Model-checking in hide and seek games with imperfect information

Ansuman Banerjee, Soham Banerjee, and Sujata Ghosh

Indian Statistical Institute, India

Abstract. We consider a logical framework to express reasoning about actions and strategies in a hide and seek game with imperfect information. In this version of the game, the seeker(s) and the hider move alternately with the seekers moving simultaneously on their turn. A win for the seekers is described in terms of the information they have about the position of the hider. We assume limited observational powers of the players. We provide model-checking algorithms for the proposed framework, and analyze the complexity of those algorithms. We also showcase an implementation of a model-checking tool that can be used for verifying winning strategies in these games.

Keywords: Hide and Seek · Cops and Robbers · Knowledge · Model-checking · Complexity · Implementation.

1 Introduction

Graph games are widely studied in relation to various phenomena occurring in our day-to-day lives. The primary reason for such studies is the flexibility of these games that can be used to model many real life scenarios. These include domains like communication networks and grid-based problems. In particular, pursuit-evasion games play a special role in those studies, e.g., see [**?**]. These games that are also termed as 'cops and robbers' [**?**], and 'hide and seek' [**?**] in the existing literature, involve agents making moves to evade/capture other agents. Evidently, they have many variants with different rules and structures.

To give some perspective to this chapter, we first note that an epistemic approach to playing the game of hide and seek is discussed in [**?**], where the reasoning capabilities of players are explored in terms of the information available to them. This information is in turn based on the players' observational powers. A formal framework, viz., an epistemic logic of hide and seek game (ELHS) is proposed in [**?**], and various properties of the logic are studied in detail. The logic ELHS deals with $n \geq 1$ seekers and one hider. The game itself satisfies the following conditions with respect to players' action and knowledge:

- The graph structure is known to all the players
- The players move in turn following a certain order
- The players get to know when there is a move in the game
- The players can observe certain reachable positions

Evidently, a player's knowledge about another player's position depends on the observational power they have. This may lead to a scenario where, players may know that some other players have moved, but they may not know where these moves actually happened.

Analogous to the above description of the game, there exist several variants where the players may have perfect information about the game, and all the players viz., cops and robber/seekers and hider may move simultaneously. These games have been discussed in [**?**] among others, where the variant of the cops and robber game with simultaneous moves is analysed. We note that simultaneous moves in games are less tractable than alternating moves [**?**]. In our setting, we consider games where the information available to the players is not perfect and the cops and the robber move alternately, with the cops moving simultaneously from their respective positions, as a group, on their turn. Such a setting is studied in details in the traditional cops and robber games literature (see, for example, [**?**]). For the sake of simplicity, we also assume that the players can observe their own positions and also the positions that are accessible from their current positions, and not beyond that. The corresponding logic that is proposed in this chapter, referred to as Sim-ELHS throughout the chapter, is quite similar to ELHS. The only difference is in the syntax of the dynamic *move* operator, which is interpreted accordingly (cf. Section 2).

This chapter studies the computational behaviour of the proposed logic concentrating on the model-checking problem. Model-checking is advocated in [**?**] (versus theorem proving) and is a natural setting in AI, used as a basis in strategic reasoning [**?**], epistemic planning [**?**], among others. The model in these circumstances represents the game we are talking about and one can check whether certain properties hold. These properties can be defined from the perspectives of actions, strategies and knowledge of players and their interplay. In [**?**], the model-checking problem for the logic of hide and seek (LHS) is studied and is shown to be P-complete. In [**?**], the same for a fragment of ELHS (without the dynamic move operator) is studied and also shown to be P-complete. In this chapter, we show that the model-checking problem for Sim-ELHS corresponding to the game consisting of *one* seeker and *one* hider is P-complete, whereas, that for Sim-ELHS corresponding to the game consisting of *more than one* seekers and *one* hider is in EXPTIME. In addition, we also show the problem to be NP-hard. A tighter bound is targeted as future work. This shows that the problem gets more complicated computationally when we move from one seeker to more than one seekers, even though finding the hider becomes easier intuitively.

As we mentioned earlier, the hide and seek game we discuss here has been explored in detail in the literature in the forms of cops and robber games [**?**], as well as, pursuit-evasion games [**?**]. The study on cops and robber game [**?**] started off by considering 2 players, a cop and a robber, taking turns and moving across a graph. A natural extension considered $n \geq 1$ cops and a robber, moving in turn, where the cops moved simultaneously, similar to our hide and seek game under consideration. There are variants of this game with varying winning conditions [**?**] and permitted moves [**?**,**?**]. The main point of our departure, akin to [**?**], is

the consideration of varying information available to players depending on their positions.

Alongside the complexity analyses of the model-checking problems of the different versions of Sim-ELHS, we have also built a model-checker, viz. MCGG to verify various strategies of players playing different kinds of games on graphs, including hide and seek games. At present, we can only deal with hide and seek games with perfect information, but we would incorporate the games with imperfect information in the near future. We include a detailed description of the model-checker MCGG in this chapter with a special focus on hide and seek games. The purpose of the model-checker is to provide a better insight into strategies the players may apply to win, with respect to the distinct graph structures.

Outline. In Section 2, we introduce hide and seek games with imperfect information and show the interplay of knowledge and action at work. We also introduce the syntax and semantics of Sim-ELHS in this section. Section 3 analyzes the model-checking problem for Sim-ELHS with one hider and one seeker, whereas, Section 4 analyzes the same for one hider with more than one seeker. Section 5 provides a comparison with the related results in similar graph games. In Section 6, the model-checker MCGG is introduced and its workings are described with relevant performances. Section 8 brings up the conclusion.

2 An epistemic logic for hide and seek

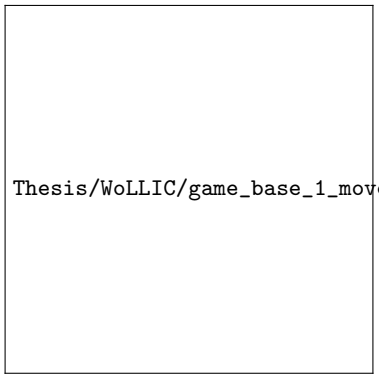
In this section, we delve into the game formally and consider a variant of the epistemic dynamic logic, ELHS, introduced in [**?**]. This is termed as Sim-ELHS to emphasize the simultaneous moves of the seekers, unlike the sequential moves of the same that is considered in ELHS. Before going into the logic, let us first describe a game of hide and seek with imperfect information that forms the basis of our study.

2.1 An example

Let us consider the following game of hide and seek, played on a connected graph with a finite number of nodes. The seeker(s) and hider take turns alternately making their moves on the game graph. Each move involves players moving to an adjacent state along an edge of the graph. All seekers move *simultaneously* followed by a move of the hider.

Figure 1a shows a graph with 6 states interconnected by directed edges with two seekers starting at w_1, w_3, respectively, and the hider starting at position w_5. Figure 1b shows the graph after the seekers have moved to the adjacent locations while the hider waits for his turn. In the game with perfect information, all players have complete information of the other players positions and the graph. In this variant, the players have the knowledge of the graph structure, and they know their own positions along with the positions of players that are at most a step away from their current locations. So, in Figure 1a, the seeker S_1 knows the position of the players at states w_1 and w_2, while S_2 knows that for the states, w_3

(a) Initial Positions



(b) After seekers' initial moves

Fig. 1: A hide and seek game with imperfect information

and w_4. So each player can have multiple equivalent configurations based on the information available to her, and her current position. For S_1, the positions of S_2 and the hider (H) are unknown as she can only see w_1 and w_2. So, according to S_1 the possible positions of the players in the form of (S_1, S_2, H) can be:

$$\begin{aligned}\text{Seeker 1} := (w_1, w_3, w_4) \mid \underline{(w_1, w_3, w_5)} \mid (w_1, w_3, w_6) \mid \\ (w_1, w_4, w_3) \mid (w_1, w_4, w_5) \mid (w_1, w_4, w_6) \mid \\ (w_1, w_5, w_3) \mid (w_1, w_5, w_4) \mid (w_1, w_5, w_6) \mid \\ (w_1, w_6, w_3) \mid (w_1, w_6, w_4) \mid (w_1, w_6, w_5)\end{aligned}$$

The underlined assignment corresponds to the actual assignment of the seekers and the hider. As mentioned earlier, these configurations are based on the information available to the players. As the players make moves along the directed edges of the graph, they gain or lose information regarding the position of the other players and their knowledge gets updated. After the move shown in Figure1b, S_1 knows the position of H as he is now at a position visible to S_1. So after the move the possible positions of the players according to S_1 are:

$$\text{Seeker 1} := \underline{(w_2, w_4, w_5)} \mid (w_2, w_6, w_5) \mid (w_2, w_1, w_5)$$

As the players move they update their knowledge based on the newly visible states as well as their previous positions. In order to talk about the interplay of moves and knowledge of various players and the game, we now consider an epistemic logic for the game of hide and seek (Sim-ELHS), a variant of ELHS. In the following we provide the details of syntax and semantics for Sim-ELHS.

2.2 Sim-ELHS

We are now ready to present the syntax of Sim-ELHS to describe the hide and seek game with imperfect information exemplified above. Let $\mathcal{P}$ be a set of

predicate symbols, V be a finite set of variables representing the players of the game. The formulas of Sim-ELHS are given by:

$$\varphi := Px \mid Rxy \mid x \equiv y \mid K_x y \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_x\varphi \mid [X]\varphi$$

where $P \in \mathcal{P}$, $x, y \in V$ and $X \subseteq V$.

We now have a look at the models to be used to represent the games. These models, similar to first-order models, are expressed using the following tuple:

$$\mathcal{M} = (D, R, \{P\}_{P\in\mathcal{P}}, \mathcal{G}, \{\sim_x\}_{x\in V})$$

where, D denotes a finite set consisting of states (or nodes) of the graph on which the game is played, $R \subseteq D \times D$ is a binary relation on D denoting the edge relation of the graph, $\{P\}_{P\in\mathcal{P}}$ is the set of interpretations of the predicate symbols in $\mathcal{P}$, $\mathcal{G}$ denotes the collection of assignment functions indicating the various positions of the players, V indicates the set of players and for each $x \in V$, $\sim_x$ is an equivalence relation on $\mathcal{G}$ satisfying the following property,

$$\text{if } g \sim_x g' \text{ then (i) } g(x) = g'(x), \text{ and, (ii) } \forall y \text{ with } g(x)Rg(y),\ g(y) = g'(y),$$

where $g, g' \in \mathcal{G}$ and $y \in V$. Since $\sim_x$ is a relation, we denote the image of an assignment g under this relation as $\sim_x (g)$.

Given the models above, a formula is evaluated in a model with respect to an assignment function. We have the following recursive definition for the evaluation of a formula in Sim-ELHS.

$$\begin{array}{lll}
M, g \vDash Px_1 \cdots x_n & \text{iff} & (g(x_1), \cdots g(x_n)) \in P \\
M, g \vDash x \equiv y & \text{iff} & g(x) = g(y) \\
M, g \vDash K_x y & \text{iff} & \text{for all } g', (g' \sim_x g) \text{ implies } g'(y) = g(y) \\
M, g \vDash \neg\varphi & \text{iff} & M, g \nvDash \varphi \\
M, g \vDash \varphi \wedge \psi & \text{iff} & M, g \vDash \varphi \text{ and } M, g \vDash \psi \\
M, g \vDash K_x\varphi & \text{iff} & \text{for all } g', (g' \sim_x g) \text{ implies } M, g' \vDash \varphi \\
M, g \vDash [X]\varphi & \text{iff} & \text{for all } g', (g' \mathbb{R}^X g) \text{ implies } M', g' \vDash \varphi
\end{array}$$

where, $g' \mathbb{R}^X g$ is defined as follows:

$$g' \mathbb{R}^X g \text{ iff } \begin{cases} g(x)Rg'(x) & \text{for } x \in X \\ g(x) = g'(x) & \text{for } x \notin X \end{cases}$$

The operator $[X]$ signifies "after all possible moves for the players in X". After making these new moves the equivalence relation between the various assignment functions have to be updated. These updated relations will form a new model M' which only differs from M in its equivalence relations $\{\sim_x\}_{x\in V}$. In the new model M', two assignments h and h' are equivalent iff,

- For $x \in X$,
  - $h(x) = h'(x)$ and $\exists g' \in \sim_x (g)$ for some assignment g, such that, $\forall y \in X$, $g\mathbb{R}^y h$ and $g'\mathbb{R}^y h'$: After a move of players in X, they remember the previous equivalent assignments.
  - $\forall z, h(x)Rh(z) \leftrightarrow h'(x)Rh'(z)$ and $h(z) = h'(z)$ if $h(x)Rh(z)$: After a move of players in X, they see the new states and reduce uncertainty.
- For $x \notin X$,
  - $\exists g' \in \sim_x (g)$ for some assignment g, such that, $\forall y \in X$, $g\mathbb{R}^y h$ and $g'\mathbb{R}^y h'$: After a move of players in X, the other players are unaware of the move.
  - $\forall z, h(x)Rh(z) \leftrightarrow h'(x)Rh'(z)$ and $h(z) = h'(z)$ if $h(x)Rh(z)$: After a move of players in X, they see the new states update their knowledge.

Using these formulas we can express various properties like "can the seekers know the location of the hider in a particular number of moves, n say" or "can the hider avoid the seekers for say, k moves". Under certain conditions, winning plays and strategies of the players may also be expressed in this logic.

3 Model-checking for games with 1 hider and 1 seeker

In this section we look at the model-checking problem for a variant of Sim-ELHS with a single seeker and a single hider. The assignments for these games will have positions for the two players, and the set X in the operator $[X]$ will always be a singleton set containing either the seeker or the hider. For checking the satisfiability of a formula in Sim-ELHS at a given model with respect to a given assignment function, we use the model-checking algorithm 1.

Algorithm 1 takes the model M, initial assignment g and formula φ as inputs. Based on the structure of the formula, it then performs the necessary steps to check the satisfiability of the formula on the model for the given assignment. This may also involve recursive calls to itself depending on the nature of the sub-formulas. For example, if the formula is of the form $\varphi := \varphi_1 \wedge \varphi_2$ then two recursive calls are made to check the validity of φ_1 and φ_2 respectively (Line 8). Some steps may also require parsing of the equivalent assignments to check the validity of some formulas (Line 10 and Line 17). The algorithm might also make calls to $updateModel$ to update the equivalence relations in the case where $\varphi := [X]\varphi_1$ (Line 26). The algorithm finally returns **True** or **False** based on the satisfiability of the formula on the given model and the given assignment.

Algorithm 2 is called by the $modelCheck$ algorithm to analyze a move made by any player. In this case the algorithm creates a new model called M' which is exactly same as M with the updated equivalence relations (Line 1). In this algorithm we consider all equivalent assignments independent of the information available to the players at the previous stage. The algorithm then filters them based on the previous information to get the set of true equivalent assignments. The initial equivalence relations are created by utilizing Algorithm 3 (Line 4 and Line 9) and the filtering is done by Algorithm 4. We then take all these assignments and filter out the impossible ones based on the previous knowledge

Algorithm 1 modelCheck (1 seeker and 1 hider)

Require: M, g, φ
Ensure: **Truth** or **Falsity** of φ at (M, g)
if $\varphi := Px_1x_2$ **then**
  return $(g(x_1), g(x_2)) \in P$
else if $\varphi := x_i \equiv x_j$ **then**
  return $g(x_i) = g(x_j)$
else if $\varphi := \neg\psi$ **then**
  return $\neg\ modelCheck(M, g, \psi)$
else if $\varphi := \psi \wedge \chi$ **then**
  return ($modelCheck(M, g, \psi)$ and $modelCheck(M, g, \chi)$)
else if $\varphi := K_i j$ **then**
  for $g' \in \sim_i (g)$ **do**
    if $g'(j) \neq g(j)$ **then**
      return False
    end if
  end for
  return True
else if $\varphi := K_i\psi$ **then**
  for $g' \in \sim_i (g)$ **do**
    if not $modelCheck(M, g', \psi)$ **then**
      return False
    end if
  end for
  return True
else if $\varphi := [X]\psi$ **then**
  for $w \in \{w \mid g(X)Rw\}$ **do** ▷ Get a list of all possible moves
    $h \leftarrow \{(V - X, g(V - X)), (X, w)\}$
    $M' \leftarrow updateModel(M, g, h, X)$ ▷ Create the updated model
    if not $modelCheck(M', h, \psi)$ **then**
      return False
    end if
  end for
  return True
end if

Algorithm 2 updateModel

Require: M, g, h, X
Ensure: M', the updated model
$M' \leftarrow M$
$i \leftarrow X$ and $j \leftarrow V - X$
Initialize $\sim_i$ and $\sim_j$
for $h' \in getEquivalent(M, h, i, j)$ **do**
  if $checkEquivalent(M, g, h, h', i, j)$ **then**
    update $\sim_i$ with h and h'
  end if
end for
for $h' \in getEquivalent(M, h, j, i)$ **do**
  if $checkEquivalent(M, g, h, h', j, i)$ **then**
    update $\sim_j$ with h and h'
  end if
end for
$M' \leftarrow \sim_i$ and $\sim_j$
return M'

Algorithm 3 getEquivalent

Require: M, h, i, j
Ensure: $\{h' | h' \sim h\}$
if $h(i)Rh(j)$ **then** ▷ The other player is visible
  return {h}
end if
$candidates \leftarrow \{\}$
for $w \in D$ **do**
  if not $h(i)Rw$ **then**
    $candidates \leftarrow candidates \cup \{(i, h(i)), (j, w)\}$
  end if
end for
return $candidates$

(Line 5 and Line 10). These new relations provide the updated knowledge of the players (Line 14).

Algorithm 3 is called by the *updateModel* algorithm to generate all possible equivalent assignments. For a single seeker and a single hider generating all equivalent assignments involves two steps. The first step is to check if player i can see player j (Line 1). In this case the equivalence relation only contains h, the current assignment (Line 2). The second step happens if the other player is not visible. In this case, we iterate over all possible positions for the other player and create a list of assignment functions to be returned (Line 5).

Algorithm 4 checkEquivalent

Require: M, g, h, h', i, j
Ensure: **True** or **False**
1: **for** $g' \in \sim_i (g)$ **do**
2: $equiv \leftarrow True$
3: **if** not $(g(i)Rh(i)$ and $g'(i)Rh'(i))$ **then**
4: $equiv \leftarrow False$
5: **end if**
6: **if** *equiv* **then**
7: **return** True
8: **end if**
9: **end for**
10: **return** False

Algorithm 4 is called by the *updateModel* algorithm to filter the assignments. This algorithm takes possibly equivalent assignments to the current assignment g and then uses the previous knowledge to check the equivalence (Line 1). If the two assignments are equivalent, the algorithm returns **True**. In any other case the algorithm returns **False**. The *updateModel* algorithm then constructs the new equivalence relation based on these assignments.

3.1 Correctness of the model-checking algorithm

Now that we have looked at the algorithm for model-checking, we need to show the correctness of the algorithm. We back up this claim by showing a recursive relation between the satisfaction of the formula and the output of the algorithm.

Theorem 1. *A formula φ is satisfied on a model M with initial assignment g iff the output of the Algorithm modelCheck (1 seeker and 1 hider) is* ***True****.*

Proof. Even though the correctness follows quite easily from the corresponding steps in the algorithm, we give it below, just for the sake of completion. To show the correctness of the algorithm we use the principle of strong induction on the size of a formula. For clarity, we define the size of a formula φ, denoted by $|\varphi|$,

using the following recursive definition.

$$\begin{aligned} |Px_1x_2| &= 1 \\ |x_1 \equiv x_2| &= 1 \\ |K_{x_i}x_j| &= 1 \\ |\neg\varphi| &= 1 + |\varphi| \\ |\varphi_1 \wedge \varphi_2| &= 1 + |\varphi_1| + |\varphi_2| \\ |K_{x_i}\varphi| &= 1 + |\varphi| \\ |[X]\varphi| &= 1 + |\varphi| \end{aligned}$$

As an example, for the formula $\varphi := (Px_1x_2) \wedge (x_1 \equiv x_2)$, $|\varphi| = 3$. For a formula φ with $|\varphi| = n$, a model M and initial assignment g, we induct on the size of the formula.

- Base case :
  - $\varphi := Px_1x_2$,
    For this case, the algorithm checks the current assignment g of the two agents against the predicate symbol P and returns true iff $(g(x_1), g(x_2)) \in P$, which is handled by Lines 1 and 2 of the algorithm.
  - $\varphi := x_1 \equiv x_2$
    For this case, the algorithm checks the current assignment g of the two agents, checks the equality and returns true iff $g(x_1) = g(x_2)$, which is handled by the Lines 3 and 4 of the algorithm.
  - $\varphi := K_{x_i}x_j$
    For this case, the algorithm checks the current assignment g of the two agents, checks the equivalent assignments of x_i (given by $\sim_i$) and returns true iff, for all equivalent assignments g', $g'(x_j) = g(x_j)$, which is handled by the Lines 9 to 15 of the algorithm.
- Induction Hypothesis:
  We assume that for all formulas φ with $|\varphi| \leq n$ the statement holds true.
- Induction Step:
  With our hypothesis, we show that for a formula φ with $\varphi = n + 1$ the statement holds true. For such a φ we have the following cases,
  - $\varphi := \neg\varphi'$
    Since $|\varphi'| = n$, by our induction hypothesis, $M, g \vDash \varphi'$ iff the algorithm returns true for φ'. For φ, if φ' evaluates to true the algorithm returns false and vice versa. This is handled by the Lines 5 and 6 of the algorithm.
  - $\varphi := \varphi_1 \wedge \varphi_2$
    Since $|\varphi_1| < n$ and $|\varphi_2| < n$, by our induction hypothesis, if any of the sub-formulas evaluate to true the algorithm also returns true for the same formula and vice versa. For φ, the algorithm will returns true iff both the sub-formulas evaluate to true. This is handled by the Lines 7 and 8 of the algorithm.
  - $\varphi := K_{x_i}\varphi'$
    Since $|\varphi'| = n$, by our induction hypothesis, for any model M' and

assignment h the algorithm will return true iff $M', h \vDash \varphi'$. The algorithm iterates over all equivalent assignments for g (from $\sim_i$) and checks the validity of φ' for each one. If any such assignment evaluates to false, the algorithm would return false for the valuation of φ. If no such assignment is found, the algorithm would return true instead. This is handled by the Lines 16 to 22 of the algorithm.

- $\varphi := [X]\varphi'$
  Since $|\varphi'| = n$, by our induction hypothesis, the satisfiability of φ' is correctly evaluated by the algorithm. For φ, the algorithm iterates over all the possible moves for the agents in X and creates an updated world (with updated equivalence relations $\sim$) for each such move. Finally the algorithm makes recursive calls with the updated models and updated assignment function for φ'. If any of the recursive calls returns false, the algorithm would return false for φ. If no such model-assignment pair exists the algorithm returns true instead. This is handled by the Lines 23 to 31 of the algorithm.

This completes the proof of correctness of the algorithm.

3.2 Complexity Analysis of the model-checking algorithm

In order to understand the complexity of the model-checking problem for Sim-ELHS with one hider and one seeker, we analyze the algorithms given above and find bounds for the number of steps involved.

Theorem 2. *modelCheck (1 seeker and 1 hider) runs in time polynomial in the size of the model and the size of the formula.*

Proof. To provide the runtime of the algorithm we look at the total number of computation steps the algorithm takes for a model $M = (D, R, \{P\}_{P\in\mathcal{P}}, \mathcal{G}, \{\sim_x\}_{x\in V})$ with initial assignment g and formula φ. This number is influenced by the number of recursive calls made by the algorithm and the number of computation steps taken by each recursive call.

- The number of recursive calls made by the algorithm can be checked by dividing the formula into various cases and sub-formulas corresponding to these cases.
  - For the cases where ($\varphi := Px_1x_2$), ($\varphi := x_1 \equiv x_2$) and ($\varphi := K_{x_i}x_j$) there are no recursive calls made by the algorithm.
  - For the cases where $\varphi := \neg\varphi'$ and $\varphi := \varphi_1 \wedge \varphi_2$ the number of recursive calls are 1 and 2 respectively.
  - For the cases where $\varphi := K_i\varphi'$ and $\varphi := [X]\varphi'$ the number of recursive calls being made are equal to the number of equivalent assignments g' to g. Since the number of equivalent assignments are linearly bounded by the number of states in the model ($|D|$) we can infer that the number of recursive calls in each occurrence of these operators is $O(|D|)$.

Now that we know the number of recursive calls for each operator we look at the number of occurrences for these operators. Since the number of subformulas is linearly bounded by the size of the formula φ, we can infer that the frequency of these operators is $O(|\varphi|)$. So the total number of recursive calls for this formula would be bounded by the product of the number of recursive calls for each operator and the frequency of that operator in the formula. This gives us the result that the total number of recursive calls by the algorithm is $O(|D| \times |\varphi|)$

- The number of steps required by each call of the algorithm can be checked for each of the possible cases.
  - For the operators P, $\equiv$, $\neg$ and $\wedge$ the algorithm only requires a constant number of steps. These steps may either include checking the predicate symbol, checking assignments of the agents or make recursive calls and return the result.
  - For the operators $K_i j$ and $K_i \psi$ the algorithm iterates over possible equivalent assignments and performs a constant number of steps for each iteration. As we have seen, the number of equivalent assignments is linearly bounded by the number of states, $|D|$. Thus, the number of steps involved can be written as $O(|D|)$.
  - For the operator $[X]$, the main computation steps involve the *updateModel* algorithm. For each call of the this algorithm, the equivalent assignments are computed for both agents and then, each such equivalent assignment is used to update the equivalence relation ($\sim$). So the total number of steps for this case can be estimated by:

$$\begin{aligned} &\text{(Number of moves for } X)\times \\ &\text{(Number of equivalent assignments for each move)}\times \\ &\text{(Update equivalence relation for each assignment)} \\ &= O(|D|) \times O(|D|) \times O(|D|) \\ &= O(|D|^3) \end{aligned}$$

Using the above arguments we can infer the total number of steps to be:

$$\begin{aligned} &\text{Total number of recursive calls} \times \text{Number of steps for each call} \\ &= O(|D| \times |\varphi|) \times O(|D|^3) \\ &= O(|D|^4 \times |\varphi|) \end{aligned}$$

So the time complexity of the algorithm is polynomial in the size of the model and the size of the formula.

As an immediate corollary of the above result, we have that:

Corollary 1. *The model-checking problem for Sim-ELHS with one seeker and one hider is in PTIME.*

4 Model-checking for Sim-ELHS with 1 hider and more than 1 seeker

From the viewpoint of hide and seek game, it might be more relevant to talk about one seeker and more than one hiders. However, as we mentioned in Section 1, the cops and robber game [**?**] is quite relevant to our study. Following the original formulation of the game with more than one cops, we consider a variant of Sim-ELHS with $n > 1$ seekers and one hider. We showcase a model-checking algorithm for this variant, where, instead of a single one, different seekers trying to find a hider in tandem. This algorithm follows a similar design to the algorithms in the previous section, with some additional computations.

Algorithm 5 modelCheck ($n > 1$ seekers and 1 hider)

Require: M, g, φ
Ensure: **Truth** or **Falsity** of φ at (M, g)
if $\varphi := Px_1x_2\cdots x_k$ **then**
  return $(g(x_1), g(x_2), \cdots, g(x_k)) \in P$
else if $\varphi := x_i \equiv x_j$ **then**
  return $g(x_i) = g(x_j)$
else if $\varphi := \neg\psi$ **then**
  return $\neg\ modelCheck(M, g, \psi)$
else if $\varphi := \psi \wedge \chi$ **then**
  return ($modelCheck(M, g, \psi)$ and $modelCheck(M, g, \chi)$)
else if $\varphi := K_i j$ **then**
  for $g' \in \sim_i (g)$ **do**
    if $g'(j) \neq g(j)$ **then**
      return False
    end if
  end for
  return True
else if $\varphi := K_i\psi$ **then**
  for $g' \in \sim_i (g)$ **do**
    if not $modelCheck(M, g', \psi)$ **then**
      return False
    end if
  end for
  return True
else if $\varphi := [X]\psi$ **then**
  for $h \in getMoves(M, g, X)$ **do** ▷ Get a list of all possible moves
    $M' \leftarrow updateModel(M, g, h, X)$ ▷ Create the updated model
    if not $modelCheck(M', h, \psi)$ **then**
      return False
    end if
  end for
  return True
end if

Algorithm 5 takes the model M, initial assignment g and a formula φ as inputs. Similar to the previous algorithms it iterates over the sub-formulas recursively and applies the necessary operations based on the structure of the sub-formulas. The algorithm makes calls to two other algorithms, *getMoves* (Line 24) and *updateModel* (Line 25) in the case of the $[X]$ operator. The algorithm finally returns true or false based on the satisfiability of the input formula in the input model.

Algorithm 6 getMoves

Require: M, g, X
Ensure: Set of moves
1: $candidates \leftarrow \{0\}$
2: **for** $i \in V$ **do**
3: **if** $i \in X$ **then**
4: $candidates \leftarrow candidates \times \{w \mid g(i)Rw\}$ ▷ all moves for i
5: **else if** $i \notin X$ **then**
6: $candidates \leftarrow candidates \times \{g(i)\}$ ▷ i remains in the same state
7: **end if**
8: **end for**
9: **return** *candidates*

Algorithm 6 is called by the *modelCheck* algorithm and is used to generate a list of all possible moves from the current position for the set of players X. The algorithm starts by considering each agent (Line 2) and checking if they are a part of X (Line 3). If not, their positions will remain the same, which is indicated by a product of the other players' positions with their current positions (Line 4). If they are part of X, all possible neighbouring states are taken into consideration and indicated by a product of all players' positions with their own possible moves (Line 6). For example, if seeker S_1 has 2 possible moves to states w_1 and w_2 and seeker S_2 has 2 possible moves to states w_3 and w_4, with hider at w_5 we get,

$$\{w_1, w_2\} \times \{w_3, w_4\} \times \{w_5\} = \{(w_1, w_3, w_5), (w_1, w_4, w_5), (w_2, w_3, w_5), (w_2, w_4, w_5)\}$$

This list corresponds to the list of new assignments after all possible moves, which is returned by the algorithm to be iterated over to check the satisfiability of the formulas in the given model.

Similar to Algorithm 2 in the previous section, Algorithm 7 is used to generate an updated model M'. Like earlier, the updated model M' differs from the model M in its equivalence relations. For each agent, the algorithm calls *getEquivalent* to create a set of equivalence assignments for the agent (Line 4). These equivalent assignments are generated independent of previous knowledge. Once the list of potential equivalent assignments are generated, they are filtered based on previous knowledge using the algorithm *checkEquivant* (Line 5). The

Algorithm 7 updateModel

Require: M, g, h, X
Ensure: M', an updated model
1: $M' \leftarrow M$
2: **for** $i \in V$ **do**
3: $\sim_i' \leftarrow \{\}$
4: **for** $h' \in getEquivalent(M, h, i)$ **do**
5: **if** $checkEquiv(M, g, h, h', X)$ **then**
6: Add h, h' to $\sim_i'$ as equivalent
7: **end if**
8: **end for**
9: Add $\sim_i$ to M'
10: **end for**
11: **return** M'

final filtered list is used to update the equivalence relation for the agent (Line 9). Once all the updated relations are generated, the updated model is returned (Line 11).

Algorithm 8 getEquivalent

Require: M, h, i
Ensure: $\{h' | h \sim h\}$
1: $candidates \leftarrow \{0\}$
2: **for** $x \in V$ **do**
3: **if** $i = x$ or $h(i)Rh(x)$ or $h(i) = h(x)$ **then**
4: $candidates \leftarrow candidates \times \{h(i)\}$
5: **else**
6: $candidates \leftarrow candidates \times \{w \mid \neg(h(i)Rw)\}$
7: **end if**
8: **end for**
9: **return** $candidates$

Algorithm 8 is called by the *updateModel* algorithm to get a list of all the equivalent assignments ignoring the previous knowledge. Since the number of players may be greater than one, we need to use products to compute the possible list of equivalent assignments. Similar to the *getMoves* algorithm, if a player is visible to others, we consider a product of its current position together with the positions of the other players (Line 4). If a player is not visible, we take all possible states where it can be and consider the product with the positions of the other players (Line 6). We then have the final list of assignments (Line 9).

Algorithm 9 is used to filter out the possible equivalent assignments based on prior knowledge. For each agent, the algorithm checks the previous knowledge and decides if the two assignments are equivalent or not (Line 4). If the assignments are equivalent, the algorithm returns true. If there exists some prior

Algorithm 9 checkEquivalent

Require: M, g, h, h', X
Ensure: **True** or **False**
for $g' \in \sim_i (g)$ **do**
  $equiv \leftarrow True$
  for $y \in X$ **do**
    if not $(g(y)Rh(y)$ and $g'(y)Rh'(y))$ **then**
      $equiv \leftarrow False$
    end if
  end for
  if $equiv$ **then**
    return True
  end if
end for
return False

knowledge (prior assignments) that disagree on the equivalence, the algorithm returns false. As explained above, various sub-formulas are checked and the final result of the algorithm is returned. The proof of correctness of Algorithm 5 is similar to that of Algorithm 1, and we leave it for our readers. The algorithms only differ in steps corresponding to computing moves and equivalent assignments, and thus we have the following result:

Theorem 3. *A formula φ is satisfied on a model M with initial assignment g iff the output of the Algorithm modelCheck ($n > 1$ seekers and 1 hider) is* ***True****.*

4.1 Complexity Analysis of model-checking

As in Section 3.2, to understand the complexity of the model-checking problem for Sim-ELHS with 1 hider and $n > 1$ seekers, we analyze the algorithms given above and find bounds on the number of steps involved.

Theorem 4. *modelCheck ($n > 1$ seekers and 1 hider) runs in exponential time.*

Proof. To estimate the number of steps required by the algorithm we look at the additional ones needed for a general model (with more than one seeker), $M = (D, R, \{P\}_{P\in\mathcal{P}}, \mathcal{G}, \{\sim_x\}_{x\in V})$ with initial assignment g and formula φ. Similar to the proof of Theorem 2, this number is influenced by the number of recursive calls made by the algorithm and the number of computation steps in each recursive call.

- The number of recursive calls made by the algorithm can be checked by dividing the formula into various cases and corresponding sub-formulas. The atomic and Boolean cases can be dealt with as earlier.
  - For the formulas $K_i\varphi'$ and $[X]\varphi'$, the number of recursive calls that are being made are equal to the number of assignments g', equivalent to g. Unlike the case with 2 agents, the number of equivalent assignments is

exponential in the size of the model ($|D|$). This is due to the fact that for every other agent, the possible number of worlds it can occupy is $O(|D|)$. So, taking into account the number n of possible agents, the total number of equivalent assignments is $O(n^{|D|})$. Thus, the number of recursive calls for these operators is also $O(n^{|D|})$.

Now that we know the number of recursive calls for each sub-formula we look at the number of occurrences of the atomic formulas, Boolean connectives and modal operators. Since the number of sub-formulas is linearly bounded by the size of the formula φ, we can infer that the frequency of these components is $O(|\varphi|)$. So the total number of recursive calls for this formula would be bounded by the product of the number of recursive calls for each such component of the formula and its frequency in the formula. This gives us the result that the total number of recursive calls by the algorithm is $O(n^{|D|} \times |\varphi|)$.

– The steps required for each call of the algorithm can be checked for the possible cases.
  - For the relation symbols P and $\equiv$, and the connectives $\neg$ and $\wedge$, the algorithm requires a constant number of steps. These steps may either include checking for the relations and assignments of the agents or make recursive calls and return the result.
  - For the knowledge operators in $K_i j$ and $K_i \psi$, the algorithm iterates over all equivalent assignments. Since the number of such assignments is $O(n^{|}D|)$ for an agent, the number of steps required for these operators is $O(n^{|D|})$ as well.
  - For the operator $[X]$, we need to consider the number of steps required to create the new model M' generated by the *updateModel* algorithm. The algorithm generates a new model for each possible move, which is again bounded by $n^{|D|}$. For each call of the algorithm, the new equivalence relation ($\sim_i$) is created by checking each new assignment (bounded by $n^{|D|}$) against each previous assignment (bounded by $n^{|D|}$) to infer equivalence. So the total number of steps for this operator is:

$$\begin{aligned}&\text{(Number of moves for } X)\times\\&\text{(Number of equivalent assignments for each move)}\times\\&\text{(Update equivalence relation for each assignment)}\\&= O(n^{|D|}) \times O(n^{|D|}) \times O(n^{|D|})\\&= O(n^{|D|})\end{aligned}$$

  So the amount of computation steps required for each call of the algorithm is $O(n^{|D|})$.

Using the above arguments, we can infer the total number of steps as:

$$\begin{aligned}&\text{Total number of recursive calls} \times \text{Number of steps in each call}\\&= O(n^{|D|} \times |\varphi|) \times O(n^{|D|})\\&= O(n^{|D|} \times |\varphi|)\end{aligned}$$

Thus, we have that the time complexity of the algorithm is exponential in the size of the model and polynomial with respect to the number of agents n and the size of the formula φ.

We have the following corollary:

Corollary 2. *The model-checking problem for Sim-ELHS with $n > 1$ seekers and 1 hider is in EXPTIME.*

We note how the intricacy of model-checking algorithms change based on the number of players, the number of seekers in particular - the run-time complexity of the algorithms jump from PTIME to EXPTIME. Intuitively this makes sense as, increasing the number of seekers would mean that the possible assignments also increase for each player. Evidently, an EXPTIME algorithm for the model-checking problem for Sim-ELHS with $n > 1$ seekers and 1 hider does not necessarily imply that the problem cannot be solved in PTIME. However, this is ensured by the result we present next.

4.2 A discussion on lower bound

In the previous section we showed that the model-checking problem for Sim-ELHS with $n > 1$ seekers and 1 robber is in EXPTIME. To get a better understanding of the complexity of the problem we now provide a lower bound for the problem. We leave the task of finding tight bounds for the future.

Theorem 5. *The model-checking problem for Sim-ELHS with $n > 1$ seekers and 1 hider is NP-hard.*

Proof. To show that the model-checking problem for Sim-ELHS is NP-Hard, we take an existing NP-Complete problem and provide a polynomial-time reduction to the said problem. The existence of such a reduction will imply that the model-checking problem in question is just as hard as the NP-Complete problem being reduced. For this case, we consider the 3-SAT problem, which is a well known NP-Complete problem. Let us consider a general case of the 3-SAT problem as follows:

$$f(x_1, \cdots, x_n) = \bigwedge (x_i \vee x_j \vee x_k)$$

where $x_1, x_2, \ldots, x_n$ are Boolean variables or their complements. The problem involves finding assignments for each of the Boolean variables such that the complete expression f is satisfied. In order to reduce this general case to our model-checking problem, we construct an instance of a Sim-ELHS game based on f in such a way that the seekers can capture the hider in the constructed game iff f is satisfiable. We first look at the construction of the game and then look at how the construction satisfies the desired claim. We have the following steps to construct the game.

- Construct a state for each Boolean variable in f – The set/collection of such states is denoted by W_s.

- Construct a pair of states for each state in W_s, forming two separate collection of states – These sets of states are denoted by W_{assign} and W'_{assign}, respectively. Then construct an edge from each state in W_s to the corresponding state in W_{assign} and that in W'_{assign} – This relation is denoted by R_{assign}.
- For each clause in f construct a corresponding state – This set of states is denoted by W_{clause}. Then construct an edge from each state in W_{assign} and W'_{assign} to the corresponding clause state in W_{clause} where the corresponding variable occurs – This relation is denoted by R_{clause}. For example, if the variable x_1 occurs in clause 2 of a formula there would be an edge from, the state in W_{assign} corresponding to x_1, to the state in W_{clause} corresponding to clause 2.
- Finally we construct a state w_h and construct edges from this state to each state in W_{clause} – This relation is denoted by R_h.

Using these constructions, we get the Sim-ELHS game as the following tuple, $\mathcal{M} = (D, R, \emptyset, \mathcal{G}, \{\sim_x\}_{x \in V})$, where,

$$D = W_s \cup W_{assign} \cup W'_{assign} \cup W_{clause} \cup \{w_h\}$$
$$R = R_{assign} \cup R_{clause} \cup R_h.$$

The set of assignments $\mathcal{G}$ can be generated accordingly. We note that the equivalence relations $\{\sim_x\}_{x \in V}$ do not play any role in this proof. The number of players for this game is $N + 1$ where N corresponds to the number of unique Boolean variables in f. Thus, we have N seekers and 1 hider. For this game, we then set the initial positions of the players as,

- Each seeker starts at a unique state in W_s.
- The hider starts at the state w_h.

This will be the initial position g. Now that we have the model $\mathcal{M}$ and the assignment g, we construct the formula to be checked as follows:

$$\varphi := \langle Seekers \rangle \, [Hider] \, \langle Seekers \rangle \, I$$

where,

$$I := \bigvee_{i \in \{1,2,\ldots,N\}} Hider \equiv Seeker_i$$

Informally this formula signifies the property, "The seekers can capture the hider in 2 moves, where they start the game". With this construction, we informally look at how the game relates to f. Each seeker is assigned to a state corresponding to each variable. A seeker, $Seeker_i$ say, can move to exactly one of 2 states, a state w_i from W_{assign} or a state w'_i from W'_{assign}. This is analogous to the 2 values that any Boolean variable can take. So the seeker moving to the state w'_i is equivalent to assigning false to the variable x_i. Once the seekers choose a state from $W_{assign} \cup W'_{assign}$, the next possible moves for the seekers would be to move

to any of the clause states from W_{clause} that the associated variable occurs in. At this point, if any clause is unreachable by the seekers, due to their assignments, the hider can move to the clause state from w_h which is unreachable, where any move by the seekers will not result in a capture. Analogous to this for the 3-SAT problem, if a particular assignment of variables does not cover every clause in f then the entire formula will evaluate to false.

For a formal argument, let us assume f is satisfiable. This would mean that there exists at least 1 assignment of the variables for which the entire formula evaluates to true. If such an assignment exists there would be a set of moves for the seekers where they can move to the states in $W_{assign} \cup W'_{assign}$ such that every clause is reachable by at least 1 seeker. So any move the hider makes will lead to a capture. So the model will satisfy the formula φ as per our construction. If f is unsatisfiable then every possible assignment of the variables will leave at least 1 clause unsatisfied. This would imply that there does not exist any set of moves for the seekers which can cover all the clause states. So the property is not satisfied. This gives us the result,

$$SAT(f) \text{ iff } M, g \vDash \varphi$$

So we can conclude that the model-checking problem for Sim-ELHS with $n > 1$ seekers and 1 hider is NP-hard.

To exemplify the proof idea, let us consider the following 3-SAT formula,

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_2} \vee x_3)$$

Figure 2 shows the constructed game model based on the formula. In this example, the seekers can move to the states corresponding to the assignments for x_1, x_2, and x_3 and cover each clause state in a way that, for every move of the hider, a seeker can make a move to ensure the capture. Inferring from these moves we also get the assignment of $x_1 = True, x_2 = True$, and $x_3 = True$ which satisfies the given 3-SAT formula.

To look at another example, let us look at the following 3-Sat formula given by,

$$\begin{aligned} g(x_1, x_2, x_3) =& (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_2} \vee x_3) \wedge (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge \\ & (\bar{x_1} \vee x_2 \vee x_3) \wedge (\bar{x_1} \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_1} \vee \bar{x_2} \vee x_3) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_3}) \end{aligned}$$

The formula is unsatisfiable as every assignment of the variables will have exactly 1 clause that evaluates to false. After reducing the formula to an instance of the game, we get the model shown in Figure 3.
For this model, let us denote the assignment states selected by the seekers to be (x'_1, x'_2, x'_3) where x'_i corresponds to either x_i or $\bar{x_i}$. For every such selection, the state corresponding to the clause $(\bar{x'_1} \vee \bar{x'_2} \vee \bar{x'_3})$ will always be safe for the hider to move to, as the seekers have moved to the complementary assignments. So, the Sim-ELHS formula given above would evaluate to false, implying that the 3-SAT formula is unsatisfiable.

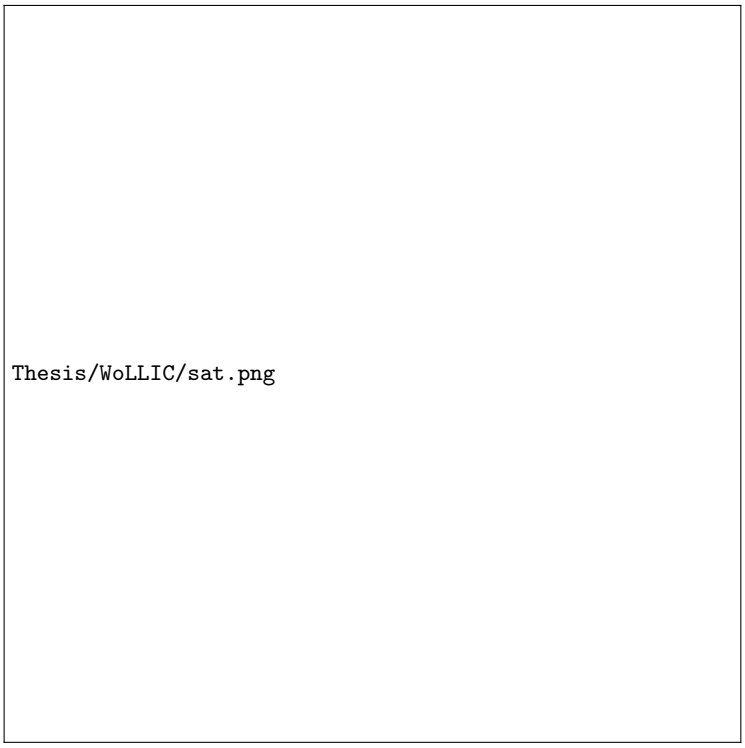


Fig. 2: Model constructed for a 3-SAT formula f

5 On related complexity results

We have explored the model-checking problem for Sim-ELHS, a logic for hide and seek games with imperfect information. As mentioned earlier, the game is a close variant of the cops and robber game [**?**]. In the following, we compare some relevant complexity results regarding solving the latter game.

In [**?**,**?**], the authors consider the problem of checking if the minimum number of cops required to guarantee the capture of the robber is at most k. The problem accepts an instance of the traditional cops and robbers game along with a positive integer k and tries to determine if, for the given game, the robber can be caught by any arrangement of at most k cops. The minimum number of cops required is called the cop number of the graph and is denoted by $c(G)$ in the literature, for example, see [**?**]. In [**?**], the author shows that the c(G)-problem for the traditional variant of the game with perfect information is EXPTIME-Complete. This result is achieved by reducing an instance of the Alternating Boolean Formula game (which is shown to be an EXPTIME-complete problem in [**?**]) to an instance of finding the minimum number of cops required to capture the robber in the traditional variant of the game. In [**?**], the authors consider a variant of the game where the number of steps each player can take is limited

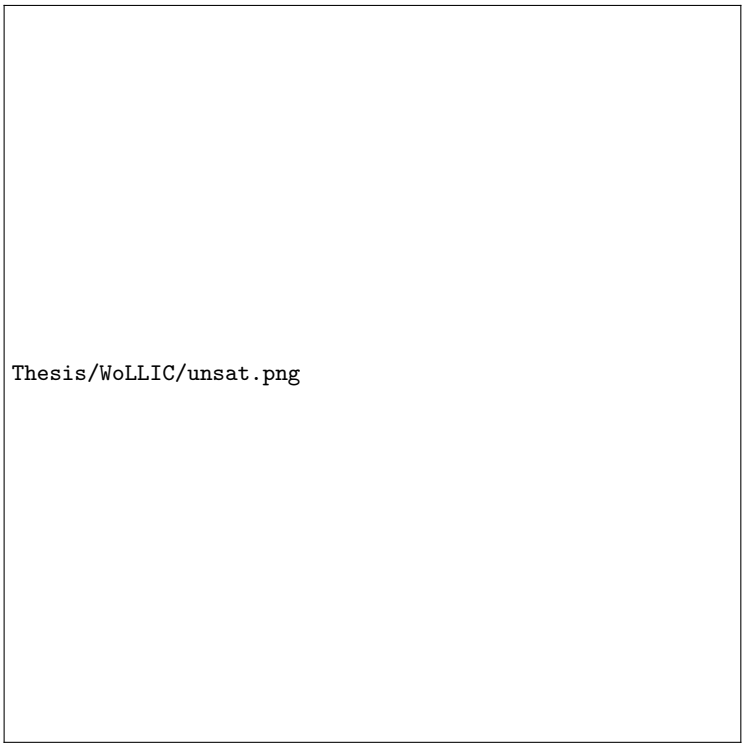


Fig. 3: Model constructed for a 3-SAT formula g

(indicated by an input parameter f). The author shows that the problem of finding the cop number for this variant is PSPACE-Hard.

In this work, we discussed the problem of model-checking Sim-ELHS formulas where one can consider various properties of the game that can be expressed using the logic. These properties can be used to express winning conditions for the players as well as other kinds of statements involving moves and strategies for the players. Some examples of winning conditions could be:

- One of the cops is on the same state as the robber after 5 moves.
- All cops know the location of the robber after 3 moves.
- No moves exist for the robber after 6 moves.

An instance of the game can then be considered to determine whether that satisfies the above-mentioned properties. Another noteworthy observation is how the complexity of the model-checking problem changes as we increase the number of seekers or the cops. For a single seeker and a single hider, the model-checking complexity is fairly straightforward and efficient (being PTIME). However, if the number of seekers increases, the problem becomes harder (being at least NP-Hard). This is due to the fact that unlike the traditional variants of the game,

Sim-ELHS is an imperfect information variant. This leads to additional computations while performing model-checking. Instead of just checking properties on the current configuration of the game, we also need to check the properties on all equivalent configurations. The number of these equivalent configurations are comparable to the different possible arrangements of the players which, grows exponentially with respect to the number of players. Thus, the proposed algorithm for the model-checking problem of Sim-ELHS runs in EXPTIME. It would be interesting to check whether the problem is EXPTIME-hard which we leave for the future.

6 Model-checking tool for graph games

Before concluding this chapter, we present MCGG, a model-checker for Graph Games (https://github.com/Soham-Banerjee/MCGG), which is a work in progress. The present version of the tool can deal with LHS [**?**], the logic of hide and seek games with perfect information. We are currently working on ELHS and Sim-ELHS. Thus, in what follows, we would be describing model-checking for LHS. Even though the games covered in this section may differ from those described in the previous sections, they serve as a good benchmark to understand the implementation aspects of model-checking. The results also showcase different optimizations and techniques that can be used to further assist in reducing the space and time required while model-checking complex games scenarios.

The tool is a symbolic model-checker [**?**] that accepts graph games and properties in the form of models and formulas. These are then processed to generate internal representations which can be used to check the satisfiability of the formula on the model for the given initial assignment. The internal representations of the models and formulas are encodings which optimize memory usage and efficiency. Since the tool is a symbolic model-checker, these internal representations are in the form of Binary Decision Diagrams (BDDs) [**?**].

A BDD is a directed, rooted and acyclic graph representation of a Boolean formula. While doing model-checking, we often check various queries like whether two states are adjacent, or, whether a propositional symbol is true at a state. These queries can be expressed as Boolean formulas [**?**], which in turn can be expressed as BDDs. The advantage of this method is that, instead of storing the entire model, we store the properties of the models as BDDs. This saves a large amount of memory and greatly reduces the query time. For the case of graph games, the model-checker constructs two BDDs, one to encode the existence of specific states in the model (D) and the other to encode the connectivity between the states (R). For example, if we need to check: "Does there exist an edge from w_1 to w_2?", we use the relation BDD and pass the encodings of the states, w_1 and w_2 to it. If the BDD returns true, we can say that an edge exists. The formulas are encoded as inorder trees for efficient storage.

The MCGG uses these principles to perform model-checking on the input model. The architecture of the model is shown in Figure 4. The tool is built on

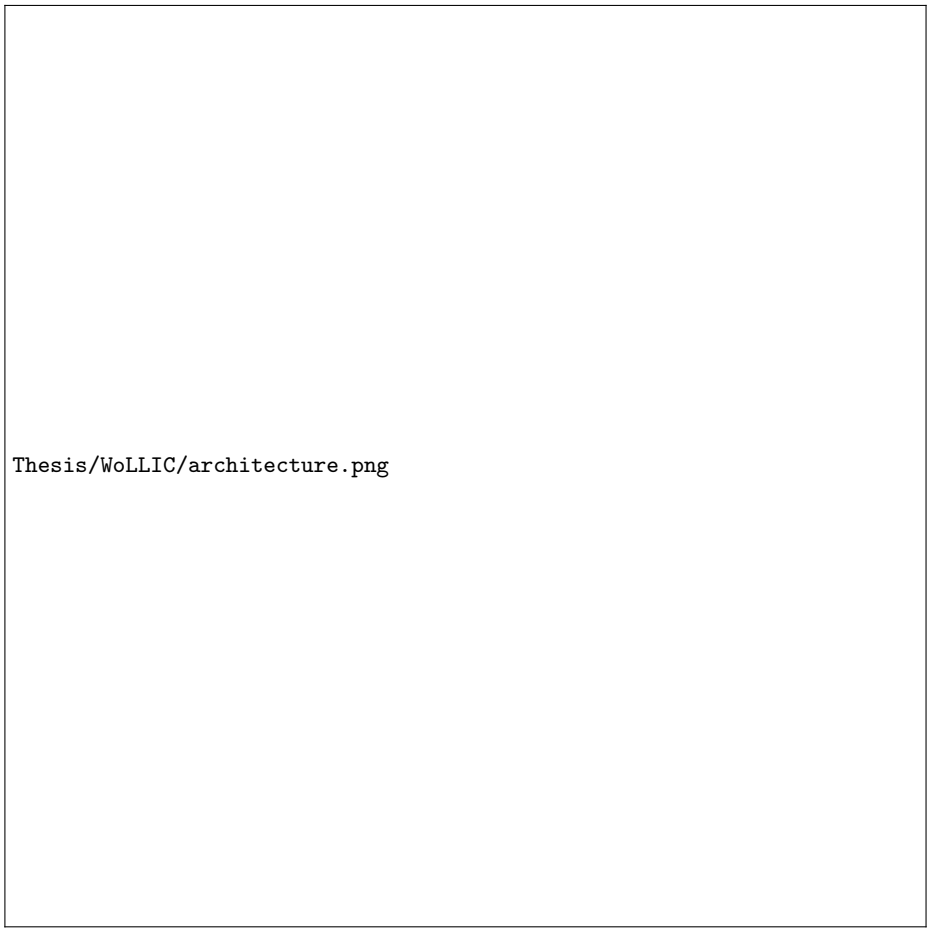


Fig. 4: Architecture of MCGG

the Python programming language and uses tulip-DD [**?**] libraries for handling BDDs. The various components of the tool are described as follows:

- **Inputs** – The tool takes as input the model description (states, relation, valuations) in a yaml file and a formula as text.
- **State Encoder** – This reads the yaml file to encode the existence of the states in the model into a Boolean formula and constructs an associated BDD. The resultant **State BDD** can be used to query the existence of a specific state in the model.
- **Edge Encoder** – This reads the yaml file to encode the relations defined over the states into a Boolean formula and constructs an associated BDD. The resultant **Edge BDD** can be used to query the existence of an edge between two states.

- **Formula Encoder** – This reads the input formula, checks the syntax of the said formula and constructs an inorder tree for aid in parsing during the model-checking process.
- **Formula Checker** – This takes the formula parse tree and generates queries to check satisfiability of the given formula. These queries are then run on the BDDs for various computations like retrieving the neighbors of a state and checking existence of a particular state in the model. These steps may involve updating the BDDs based on the nature of the graph game. The module finally returns **TRUE** or **FALSE** based on the satisfiability of the formula on the given model.

The tool accepts the input model in the form of a yaml file which is a representation of the model tuple. The formula and the initial assignment of the players are added as a separate text file. The model outputs True or False based on the satisfiability of the formula and also displays the resultant graph. Figure 6 showcases a graph game played on a 3 by 3 grid with 2 seekers and 1 hider which we use in the explanation of the results. The tool is built to support various graph games ranging from usual travel games [**?**], to hide and seek/cops and robber games [**?**] to sabotage games [**?**]. These games have different logic frameworks describing them, but the model-checking process is quite similar to the algorithms highlighted in the earlier sections.

For this chapter, we focus on the tool verifying properties of the hide and seek games with perfect information [**?**]. To exhibit how the tool can aid in verifying properties of the said games with imperfect information, we note that the tool already uses BDD updates to incorporate structural changes in the graphs, to deal with sabotage games [**?**]. The same technique can be used to deal with the games with imperfect information, so as to bring ELHS and Sim-ELHS within the purview of the MCGG tool.

6.1 Tool Demonstration

In this section, we consider various examples of hide and seek games with the seeker(s) moving first, to demonstrate the performance of the tool.

- A triangle with an external state, as given in Figure 5. The game has one seeker at w1 and the hider at w4.
- A 3 by 3 grid, as given in Figure 6. The game has two seekers at opposite corners of the grid with the hider in the center.
- A 5 by 5 grid, as given in Figure 7. The game has two seekers at the center of left and top edges, respectively and one seeker at the opposite corner, and the hider in the middle state.
- An octagonal graph, as given in Figure 8. The two seekers are at w3 and w6, respectively, and the hider is on w1.

For these graphs we look at the following three properties to be checked:

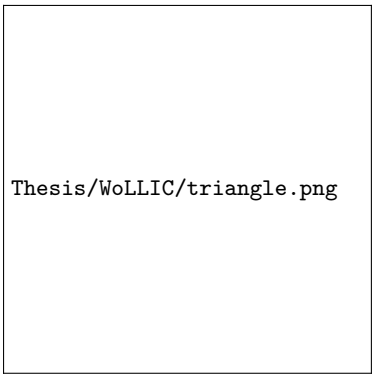


Fig. 5: A hide and seek game on a small graph

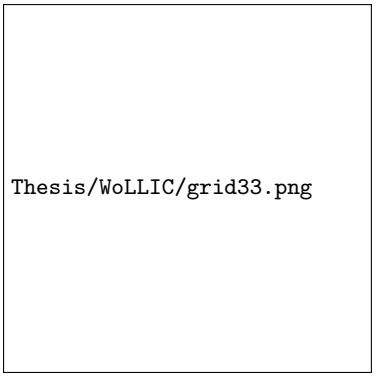


Fig. 6: A hide and seek game on a 3 by 3 grid

– **Can the seekers capture the hider in 3 moves?** This property is given by the following formula:

$$\varphi_1 := (\langle S \rangle [H])^3(I) \vee (\langle S \rangle [H])^2(I) \vee (\langle S \rangle [H])(I) \vee I$$

– **Can the seekers capture the hider in 5 moves?** This property is given by the following formula:

$$\varphi_2 := \bigvee_{i=0}^{5} (\langle S \rangle [H])^i(I)$$

– **Can the seeker avoid capture by not moving for 2 moves?** This property is given by the following formula,

$$\varphi_3 := (\langle S \rangle)^2(I) \vee (\langle S \rangle)(I) \vee I$$

Here, $\langle S \rangle$ corresponds to some move for all seekers, $[H]$ corresponds to all possible moves for the hider and I corresponds to the case where any of the seekers is on

Fig. 7: A hide and seek game on a 5 by 5 grid

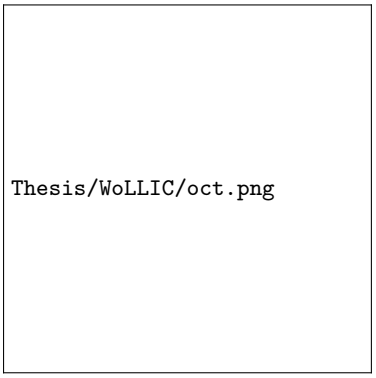


Fig. 8: A hide and seek game on an octagonal graph

the same state as the hider given by,

$$I := \bigvee_{i=1}^{n} (s_i \equiv h)$$

Table 1 tabulates the results obtained on the aforementioned examples. For each of the properties the table indicates if the given property is satisfied by the model and shows the amount of time taken by the tool to compute the results.

7 Conclusion

In this chapter we looked at a variant of the hide and seek game: hide and seek games with imperfect information from a model-checking perspective and

Graph	φ_1		φ_2		φ_3	
	Result	**Time**	**Result**	**Time**	**Result**	**Time**
Small Graph	True	0.3 sec	True	0.3 sec	False	0.2 sec
Octagonal Graph	False	0.4 sec	True	0.6 sec	True	0.3 sec
3x3 Grid	True	0.6 sec	True	0.7 sec	False	0.4 sec
5x5 Grid	False	1.1 min	True	2 mins	False	45 sec

Table 1: Results table for the hide and seek games

explored the run-time complexity of the algorithms. In process, we obtained some complexity bounds on the model-checking problems for the logic Sim-ELHS describing the said game. We showed that while the model-checking problem for the game with one seeker and one hider is PTIME, with more than one seeker and one hider, the problem becomes NP-hard. In fact, we presented an EXPTIME algorithm for the latter problem. We finished the chapter by showcasing a model-checking tool for graph games to get a sense of a practical implementation of the hide and seek game.

There are various avenues of future work, finding tight bounds for the model-checking problem for Sim-ELHS with more than one seekers is of immediate concern among them. We conjecture that the problem is EXPTIME-complete. Moving away from the computational behavior, there are various properties of Sim-ELHS that can be explored, e.g., finite axiomatization, model equivalence vs. modal equivalence, among others. From the cops and robber game literature, we can also consider different variants of the game, and explore their inter-relationships with respect to the reasoning capacities of the players. We can also look at games with simultaneous moves for all the players and explore how such moves affect the properties of these games.

Acknowledgements. We thank the reviewer for the constructive comments that helped to improve the paper. Sujata Ghosh acknowledges Department of Science and Technology, Government of India for financial support vide Reference No DST/CSRI/2018/202 under Cognitive Science Research Initiative (CSRI) to carry out this work.