Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

# Formulating Robot Pursuit-Evasion Strategies by Model Checking [⋆]

**Hongyang Qu** [*] **Andreas Kolling** [*] **Sandor M. Veres** [*]

[*] *Department of Automatic Control and Systems Engineering*
*University of Sheffield, UK*
*e-mail: {h.qu, a.kolling, s.veres}@sheffield.ac.uk*

**Abstract:** This paper presents an application of a model checking framework to robotic search, particularly search problems known as pursuit-evasion that assume a smart, fast and evading target. Within the framework we can model different pursuit-evasion problems and thereby enable a direct and rigorous comparison between different problem formulations and their respective properties and algorithms. In addition, we enable the computation of new kinds of solutions to pursuit-evasion problems, so called strategies, that can consider multiple criteria, e.g. order of vertices or connectedness. These strategies are computed by satisfying a temporal logic formula by model checking. We present theorems that show the connection between a strategy and a temporal logic formula. We demonstrate our approach by applying it to two different graph-based pursuit-evasion problems and show how to enable a comparison. This work presents the first step and basis for further investigations of more rigorous and unified approaches to understand, compare and design pursuit-evasion models.

Keywords: Distributed robot systems; pursuit-evasion games; verification; graph-searching, mobile agent.

## 1. INTRODUCTION

The study of surveillance tasks goes back a long way in the robotics literature. Initially, there have been visibility-based pursuit-evasion games, first investigated by in Suzuki and Yamashita [1992], for detecting targets with an unlimited-range beam sensor. Following this many variants of their problem formulation were studied; for instance robots with an unlimited range of omnidirectional gap sensors by Sachs et al. [2004], which can detect intruders robustly and in varied environments. A different model for the problem was used by Parker [2002] who investigated the surveillance of multiple moving targets in simple planar environments by large robot teams. Bopardikar et al. [2007] presented a capturing strategy in open planar environments in which robots form a "trapping chain" to ensure that the target once detected by any robot in the chain will subsequently be caught. A probabilistic approach is presented by Moors et al. [2005] that take into account uncertainties of measurements and movements. A summary of the vast variety of surveillance tasks and approaches from a robotics perspective is presented by Chung et al. [2011].

For large-scale pursuit-evasion, however, graph-based models, first introduced by Parsons [1976], have proven to be useful. An overview of theoretical work on graph-based pursuit-evasion is given by Fomin and Thilikos [2008]. In a robotics context new models, such as the Graph-Clear model based on surveillance graphs (SG), have been investigated intensively by Kolling and Carpin [2010a,b]. For robotics applications, Kolling and Carpin [2008] thoroughly investigated the practicality of graph-based models by showing how it is possible to algorithmically extract suitable graphs from occupancy grid maps produced by robots. Generalized Voronoi diagrams have been used to obtain SGs from maps continuously updated by mobile robots exploring indoor environments. This allows the application of numerous algorithms (see Fomin and Thilikos [2008], Chung et al. [2011] for an overview) that were designed to solve pursuit-evasion problems under different constraints. The large variety of different models, however, make it difficult to compare them. They differ with regard to the actions that searchers can execute, visibility requirements for the targets and the locations targets can occupy in the graph. In addition, the algorithms that compute solutions to these pursuit-evasion problems, also known as strategies, also vary with regard to their optimization criteria, complexity, and further assumptions, e.g., some require connectedness Barrière et al. [2002].

In this paper, we propose a new framework for modelling pursuit-evasion problems and algorithms in a unified manner that allows a rigorous and comprehensive comparison. The framework is based on model checking, i.e. algorithms of formal verification. Model checking allows a strategy to be specified as a temporal logic formula. This addresses one of the problems with prior pursuit-evasion models, namely that they only consider single criteria, usually the minimal number of searchers. Adding different criteria is not possible within the context of a particular pursuit-evasion model without changing its formalization. Due to the versatility of temporal logics we can compute multiple strategies, each of which can satisfy different criteria, e.g., to require searching a subset of nodes in a certain order.

[⋆] This work was supported by the EPSRC project EP/J011894/2.

The specific actions that a pursuit-evasion model permits also determine how applicable it is for implementations on robotic searchers. Applicability can vary greatly depending on the type of environment, the method with which graphs are extracted from the map and the type of robotic searchers that are available. The ultimate goal of our framework is to formulate a pursuit-evasion model that is flexible enough to allow multiple criteria, multiple and different kinds of actions from multiple previous models and heterogenous search teams. As a first step towards this goal we present our framework and show how to model two different graph-based pursuit-evasion problems and enable a comparison, e.g., of the number of states and computation time for strategies.

The remainder of this paper is structured as follows. In Section 2, we give a brief discussion of two graph-based pursuit-evasion models: Graph-Clear (GC) and Weighted Edge Searching (WES). Section 3 presents an introduction to the core of the framework of: the model checker of multiagent systems MCMAS [Lomuscio et al., 2009], including its modelling and specification language. In Sections 4 and 5 rules to model the GC and WES algorithm are presented with main theorems on the correctness of the algorithms. Section 6 concludes the paper.

## 2. PURSUIT-EVASION PROBLEM

There can be a variety of formulations for pursuit-evasion problems. One of the common assumptions, which most of these formulations share, is the search for an omniscient and smart target that moves at unbounded speed. This target is then represented by the formal concept of contamination by Parsons [1976] as defined below. The unbounded speed assumption [Suzuki and Yamashita, 1992] simplifies the problem setup and makes solutions conservative: unbounded speed is the worst one can assume about spread of contamination.

The second common assumption is that the searchers can execute actions, usually in the form of motion, that clears contamination or block it from spreading through parts of the environment. Variants of pursuit-evasion problems consider different kinds of clearing and blocking actions. For our purposes we consider the Graph-Clear (GC) model introduced and formalized in Kolling and Carpin [2010b] that has proven to be a useful model for robot search and the Weighted Edge-Searching (WES) model introduced by Barrière et al. [2002]. Both models consider that some basic actions may require multiple searchers in order to be executed. We will briefly describe these two models below.

In Graph-Clear the environment is given by a surveillance graph which is an undirected weighted graph $G = (V, E, w)$ with set $V$ of vertices, set $E$ of edges, and $w : V \cup E \to \mathbb{N}^+$ defined as a weighting function. To model contamination, and how it is spreading, the vertices and edges have an associated state. Vertices are either *clear* or *contaminated* and edges are either *clear*, *contaminated*, or *blocked*. These are abbreviated as $\mathcal{R}, \mathcal{C}$, and $\mathcal{B}$ for clear, contaminated, and blocked, respectively. The state space of the surveillance graph with $n$ vertices and $m$ edges is then given by $\nu \in \mathcal{V}(G) = \{\mathcal{R}, \mathcal{C}\}^n \times \{\mathcal{R}, \mathcal{C}, \mathcal{B}\}^m$. As a shorthand, we write $\nu(v_i)$ and $\nu(e_j)$ for the state of a particular vertex or edge. Contamination spreads on recontamination paths. These are paths of vertices and edges on which no edge is blocked.

Finally, searchers can execute actions which are either a sweep on a vertex or a block on an edge. The executed actions on $G$ can be represented by $a = \{a_1, \ldots, a_{n+m}\} \in \{0,1\}^{n+m} = \mathcal{A}(G)$ where a 1 for an associated vertex indicates a sweep and a 1 for an associated edge indicates a block. The cost of an action $a$ is given by $c(a) = \sum_{i=1}^{n} a_i w(v_i) + \sum_{j=1}^{m} a_{n+j} w(e_j)$, representing the number of robots needed to execute all sweeps and blocks for the action. The spread of contamination and the clearing of actions can now be formalized via a transition function $\zeta$, defined in Kolling and Carpin [2010b], as follows:

*Definition 1.* [Transition function] Let $G$, $\mathcal{V}(G)$ and $\mathcal{A}(G)$ be defined as above. The *transition function* $\zeta$ maps a state and an action into a new state:

$$\zeta : \mathcal{V}(G) \times \mathcal{A}(G) \to \mathcal{V}(G).$$

Given $a \in \mathcal{A}(G)$ and $\nu \in \mathcal{V}(G)$, the new state $\nu'$ is defined as follows:

(1) if $a_{n+j} = 1$, $1 \le j \le m$, then $\nu'(e_j) = \mathcal{B}$
(2) if $a_i = 1$, $1 \le i \le n$, then $\nu'(v_i) = \mathcal{R}$
(3) if $\nu_{n+j} = \mathcal{B}$, $a_{n+j} = 0$, $1 \le j \le m$, and no recontamination path between $e_j$ and $x \in V \cup E$ with $\nu(x) = \mathcal{C}$ exists, then $\nu'_{n+j} = \mathcal{R}$
(4) if there exists a recontamination path between $x \in V \cup E$ and $y \in V \cup E$ with $\nu(y) = \mathcal{C}$, then $\nu'(x) = \mathcal{C}$
(5) $\nu'_i = \nu_i$ otherwise.

In colloquial terms the above simply describes the following rules:

(1) edges where a block action is applied become blocked;
(2) vertices where a sweep action is applied become clear;
(3) blocked edges where a block action is not applied anymore become clear if there is no recontamination path involving them;
(4) vertices or edges for which a recontamination path towards a contaminated vertex or edge exists become contaminated;
(5) vertices or edges maintain their previous state if none of the former cases apply.

A strategy to clear an initially fully contaminated surveillance graph $G$ is a sequence of actions $\mathcal{S} = \{a_1, a_2, \ldots, a_k\}$, and the cost of the strategy is the maximum cost of executing one of its actions, i.e., $\max_{i=1\ldots k} c(a_i)$. In Kolling and Carpin [2010b] it has been shown that solving the GC problem on graphs is NP-hard and a polynomial time algorithm for trees exists. The algorithm has been applied to robotic search in Kolling and Carpin [2009, 2010a,b], Kolling and Kleiner [2013]. In Kolling and Carpin [2008] a method to extract instances of the GC problem from robot maps was presented, validating the graph-based model for practical use.

The above definitions are best illustrated with the simple example shown in Fig. 1, where vertices are associated with rooms, and edges with connections between rooms. Edges between vertices are blocked by placing a robot in the connection between rooms. All contaminated parts can hide an intruder while cleared parts are guaranteed to be free of undetected intruders. A room is cleared by using the specified number of robots to sweep through it.

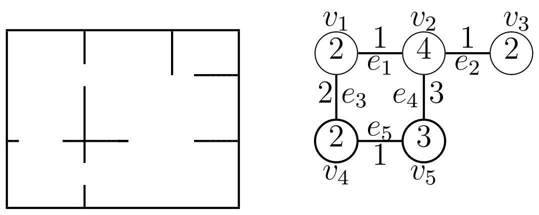

Fig. 1. A simple example environment and one of the possible surveillance graphs that can model the search for a target. Numbers on vertices are clearing costs and numbers on edges are blocking costs.

Fig. 2 presents a strategy to clear the graph associated with the environment shown in Fig. 1. The first column displays the status of the graph in the form of "$\nu(v_1)\cdots\nu(v_5)$ $\nu(e_1)\cdots\nu(e_5)$", the second the applied action of the form "$a_{v_1}\cdots a_{v_5}$ $a_{e_1}\cdots a_{e_5}$", and the third the cost. The reader should note that in the third row an action of sweeping two vertices at the same time is applied, and that a final action removing all blocks is executed in the end (with 0 cost). The cost of this strategy is 12, i.e., the maximum value read in the third column, and as such it is not optimal. An optimal strategy can be found in Section 4 with 9 cost.

| $\nu(G)$ | $a$ | $c(a)$ |
|---|---|---|
| $\mathcal{CCCCC\ CCCCC}$ | 10000 10100 | 5 |
| $\mathcal{RCCCC\ BCBCC}$ | 00010 10101 | 6 |
| $\mathcal{RCCRC\ BCBCB}$ | 01100 11011 | 12 |
| $\mathcal{RRRRC\ BBRBB}$ | 00001 00011 | 7 |
| $\mathcal{RRRRR\ RRRBB}$ | 00000 00000 | 0 |
| $\mathcal{RRRRR\ RRRRR}$ | | |

Fig. 2. A Graph-Clear strategy.

Much of the prior work in graph-searching by Fomin and Thilikos [2008] focused on actions that are executed by a single searcher, while the GC model explicitly considers actions that require multiple searchers. An extension of edge-searching WES proposed in Barrière et al. [2002] also considers weighted actions. Despite the claim in Barrière et al. [2002] that WES can be solved on trees in polynomial time, it was later shown to be NP-hard even on trees in Dereniowski [2010].

The WES problem is also defined as an undirected weighted graph $G = (V, E, w)$ but with actions that allow multiple searchers to be 1) placed on a vertex or 2) to be moved along an edge, as presented by Barrière et al. [2002]. A vertex $v$ with $w(v)$ searchers placed on it is said to be *guarded* and prevents recontamination through all edges adjacent to this node. An edge $e$ with $w(e)$ searchers moving along it becomes clear after the move. For the purposes of this paper we consider connected searching (sometimes also named as contiguous search) to be a search in which the cleared edges and vertices always form a connected subgraph. The addition of weights by Barrière et al. [2002] is a straightforward extension of the classical edge-searching problem studied since Parsons [1976]. The type of actions and the spread of contamination are identical with those of GC, with the only exception that $w(v)$ or $w(e)$ searchers are necessary instead of one.

To provide an intuition on how WES works, we show a strategy in Fig. 3 that solves the WES problem for the example in Fig. 1. For simplicity, we assume that all robots take actions simultaneously. Note that in WES a guarded vertex effectively blocks the contamination from spreading and it is hence not obligatory in WES to block every connected edge when guarding a node. This key difference to GC in the formulation means that we may require fewer robots to clear the graph from Fig. 1, but it has implications with regard to the applicability on real robots. In practice, the guarding of a vertex requires the clearing of its associated area or room while simultaneous guaranteeing that no target can move through it from any of its edges to any other. In GC the sweeping of a vertex is only a clearing action and not a blocking action. As a consequence, some of the algorithms for visibility-based pursuit-evasion, e.g., the algorithms from Sachs et al. [2004], can be used in GC for sweeping rooms but cannot be used for guarding a room in WES. The former may require fewer searchers, but makes the implementation of vertex guarding more difficult for robots. The latter may require additional robots for blocking edges but simplifies the implementation of sweeping vertices.

| $\nu(G)$ | $a$ | $c(a)$ |
|---|---|---|
| $\mathcal{CCCCC\ CCCCC}$ | 00001 00000 | 3 |
| $\mathcal{CCCCR\ CCCCC}$ | 00000 00011 | 4 |
| $\mathcal{CCCCR\ CCCBB}$ | 00010 00010 | 5 |
| $\mathcal{CCCRR\ CCCBR}$ | 01010 00000 | 6 |
| $\mathcal{CRCRR\ CCCRR}$ | 01000 00100 | 6 |
| $\mathcal{CRCRR\ CCBRR}$ | 11000 00000 | 6 |
| $\mathcal{RRCRR\ CCRRR}$ | 00000 11000 | 2 |
| $\mathcal{RRCRR\ BBRRR}$ | 10000 00000 | 2 |
| $\mathcal{RRRRR\ RRRRR}$ | 00000 00000 | |

Fig. 3. A WES strategy with cost 6.

## 3. MODEL CHECKING BY MCMAS

MCMAS is a symbolic model checker developed for verification of multiagent systems [Lomuscio et al., 2009] and has been applied to many scenarios, e.g., contract [A. Lomuscio, 2012] and commitment [Bentahar et al., 2012]. In recent years, the control community has witnessed a growth of successful case studies using MCMAS [Ezekiel et al., 2011, Molnar and Veres, 2011]. MCMAS adopts the binary decision diagram (BDD) [Bryant, 1986] as the basis for verification algorithms. BDD allows a compact and unique representation for a Boolean formula, which is very effective in reducing memory consumption, and therefore, enables verification of real world systems. On top of BDD, various model checking techniques, such as symmetry reduction [Cohen et al., 2009a,b], abstraction [Lomuscio et al., 2010] and parallel computation [Kwiatkowska et al., 2010], have been developed for MCMAS to speed up verification computationally.

The MCMAS input language ISPL was designed particularly for modelling multiagent systems. An agent is composed of a set of *variables*, which record the local state of the agent, a set of *actions* the agent has, a *protocol* deciding when an action can be executed, and an *evolution function* specifying the transitions among the states. The variables are internal to the agent in the sense that other agents cannot observe their value. This constraint, although restrictive, makes the model very close the real world. Communications among agents is conducted by agents' actions.

In other words, when an agent makes a transition, the new local state it will locate in will be determined by not only its own action, but also other agents' action. The environment surrounding agents is modelled as a special agent *Environment*, whose variables are partitioned into two sets: one can be observed by normal agents, and the other cannot. The observable variables are part of the local states of each normal agent.

MCMAS generates a Cartesian product from the specification of the agents and Environment. The product is a transition system encoding the evolution of the system from the global point of view. The properties are checked in this product. MCMAS supports *Computation Tree Logic* (CTL) and other logics. In this paper, we only need CTL to search for a graph clearing strategy for robots. Formally, CTL can be defined by the following BNF grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid EX\ \varphi \mid EG\ \varphi \mid E(\varphi\ U\ \varphi),$$

where $p$ is an atomic proposition. $EX\varphi$ means that there exists a path $\rho$ starting from the current state $s$ such that $\varphi$ holds in the successor state of $s$; $EG\varphi$ specifies that all states in $\rho$ satisfy $\varphi$; $E(\varphi_1\ U\ \varphi_2)$ says that there exists a state $s'$ in $\rho$ satisfying $\varphi_2$ and all states before $s'$ satisfy $\varphi_1$. Some other operators that are used frequently in practice, such as temporal operator $F$ and universal path quantifier $A$, can be expressed by the combinations of $E$, $X$, $G$ and $U$. For example, $EF\varphi \equiv E(true\ U\ \varphi)$ asks for a path $\rho$ such that $\varphi$ holds in a state in $\rho$. $AG\varphi \equiv \neg EF\ \neg\phi$ requires that, in all path starting from $s$, every state satisfies $\varphi$. More details can be found in [Clarke et al., 1999].

Given a multiagent system that is composed of $n$ agents $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ and the Environment $\mathcal{A}_E$, the semantics of CTL is interpreted on its underlying transition system. We first present the formal definition of transition systems, and afterwards, the CTL semantics.

*Definition 2.* [Transtion system] A transition system $M$ is a tuple $\langle S, S_0, T, A, H\rangle$ such that

- $S$ is a finite set of states; A state $s \in S$ is composed of local states from agents and the Environment, i.e., $s = (l_e, l_1, \ldots, l_n)$, where $l_e$ is the local state of the Environment and $l_i$ the local state of agent $i$.
- $S_0 \subseteq S$ is a set of initial states;
- $T \subseteq S \times S$ is the transition relation; A transition $(s^i, s^j) \in T$ is composed of a set of local transitions $\{(l^i_e, l^j_e), (l^i_1, l^j_1), \ldots, (l^i_n, l^j_n)\}$ from the Environment and the agents, and each local transition is permitted by its agent's protocol and evolution function.
- $A$ is a set of atomic propositions;
- $H : S \to 2^A$ is a labelling function mapping states to the set of atomic propositions $A$. We denote the set of atomic propositions holding in state $s$ by $H(s)$.

A path in $M$ is a sequence of states $\rho = s_0, s_1, \ldots, s_n, \ldots$ such that for all $i \geq 0$, $(s_i, s_{i+1}) \in T$. The state at place $k$, i.e., $s_k$, is also written as $\rho(k)$.

*Definition 3.* [Satisfaction] Let $s \in S$ be a state in the transition system $M$. The satisfaction of a CTL formula $\varphi$ at $s$, written as $M, s \models \varphi$, is recursively defined as follows.

- $M, s \models p$ iff $p \in H(s)$;
- $M, s \models \neg\varphi$ iff it is not the case that $M, s \models \varphi$;
- $M, s \models \varphi_1 \wedge \varphi_2$ iff $M, s \models \varphi_1$ and $M, s \models \varphi_2$;
- $M, s \models EX\varphi$ iff there exists a path $\rho$ starting at $s$ such that $M, \rho(1) \models \varphi$.
- $M, s \models EG\varphi$ iff there exists a path $\rho$ starting at $s$ such that $M, \rho(k) \models \varphi$ for all $k \geq 0$;
- $M, s \models E(\varphi U \psi)$ iff there exists a path $\rho$ starting at $s$ such that for some $k \geq 0$, $M, \rho(k) \models \psi$ and $M, \rho(j) \models \varphi$ for all $0 \leq j < k$.

If for all initial states $s_0 \in S_0$, we have $M, s_0 \models \varphi$, then we say that $\varphi$ is satisfied in $M$, written as $M \models \varphi$. The set of all formulae, of which the satisfaction can be derived from $A$, will be denoted $\Phi \supset A$.

## 4. MODELLING GRAPH-CLEAR

We model all robots as a whole by a *Robots* agent, and all intruders by an *Intruders* agent. In addition, we need the *Environment* to control the execution order between *Robots* and *Intruders* and update the status of vertices and edges. We could model each robot individually by an agent, but this would blow up the state space and prevent large graphs from being verified.

To find the minimum number of robots required, we start by verifying the graph for $d \geq 1$ robots. If no strategy can be found for $d$ robots, then we try $d+1$ robots in the next round. This process is continued until a strategy is found [1]. Let $\mathcal{N}_{v_i}$ be the minimum number of robots needed to sweep vertex $v_i$ (which includes the robots blocking all adjacent edges), and $\mathcal{N}_G = \max\{\mathcal{N}_{v_i} \mid 1 \leq i \leq n\}$. It is apparent that at least $\mathcal{N}_G$ robots are required to clear the graph. Hence, we set $d$ to $\mathcal{N}_G$ in the first round.

### 4.1 Definition of the Environment agent

*Variables.* For each vertex $v_i$ (edge $e_i$, resp.) in the pursuit-evasion graph, we assign a variable $\mathtt{v}_i$ ($\mathtt{e}_i$, resp.) to record its state.

$$\mathtt{v}_i = \begin{cases} \mathcal{R} & \text{if the vertex is clear,} \\ \mathcal{C} & \text{if contaminated.} \end{cases}$$

$$\mathtt{e}_i = \begin{cases} \mathcal{R} & \text{if the edge is clear,} \\ \mathcal{C} & \text{if it is contaminated,} \\ \mathcal{B} & \text{if it is blocked.} \end{cases}$$

We also define a variable $\mathtt{nv}_i$ for vertex $v_i$ to indicate whether the vertex is being swept during deployment of agents (i.e., when it is the robots' turn to move). $\mathtt{nv}_i$ is reset to zero when the Environment updates the status of vertices and edges.

$$\mathtt{nv}_i = \begin{cases} \mathtt{1} & \text{if } v_i \text{ is being swept,} \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

The variable $\mathtt{ne}_i$ is defined for edge $e_i$ in the same way.

$$\mathtt{ne}_i = \begin{cases} \mathtt{1} & \text{if } e_i \text{ is being blocked,} \\ \mathtt{0} & \text{otherwise.} \end{cases}$$

We need a variable `turn` to schedule the turn of each agent: robots, intruders, or the Environment itself.

$$\mathtt{turn} = \begin{cases} \mathtt{robots} & \text{if it is the robots' turn,} \\ \mathtt{intruders} & \text{if it is the intruders' turn,} \\ \mathtt{env} & \text{if it is the Environment's turn,} \\ \mathtt{stop} & \text{if the graph is cleared completely.} \end{cases}$$

[1] This process will terminate in finite time as only a finite number of robots are needed to clear the graph.

Table 1. Protocol in Environment.

| | |
|---|---|
| robots | robots_actions |
| intruders | intruders_actions |
| env | env_actions |
| stop | null |

*Actions and Protocol.* The Environment have four actions: robots_actions, intruders_actions, env_action, and null. They are enabled according to the values of variable turn as shown in Table 1.

*Evolution function.* The evolution function defines how to update a variable $a$ individually. Let $a'$ denote the new value updated from $a$.

The evolution of variable turn is illustrated in Fig. 4. The ellipse node represents the initial value.

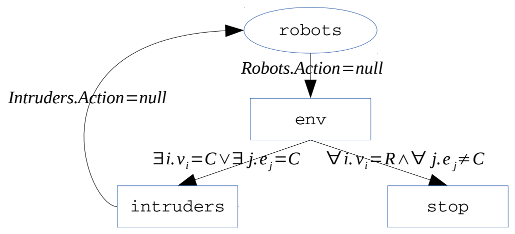

Fig. 4. Evolution of variable turn.

Fig. 5 demonstrates how the value of $\mathtt{v}_i$ and $\mathtt{e}_i$ is changed. An edge changes its status from $\mathcal{B}$ to $\mathcal{R}$ after robots blocking it leave the edge. In this figure, the formula $\psi_1$ is customised for each vertex $v_i$. Let $\bar{E} = \{\bar{e}_1, \ldots, \bar{e}_k\}$ be the set of adjacent edges of $v_i$, and $\overline{\mathtt{ne}}_j$ the corresponding ne variable for $\bar{e}_j$. The formula $\psi_1$ is defined as follows.

$$\psi_1 \equiv \bigwedge_{j=1}^{k} \overline{\mathtt{ne}}_j = 1. \tag{1}$$

The condition $\mathtt{nv}_i = 1 \wedge \psi_1$ expresses the requirement of sweeping operation on $v_i$: the vertex itself is swept by the robots and all entrances are blocked.

Fig. 6 demonstrates how the value of $\mathtt{nv}_i$ and $\mathtt{ne}_i$ is changed. $\mathtt{nv}_i$ is set to one when it is being swept by the robots and reset to zero by the Environment. An edge can be automatically blocked if one of its vertices is under sweep, or blocked by the robots on purpose. Therefore, the condition $\psi_2$ is defined as follows:

$$\begin{aligned}\psi_2 \equiv & Robots.Action = sweep\ v_j \vee \\ & Robots.Action = sweep\ v_k,\end{aligned} \tag{2}$$

where $v_j$ and $v_k$ are vertices that edge $e_i$ connects.

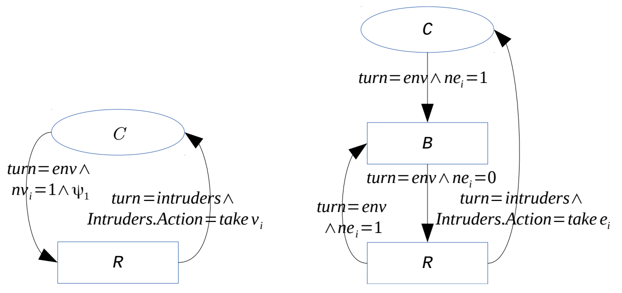
C
turn=env∧ nv_i=1∧ψ1
turn=intruders∧ Intruders.Action=take v_i
R
C
turn=env∧ne_i=1
B
turn=env∧ne_i=0
turn=env ∧ne_i=1
turn=intruders∧ Intruders.Action=take e_i
R

Fig. 5. Evolution of variable $\mathtt{v}_i$ (left) and $\mathtt{e}_i$ (right)

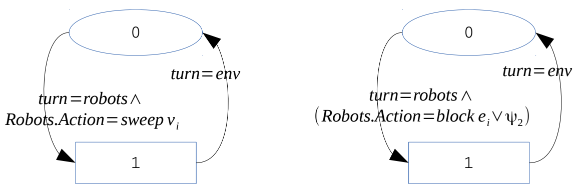
0
turn=env
turn=robots∧ Robots.Action=sweep v_i
1
0
turn=env
turn=robots∧ (Robots.Action=block e_i∨ψ2)
1

Fig. 6. Evolution of $\mathtt{nv}_i$ (left) and $\mathtt{ne}_i$ (right)

### *4.2 Definition of the Robots agent*

*Variables.* Only one variable is needed to represent the presence of robotic agents. It is defined as a bounded integer: $\mathtt{n} = 0..d$. The value of n records the number of robots available for deployment.

*Actions.* This agent has actions *sweep* $v_i$ to sweep vertex $v_i$ and actions *block* $e_j$ to block edge $e_j$. Action *null* is enabled when no other actions can be enabled.

*Protocol.* At the beginning of the clearing process, all vertices are contaminated, i.e.,

$$\begin{gathered} Environment.\mathtt{turn} = \mathtt{robots} \wedge \\ \mathtt{n} = d \wedge \sum_{i=1}^{n} Environment.\mathtt{v}_i = \mathcal{C}, \end{gathered} \tag{3}$$

and all *sweep* $v_i$ actions are enabled, while all *block* $e_j$ actions are disabled. This arrangement indicates that we can always begin with a sweeping action to clear the graph if there exits a strategy to do so. In other cases, *sweep* $v_i$ is enabled if vertex $v_i$ is contaminated, and one of its adjacent vertices $\bar{v}_j$ is clear, i.e.,

$$\begin{gathered} Environment.\mathtt{turn} = \mathtt{robots} \wedge \\ Environment.\mathtt{v}_i = \mathcal{C} \wedge \bigvee_{j=1}^{k} Environment.\bar{\mathtt{v}}_j = \mathcal{R}, \end{gathered} \tag{4}$$

where $\bar{\mathtt{v}}_1, \ldots, \bar{\mathtt{v}}_k$ are the adjacent vertices of $v_i$. Once a vertex is selected to be swept, the agent can choose to block some edges that are not connected to the vertex if there are robots available to do so. To guarantee contiguous strategies, we require that one of its end vertex needs to have been cleared already. However, the agent also has the right not to do any block actions if necessary. Formally, both *block* $e_j$ and *null* are enabled if

$$\begin{aligned} & Environment.\mathtt{turn} = \mathtt{robots} \wedge k \leq \mathtt{n} < d \wedge \\ & Environment.\mathtt{ne}_j = 0 \wedge \\ & ((Environment.\mathtt{v}_p = \mathcal{R} \wedge Environment.\mathtt{v}_q = \mathcal{C}) \vee \\ & (Environment.\mathtt{v}_p = \mathcal{C} \wedge Environment.\mathtt{v}_q = \mathcal{R})), \end{aligned} \tag{5}$$

where $v_p$ and $v_q$ are end vertices of $e_j$, and $k$ is the number of robots needed to block $e_j$. In all other cases, only *null* is enabled.

*Evolution function.* For each sweep action *sweep* $v_i$, the new value of n is defined as follows:

$$\mathtt{n}' := \mathtt{n} - k, \tag{6}$$

where $k$ is the number of robots needed to sweep $v_i$ and block all adjacent edges. For each block action *block* $e_j$, we have as follows:

$$\mathtt{n}' := \mathtt{n} - t, \tag{7}$$

where $t$ is the number of robots needed to block $e_i$. Note that when a vertex of $e_j$ is under sweep, *block* $e_j$ is disabled automatically. When no vertices need to be swept and no edges need to be blocked, i.e., when action *null* is enabled, `n` is reset to the initial value (the total number of robots).

### *4.3 Definition of the Intruders agent*

*Variables.* The Intruders agent has only one variable `recontamination`, which is of Boolean type. The truth value indicates whether recontamination occurs after the robots are deployed in each iteration.

*Actions.* This agent has actions *take* $v_i$ to recontaminate vertices $v_i$ and *take* $e_j$ to recontaminate edges $e_j$. In addition, action *null* means the agent cannot recontaminate vertices or edges any more in each iteration.

*Protocol* Action *take* $v_i$ is enabled if

$$\mathit{Environment}.\mathtt{turn} = \mathtt{intruders} \wedge$$
$$\mathit{Environment}.\mathtt{v}_i = \mathcal{R} \wedge \bigvee_{j=1}^{k} \mathit{Environment}.\bar{\mathtt{e}}_j = \mathcal{C},$$

where $\bar{e}_1, \ldots, \bar{e}_k$ are adjacent edges. Similarly, *take* $e_j$ is enabled if

$$\mathit{Environment}.\mathtt{turn} = \mathtt{intruders} \wedge \mathit{Environment}.\mathtt{e}_j = \mathcal{R} \wedge$$
$$(\mathit{Environment}.\bar{\mathtt{v}}_1 = \mathcal{C} \vee \mathit{Environment}.\bar{\mathtt{v}}_2 = \mathcal{C})$$

where $\bar{v}_1$ and $\bar{v}_2$ are adjacent vertices.

*Evolution function.* `recontamination` is *true* when recontamination occurs, and *false* once the Intruders agent finishes recontamination, as illustrated in Fig. 7.

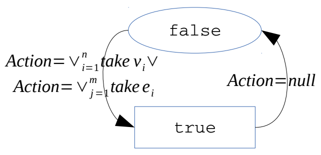
false
Action= $\vee_{i=1}^{n}$ take $v_i$ $\vee$
Action= $\vee_{j=1}^{m}$ take $e_i$
Action=null
true

Fig. 7. Evolution of variable `recontamination`

Note that the turn-based scheduling allows efficient verification, but it does not preserve the unbounded speed of recontamination. However, as stated in Theorem 1, the intruders are not able to recontaminate the graph in the middle of deployment of the robots even with the unbounded speed.

### *4.4 Specification*

We first define two atomic propositions in order to construct a strategy specification:

- *recontaminated* holds whenever `recontamination` in the Intruders agent is set to *true*.
- *graph_cleared* becomes *true* when all vertices and edges are free of contamination, i.e.,

$$\bigwedge_{i=1}^{n} \mathtt{v}_i = \mathcal{R} \wedge \bigwedge_{j=1}^{m} (\mathtt{e}_j = \mathcal{R} \vee \mathtt{e}_j = \mathcal{B}).$$

We ask MCMAS to generate a clearing strategy using the following CTL formula.

$$E(\neg \mathit{recontaminated}\ U\ \mathit{graph\_cleared}). \quad (8)$$

This formula specifies whether there exists a path that clears the whole graph without incurring any recontamination. Such a path is a clearing strategy for the robots.

The following theorem shows the correctness of the ISPL model constructed in this section.

*Theorem 1.* If formula (8) is satisfied by the model, then every path satisfying it is a contiguous strategy for the robots to clear the graph, and no recontamination can occur during the clearing process.

*Proof 1.* Contiguity is guaranteed by the protocol in the Robots agent. We begin with a close look at the clearing process to prove that recontamination cannot occur.

When the clearing process begins, it is the robots' turn to deploy, i.e., choose a vertex to sweep (including blocking the adjacent edges) and some extra edges to block if possible. Once the deployment is complete, the Environment set the status of vertices and edges accordingly. Next, it is the Intruders' turn to recontaminate the graph when possible. After the recontamination is done, a new iteration begins just like the cycle in Fig. 4 suggests.

The $U$ operator in Formula (8) eliminates the possibility of recontamination after the deployment of the robots is complete. Now we need to show there is no chance to recontaminate the graph during the deployment process, which can be done by induction. At the beginning of the clearing process, all vertices and edges are contaminated. Hence the first deployment does not cause recontamination.

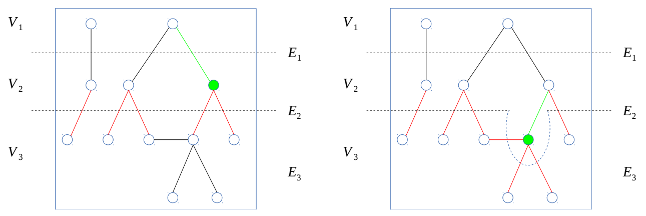

Fig. 8. Clearing step $i$ (left) and $i + 1$ (right)

Assume no recontamination occurs up to the $i$-th deployment, which can be illustrated in Fig. 8. We partition vertices into three sets.

- $V_1$ contains cleared vertices that do not have contaminated adjacent vertices;
- $V_2$ contains cleared vertices that have contaminated adjacent vertices;
- $V_3$ contains contaminated vertices.

Recall that in each deployment, we choose a vertex to sweep. Such a vertex is marked in green in the figure. Similarly, the edges are partitioned into three sets as well.

- $E_1$ contains cleared edges that do not have contaminated adjacent vertices;
- $E_2$ contains blocked edges (in red) that have a contaminated adjacent vertex, and blocked edges (in green) due to the requirement of sweeping;
- $E_3$ contains contaminated edges.

Note that our model does not allow robots to block edges in $E_1$. If an edge in $V_2$ is not blocked, then it can be recontaminated between the $i$-th and $(i + 1)$-th deployment. This situation has been taken care of by the $U$ operator.

In the $(i + 1)$-th deployment, we first choose a vertex in $V_3$ to sweep, which is shown in Fig. 8 on the right. As all edges in $E_2$ in the $i$-th deployment are still blocked, which means that the robots occupying those edges do not move, no recontamination can occur during the $(i+1)$-th deployment. □

*Example* Fig. 9 displays a clearing strategy for the example in Section 2 produced by MCMAS. The experiment was carried out in a computer with dual Intel Xeon E5-2630 processors and 32 GB memory. The reachable state space in this model is 2640, and it took MCMAS less than one second to compute the strategy.

| $\nu(G)$ | $a$ | $c(a)$ |
|---|---|---|
| $\mathcal{CCCCC}\ \mathcal{CCCCC}$ | 00001 00011 | 7 |
| $\mathcal{CCCCR}\ \mathcal{CCCBB}$ | 00010 00111 | 8 |
| $\mathcal{CCCRR}\ \mathcal{CCBBB}$ | 10000 10110 | 8 |
| $\mathcal{RCCRR}\ \mathcal{BCBBR}$ | 01000 11010 | 9 |
| $\mathcal{RRCRR}\ \mathcal{BBRBR}$ | 00100 01000 | 3 |
| $\mathcal{RRRRR}\ \mathcal{RBRRR}$ | | |

Fig. 9. A Strategy for Graph-Clear.

## 5. MODELLING WEIGHTED EDGE-SEARCHING

Just as for modelling Graph-Clear, we put all robots in the *Robots* agent, all intruders in the *Intruders* agent, and the graph in the *Environment*. We also verify the graph iteratively starting from $d$ robots. However, $d$ is not equal to $\mathcal{N}_G$ anymore. Instead, it is the maximum weight a vertex can have.

To find a strategy using WES to clear the graph, we define the same set of variables as when modelling GC: $\mathtt{v}_i$, $\mathtt{e}_i$, $\mathtt{nv}_i$, $\mathtt{ne}_i$, and $\mathtt{turn}$. The variables have the same domain as well, except that $\mathtt{v}_i$ has one more value.

$$\mathtt{v}_i = \begin{cases} \mathcal{R} & \text{if } v_i \text{ is clear,} \\ \mathcal{C} & \text{if } v_i \text{ is contaminated,} \\ \mathcal{G} & \text{if } v_i \text{ is guarded.} \end{cases}$$

The evolution of variables $\mathtt{turn}$, and $\mathtt{e}_i$ is the same as before. The evolution of $\mathtt{v}_i$, shown in Fig. 10 is now close to $\mathtt{e}_i$ as $\mathtt{v}_i$ has three values.

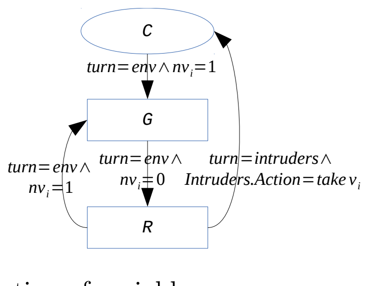

Fig. 10. Evolution of variable $\mathtt{v}_i$

The evolution of $\mathtt{nv}_i$ and $\mathtt{ne}_i$ is changed slightly. There is no $\psi_2$ in the enabling condition of the transition from $\mathtt{ne}_i = 0$ to $\mathtt{ne}_i = 1$, as a guard action does not enforce block actions any more. The definition of the Intruders agent is the same as that in Graph-Clear in Section 4.

### 5.1 Definition of the Robots agent

This agent is quite similar to that in GC. The variables are the same; the actions are the same as well except *sweep* $v_i$ is replaced by *guard* $v_i$. The evolution function is defined

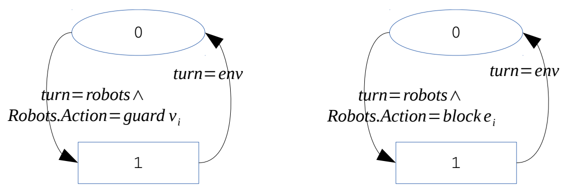
0
turn=env
turn=robots∧ Robots.Action=guard v_i
1
0
turn=env
turn=robots∧ Robots.Action=block e_i
1

Fig. 11. Evolution of $\mathtt{nv}_i$ (left) and $\mathtt{ne}_i$ (right)

in the same way too except that when executing an action *guard* $v_i$, only the number of robots needed for guarding the vertex is reduced from the robots pool. The protocol also reflects this difference, as well as the change of the value domain of $\mathtt{v}_i$.

*Protocol.* When the whole graph is contaminated, all *guard* $v_i$ actions are enabled, while all *block* $e_j$ actions are disabled. This constraint is the same as in GC, as it is still possible to clear the graph by clearing nodes sequentially.

After the clearing process starts, *guard* $v_i$ is enabled at the beginning of each clearing iteration if the following condition is satisfied:
$Environment.\mathtt{turn} = \mathtt{robots} \wedge \mathtt{n} = d \wedge$ $((Environment.\mathtt{v}_i = \mathcal{C} \wedge \psi_3) \vee (Environment.\mathtt{v}_i = \mathcal{G} \wedge \psi_4))$, where $\psi_3$ encodes that an edge connected to $v_i$ is blocked, and $\psi_4$ encodes that an edge has been cleared. Formally,

$$\psi_3 = \bigvee_{j=1}^{k} Environment.\bar{\mathtt{e}}_j = \mathcal{B},$$

$$\psi_4 = \bigvee_{j=1}^{k} Environment.\bar{\mathtt{e}}_j = \mathcal{C},$$

where $\bar{e}_1, \ldots, \bar{e}_k$ are edges connected to $v_i$. The condition $Environment.\mathtt{v}_i = \mathcal{C} \wedge \psi_3$ indicates that $v_i$ can be guarded if it is contaminated and the robots can enter it via a blocked edge, which guarantees the contiguity. The condition $Environment.\mathtt{v}_i = \mathcal{G} \wedge \psi_4$ allows $v_i$ to remain guarded to avoid recontamination, as one of its edges is contaminated. In other cases, we do not allow the robots to guard $v_i$ in order to reduce the number of possible strategies and speed up verification.

In WES, sometimes it is necessary to allow the robots to only block edges in certain iterations in order to minimise the number of robots needed. The enabling condition of *block* $e_i$ is defined as follows:

$$\begin{aligned} &Environment.\mathtt{turn} = \mathtt{robots} \wedge \mathtt{n} \geq k \\ &\wedge Environment.\mathtt{ne}_i = 0 \\ &((Environment.\mathtt{v}_p \neq \mathcal{C} \wedge Environment.\mathtt{v}_q = \mathcal{C}) \vee \\ &(Environment.\mathtt{v}_p = \mathcal{C} \wedge Environment.\mathtt{v}_q \neq \mathcal{C})), \end{aligned} \tag{9}$$

where $k$ is the number of robots needed to block the edge, and $v_p$ and $v_q$ are end vertices of $e_i$. The last two lines in the condition enforces the contiguity. Note that action *null* is also enabled with *block* $e_i$ to give robots freedom to choose whether to block $e_i$. Indeed, this freedom can avoid unnecessary movement/deployment of robots.

### 5.2 Specification

We use the same CTL formula as before to look for a strategy to clear the graph. Similar to Section 4, we can prove that Theorem 2 holds for this model.

*Theorem 2.* If formula (8) is satisfied by the model, then every path satisfying it is a contiguous strategy for the robots to solve the weighted edge-searching problem, and no recontamination can occur during the clearing process.

*Example* Fig. 12 shows a clearing strategy for the example in Section 2 using WES. Similarly, this strategy was computed within two seconds for 7029 reachable states.

| $\nu(G)$ | $a$ | $c(a)$ |
|---|---|---|
| $\mathcal{CCCCC}$ $\mathcal{CCCCC}$ | 00001 00000 | 3 |
| $\mathcal{CCCCG}$ $\mathcal{CCCCC}$ | 00000 00011 | 4 |
| $\mathcal{CCCCR}$ $\mathcal{CCCBB}$ | 01000 00001 | 5 |
| $\mathcal{CGCCR}$ $\mathcal{CCCRB}$ | 00010 11001 | 5 |
| $\mathcal{CRCGR}$ $\mathcal{BBCRB}$ | 10000 01100 | 5 |
| $\mathcal{GRCRR}$ $\mathcal{RBBRR}$ | 00100 01000 | 3 |
| $\mathcal{RRGRR}$ $\mathcal{RBRRR}$ | | |

Fig. 12. A Strategy for Weighted Edge-Searching.

## 6. CONCLUSION

This paper lays the foundations of using model checking to find strategies for different pursuit-evasion models and to describe and compare their properties. In addition, it offers the potential to consider additional criteria other than the number of searchers for the computation of strategies, such as time or constraints on the order of vertices.

We have shown how to express two pursuit-evasion models in our framework and used the multiagent model checker MCMAS to find solutions to the Graph-Clear and the Weighted Edge Searching problems.

Two main theorems have been presented which guarantee the correctness of the methods proposed. In addition, we discussed how the model checking methods proposed enable a comparison of algorithmic complexity of the different pursuit-evasion models.

To conclude, the presented work can be a starting point for further investigations regarding a more rigorous and unified approach to understand, compare and design pursuit-evasion models.

## REFERENCES

M. Solanki A. Lomuscio, H. Qu. Towards verifying contract regulated service composition. *Autonomous Agents and Multi-Agent Systems*, 24(3):345–373, 2012.

L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *Proceedings of SPAA'02*, pages 200–209. ACM Press, 2002.

J. Bentahar, M. El-Menshawy, H. Qu, and R. Dssouli. Communicative commitments: Model checking and complexity analysis. *Knowl.-Based Syst.*, 35:21–34, 2012.

S. D. Bopardikar, F. Bullo, and J. P. Hespanha. Cooperative pursuit with sensing limitations. In *American Control Conference*, pages 5394–5399, 2007.

R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, 35(8): 677–691, 1986.

T.H. Chung, G.A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.

E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A symmetry reduction technique for model checking temporal-epistemic logic. In *Proc of IJCAI'09*, pages 721–726, 2009a.

M. Cohen, M. Dam, A. Lomuscio, and H. Qu. A data symmetry reduction technique for temporal-epistemic logic. In *Proc of ATVA'09*, volume 5799 of *LNCS*, pages 69–83. Springer, 2009b.

D. Dereniowski. Connected searching of weighted trees. *Mathematical Foundations of Computer Science 2010*, pages 330–341, 2010.

J. Ezekiel, A. Lomuscio, L. Molnar, and S. M. Veres. Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In *Proceedings of IJCAI'11*, pages 1659–1664. IJCAI/AAAI, 2011.

F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *Proceedings of IROS'08*, pages 2323–2328, 2008.

A. Kolling and S. Carpin. Surveillance strategies for target detection with sweep lines. In *Proceedings of IROS'09*, pages 5821–5827, 2009.

A. Kolling and S. Carpin. Multi-robot pursuit-evasion without maps. In *Proceedings of ICRA'10*, pages 3045–3051, 2010a.

A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE T. Robot.*, 26(1):32–47, 2010b.

A. Kolling and A. Kleiner. Multi-uav motion planning for guaranteed search. In *Proceedings of AAMAS'13*, pages 79–86, 2013.

M. Kwiatkowska, A. Lomuscio, and H. Qu. Parallel model checking for temporal epistemic logic. In *Proc of ECAI'10*, pages 543–548. IOS Press, 2010.

A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc of CAV'09*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

A. Lomuscio, H. Qu, and F. Russo. Automatic data-abstraction in model checking multi-agent systems. In *Proc of MoChArt'10*, volume 6572 of *LNCS*, pages 52–68. Springer, 2010.

L. Molnar and S. M. Veres. Hybrid automata dicretising agents for formal modelling of robots. In *Proc of the 18th IFAC World Congress*, pages 49–54. IFAC, 2011.

M. Moors, T. Röhling, and D. Schulz. A probabilistic approach to coordinated multi-robot indoor surveillance. In *Proceedings of IROS'08*, pages 3447–3452, 2005.

L.E. Parker. Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous Robots*, 12:231–255, 2002.

T.D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Applications of Graphs*, volume 642, pages 426–441. Springer, 1976.

S. Sachs, S. M. LaValle, and S. Rajko. Visibility-based pursuit-evasion in an unknown planar environment. *Int. J. Robot. Res.*, 23(1):3–26, 2004.

I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, 1992.