Model Checking Knowledge in Pursuit Evasion Games *

Xiaowei Huang
Computer Science and Engineering,
University of New South Wales, Australia

Patrick Maupin
Defence R&D Canada
Valcartier, Quebec

Ron van der Meyden
Computer Science and Engineering,
University of New South Wales, Australia

Abstract

In a pursuit-evasion game, one or more pursuers aim to discover the existence of, and then capture, an evader. The paper studies pursuit-evasion games in which players may have incomplete information concerning the game state. A methodology is presented for the application of a model checker for the logic of knowledge and time to verify epistemic properties in such games. Experimental results are provided from a number of case studies that validate the feasibility of the approach.

1 Introduction

Pursuit-Evasion games are a type of multi-player game in which one or more pursuers have the objective of identifying the presence of one or more evaders, and of capturing them. Solutions to pursuit-evasion problems have multiple applications, including air and naval combat, ship navigation, [Isaacs, 1965], automatic car collision avoidance systems [Lachner *et al.*, 2000], air traffic control, and unmanned aerial vehicle control [Vidal *et al.*, 2002]. *Epistemic model checking* is an approach to automated verification, in which one checks whether a model satisfies a formula that describes how the knowledge of agents evolves over time.

Our contribution in this paper is to develop examples of the use of epistemic model checking in pursuit-evasion problems. In particular, we identify specifications requiring the use of the expressiveness provided by epistemic logic, and conduct some experiments in which we apply the model checker MCK [Gammie and van der Meyden, 2004] on some small scale examples. In particular, we compare the performance of two epistemic model checking approaches provided by MCK: BDD based model checking and bounded model checking.

We define the pursuit-evasion games with incomplete information that we study in Section 2. We explain epistemic model checking in Section 3. Section 4 studies the most basic question in pursuit-evasion games – can the pursuers capture an evader – in the context of full-information games. In Section 5, we consider games where the players do not have full visibility: here we argue that epistemic specifications are of particular interest. We discuss related work in Section 6, and conclude with ideas for future work in Section 7.

*Work supported by Australian Research Council Linkage Grant LP0882961 and Defence Research and Development Canada (Valcartier) contract W7701-082453.

2 Pursuit-Evasion Games

The pursuit-evasion games we consider in this paper are discrete games with limited visibility in which the agents have complete knowledge of the map, which is represented as a graph, with players able to move to an adjacent node in each step. Our approach involves consideration of a number of different memory assumptions.

More precisely, each game is played on a map $\mathcal{M} = (V, E_m, E_v)$ where V is a set of vertices (or positions), $E_m \subseteq V \times V$ is a set of adjacency edges, along which players can move, and $E_v \subseteq V \times V$ is a set of edges indicating visibility: a player at position u can see another player at position v just in case $(u, v) \in E_v$. Most of our examples use undirected graphs, so that both relations E_m, E_v are symmetric. Generally, we also have that E_m is reflexive, so that staying at the same location is a valid move.

A game state on a map $\mathcal{M} = (V, E_m, E_v)$ consists of a tuple $(A_P, A_E, posn, turn)$, where A_P is a set of pursuers, A_E is a set of evaders, $posn : A_P \cup A_E \to V$ is a function giving the location of each pursuer and evader, and $turn \subseteq A_P \cup A_E$ indicates the set of scheduled agents. A *game* is given by a tuple $\mathcal{G} = (\mathcal{M}, I, sched)$, where $\mathcal{M}$ is a map, I is a set of initial game states on M, and *sched* is a scheduler, which maps each finite play of the game to a set of agents: the agents that enabled to make a move in the next step. We consider two types of schedulers in this paper. The *synchronous* scheduler enables all players at all moves, and the *turn-based* scheduler alternates scheduling of all pursuers with scheduling of all evaders. We consider a variety of game objectives in our examples, so do not include winning conditions in this definition of a game.

A play of the game consists of a finite or infinite sequence $s_0, s_1, \ldots$ where $s_0 \in I$, and at each step i, if $s_i = (A_P, A_E, posn, turn)$, and $s_{i+1} = (A'_P, A'_E, posn', turn')$, then $A'_P = A_P$, $A'_E = A_E$, $turn'$ is determined from $turn$ by the scheduler rule *sched*, and the scheduled agents move across an edge in the adjacency graph, i.e., for each agent $a \in turn$, we have $(posn(a), posn'(a)) \in E_m$, and for each agent $a \in (A_P \cup A_E) \setminus turn$, we have $posn'(a) = posn(a)$.

In each game state, each player a makes an *observation* of the game state. This is captured by an observation function O_a with domain the set of game states. We define $O_a((A_P, A_E, posn, turn)) = (A_P, A_E, posn_a, turn)$, where $posn_a(b) = posn(b)$ if player b is either equal to a or visible from player a's position, i,.e., $(posn(a), posn(b)) \in E_v$, and $posn_a(b) = \perp$ otherwise, indicating an unknown position. The player's *perfect recall view* of a play $s_0 s_1 \ldots$ is the sequence of observations $view_a(\alpha) = O_a(s_0), O_a(s_1), \ldots$.

A *strategy* for player a is a function σ_a mapping each possible view α of the player, ending in an observation $(A_P, A_E, posn_a, turn)$ where that player is scheduled to move (i.e. $a \in turn$), to a set of positions $\sigma_a(\alpha)$ to which that player is able to move, i.e., such that $(posn_a(a), x) \in E_m$ for all $x \in \sigma_a(\alpha)$. Note that the strategy maps to a set of positions in order to allow the strategy to be to make a random move. A *strategy assignment* is a mapping σ associating a strategy σ_a to each agent $a \in A_P \cup A_E$. A play $s_0 s_1 \ldots$ is *consistent* with a strategy assignment σ if for each player a and each step from $s_i = (A_P, A_E, posn, turn)$ to $s_{i+1} = (A_P, A_E, posn', turn')$, if a moves in this step (i.e. $a \in turn$) then it moves to one of the positions selected by its strategy (i.e., $posn'(a) \in \sigma_a(view_a(s_0 \ldots s_i))$).

3 Epistemic Model Checking

Model checking [Clarke *et al.*, 1999] is an automated method for formal verification, in which an algorithm is used to check whether a specification expressed in a formal logic holds in a particular model that represents the system to be verified. Model checking is well developed for temporal logic specifications and finite state automata models. Epistemic logic [Fagin *et al.*, 1995] provides a formal language in which one can specify how the information possessed by agents in a distributed or multi-agent system changes over time. In recent years, a number of model checkers have been developed for specifications that combine temporal and epistemic expressiveness [Gammie and van der Meyden, 2004; Lomuscio *et al.*, 2009; Eijck, 2004; Su *et al.*, 2007].

We work in this paper with the epistemic model checker MCK, which supports a specification language that combines operators from the branching time logic CTL* with operators from the logic of knowledge. The fragment relevant to our purposes has syntax given by the following grammar:

$$\phi \;=\; p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid A\phi \mid X\phi \mid G\phi \mid K_i\phi$$

where p is an element of the set Prop of atomic propositions and i is an element of the set of agents $Agt = A_P \cup A_E$. Intuitively, $A\phi$ says that ϕ holds in all possible futures, $X\phi$ says that ϕ holds at the next moment of time, $G\phi$ says that ϕ holds now and at all future times, and $K_i\phi$ says that agent i knows that ϕ is true. Other operators that can be defined from this set include $E\phi = \neg A\neg\phi$ (ϕ holds in some possible future), and $F\phi = \neg G\neg\phi$ (ϕ holds at some future time).

Semantics of the logic can be given using *interpreted systems* [Fagin *et al.*, 1995]. Let S be a set, which we call the set of environment states. A *run* over environment states S is a function $r : \mathbf{N} \to S \times \Pi_{i \in Agt} L_i$, where each L_i is some set, called the set of *local states of agent i*. These local states are used to concretely represent the information on the basis of which agent i computes its knowledge. Given a run r, agent i and time m, we write $r_i(m)$ for the corresponding component (in L_i) of $r(m)$, and $r_e(m)$ for the first component (in S). An *interpreted system* over environment states S is a tuple $\mathcal{I} = (\mathcal{R}, \pi)$, where $\mathcal{R}$ is a set of runs over environment states S, and $\pi : \mathcal{R} \times \mathbf{N} \to \mathcal{P}(Prop)$ is an interpretation function.

A *point* of $\mathcal{I}$ is a pair (r, m) where $r \in \mathcal{R}$ and $m \in \mathbf{N}$. The semantics of CTL*K$_n$ is given by a relation $\mathcal{I}, (r, m) \models \phi$, where $\mathcal{I}$ is an intepreted system, (r, m) is a point of $\mathcal{I}$, and ϕ is a formula, defined inductively as follows (we omit details for the boolean operators):

- $\mathcal{I}, (r, m) \models p$ if $p \in \pi(r, m)$,
- $\mathcal{I}, (r, m) \models A\phi$ if for all runs $r' \in \mathcal{R}$ with $r'(k) = r(k)$ for all $k = 0 \ldots m$, we have $\mathcal{I}, (r', m) \models \phi$,
- $\mathcal{I}, (r, m) \models X\phi$ if $\mathcal{I}, (r, m+1) \models \phi$,
- $\mathcal{I}, (r, m) \models G\phi$ if for all times $m' \geq m$ we have $\mathcal{I}, (r, m') \models \phi$,
- $\mathcal{I}, (r, m) \models K_i\phi$ if for all points (r', m') of $\mathcal{I}$ such that $r_i(m) = r'_i(m')$ we have $\mathcal{I}, (r', m') \models \phi$.

The systems of interest in this paper have runs derived from all plays of a game $\mathcal{G}$, and local states assigned using what MCK call the *observational semantics*. We allow agents to retain some memory of their past observations, represented by a memory assignment μ that associates to each agent a a function μ_a mapping perfect recall views of a to some set of memory values. We require that this function have the following Markov property: there exists a function f such that for all views α and observations o, $\mu_a(\alpha o) = f(\mu_a(\alpha), o)$. This permits memory to be incrementally maintained.

Suppose $\rho = s_0 s_1 \ldots$ is an infinite play of the game consistent with strategy assignment σ. Given a memory assignment μ, we construct a run $r = \rho^{\mathsf{obs},\mu}$ by defining $r_e(m) = s_m$ and $r_i(m) = (O_i(s_m), \mu_i(view_i(s_0 \ldots s_{m-1})))$ for all $i \in Agt$ and $m \in \mathbf{N}$. That is, each player's local state is its current observation, plus its memory of its previous observations. We write $\mathcal{I}^{\mathsf{obs}}(\mathcal{G}, \sigma, \mu)$ for the system in which the set of runs consists of $\rho^{\mathsf{obs},\mu}$ for all infinite plays ρ of $\mathcal{G}$ consistent with σ.

The propositions of interest for our analysis are as follows:

- $caught_e$: the evader e has been caught. In case of the turn-based scheduler, this means that for some pursuer $p \in A_P$, we have $posn(p) = posn(e)$. When we use the synchronous scheduler, it is possible that a pursuer and an evader traverse an edge in opposite directions in the same step, i.e., $posn'(p) = posn(e) \wedge posn'(e) = posn(p)$. In this case, this is also treated as a capture.
- $pos_i \in X$, where $i \in Agt$ and $X \subseteq V$ is a set of game positions. This is true just in case $posn(i) \in X$.

The interpretation π is constructed so as to associate these meanings with these variables.

Given a finite state environment E, and an assignment of protocols $\mathbf{P}$ being executed by the agents in this environment, and a formula ϕ of the logic of knowledge and time, MCK builds an interpreted system $\mathcal{I}$, and checks whether ϕ holds at time 0 in all runs of $\mathcal{I}$, i.e., whether $\mathcal{I}, (r, 0) \models \phi$ for all runs r of $\mathcal{I}$. In the applications of interest in the present paper,

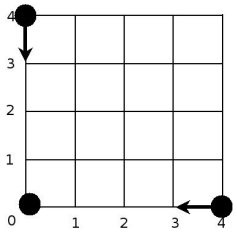

Figure 1: Grid Scenario

the environment corresponds to a game $\mathcal{G}$, the assignment c protocols corresponds to the strategy assignment σ, and the system $\mathcal{I}$ is $\mathcal{I}^{\mathsf{obs}}(\mathcal{G}, \sigma, \mu)$ for some memory assignment μ.

MCK supports several different algorithmic approaches for solving the model checking problem. We use two in the present paper: 1) Ordered binary decision diagram (BDD) based model checking, (MCK's algorithms `spec_obs_ctl` and `spec_obs_ctls`) and 2) Bounded model checking (BMC) (MCK's algorithm `spec_obs_bmc`), which is based on translation to a propositional logic satisfiability problem and works on the universal fragments of the logic by finding a counterexample in case the specification fails. MCK supports diagnosis by listing and visualization of counterexamples.

In the following sections, we present examples of the application of epistemic model checking using MCK where the system is a pursuit-evasion game. Our focus is to identify epistemic specifications of interest in this domain, and to obtain some preliminary results on the performance of MCK on these examples. The experiments were conducted on an Apple iMac with a 3.06GHz Intel i3 processor and 4G memory.

4 Pursuit Verification

The first example models a full-information pursuit-evasion game where the arena is a (m×n)-grid, and two pursuers chase an evader. In a full-information game, agents can observe the whole system, Thus, the map $\mathcal{M} = (V, E_m, E_v)$ is given by $V = [0 \dots m-1] \times [0 \dots n-1]$, $E_m = \{((x,y),(x',y')) \in V^2 \mid (x = x' \vee y = y') \wedge |x - x'| \leq 1 \wedge |y - y'| \leq 1\}$ and $E_v = V \times V$. Figure 1 presents the case of a (5×5)-grid. The evader is initialized at position $(0, 0)$ and the pursuers are initialized at position $(4, 0)$ and $(0, 4)$, respectively.

We consider two variants of the game, with strategies depending on the type of scheduler used. The strategy of the evader is always to move randomly to an adjacent vertex.

- synchronous scheduling: The pursuers choose any move in the direction of the current evader position.
- turn-based: The pursuers execute a pincer movement: one prioritizes moving towards the horizontal position of the evader, the other prioritizes the vertical. Once horizontally/vertically aligned, the pursuer may move towards the evader in the other dimension.

The point of these examples is to demonstrate some basic pursuit issues: is it possible for the evader to avoid capture when the pursuers are following the particular strategy prescribed for them? Thus, the formula

$$AF\ caught_e \tag{1}$$

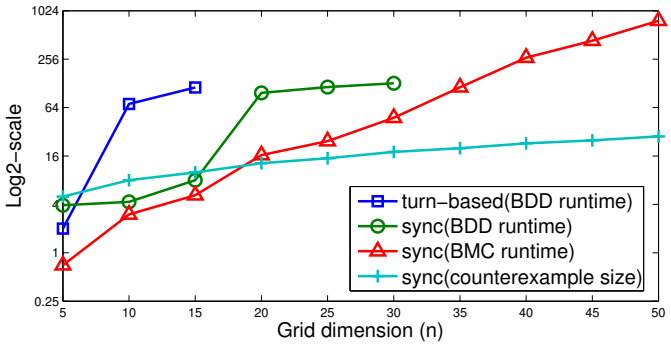

Figure 2: Grid Pursuit, Specification (1)

expresses that the evader cannot avoid capture (on all paths there exists a future time where the evader has been caught).

We may model check this using BDD-based model checking. Alternately, since this is a universal formula, we may also use bounded model checking. In this case, if the formula is false and a large enough bound N is selected, then the model checker returns a counterexample trace that satisfies $G\neg caught_e$, i.e. on which the evader avoids capture.

In the turn-based game, model checking verifies that capture is guaranteed. However, in the synchronous game, the specification fails to hold: the evader can escape capture if the pursuers choose their moves poorly. The counterexample facilities can be used to investigate the failure, and reveal the following pattern of movement in which the evader avoids capture in a loop such as $((1,2),(0,1),(2,1)) \rightarrow ((1,1),(0,2),(2,2)) \rightarrow ((1,2),(0,1),(2,1))$ with the triples $(posn(e), posn(p_1), posn(p_2))$ denoting the positions of evader and pursuers.

The experimental results for model checking specification (1) in the grid game are shown in Figure 2, which gives a logscale plot of model checking runtime against the grid dimension n for an $n \times n$ grid. The specification is false in the synchronous case, and we also plot the counter-example size in number of steps: this turns out to be roughly half the grid size plus 4. Timing results are omitted in the case of BMC and the turn-based game, since there exists no counterexample to be found, so the BMC approach does not terminate (MCK does not attempt to determine a sufficient bound for termination of BMC). In this example, as we increase the problem size, the BMC-based model checking scales well, while BDD model checking runtime becomes large (we terminated the computation at 2 hours) at thresholds that appear to be related to an increase in the number of bits required to represent the dimension.

Note that this analysis uses temporal specifications only, so any temporal logic model checker could be used to do the same analysis. The main point of these examples is to check how the two algorithm types compare and how well the epistemic model checker MCK runs as we scale the examples. It appears the approach is feasible for moderate sized examples.

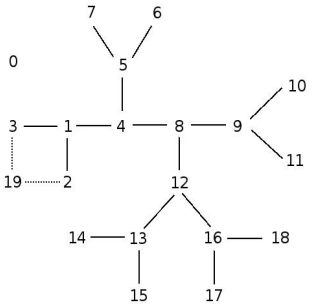

Figure 3: Region Clearing Scenario

5 Region Clearing and Pursuit Verification

We now consider some examples where the agents do not have complete visibility, and some game objectives become of interest, for which epistemic model checking is required, since temporal model checking does not suffice.

In all the games in this section, a single pursuer chases a single evader, and scheduling is turn-based. The map in these examples is based on variants of the undirected movement graph (V, E_m) depicted in Figure 3. (We explain the dotted edges below.) Initially, the pursuer is at position 1, and the evader can be at any position, including a special position 0, which indicates that the pursuer is not present in the game. Let Q be the set of all positions other than 0.

The visibility graph in the examples is derived from Figure 3 by $(u, v) \in E_v$ if u and v are connected in the figure by a sequence of edges forming a straight line (e.g., we have $(3, 9), (8, 12) \in E_v$, but $(3, 11), (5, 8) \notin E_v$).

In these examples, the evader's strategy is to move randomly through the graph, and we define a particular strategy for the pursuer, to be verified for correctness. The pursuer's strategy is defined in two parts, corresponding to two game objectives: first, to determine whether the evader is present in the region Q, and, second, to capture the evader once it is known to be present. The first part of the strategy is constructed so that the pursuer moves along a prescribed surveillance route, until the evader is seen. At this point the strategy switches to the second part, in which the pursuer gives chase by moving, at each turn, towards the location where it last saw the evader. For example, suppose the pursuer is at position 1, it sees the evader is at position 8, and it is the evader's turn to move. If the evader moves to position 12 it is no longer in view of the pursuer, so the pursuer's move would be to position 4, in the direction it last saw the evader.

In this set of examples, the memory assignment for the evader is the null function, and the memory assignment for the pursuer always retains the position where the evader was last seen (else $\perp$) and a boolean variable indicating whether the evader has been caught, plus any additional values described below.

5.1 Version 1: Catch or Clear

In this version of the example the pursuer can both clear the region and capture the evader if present. We drop from the map the position 19 and the edges connected to it. The prescribed surveillance route is 1-4-5-4-8-9-8-12-13-12-16.

This example helps to illustrate the usefulness of epistemic specifications in a situation with partial observability. Since the evader may or may not be in the arena, the pursuer would like to sweep the arena to determine if the evader is present in the arena, and to catch it if so. This objective can be expressed by the specification

$$AF(caught_e \vee K_p(pos_e \notin Q)) \tag{2}$$

which states that the pursuer is guaranteed to eventually capture the evader, or else know that the evader is not in the arena. Alternately, if we do not require capture, but merely that the pursuer knows whether the evader is in the arena, then we could check

$$AF((K_p(pos_e \in Q)) \vee K_p(pos_e \notin Q)) \tag{3}$$

Note that it would not suffice to write the purely temporal specification $AF(caught_e \vee pos_e \notin Q)$. While this does imply that the evader is caught whenever it is present, it does not imply that the pursuer can terminate the search. Possibly the evader is not in Q but the pursuer will never know this fact.

In this version of the game, the pursuer is guaranteed to either clear the region or capture the evader. Model checking confirms that Specifications (2) and (3) both hold. Table 1 lists the experimental results for the BDD-based model checker. The BMC model checker is not included because it is nonterminating for a specification that holds.

5.2 Version 2: Detecting Evadability

We next consider a version where the region can be cleared, but capture is not guaranteed. In this variant, we include position 19 and the edges from it to position 2 and position 3. Since now the evader may be hiding at position 19, not visible from position 1, we adapt the prescribed surveillance route for the pursuer to 1-3-1-4-5-4-8-9-8-12-13-12-16, where we precede the previous route by a move to position 3, where position 19 is visible. In this scenario, the pursuer is able to clear the region, but there exists an infinite chase, in which the pursuer and the evader both move in the same direction along the cycle 1-2-19-3-1. Specification (2) becomes false, but (3) continues to hold.

The bounded model checker can be used to find a counterexample for the specification (2), although it is much slower than BDD-based model checker because of the length of the counterexample.

In this scenario, it is of interest for the pursuer, when it knows that the evader is in the arena, to determine if it is possible for the evader to evade capture. The formula

$$AF((K_p(pos_e \notin Q)) \vee caught_e \vee K_p(pos_e \in Q \wedge EG\neg caught_e)) \tag{4}$$

says that the pursuer is guaranteed to learn whether the evader is in the region, and if it is in the arena, the pursuer will either capture the evader, or eventually know that the evader can (in some cases) evade capture. An arguably clearer, but equivalent, formulation of this is

$$\begin{aligned}(pos_e \notin Q \Rightarrow AF(K_p(pos_e \notin Q))) \wedge (pos_e \in Q \Rightarrow \\ A((F caught_e) \vee FK_p(pos_e \in Q \wedge EG\neg caught_e)))\end{aligned} \tag{5}$$

However, the temporal operators here are not in CTL so it runs a little slower. Experimental results are shown in Table 1.

Spec	Version	Sat	Checker	Time(s)
(2)	1	Y	BDD	3.1
(3)	1	Y	BDD	2.6
(2)	2	N	BDD	3.6
(3)	2	Y	BDD	3.1
(2)	2	N	BMC(Cex = 10)	367.9
(4)	2	Y	BDD	2.4
(5)	2	Y	BDD	66.1
(2)	3	Y	BDD	16.6
(3)	3	Y	BDD	11.3

Table 1: Experimental Results: Region Clearing and Pursuit

5.3 Version 3: Capture with Assistance

In the previous version, the pursuer can discover that although the evader is present, it may be able to avoid capture. To address this, the pursuer may call in assistance with the chase. We model this by adding to the map a new position 20, where the assistant waits until it is called, as well as a unidirectional edge from position 20 to position 1. The pursuer calls the assistant at its first turn where it has last seen the evader at one of the positions on the loop where an infinite pursuit may happen, i.e., 1,2,3 or 19. We add a new variable to the pursuer's memory, which records whether it has called for assistance.

The assistant follows a prescribed strategy to help the pursuer catch the evader. It has been waiting for the call at a special position 20. Once called, it moves to position 1 at its next turn. It then moves along the loop 1-2-19-3 in clockwise direction. The pursuer, after calling for assistance, moves towards the loop and then around it in anti-clockwise direction.

In this scenario, the formula (2) holds, as does formula (3). Note that (4) and (5) hold trivially when (2) holds.

6 Related Work

The games we have studied in this paper are a special case of motion and sensing problems, which [Isler, 2004] groups into sensor assignment, sensor placement, exploration, and pursuit-evasion. These four problems can be seen as special cases of a more general problem, involving either single or multiple sensors, static or mobile sensors, and a search goal that is either punctual or distributed across the environment. This paper deals primarily with pursuit-evasion games, and the closely related problem of exploration (termed environment clearing above), but we believe epistemic model checking may also be applicable to other problems in this area.

Depending on the scientific community many synonyms are used for somewhat similar problems: art gallery problems in computational geometry [Chvàtal, 1975], [Bjorling-Sachs and Souvaine, 1995], graph searching in computer science [Megiddo *et al.*, 1988], [Goldstein and Reingold, 1995] and [Gal, 2005], rendezvous problems in operations research [Lim, 1997], [Alpern and Lim, 1998] and [Alpern, 2002], and differential games in control theory [Isaacs, 1965].

The literature on pursuit-evasion games studies a range of models and issues. One of the objectives is to develop general strategies for a class of games, to prove their correctness and analyze their efficiency. For example, [Adler *et al.*, 2004] provides bounds on expected time to capture for randomized pursuit-evasion protocols for graphs involving one pursuer and one evader. Here neither agent has visibility of the neighboring nodes. The scheduler is synchronous. Isler [Isler, 2004] studies expected time to capture for agents with local visibility of nodes adjacent to their location. He considers reactive evaders, which only move when they see a pursuer, as well as algorithms for classifying graphs according to whether they admit a winning strategy. Other important results are found in [Gal, 2005], [Megiddo *et al.*, 1988] and [Alpern, 1995; 2002].

Our work in this paper differs from the above in that we are interested in obtaining a *formal verification* of the correctness of a strategy in a *particular game*, rather than in a class of games. This may be useful even when applied to games for which a theoretical analysis exists, since theoretical proofs are sometimes flawed. Further, a strategy that is asymptotically optimal for a class of games may not be optimal for a specific game in that class, leading mission planners to deviate from the theoretical strategy, so that the proof guarantees no longer apply.

Another benefit of a model checking approach is that we can flexibly handle a larger range of game models than studied in the theoretical literature, investigating questions such as how a game is impacted by changes in the sensor models, player capabilities or game objectives. Such changes may invalidate a theoretical analysis, requiring a lengthy and labor-intensive intellectual effort to develop the theory in the new setting. On the other hand, model checking promises to provide an efficient, automated approach to the analysis of particular variants of interest.

A knowledge-based approach to incomplete-information robot motion planning problems is proposed in [Brafman *et al.*, 1997], and one of the examples from this paper has been studied using MCK [Gammie and van der Meyden, 2004]. Model checking based on the interpreted systems model for uncertainty, as a general solution procedure to situation analysis problems in the military domain, has been proposed in [Maupin and Jousselme, 2005] and [Maupin *et al.*, 2010]. So far as we know, our work in this paper is the first detailed study of the application of an *epistemic* model checker to pursuit-evasion games.

However, others have investigated model checking of specifications in temporal logic as an approach to the analysis of pursuit-evasion games. [Bohn, 2004] uses the symbolic model checker SMV to synthesize a strategy for a pursuer to capture an evader when possible. The grid scenario in this work is different from ours in that the pursuer and evader observe only their current positions and the pursuer has a speed advantage over the evader. The pursuer's knowledge is implicitly encoded in the model with a boolean variable $occupied_x$ for every position x, denoting that the pursuer considers it possible that the evader is at position x. Since this encoding is done by hand, the approach is likely to be difficult to follow in more complex examples, and it leaves the possibility of human error. By comparison, our epistemic model checking approach automates such reasoning about knowledge. [Sirigineedi *et al.*, 2009] uses SMV to check the behavior of a UAV performing a cooperative search mission.

In [Moulin *et al.*, 2003], a bounded model checker is used to check several universal properties in maneuvering target tracking in a planar air-to-air scenario.

7 Conclusion

Our objective in this paper was to clarify the ways that epistemic model checking might be beneficial for the analysis of pursuit-evasion games, and to conduct a number of case studies of small scale in order to confirm its relevance and feasibility. We feel that the results are promising and justify the development of more realistic, larger scale case studies. Encoding of the examples as MCK input scripts was done by hand, and larger scale studies will require tools for automatic generation of model checker input from geographic information systems. We presently have such tools under development and hope to report results in the near future.

Several other directions would be interesting to pursue in future research. We have identified knowledge conditions of interest in the pursuit-evasion setting. Determining explicitly the exact conditions under which an agent has this knowledge would be useful, e.g. for terminating a search or calling in assistance at the earliest possible time. It would also be interesting to develop automated approaches for the synthesis of player strategies in these games, and to apply our approach to strategies from the existing theoretical literature.

References

[Adler *et al.*, 2004] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking. Randomized pursuit-evasion in graphs. *Combinatorics, Probability and Computing*, 12:225–244, 2004.

[Alpern and Lim, 1998] S. Alpern and W. S. Lim. The symmetric rendezvous-evasion game. *SIAM J. Control Optim.*, 36(3):948–959, 1998.

[Alpern, 1995] S. Alpern. The rendezvous search game. *SIAM J. Control Optim.*, 33(3):673–683, 1995.

[Alpern, 2002] S. Alpern. Rendezvous search: A personnal perspective. *Oper. Res.*, 50(5):772–795, 2002.

[Bjorling-Sachs and Souvaine, 1995] I. Bjorling-Sachs and D. Souvaine. An efficient algorithm for guard placement in polygons with holes. *Discrete Comp. Geom.*, 13:77–109, 1995.

[Bohn, 2004] C. A. Bohn. *In Pursuit Of A Hidden Evader*. PhD thesis, The Ohio State University, 2004.

[Brafman *et al.*, 1997] R. I. Brafman, J-C. Latombe, Y. Moses, and Y. Shoham. Applications of a logic of knowledge to motion planning under uncertainty. *JACM*, 44(5), 1997.

[Chvàtal, 1975] V. Chvàtal. A combinatorial theorem in plane geometry. *J. Combin. Theo. (Series B)*, 18:39–44, 1975.

[Clarke *et al.*, 1999] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[Eijck, 2004] D.J.N. Eijck. Dynamic epistemic modelling. *CWI. Software Engineering [SEN]*, (E 0424):1–112, 2004.

[Fagin *et al.*, 1995] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. The MIT Press, 1995.

[Gal, 2005] S. Gal. Strategies for searching graphs. In *Graph Theo. Combin. Alg.*, pages 189–214. 2005.

[Gammie and van der Meyden, 2004] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In *CAV*, pages 479–483, 2004.

[Goldstein and Reingold, 1995] A. S. Goldstein and E. M. Reingold. The complexity of pursuit on a graph. *Theo. Comp. Sci.*, 143:93–112, 1995.

[Isaacs, 1965] R. Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. Wiley and Sons (1965), Dover (1999), 1965.

[Isler, 2004] I. V. Isler. *Algorithms for Distributed and Mobile Sensing*. PhD thesis, Uni. of Pennsylvania, 2004.

[Lachner *et al.*, 2000] R. Lachner, M. H. Breitner, and H. J. Pesch. Real-time collision avoidance: Differential game, numerical solution, and synthesis of strategies. In *Adv. in Dynamic Games and Applications*, pages 115–135. 2000.

[Lim, 1997] W. S. Lim. A rendezvous-evasion game on discrete locations with joint randomization. *Adv. Appl. Probab.*, 29:1004–1017, 1997.

[Lomuscio *et al.*, 2009] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *LNCS 5643*, pages 682–688, 2009.

[Maupin and Jousselme, 2005] P. Maupin and A. Jousselme. A general algebraic framework for situation analysis. In *Fusion*, 2005.

[Maupin *et al.*, 2010] P. Maupin, A. Jousselme, H. Wehn, S. Mitrovic-Minic, and J. Happe. A situation analysis toolbox: Application to coastal and offshore surveillance. In *Fusion*, 2010.

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

[Moulin *et al.*, 2003] M. Moulin, L. Gluhovsky, and E. Bendersky. Formal verification of maneuvering target tracking. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 1–8, 2003.

[Sirigineedi *et al.*, 2009] G. Sirigineedi, A. Tsourdos, R. Zbikowski, and B. A. White. Modelling and verification of multiple uav mission using smv. In *Formal Methods for Aerospace Workshop, Formal Methods*, 2009.

[Su *et al.*, 2007] K. Su, A. Sattar, and X. Luo. Model checking temporal logics of knowledge via OBDDs. *The Computer Journal*, 50(4):403–420, 2007.

[Vidal *et al.*, 2002] R. Vidal, O. Shakernia, J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *Int. J. Comp. Geo. App.*, 18(5):662–669, 2002.