Experiments on Epistemic Model Checking in Pursuit-Evasion Games

X. Huang and R. van der Meyden
School of Computer Science and Engineering,
University of New South Wales

January 10, 2012

Abstract

A framework for pursuit-evasion scenario verification and analysis has been proposed by Huang, Maupin and van der Meyden, IJCAI 2011. This report extends this line of work by describing a set of experiments that test the approach on a variety of examples of larger scale.

1 Introduction

An approach to model checking epistemic properties in pursuit-evasion games with incomplete information has been proposed in [1], which argues for the relevance of epistemic properties in this class of problems, and gives some small scale examples of graph-based pursuit-evasion game in which the model checker MCK is applied to verify these properties.

In this report, we describe some experiments that folllow the framework of [1] but consider some larger scale problems. Our main objective in this work is to evaluate how the performance of the proposed approach scales with the size of the problem. We refer the reader to [1] for details of the general framework that have been omitted in this report. It is anticipated that a revised version of this work will be combined with the work in [1] to form a paper that will be submitted to a journal.

We have conducted experiments in which we apply MCK to pursuit-evasion games on two classes of randomly generated maps: random graphs and graphs in the form of Manhattan grids with randomly placed holes. The details are given in Section 3. Both types of game models correspond to vertex-based pursuit-evasion games: at each game state, each of the players inhabits some vertex, and a transition is made by moving along the edges to another vertex.

For each of these types of graphs, we generate scenarios by varying a number of other parameters. In all scenarios there is just one evader. We vary the scheduling policy, the number of pursuers, the information observable to the pursuers, the strategy used by the pursuers for region clearing, and the strategy used by the pursuers to chase the evader once detected. The details depend on the type of graph, and are discussed below. In general, the choice of pursuer strategy is either random or a simple rule, not necessarily optimal or correct. We make no attempt to identify a correct or optimal strategy: our objective is to evaluate the performance of our model checking approach to strategy verification, which should work independently of whether the given strategy satisfies the formula being verified.

This report is structured as follows. Section 2 describes the specifications of pursuit-evasion problems that we analyse. The two classes of models of such scenarios are given in Section 3. We report the results of model checking experiments in Section 4. Section 5 makes some concluding remarks.

2 Specifications

Our discussion in this section is largely based on the prior work in [1]. In Section 2.1 we describe the specifications for pursuit-evasion problems that we consider, with respect to an epistemic analysis based on an observational interpretation of knowledge. MCK also supports richer semantics for knowledge, but requires syntactic restrictions on

specifications when these semantics are used. Variants of the specifications for the richer clock, observational and probabilistic semantics are described in Section 2.2.

In the specifications in this section, we refer to an agent p, representing the pursuers. Our models may have multiple pursuers in a given scenario, but are constructed with the additional agent p that observes all variables that are observable to any pursuer. Thus, the epistemic operator K_p represents the distributed knowledge of the pursuers, i.e., what they would know if they were to pool their knowledge [2]. (If there is just one pursuer, then p is identical to this pursuer.)

2.1 Specifications: Observational Semantics

We first describe the specification formulas we consider with respect to the observational semantics, which allows model checking for the richest temporal expressiveness.

Each pursuit game has two phases. The first phase is region clearing, during which the pursuers seek to ascertain whether there is an evader is present in the scenario. In this phase we check the formula

$$AF(K_p(pos_e \notin Q) \vee K_p(pos_e \in Q)) \tag{1}$$

Here pos_e is the position of the evader, and the set of positions Q represents the set of positions of the space to be searched. The formula says that it is guaranteed that eventually the pursuers will either know that the evader is present in this space, or know that the evader is not present in this space.

In case the evader is detected, we switch to the second phase, in which the pursuers chase the evader. For this phase, the formulas we verify are

$$AF caught \tag{2}$$

(which says that it is guaranteed that the evader will eventually be caught), and

$$AF(caught \vee K_p(EG\neg caught)) \tag{3}$$

(which says that it is guaranteed that the evader will eventually be caught, or the pursuers will know that it is possible that the evader will never be caught.)

If we model both game phases in one model, and then use this model to verify a formula that refers to single phase, the performance of the verification must pay the price of reasoning about states belonging to the other phase. This can lead to a significant performance penalty.

However, it is possible to decompose the game model into two smaller models, one for each phase, and verify the formulas referring to a phase in the model relevant to that phase. This proves to be more efficient. In particular, we assume in the region clearing model that the evader starts at an unknown position (possibly off the scenario) and the pursuers all start at a fixed location. In the chase model, we can assume that the evader is present and the pursuers start at an arbitrary location. This is an over-approximation of the possible states at which the evader is first detected in a given region clearing model. However, the over-approximation is safe: if the formulas hold in this over-approximation then they will also hold in the more restricted model.

For comparison, we consider also the model checking performance when the two phases are combined in a single model. The formulas to be checked are

$$AF(caught \vee K_p(pos_e \notin Q)) \tag{4}$$

$$AF(caught \vee K_p(pos_e \notin Q) \vee K_p(pos_e \in Q \wedge EG\neg caught)) \tag{5}$$

We verify the specification formulas using MCK's BDD-based model checking algorithms, as well as, wherever possible, using MCK's bounded model checking (BMC) algorithms. Specifically, all the above formulas can be checked using BDD-based algorithms. Formulas (1) and (2) can be checked with BMC algorithms, the rest cannot.

The BDD-based model checking algorithms operate in two stages. In the first stage a BDD for the model is constructed (including the transition relation and reachable states). This is combined with a BDD for the formula being verified in the second stage. Since the cost of the first stage is the same for all formulas, we estimate the cost of the first phase by verifying the following formula

$$True \tag{6}$$

Subtracting this from the total running time for the two stages gives an estimate of the cost of the second stage.

2.2 Specifications: Richer Semantics

All the specifications in the previous section (and in the earlier paper [1]) assume the observational semantics for the knowledge operator. Additionally, we perform some experiments based on the clock semantics, synchronous perfect recall semantics, and their probabilistic variants. The algorithms for these semantics are supported in MCK only for specifications of the form $X^k\phi$, where ϕ is a formula that talks about the (probabilistic) knowledge of a single agent.

Thus, we need to set a bound on the number of steps for which we evaluate the game. This is reasonable. The number of steps required by a successful clearing strategy for a single pursuer, could be, in the worst case, exponential in the number of vertices in the graph. However, it is always possible to reduce the length of the required strategy by adding extra pursuers: plainly, once we have as many pursuers as vertices, a strategy of length zero suffices.

For the purposes of the experiment will assume that the number k considered in the specification, satisfies

$$k \leq 2 \cdot \frac{|\text{number of vertices}|}{|\text{number of pursuers}|}.$$

We call the number k the *path length*.

We verify the following two formulas for the clearing phase, for $n \leq k$:

$$X^n(K_p(pos_e \notin Q) \vee K_p(pos_e \in Q)) \tag{7}$$

$$X^n(pos_e \notin Q \Rightarrow Prob_p(pos_e \notin Q) \geq 0.9) \tag{8}$$

The first of these says that after n steps the pursuers have either detected the evader or know the evader to be absent. The second says that if the evader is absent, then after n steps the pursuers will have high probability that the evader is absent. Given the way that the probabilistic model checking algorithm works, the number 0.9 used in the latter specification could be replaced with any other without changing the expected running time of the algorithm, so it is not necessary to consider other numbers for our purposes (even if it may be significant for the actual application.)

The formulas above for the chase phase are either purely temporal (which does not require the richer epistemic semantics we consider in the present section) or have the subformula $K_p(EG\neg caught)$, which is beyond the scope of the algorithms for the richer semantics. We therefore do not have formulas for this phase using the richer semantics.

For the combined game we consider the formula

$$X^n(caught \vee K_p(pos_e \notin Q)) \tag{9}$$

which states that after n steps, the evader has been either caught, or the pursuers know the evader to be absent, and its probabilistic variant

$$X^n(caught \vee Prob_p(pos_e \notin Q) \geq 0.9)\ . \tag{10}$$

As before, we verify all these formulas using the BDD-based algorithms, as well as the BMC algorithms whenever possible; in particular, formulas (8) and (10) can be checked using the BMC algorithms, but the probabilistic formulas (9) and (11) cannot.

3 Game Models

We now give the details of the two classes of game models that we consider in our experiments. Section 3.1 describes the random graph based models and Section 3.2 describes the models where the graph is in the form of a grid with holes.

3.1 Random Graphs

The class of connected random graphs $\mathcal{G}_{rand}(n, m)$ is generated using two parameters: the number of nodes n and the number of edges m. Thus, for a game of h pursuers, the path length bound is $k \leq \frac{2n}{h}$ for random graphs in $\mathcal{G}_{rand}(n, m)$ In our experiments the random graphs are generated by the routine gnm_random_graph from the software networkx

(http://networkx.lanl.gov/). To make sure that a graph is connected, we set up an iteration process to ignore all non-connected graphs and keep the first connected one. In order to model that an evader is absent from the scenario, we add a disconnected node 0 for the evader position in this case.

We use the synchronous scheduler on these random graphs. The initial position of the evader may be any node, including the node 0 that is outside the scenario. The pursuers are initially all placed at the same fixed node. The evader strategy is to move randomly. Pursuer visibility in these random graph models is defined to be the current node and the adjacent nodes. We assume that the pursuers always move at unit speed, i.e. may traverse a single edge in each transition. We consider two possible speeds for the evader: unit speed or double-speed, i.e., up to two edges traversed in a single transition.

The three different models for this class of graphs are based on these assumptions, together with the following:

- **Clearing:** In the clearing phase, each of the pursuers follows a fixed surveillance route through the graph. This route is randomly generated at the time of model construction, with a strong bias (probability 99%) to choice of an adjacent node that has not previously been visited, when one exists. The number of steps in this path is fixed in each model; when the pursuer reaches the end of the path it remains at the final node for the rest of time.

- **Chasing:** Given the assumptions above, a possible initial state for the chasing phase model is one in which the pursuers are placed arbitrarily and the evader is adjacent to one of the pursuers. (Note that in this model, *individual* pursuers may not know the location of the other pursuers. However, given the determinism of the surveillance route and the synchrony assumption, they would have this knowledge at the end of the clearing phase in the clearing model. This would effect the result of the subformula $K_p(pos_e \in Q \wedge EG\neg caught)$) if p were to represent the knowledge of a single pursuer. However, since agent p represents distributed knowledge of the pursuers, the location of all the pursuers is in fact available to this agent, since each individual pursuer observes its own position.)

  In the chasing phase, one of the pursuers who detected the evader chases the evader, while the others remain static at the position they occupied when the evader was first observed by any pursuer. The details of the pursuer strategy depends on the evader speed.

  If the evader moves at unit speed, the chasing pursuer's strategy is simply to move to the node occupied by the evader. Since the evader also simultaneously moves, the evader may no longer be at that position in the new state. However, the invariant that the chasing pursuer and the evader are at adjacent nodes is preserved.

  If the evader moves at double speed, it is possible for the evader to move out of the range of visibility of the pursuer. To avoid turning the problem back into the problem of detecting the evader, we assume that the evader leaves a trail, which the pursuer may observe and follow. Specifically, any edge traversed by the evader is marked with the direction of traversal. The pursuer erases these markings when they follow this edge. In case multiple edges from a node are marked, the pursuer chooses the one to follow randomly.

- **Combined:** The combined model brings together the above strategies for the two phases in a single model. It has a smaller set of possible states for the start of the chase than the Chasing model. However, as discussed above, the Chasing model is a safe approximation.

The information observable to a pursuer consists of whichever of the following are relevant to the given model type: its current position, the location of the evader, when this is in the range of visibility, a counter representing the position on its search path, the contamination on the local trails, and (in the combined model) the phase it is currently in.

3.2 Manhattan Grids with Holes

Random graphs do not have regularities of structure of the sort that give advantages to BDD-based model checking algorithms. A reasonable hypothesis is that a more regular class of graphs, that does have some regularities, can be handled more efficiently by BDD-based model checking algorithms, which have the potential of exploiting such regularities to construct more compact BDD representations. To investigate this question, we consider the class of randomly generated graphs in the form of Manhattan Grids with Holes. The class $\mathcal{G}_{grid}(n, m, l)$ consists of graphs of

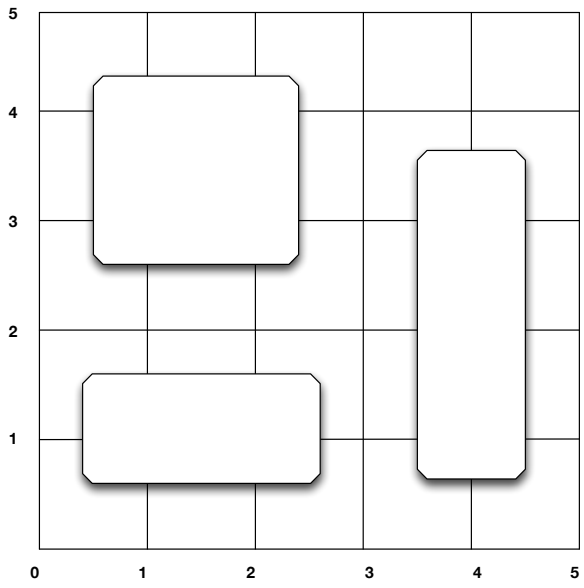

Figure 1: A Manhattan grid with holes

the form of $n \times n$ grids with m holes, each of which has a size of at most $l \times l$. Figure 1 gives an example of a grid of size 7x7 with three holes of size at most 2x3, i.e., a graph in $\mathcal{G}_{grid}(7, 2, 3)$. Since a grid of this form may have at most n^2 nodes, the path length we consider for these experiments is $k \leq \frac{2n^2}{h}$.

The random process for generating graphs in $\mathcal{G}_{grid}(n, m, l)$ works as follows. We start with the full grid of size $n \times n$, represented by nodes (x, y) for all $0 \leq x \leq n-1$ and $0 \leq y \leq n-1$. We then randomly generate the holes one by one and make sure at each step that the newly generated hole does not overlap with any existing hole; if so we regenerate a hole until one is found that satisfies this condition. To get one hole, we first obtain its real size $l_1 \text{x} l_2$ by letting l_1 and l_2 be uniformly chosen random numbers over the range $[1..l]$. Given the size $l_1 \text{x} l_2$ of a hole, we uniformly at random choose a node (x, y) in the grid. To check the non-overlapping condition we make sure that all the nodes $(x + i, y + j)$ with $0 \leq i \leq l_1$ and $0 \leq j \leq l_2$ do not appear in previous holes.

For the straight-line game, the holes are generated within the regions.

We use the turn-based scheduler for all models in this class. All players move at unit speed on the grid, with moves into or crossing the holes prohibited.

The evader is initialized at any node. If it is in a hole, and thus is outside the scenario, then it can not move. Otherwise, it moves randomly on the graph. The pursuers are initially placed at different positions on the grid, selected to enable them to patrol in a rectangular pattern around some hole (see below).

We consider two types of visibility relation for the pursuers:

- **local:** in this class of models, a pursuer is able to observe the node it occupies plus adjacent nodes.
- **straight-line:** in this class of models, a pursuer is able to see to any distance along a Manhattan grid straight line, but not through holes. That is, from position (x, y) it can see an evader at position (u, v), provided $x = u$ or $y = v$, provided that there is not an intermediate point between these two points that lies inside a hole.

The models for each of the phases are constructed as follows:

- **Clearing:** In clearing phase, the h pursuers follow a simple strategy, that depends on which case of the visibility relation applies.

  For local visibility game, starting from different initial nodes, the pursuers patrol the whole grid by a strategy that sweeps the entire grid. That is, if the position of pursuer p is given by $pos_p = (i, j)$, then the position pos'_p of that pursuer after a move is given by:

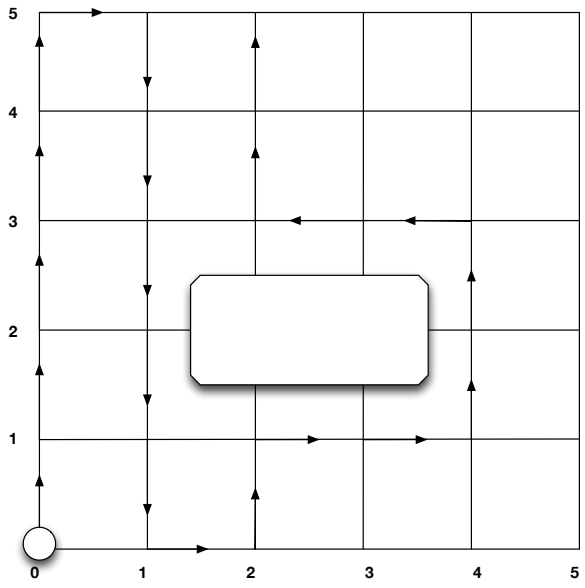

Figure 2: Grid-sweeping strategy

- if i is even and $j < n - 1$ then let $pos'_p = (i, j + 1)$,
- if i is odd and $j > 0$ then let $pos'_p = (i, j - 1)$, and
- if i is even and $j = n - 1$ or i is odd and $j = 0$ then let $pos'_p = (i + 1, j)$.

This strategy is depicted in Figure 2. If the prescribed move is illegal because the path is blocked by a hole then the pursuer will instead take a number of steps anti-clockwise around the borders of the hole, in order to move to the first position on the other side of the hole that it would have reached if following the strategy above. It then continues with the above strategy.

For the straight-line visibility game, we split the grid into h non-overlapping regions, each of which is patrolled by a pursuer. These regions are selected so that each hole lies strictly inside one of the regions. Each pursuer is initially at a fixed node on the border of its allocated region. The strategy of the pursuer is then to cyclically patrol the border of its region. (This simple strategy avoids the more complicated question of how a pursuer should behave when it reaches a hole.)

- **Chasing:** In chasing phase,we use a simple strategy in which all pursuers move towards the current position of the evader at each step. In order to do so, they need to know the position of the evader. However, the evader may move from a position where it is visible to where it is not visible to any evader. To simplify the programming of the model in this case, we assume that once evader is detected, the pursuers receive instructions from an oracle that provides them with the correct move to make in order to move closer to the evader. (One plausible explanation of this is that, once detected by a ground-based pursuer, a helicopter or UAV is called in to assist the ground-based pursuers in the chase, by providing location information, and is able to move so that it always sees the evader.) As in the clearing phase, if a move cannot be made directly because of a hole, the pursuer moves around the hole instead.

- **Combined:** The combined model contains both the clearing and chase strategies described above. As with the case of random graphs, this means that the combined model has a smaller set of states at the transition between the two phases, but the clearing and chase models together form a sound approximation to the more accurate combined model. A pursuer can observe its current position, the phase it is currently in, and whether it can see the evader in current round.

4 Results

The experiments are conducted on four kinds of games:

- random graph games in which the evader moves at unit speed (1speed game, for short),
- random graph games in which the evader moves at double speed (2speed game, for short),
- Manhattan grid games in which the pursuers have straight-line visibility (line game, for short), and
- Manhattan grid games in which the pursuers have local visibility (local game, for short),

each of which has three models: clearing phase model, chasing phase model, and combined model. The experimental results are collected on two machines: an Apple iMac (3.06GHz Intel Core i3 with 4G memory) for the random graph games and a Ubuntu Linux system (3.06GHz Intel Core i3 with 4G memory) for the Manhattan grid games. Each process is allocated up to 500M memory. In general, the rate of growth in the computation time in our experiments is exponential, so we plot log base 2 of the computation time, and a straight line in the diagram would indicate exponential growth. To keep the total cost of the experiments within reasonable bounds, we have generally scaled the experiments to the extent possible up to a maximal computation time of around 2^{14} seconds, i.e., roughly 4.5 hours.

In the case of the BDD-based model checking algorithms, there is some randomness in the computation time, that arises from the nondeterministic behaviour of the CUDD BDD package that MCK uses. This can cause some significant lack of smoothness in the plots for our experiments, even when we average over a number of runs of the computation, since the distribution may contain some cases of unusually large computation times. Figure 3 is an example of this behaviour: here we plot the average, median and minimum computation time over 11 runs as we vary the graph size. The plots of the median and minimum computation times give significantly smoother curves, which are similar to each other, whereas the average computation times are often much higher, but quite erratic.

We have chosen to handle this behaviour of the data by reporting the minimum computation time over a number of runs in each of our experiments. This choice can be justified as being the running time that would be obtained if we were to address the possibility of an unusually large computation time by running several copies of the computation in parallel, and terminating the computation at the earliest time that any copy returns an answer.

4.1 Random Graph Games

We consider instances of the random graph model in which the number of edges is equal to the number of vertices plus two. We divide the discussion of results into two sections, on model checking with respect to the observational semantics, and on the richer (clock, perfect recall and probabilistic) semantics, corresponding to Sections 2.1 and 2.2.

4.1.1 Observational semantics

Computation time versus graph size: The first question we consider is how computation time scales as we increase the size of the graph.

Clearing Phase: Figure 4 plots model checking computation time as a function of the number of nodes (nbrNodes) for the clearing phase in the 2speed game. The game has only one pursuer, which searches for 20 steps. We can see that, the BDD-based model checker scales well in checking the presence of the evader in this scenario: it takes several seconds to work on a map of 46 nodes and 48 edges. We plot also the computation time for formula (6), as a measure of how much of the computation is taken up by model construction. The results indicate that most of the computation time for formula (1) is taken up in building BDDs for the models, rather than in processing the formula.

Chasing Phase: Figure 5 plots the model checking computation time as a function of the number of nodes for the chasing phase in the 2speed game. The game has only one pursuer. Here we find that larger computation times are required, and we are not able to reach a scale comparable to the clearing phase. A likely explanation of this is that the set of initial states is much larger in this case: conceivably a closer approximation of the initial states for the chasing phase might yield better performance. The plot displays that computation time grows exponentially in the number of nodes. Considering the plot for formula (6), we also find that relatively more computation time is spent on the formula, with model construction taking roughly of the order of the square root of the total computation time.

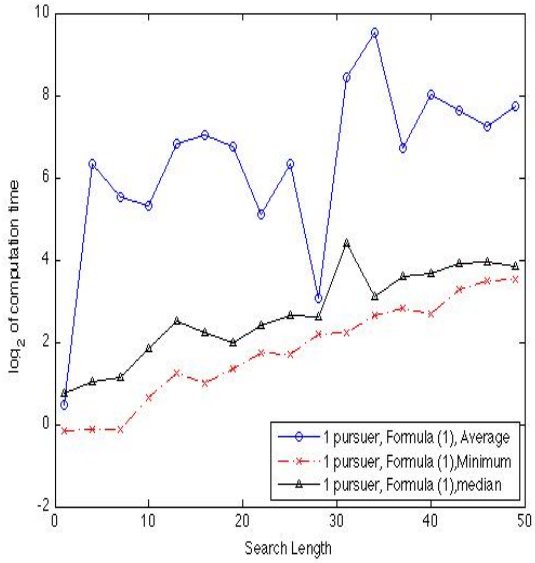

Figure 3: Computation times over 11 runs, BDD-based model checking of clearing phase model of a graph in $\mathcal{G}_{rand}(40, 42)$.

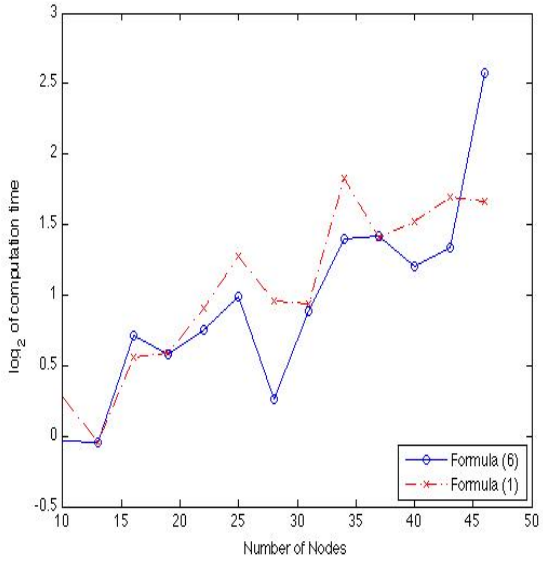

Figure 4: Computation time versus graph size: BDD-based model checking of $\mathcal{G}_{rand}(n, n+2)$, 2-speed clearing phase model, 1 pursuer, search length = 20, observational semantics.

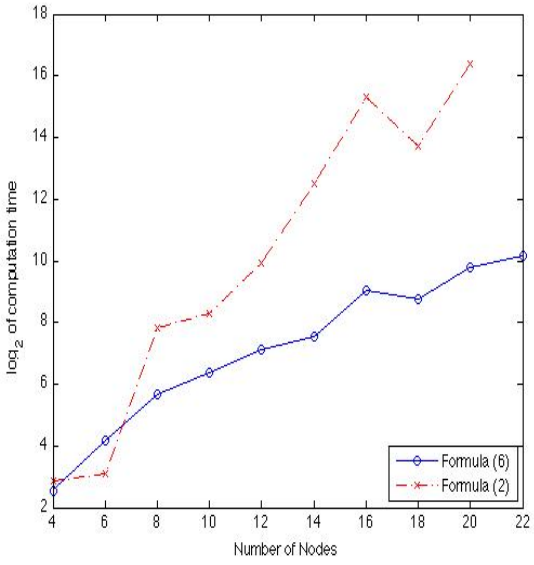

Figure 5: Computation time versus graph size: BDD-based model checking of $\mathcal{G}_{rand}(n, n+2)$, 2-speed chasing phase model, 1 pursuer, observational semantics.

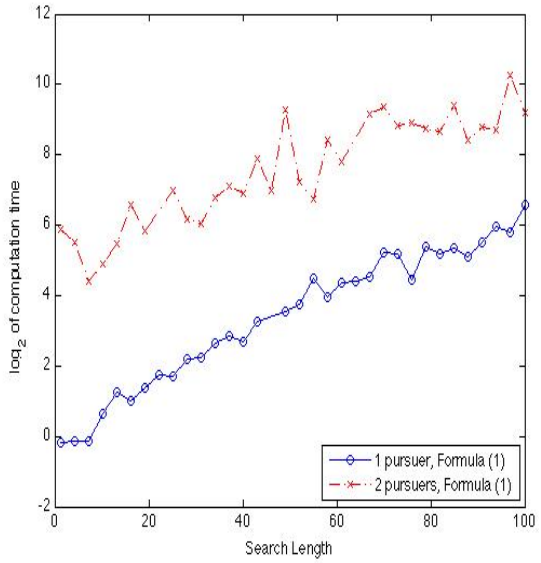

Figure 6: Computation times for increasing search path length in a fixed graph in $\mathcal{G}_{rand}(40, 42)$: BDD-based model checking of clearing phase model in 2speed games, observational semantics.

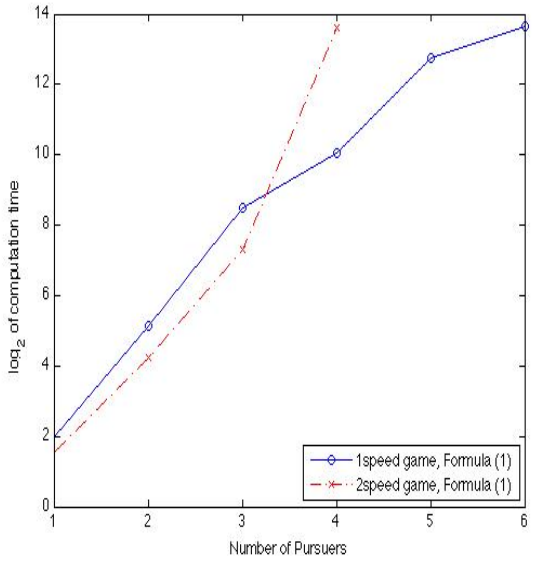

Figure 7: Computation times for increasing numbers of pursuers in a fixed graph in $\mathcal{G}_{rand}(40, 42)$: BDD-based model checking of clearing phase model in 2speed games, search path length 10, observational semantics.

Impact of Search Strategy Length: Figure 6 shows the effect on model checking performance of increasing the length of the search path in the clearing phase, with the map fixed. The data are for the 2speed game, playing on a map of 40 nodes and 42 edges. We give the data for the games of one pursuer and two pursuers. The results indicate that computation time increases exponentially with search path length. As expected, the two pursuer model requires greater computation time.

Impact of Number of Pursuers and Evader Speed: Figure 7 compares model checking computation times for the clearing phase of the 1speed game with the 2speed game in terms of the number of pursuers. The data is based on a fixed map of 40 nodes and 42 edges, with the pursuers searching for 10 steps. The 1speed game scales better than the 2speed game. The results indicate that computation time grows exponentially in the number of pursuers, and that the 2-speed model becomes significantly more difficult than the 1-speed model after 3 pursuers.

BDD versus BMC model Checking: Figure 8 and Figure 9 compare the performance of BDD-based model checking and bounded model checking in the observational semantics. The results indicate that neither approach is optimal in all circumstances.

The model in Figure 8 is a clearing phase model in the 2speed game of 1 pursuer and a map of 15 nodes and 17 edges. With the increase on the length of search path, the computation time for bounded model checking grows exponentially. The BDD-based model checking is less sensitive to the length of search path.

The model in Figure 9 is a chasing phase model in the 2speed game of 2 pursuers. The length of search path is fixed at 10. The BDD-based model checking does not scale well with the increase of the graph size. The bounded model checker performs much more efficiently, easily finding a counterexample with bounded model checking parameter 3 (the length of the run fragments used in constructing the counterexample.) However, we note that a factor in this result is the fact that the formulas being verified are false in the given model. If they were true, then bounded model checking would need to be carried out to a much larger "completeness bound" on run fragment length in order to confirm that

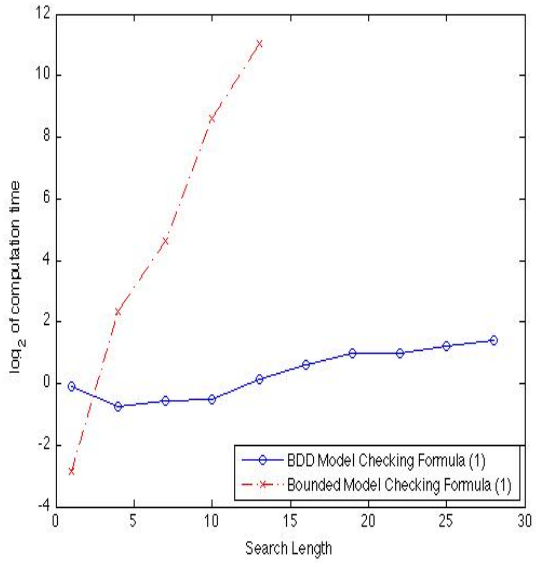

Figure 8: Computation times for BDD-based model checking versus bounded model checking as a function of search path length: clearing phase model for a graph in $\mathcal{G}_{rand}(15, 17)$ in 2speed game, 1 pursuer, observational semantics.

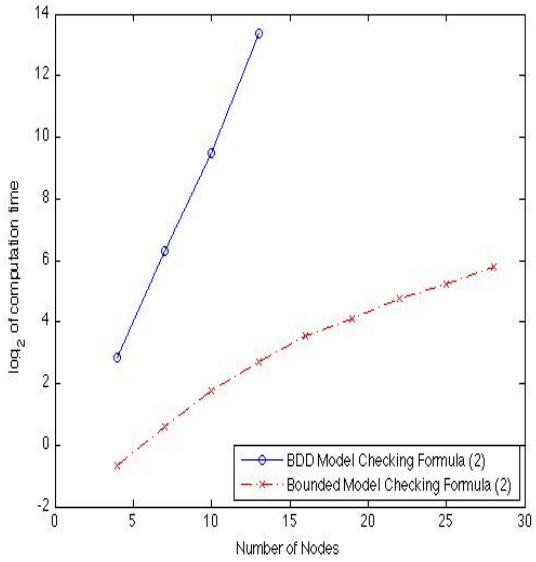

Figure 9: Computation times for BDD-based model checking versus bounded model checking as a function of graph size ' n: graphs in $\mathcal{G}_{rand}(n, n+2)$, chasing phase model in 2speed game, 2 pursuers, search path length 10, observational semantics.

the formula is true. In MCK's current implementation, which does not include calculation of this completeness bound, bounded model checking would in fact be non-terminating for formulas that are true.

4.1.2 Richer semantics

The following three figures show experimental results on richer semantics of knowledge formulas. We focus here on the clearing phase model since the key search phase formulas either do not involve knowledge, or involve knowledge in a form that the algorithms for the richer semantics cannot handle.

Clock Semantics: Figure 10 compares the BDD model checker and bounded model checker over clock semantics as we vary the search path length in a fixed model. The temporal depth of the formula used is the same as the search path length. In this experiment, we deploy two pursuers for the clearing phase in the 2speed game for a graph in $\mathcal{G}_{rand}(40, 42)$. It is interesting that whereas bounded model checking initially performs well, as the search path length increases, eventually it is outperformed by BDD-based model checking, which has a relatively slowly growing computation time. A possible explanation for this is that bounded model checking is searching in an unstructured way for an evader evasion path of increasing length as we increase the search, whereas the BDD-based algorithm is computing the set of possible states at each time in a structured way, with small incremental cost at each step once the cost of constructing BDDs for the transition relation and initial states has been paid.

Perfect Recall Semantics: Figure 11 compares the performance of BDD-based model checking for the clock semantics and synchronous perfect recall semantics. Here we consider the clearing phase for a fixed graph of 35 nodes and 37 edges. Only one pursuer is deployed and the game is the 2speed game. As expected, perfect recall requires more computation time, but both are able to handle reasonable size graphs.

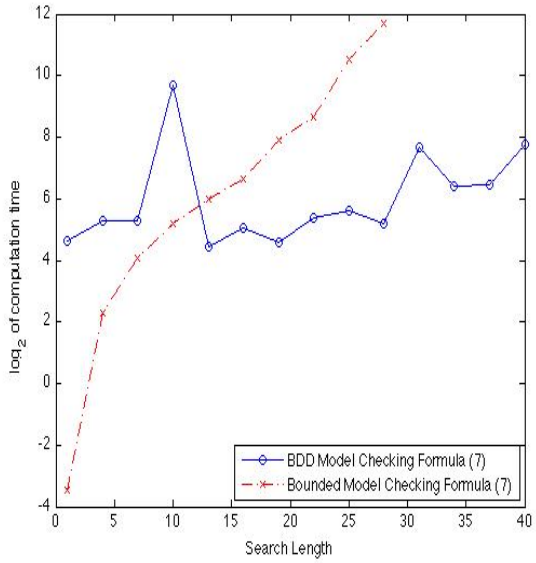

Figure 10: Computation time for BDD-based model checking versus bounded model checking as a function of search path length, for clock semantics, clearing phase model in $\mathcal{G}_{rand}(40, 42)$, 2 pursuers, 2speed game.

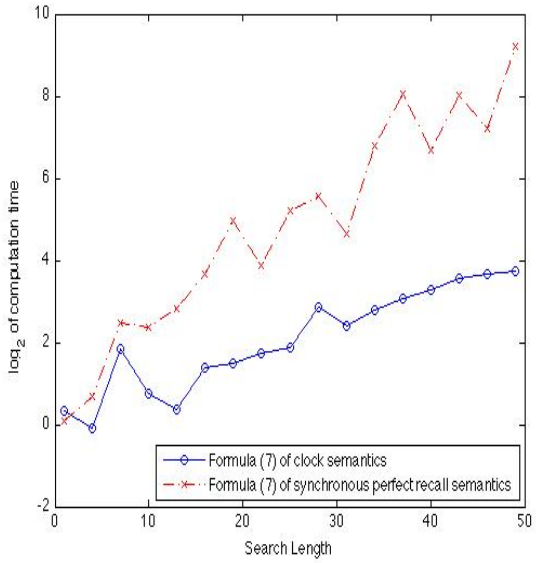

Figure 11: Computation time for BDD-based model checking for clock semantics and synchronous perfect recall semantics for a graph in $\mathcal{G}_{rand}(35, 37)$, clearing phase model, 1 pursuer, 2speed game.

Probabilistic Model Checking: Figure 12 plots computation time for model checking probabilistic knowledge using MCK's MTBDD-based probabilistic model checking algorithm, as a function of search path length (equal to formula depth) in a fixed 2speed search phase model with one pursuer for a graph in $\mathcal{G}_{rand}(15, 17)$. Both the clock semantics and synchronous perfect recall semantics are considered. As expected, model checking for the clock semantics is more efficient.

4.2 Manhattan Grid Games

When dealing with the Manhattan grid models, we have more structured strategies that can run for an arbritrary amount of time, so the issue of search path length considered above in the case of random graphs does not apply. As above, we structure the discussion into the cases of observational and richer semantics.

4.2.1 Observational semantics

Computation time versus grid dimension: Figure 13 gives the scalability of BDD-based model checker in terms of the grid size in Manhattan grid games by comparing the *clearing* phases of the straight-line visibility game and local visibility games. In both games, we check the presence of the evader in the scenario in observational semantics. It is apparent that the local game is harder than the line game.

Computation time versus number of pursuers: Figure 14 shows the dependence of BDD-based model checking computation time on the number of pursuers in the grid game. The formula considered is the simplest formula (6), which captures just the time to construct the BDDs for the model itself. Two versions are considered: the game with straight-line visibility and the game with local visibility. These give rise to similar curves but with respect to different grid sizes: for local visibility the game is played on a 5x5 grid, and for straight-line visibility the game is played on a 30x30 grid. While the number of data points is too small for a definitive conclusion, the results are strongly suggestive

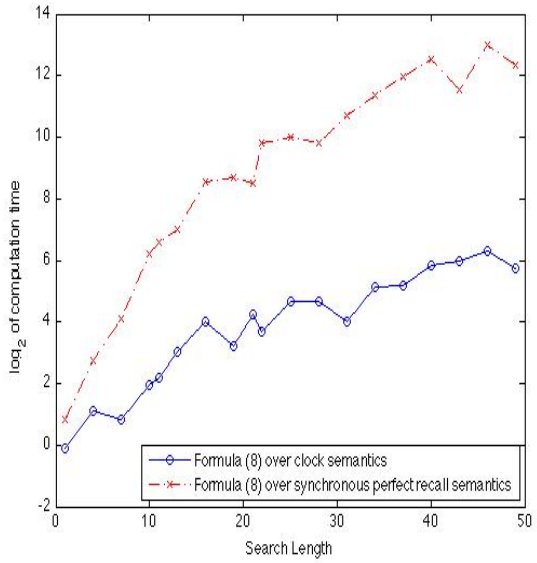

Figure 12: Computation time for MTBDD-based probabilistic model checking for clock semantics and synchronous perfect recall semantics, clearing phase model of a graph in $\mathcal{G}_{rand}(15, 17)$, 1pursuer, 2speed game.

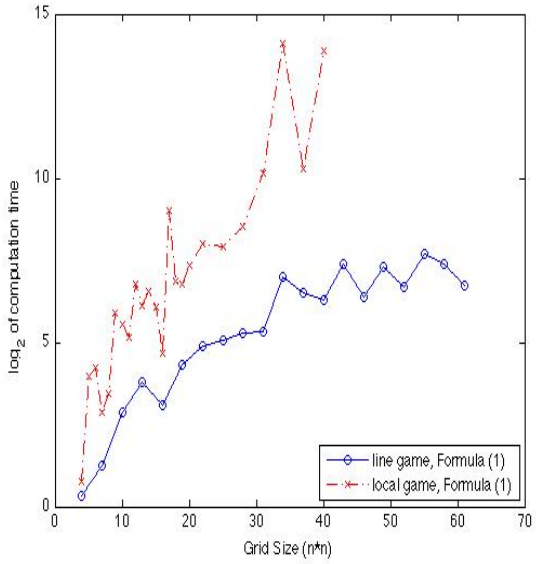

Figure 13: Computation time versus grid dimension n of BDD-based model checking, observational semantics, clearing phase of graphs in $\mathcal{G}_{grid}(n, 2, 2)$, 1 pursuer.

of an exponential dependence of computation time on the number of pursuers.

Computation time versus number of holes: Figure 15 plots the computation time of BDD-based model checking of a fixed size (20×20) grid for the local observability game as we increase the number of holes in the grid. Both the clearing phase model and the combined model are considered. In the clearing phase, we check the presence of the evader in the scenario, while in the combined phase, we check the possibility of either catching the evader or clearing the grid. The flatness of the curves (compared to some of our other graphs) suggests that there is not a strong dependence of computation on this parameter.

4.2.2 Richer semantics

The following three figures show experimental results on model checking the grid games with respect to the richer semantics of knowledge.

Clock Semantics: Figure 16 plots the computation time of BDD-based model checking with respect to clock semantics in terms of the size of grid. The formulas checked have temporal depth 16. Both the clearing phase model and the combined model are checked, for the local visibility game. In the clearing phase, we check the presence of the evader in the scenario, while in the combined phase, we check the possibility of either catching the evader or clearing the grid. The computation times obtained, even when taking minimum times, are somewhat erratic, but suggestive of an exponential growth rate.

Perfect Recall Semantics: Figure 17 presents computation times for model checking with respect to the synchronous perfect recall semantics versus grid size n, for graphs in $\mathcal{G}_{grid}(n, 3, 2)$ with 1 pursuer with local visibility. We consider the combined model, and the formula checks whether it is guaranteed to either clear the grid or capture the evader. We compare the BDD-based model checker with the bounded model checker. As we would expect, the bounded model

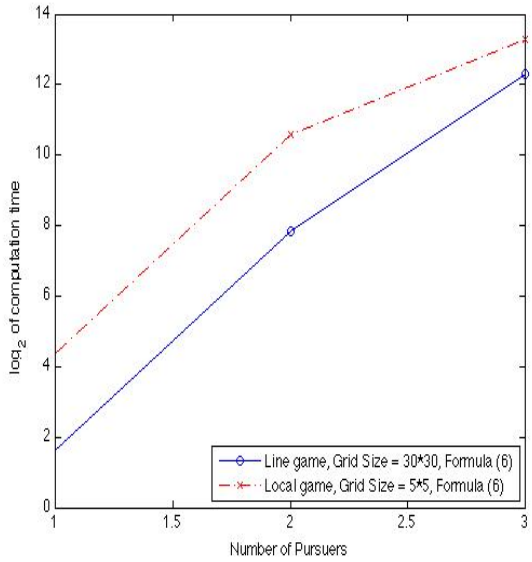

Figure 14: Computation time for BDD-model construction versus number of pursuers, clearing phase model, graphs in $\mathcal{G}_{grid}(n, 1, 1)$, observational semantics.

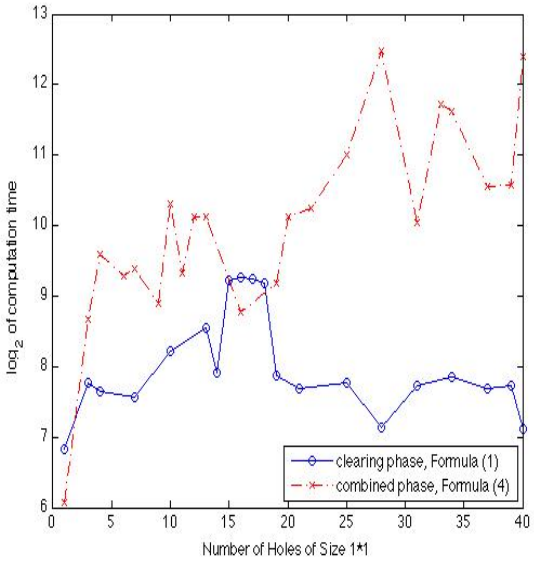

Figure 15: Computation time of BDD-based model checking versus the number h of holes, local visibility game, graphs in $\mathcal{G}_{grid}(20, h, 1)$, 1pursuer, observational semantics.

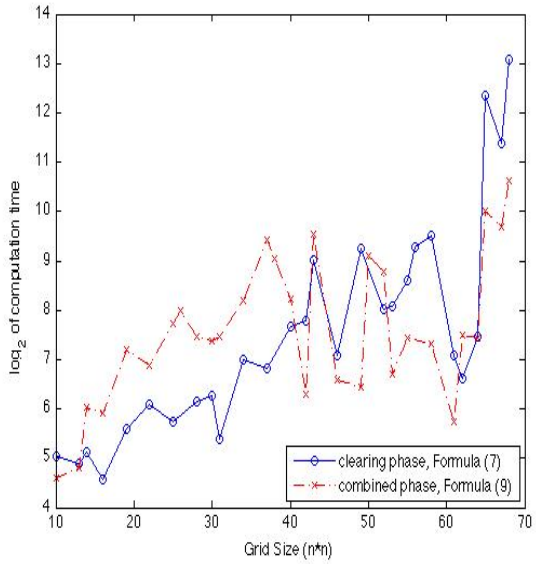

Figure 16: Computation time of BDD-based model checking for clock semantics versus grid dimension n, graphs in $\mathcal{G}_{grid}(n, 1, 1)$, local visibility, 1 pursuer, formula temporal depth 16.

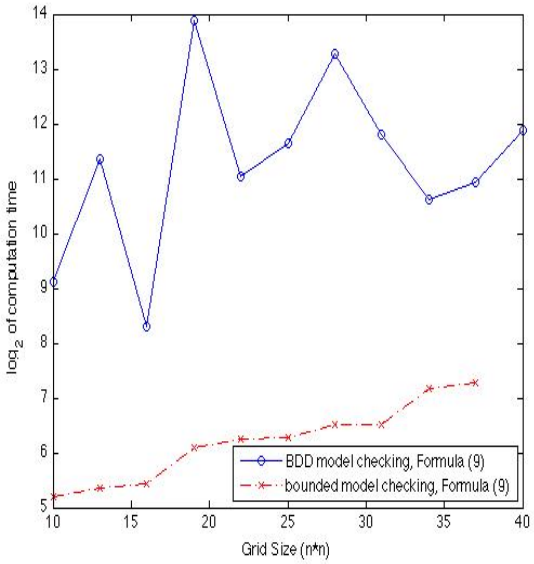

Figure 17: Computation time of model checking for synchronous perfect recall semantics versus grid size n, for graphs in $\mathcal{G}_{grid}(n, 3, 2)$, 1 pursuer, combined model, local visibility, formula temporal depth 16.

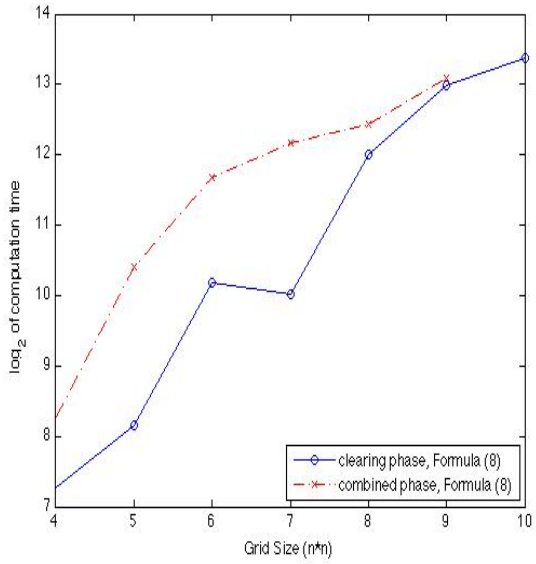

Figure 18: Computation time of MTBDD-based probabilistic model checking for clock semantics versus grid dimensions, graphs in $\mathcal{G}_{grid}(n, 1, 1)$, local visibility, formula temporal depth 16.

checker is more efficient.

Probabilistic Model Checking: Figure 18 plots the computation time of MTBDD-based probabilistic model checking in clock semantics in terms of the size of the grid. We check whether the pursuer has high probability (0.9) of clearing the grid in 16 steps in the 2speed game if the evader is not present. We consider both the clearing phase model and the combined model, which give comparable results, but as expected, the combined model requires more computation time. The scale of examples we can handle in this case is somewhat smaller than for the weaker semantics, with grids of 10×10 nearing our upper bound of 2^{14} seconds.

5 Conclusion

For the specific questions we ask, in the specific classes of models, it may be possible to tailor special purpose algorithms that yield better computation times than the results we have reported here. However, this is to be expected: an epistemic model checker such as MCK is a *general purpose* instrument, with the advantage of enabling a broad range of questions to be answered, in a very convenient way. Instead of having to design a special purpose algorithm, the user need only represent the model in the form of a program describing agent behavior, and express the question of interest as formula in a declarative specification language: the model checker then takes care of working out how to compute the answer to this question without further input from the user. The experiments we have conducted in this paper given an indication of how well this general purpose instrument performs on the analysis of a range of questions in a variety of pursuit-evasion scenarios.

In general, the experiments indicate that examples of non-trivial but moderate scale can be handled using epistemic model checking, but that the model checking computation times for our formulas in these models scale exponentially with respect to most of the parameters of interest: i.e., graph size, search strategy size, formula temporal depth, and number of pursuers. (This result is not unexpected, since similar problems have been shown to have NP-complete

complexity.)

Bounded model checking, in the case where the formula is false, generally outperforms BDD-based model checking, but there are cases where the reverse obtains, and only BDD-based model checking can give a definitive answer in the case where the formula is true.

The weaker observational semantics generally allows larger scale examples to be handled, but problems of interesting scale can be analysed with respect to the richer clock, perfect recall and probabilistic epistemic semantics.

Given the large number of parameters of the problems we have considered, our results represent only a few slices of a complex problem space. Many other specific questions can be asked about the specific pursuer and evader models we have introduced, and numerous variants of these pursuit-evasion scenarios can be contemplated.

In general, to the extent that the scale of example we have shown the approach to be capable of handling is of practical interest, our results indicate that further studies of this type are warranted. At the very least, the scenarios of the type we have considered here can provide challenge problems for the epistemic model checking community, so that subsequent work on improvement of this technology may increase its applicability to this problem space.

References

[1] X. Huang and P. Maupin and R. van der Meyden. Model Checking Knowledge in Pursuit-Evasion Games, In *International Joint Conference on Artificial Intelligence,* IJCAI 2011, Barcelona, Jul 2011.

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