards a high-priority safehouse or following a safe path towards a low-priority safehouse, and ensuring that any low-priority safehouse is still more preferable than a non-goal node which is a neighbor of some high-priority safehouses.

References

1. Adler, M., Räcke, H., Sivadasan, N., Sohler, C., Vöcking, B.: Randomized pursuit-evasion in graphs. Combinatorics, Probability and Computing **12**(03), 225–244 (2003)
2. Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Applied Mathematics **8**(1), 1–12 (1984)
3. Alexopoulos, A., Schmidt, T., Badreddin, E.: Cooperative pursue in pursuit-evasion games with unmanned aerial vehicles. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS). IEEE (2015)
4. Alpern, S., Fokkink, R., Gal, S., Timmer, M.: On search games that include ambush. SIAM Journal on Control and Optimization **51**(6), 4544–4556 (2013)
5. Alspach, B.: Searching and sweeping graphs: a brief survey. Le matematiche **59**(1, 2), 5–37 (2006)
6. Barrett, S., Stone, P., Kraus, S.: Empirical evaluation of ad hoc teamwork in the pursuit domain. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)-Volume 2, pp. 567–574. International Foundation for Autonomous Agents and Multiagent Systems (2011)
7. Basilico, N., De Nittis, G., Gatti, N.: A security game combining patrolling and alarm–triggered responses under spatial and detection uncertainties. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 397–403. AAAI Press (2016)
8. Basilico, N., Gatti, N., Amigoni, F.: Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems(AAMAS)-Volume 1, pp. 57–64. International Foundation for Autonomous Agents and Multiagent Systems (2009)
9. Bhadauria, D., Klein, K., Isler, V., Suri, S.: Capturing an evader in polygonal environments with obstacles: The full visibility case. The International Journal of Robotics Research pp. 1176–1189 (2012)
10. Bishop, C.: Pattern recognition and machine learning (information science and statistics). Springer, New York (2007)

11. Boidot, E., Marzuoli, A., Feron, E.: Optimal navigation policy for an autonomous agent operating in adversarial environments. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3154–3160. IEEE (2016)
12. Borie, R.B., Tovey, C.A., Koenig, S.: Algorithms and complexity results for pursuit-evasion problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), vol. 9, pp. 59–66 (2009)
13. Brémaud, P.: Markov chains: Gibbs fields, Monte Carlo simulation, and queues, vol. 31. Springer Science & Business Media (2013)
14. Chen, H., Zhang, F.: The expected hitting times for finite markov chains. Linear Algebra and its Applications **428**(11-12), 2730–2749 (2008)
15. Chung, T.H., Hollinger, G.A., Isler, V.: Search and pursuit-evasion in mobile robotics. Autonomous Robots **31**(4), 299–316 (2011)
16. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a nash equilibrium. SIAM Journal on Computing **39**(1), 195–259 (2009)
17. De Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theoretical Computer Science **386**(3), 188–217 (2007)
18. Dereniowski, D., Dyer, D., Tifenbach, R.M., Yang, B.: Zero-visibility cops and robber game on a graph. In: Frontiers in Algorithmics and Algorithmic Aspects in Information and Management, pp. 175–186. Springer (2013)
19. Foderaro, G., Raju, V., Ferrari, S.: A cell decomposition approach to online evasive path planning and the video game ms. pac-man. In: IEEE International Symposium on Intelligent Control (ISIC), pp. 191–197 (2011)
20. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theoretical Computer Science **399**(3), 236–245 (2008)
21. Geraerts, R., Schager, E.: Stealth-based path planning using corridor maps. In: Proceedings of Computer Animation and Social Agents (CASA) (2010)
22. Gibbs, A., Su, F.: On choosing and bounding probability metrics. International statistical review **70**(3), 419–435 (2002)
23. Gmytrasiewicz, P.J., Doshi, P.: A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research (JAIR) **24**, 49–79 (2005)
24. Göbel, F., Jagers, A.: Random walks on graphs. Stochastic processes and their applications **2**(4), 311–336 (1974)
25. Guibas, L.J., Latombe, J.C., LaValle, S.M., Lin, D., Motwani, R.: A visibility-based pursuit-evasion problem. International Journal of Computational Geometry & Applications **9**(4 & 5), 471–493 (1999)
26. Huang, H., Ding, J., Zhang, W., Tomlin, C.J.: Automation-assisted capture-the-flag: a differential game approach. IEEE Transactions on Control Systems Technology **23**(3), 1014–1028 (2015)
27. Jain, M., Conitzer, V., Tambe, M.: Security scheduling for real-world networks. In: Proceedings of the 2013 international conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 215–222 (2013)
28. Karlin, S.: A first course in stochastic processes. Academic press (2014)
29. Latombe, J.C.: Robot Motion Planning. Boston: Kluwer Academic Publishers (1991)
30. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Proceedings of the eleventh international conference on machine learning, vol. 157, pp. 157–163 (1994)
31. Lubiw, A., Vosoughpour, H.: Visibility graphs, dismantlability, and the cops and robbers game. 26th Canadian Conference on Computational Geometry (2014)
32. Marzouqi, M., Jarvis, R.A.: Covert path planning for autonomous robot navigation in known environments. In: Proceedings of the Australasian Conference on Robotics and Automation, Brisbane. Citeseer (2003)
33. Marzouqi, M., Jarvis, R.A.: Covert robotics: Covert path planning in unknown environments. In: Proceedings of the Australasian Conference on Robotics and Automation, Brisbane, Australia. Citeseer (2003)
34. Maskin, E., Tirole, J.: Markov perfect equilibrium, i: Observable actions. Journal of Economic Theory **100**(2), 191–219 (2001)
35. McCarthy, S., Tambe, M., Kiekintveld, C., Gore, M.L., Killion, A.: Preventing illegal logging: Simultaneous optimization of resource teams and tactics for security. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 3880–3886 (2016)
36. Nearchou, A.C.: Path planning of a mobile robot using genetic heuristics. Robotica **16**(05), 575–588 (1998)

37. Nowakowski, R., Winkler, P.: Vertex-to-vertex pursuit in a graph. Discrete Mathematics **43**(2), 235–239 (1983)
38. Örgen, P., Winstrand, M.: Minimizing mission risk in fuel constrained uav path planning. AIAA Journal of Guidance, Control, and Dynamics **31**(5), 1497–1500 (2008)
39. Petres, C., Pailhas, Y., Patron, P., Petillot, Y., Evans, J., Lane, D.: Path planning for autonomous underwater vehicles. IEEE Transactions on Robotics **23**(2), 331–341 (2007)
40. Raghavan, T.: Zero-sum two-person games. Handbook of game theory with economic applications **2**, 735–768 (1994)
41. Sgall, J.: Solution of david gale's lion and man problem. Theoretical Computer Science **259**(1-2), 663–670 (2001)
42. Shoham, Y., Leyton-Brown, K.: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press (2008)
43. Stentz, A.: Optimal and efficient path planning for partially-known environments. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 3310–3317 (1994)
44. Sutton, R.S., Barto, A.G.: Introduction to reinforcement learning. MIT Press (1998)
45. Tews, A., Matarić, M.J., Sukhatme, G.S.: Avoiding detection in a dynamic environment. In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS), vol. 4, pp. 3773–3778 (2004)
46. Tews, A., Sukhatme, G.S., Matarić, M.J.: A multi-robot approach to stealthy navigation in the presence of an observer. In: Proceedings of the International Conference on Robotics and Automation, pp. 2379–2385 (2004)
47. Yehoshua, R., Agmon, N.: Adversarial modeling in the robotic coverage problem. In: Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) (2015)

10 Appendix

We shall now introduce the full proof of Theorem 4:

Theorem 4 *Denote the actual probability distribution over* C*'s location at time* t *as* $P_{\mathrm{C}}^{(t)}$ *(i.e.,* $P_{\mathrm{C}}^{(t)}$ *is* 1 *for* C*'s location and* 0 *for any other node).* $\tilde{P}_{\mathrm{C}}^{(t)}$ *denotes* R*'s estimation for* $P_{\mathrm{C}}^{(t)}$ *before updating it with the information received from* $V_{visible}(v_{\mathrm{R}}^t)$*, while* $\hat{P}_{\mathrm{C}}^{(t)}$ *denotes* $\tilde{P}_{\mathrm{C}}^{(t)}$ *after being updated. Incorporating the information received from the visibility edges for updating* $\hat{P}_{\mathrm{C}}^{(t)}$*, improves the accuracy of this estimation. Namely,* $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) \geq H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$.

Proof As stated in Section 6.2, if $\sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] = 0$ then $\hat{P}_{\mathrm{C}}^{(t)} = \tilde{P}_{\mathrm{C}}^{(t)}$. Hence, $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) = H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$.

Now, assume $\sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] > 0$ (a sum of probabilities is either 0 or positive). If C has been observed ($v_{\mathrm{C}}^t \in V_{visible}(v_{\mathrm{R}}^t)$), then $\hat{P}_{\mathrm{C}}^{(t)} = P_{\mathrm{C}}^{(t)}$.

Otherwise, $v_{\mathrm{C}}^t \notin V_{visible}(v_{\mathrm{R}}^t)$. Let us compute the Hellinger distance between the actual distribution over C's location to R' estimated distribution prior being updated, $H\left(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right)$:

$$H\left(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) = \sqrt{\sum_{v \in V}\left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v]} - \sqrt{P_{\mathrm{C}}^{(t)}[v]}\right)^2} =$$

$$\left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)}\left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v]} - \sqrt{P_{\mathrm{C}}^{(t)}[v]}\right)^2 + \right.$$

$$\left.\sum_{v \in V_{visible}(v_{\mathrm{R}}^t)}\left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v]} - \sqrt{P_{\mathrm{C}}^{(t)}[v]}\right)^2 + \left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - \sqrt{P_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]}\right)^2\right)^{1/2}$$

C resides at v_{C}^t, hence $\forall v \in V \setminus \{v_{\mathrm{C}}^t\}$, $P_{\mathrm{C}}^t[v] = 0$, $P_{\mathrm{C}}^t[v_{\mathrm{C}}^t] = 1$. Therefore:

$$H\left(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) = \left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)}\left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v]}\right)^2 + \right.$$

$$\left.\sum_{v \in V_{visible}(v_{\mathrm{R}}^t)}\left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v]}\right)^2 + \left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - 1\right)^2\right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - 1\right)^2\right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus \left(V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\}\right)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \tilde{P}_{\mathrm{C}}^{(t)} - 2\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} + 1\right)^{1/2}$$

Now, we shall compute the Hellinger distance between the actual distribution over C's location to R' estimated distribution after being updated, $H\left(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right)$:

$$H\left(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) = \left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \left(\sqrt{\hat{P}_{\mathrm{C}}^{(t)}[v]} - \sqrt{P_{\mathrm{C}}^{(t)}[v]} \right)^2 + \right.$$

$$\left. \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \left(\sqrt{\hat{P}_{\mathrm{C}}^{(t)}[v]} - \sqrt{P_{\mathrm{C}}^{(t)}[v]} \right)^2 + \left(\sqrt{\hat{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - \sqrt{P_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} \right)^2 \right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \hat{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \hat{P}_{\mathrm{C}}^{(t)}[v] + \left(\sqrt{\hat{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - 1 \right)^2 \right)^{1/2}$$

C resides at v_{C}^t and $v_{\mathrm{C}}^t \notin V_{visible}(v_{\mathrm{R}}^t)$, hence:

$$\forall v \in V_{visible}(v_{\mathrm{R}}^t) : \hat{P}_{\mathrm{C}}^{(t)}[v] = 0$$

$$\forall v \in V \setminus V_{visible}(v_{\mathrm{R}}^t) : \hat{P}_{\mathrm{C}}^{(t)}[v] = \tilde{P}_{\mathrm{C}}^{(t)}[v] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \hat{P}_{\mathrm{C}}^{(t)}[v]}$$

Therefore, we obtain:

$$H\left(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}\right) =$$

$$\left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \hat{P}_{\mathrm{C}}^{(t)}[v] + \left(\sqrt{\hat{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]} - 1 \right)^2 \right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \left(\tilde{P}_{\mathrm{C}}^{(t)}[v] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} \right) + \right.$$

$$\left. \left(\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} - 1 \right)^2 \right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \left(\tilde{P}_{\mathrm{C}}^{(t)}[v] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} \right) + \right.$$

$$\left. \tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} - 2\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} + 1 \right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} + \right.$$

$$\left. \tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} - 2\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} + 1 \right)^{1/2} =$$

$$\left(\sum_{v \in V \setminus (V_{visible}(v_{\mathrm{R}}^t) \cup \{v_{\mathrm{C}}^t\})} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v] + \tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \right.$$

$$\left. -2\sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} + 1 \right)^{1/2}$$

If $\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] = 0$ then $\hat{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] = \tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]$ and $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) = H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$. Otherwise:

$$\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] > 0 \Rightarrow \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]} > 0$$

$$\Rightarrow \sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] + \frac{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t] \cdot \sum_{v \in V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}{\sum_{v \in V \setminus V_{visible}(v_{\mathrm{R}}^t)} \tilde{P}_{\mathrm{C}}^{(t)}[v]}} > \sqrt{\tilde{P}_{\mathrm{C}}^{(t)}[v_{\mathrm{C}}^t]}$$

$$\Rightarrow H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) > H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$$

In conclusion, we obtain that $H(\tilde{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)}) \geq H(\hat{P}_{\mathrm{C}}^{(t)}, P_{\mathrm{C}}^{(t)})$, hence R's estimated probability distribution over C's location is closer to $P_{\mathrm{C}}^{(t)}$ after incorporating the information gained from the visibility edges.

Proving Theorem 4 for C is similar.