Journal of Artificial Intelligence Research 66 (2019) 901-935 Submitted 12/2018; published 12/2019

General Game Playing with Imperfect Information

Michael Schofield MSCHOFIELD@CSE.UNSW.EDU.AU
Michael Thielscher MIT@CSE.UNSW.EDU.AU
School of Computer Science and Engineering
UNSW Sydney, Australia

Abstract

General Game Playing is a field which allows the researcher to investigate techniques that might eventually be used in an agent capable of Artificial General Intelligence. Game playing presents a controlled environment in which to evaluate AI techniques, and so we have seen an increase in interest in this field of research. Games of imperfect information offer the researcher an additional challenge in terms of complexity over games with perfect information. In this article, we look at imperfect-information games: their expression, their complexity, and the additional demands of their players. We consider the problems of working with imperfect information and introduce a technique called HyperPlay, for efficiently sampling very large information sets, and present a formalism together with pseudo code so that others may implement it. We examine the design choices for the technique, show its soundness and completeness then provide some experimental results and demonstrate the use of the technique in a variety of imperfect-information games, revealing its strengths, weaknesses, and its efficiency against randomly generating samples. Improving the technique, we present HyperPlay-II, capable of correctly valuing information-gathering moves. Again, we provide some experimental results and demonstrate the use of the new technique revealing its strengths, weaknesses and its limitations.

1. Introduction

General Game Playing (GGP) is becoming popular with Artificial Intelligence researchers as it presents the challenge of designing a system capable of playing a game, without prior knowledge or learning. Such games are described in the Game Description Language, specifying the initial state, legal moves, percepts (signals), terminal conditions and reward structure (Genesereth, Love, & Pell, 2005). Games are invented that test the boundaries of computing technology with intractable search spaces and no clear heuristic or strategy for winning. In this arena, imperfect-information games are the most complex to play and the least explored by researchers.

In this article we present the HyperPlay technique as a "bolt-on" solution to convert a General Game Player designed to play perfect-information games into a player capable of playing imperfect-information games. HyperPlay is not a move planning technique, it is a method of sampling an information set so that the host player can plan moves using the samples. In this paper we choose a simple host based on a Monte Carlo player. We use a Monte Carlo host as it is not a "good" player; we resource the host so that it makes a large variety of moves (from one playout to the next) and occasionally makes poor moves, testing the robustness of the HyperPlay technique across the full extent of the game tree.

©2019 AI Access Foundation. All rights reserved.

We extend the formalism for imperfect-information games and players, we then introduce the technique in sufficient detail so that it can be reproduced, we detail several implementations of the technique and experimental data that highlights its strengths, weaknesses, limitations, and investigate its effectiveness and its efficiency.

A note on terminology: In the game playing community the term “imperfect” is used where general AI often uses the term “incomplete.” We will use “imperfect” information to mean that the game play may be hidden from one or more players. The initial state, reward structure, etc. are declared in the rules which are fully known to all players.

1.1 Related Research

The General Game Playing community has been slow to pick up the challenge of imperfect-information games and only a few players have been implemented with even fewer competitions being conducted. The published approaches to designing general game players for GDL-II show how existing perfect-information GDL players can be lifted to play general imperfect-information games by using models or sampling of an information set as the starting states for a perfect-information search (Edelkamp, Federholzner, & Kissmann, 2012; Schofield, Cerexhe, & Thielscher, 2012). This takes advantage of previous research into information set generation for games specified in a variant of the Planning Domain Definition Language (PDDL) (Richards & Amir, 2012) and the use of particle system techniques (Silver & Veness, 2010) with the sample of an information set either generated, or maintained from one round to the next. Application of these techniques in the General Game Playing setting requires their adaptation to the general Game Description Language (GDL), which, in contrast to PDDL, supports the syntax of logic programming to define game rules, including negation-by-failure, and can handle arbitrary state- and action-dependent percepts, aka. observation tokens (Engesser, Mattmüller, Thielscher, & Nebel, 2018). The latter also distinguishes model sampling in GGP from information set sampling for specific games in which a small, pre-defined set of possible observations can be sampled directly, e.g. in Kriegspiel (Ciancarini & Favini, 2010).

The GGP-II international competitions have sparked the development of several players. Edelkamp et al. (2012) have built a GDL-II player Nexusbaum based on perfect-information Monte Carlo sampling, which they compared against Fluxii, an imperfect-information extension of Fluxplayer (Schiffel & Thielscher, 2007) that maintains the full information set throughout a game. An early development version of a HyperPlayer also competed in the first international competition, coming second out of only three competitors. At the Australasian Joint Conference on Artificial Intelligence[1] there was GDL-II track encouraging entries by CadiaPlayer (Bjornsson & Finnsson, 2009), LeJoueur (Hufschmitt, 2014) and Nexusbaum (Edelkamp et al., 2012).

Outside of General Game Playing, the Monte Carlo tree search technique based on a sample of an information set has been applied to a variety of specific perfect-information and imperfect-information games alike (Browne et al., 2012). Frank and Basin (1998) have presented a ‘game-general’ tree search algorithm for the card game Bridge that exploits a number of imperfect-information heuristics. For the same game, Ginsberg (2001) has applied perfect-information Monte Carlo sampling. Similar special case applications of

1. See https://wiki.cse.unsw.edu.au/ai2012/GGP.

sampling to reduce imperfect to perfect information can be found in Kupferschmid and Helmert (2007). The Alberta Computer Poker Research Group has developed systems at the forefront of computer Poker players (Billings et al., 2006). This is a challenging domain combining imperfect and misleading information, opponent modeling, and a large state space. They describe several techniques that could generalize to this field, including *miximix*, fast opponent modeling, and Nash equilibrium solutions over an abstracted state space.

An obvious criticism of these sampling methods is that they value information at zero because each individual sample assumes perfect information at all times (Frank & Basin, 2001). This was confirmed at the imperfect-information track of the GGP competition at the Australasian Joint Conference on Artificial Intelligence, where three games, NumberGuessing, BankerAndThief, BattleshipsInFog, were specifically designed to test the ability of players to value information. None of the competitors were able to do so. This so-called *strategy fusion error* has been identified as a main issue with straightforward perfect-information model sampling (Frank & Basin, 1998). Long, Sturtevant, Buro, and Furtak (2010) analyze the impact of this error and present three conditions on game trees under which it does not have a strong adverse effect on the quality of play. For other games, the strategy fusion error has led to variations of perfect-information sampling that have been successfully applied to other card games (Wisser, 2015).

Another recent idea by Cowling, Powley, and Whitehouse (2012) is Information Set Monte Carlo Tree Search (IS-MCTS) currently being applied to many deterministic games of imperfect information. Instead of searching game trees constructed from game states and moves the IS-MCTS algorithms search trees constructed of information sets providing an alternative analysis of the game. In doing so the technique takes into consideration aspects of game-play that might be outside the domain of other reasoners. IS-MCTS relies on randomly guessing the missing information but considers each sample as an independent game to be analysed and to guide a single iteration of MCTS. Instead of keeping the search trees independent, IS-MCTS creates a conglomerate tree which includes portions of the game tree known to be unreachable yet delivers statistically significant results. A later variant SO-MCTS maintains role independent trees. There are strong parallels between this approach and the Imperfect Information Simulation in section 6. Importantly IS-MCTS and SO-MCTS still requires a deterministic sample of an information set. Taking deterministic samples of an information set is what HyperPlay does very efficiently.

1.2 Contribution

This article draws on the General Game Playing research for perfect-information games and extends it to imperfect-information games. We take the simple technique of using a sample of the player's information set[2] and confirm that only a weighted sample will provide the correct results. We then devise a formulation for calculating weightings for the particle filter from partial probabilities using Ockham's Razor.

The idea of sampling an information set is then extended to search spaces where such sampling would normally be intractable. We offer a technique (HyperPlay) that maintains a bag of samples from one information set to the next in a computationally efficient and

2. Sometimes this is referred to as a "particle filter" (Silver & Veness, 2010).

effective manner. This allows us to generate samples of an information set that would otherwise be impossible within the time budget given for each move. The technique is proven to be sound and complete for all imperfect-information games in General Game Playing. A formalism for an implementation of the technique is presented and tested to reveal its strengths, weaknesses, limitations, and its efficiency over randomly generated samples. Experimental data is presented to support the conclusions.

The incorrect valuation of information-gathering moves is clearly identified as a limitation to the basic technique, and so information set sampling is lifted to imperfect-information models to overcome the problems that occur when sample is elevated to fact[3]. This improvement ensures that the player correctly values all information-gathering moves. Again, a formalism for an implementation is presented and tested. Again, experimental data is used to identify the strengths, weaknesses and limitations of the new technique.

We begin with an overview of General Game Playing in section 2, followed by the introduction of the HyperPlay technique and an analysis of an information set sampling and the use of a weighted particle filter for general GDL-II games in section 3. The implementation of HyperPlay is set out in section 4 with some experimental results, concluding with an example of its limitations. Section 5 shows the efficiency of the technique over random sampling and demonstrates its ability to maintaining viable models in a truly massive search space.

In section 6, we lift model sampling to imperfect-information models and present a formalism for HyperPlay-II along with experimental result. The scalability of the new technique is challenged in section 7 to reveal the computational resource required for play large games. Finally, we draw some conclusions in section 9. Experimental results and the imperfect-information game topologies used for testing are presented in the electronic appendix to the paper.

2. General Game Playing

In this section, we introduce the basics of General Game Playing (GGP). We review the Game Description Language - Imperfect Information (GDL-II) and show its use in describing a game. We introduce the Monty Hall game as a running worked example to illustrate the technical sections. Finally, we define a formalization for an imperfect-information game that is used to underpin the proofs and experimental findings presented in later section.

2.1 Game Description Language

The standardization of the Game Description Language (GDL) (Love, Hinrichs, Schkufza, & Genesereth, 2006) and its widespread adoption has led to an increase in the research of General Game Playing. Each year we see competitions being held at AI conferences, beginning with AAAI GGP Competition (Genesereth et al., 2005) with successful players employing a variety of techniques including: automatically generated evaluation functions (Clune, 2007; Schiffel & Thielscher, 2007); or some form of Monte Carlo technique such as the modern UCT (Bjornsson & Finnsson, 2009; Mehat & Cazenave, 2011). However, these

3. Elevating sample to fact is an example of the *strategy fusion error*(Frank & Basin, 1998)

advancements have been in the field of perfect-information games, only a few advancements have been made in the field of imperfect-information games.

Imperfect-information games were set as a challenge for existing AI systems (Thielscher, 2010; Schiffel & Thielscher, 2014) with a specification for the extension of the Game Description Language (Quenault & Cazenave, 2007; Thielscher, 2011) to encompass "Imperfect Information" (GDL-II) being accepted by the General Game Playing community. These games present several new challenges for the AI player. Firstly, the search space is often larger than similar perfect-information games as the player must search parts of the game tree that would otherwise be known to be inaccessible. Secondly, the player must reason across an information set of all possible game states that satisfies the percepts received by the player and choose the move that is most likely to give a positive outcome.

The GDL is used to specify a set of declarative statements (rules) with a programming-like syntax. Originally designed for games of perfect information, the GDL was formalized by Love et al. (2006) and later enhanced to include games of imperfect information (Thielscher, 2010; Schiffel & Thielscher, 2014) with percepts and a role for the random player who often represents 'nature'. The syntax and semantics of GDL are detailed in Appendix A.

```
 1 (role random)
 2 (role contestant)
 3
 4 (init (closed 1))
 5 (init (closed 2))
 6 (init (closed 3))
 7 (init (round 1))
 8
 9 (<= (legal random (hide_car ?d))
10     (true (round 1))
11     (true (closed ?d)))
12 (<= (legal random (open_door ?d))
13     (true (round 2))
14     (true (closed ?d))
15     (not  (true (car ?d)))
16     (not  (true (chosen ?d))))
17 (<= (legal random noop)
18     (true (round 3)))
19 (<= (legal contestant (choose ?d))
20     (true (round 1))
21     (true (closed ?d)))
22
23 (<= (legal contestant noop)
24     (true (round 2)))
25 (<= (legal contestant noop)
26     (true (round 3)))
27 (<= (legal contestant switch)
28     (true (round 3)))
29
30 (<= (sees contestant ?d)
31     (does random (open_door ?d)))
32 (<= (sees contestant ?d)
33     (true (round 3))
34     (true (car ?d)))
35 (<= (next (car ?d))
36     (does random (hide_car ?d))
37 (<= (next (car ?d))
38     (true (car ?d)))
39 (<= (next (closed ?d))
40     (true (closed ?d))
41     (not  (does random open_(door ?d))))
42 (<= (next (chosen ?d))
43     (does contestant (choose ?d)))
44
45 (<= (next (chosen ?d))
46     (true (chosen ?d))
47     (not  (does contestant switch)))
48 (<= (next (chosen ?d))
49     (does contestant switch)
50     (true (closed ?d))
51     (not  (true (chosen ?d))))
52 (<= (next (round 2))
53     (true (round 1)))
54 (<= (next (round 3))
55     (true (round 2)))
56 (<= (next (round 4))
57     (true (round 3)))
58
59 (<= terminal
60     (true (round 4)))
61
62 (<= (goal contestant 100)
63     (true (chosen ?d))
64     (true (car ?d)))
65 (<= (goal contestant  0)
66     (true (chosen ?d))
67     (not  (true (car ?d))))
68 (goal random 0)
```

Figure 1: A GDL-II description of the Monty Hall game (Rosenhouse, 2009) adapted from Thielscher (2011).

Example 1. To illustrate we use a running example based on the GDL-II rules in Figure 1, which formalize the simple Monty Hall game. The intuition behind the rules follows.

Lines 1 – 2 introduce the players' names (the game host is modeled by `random`).

Lines 4 – 7 define the four features that comprise the initial game state.

The possible moves are specified by the rules for `legal`: in round 1, the `random` player must decide where to place the car (lines 9 – 11) and, simultaneously, the contestant chooses a door (lines 19 – 21); in round 2, `random` opens one of the other doors (lines 12 – 16) to reveal a goat; finally, the contestant can either stick to their earlier choice (`noop`) or switch to the other, yet unopened door (lines 25 – 26, 27 – 28).

The contestant's only percepts are: the door opened by the host (lines 30 – 31); and the location of the car at the end of the game (line 32 – 34).

The remaining rules specify the state update (rules for `next`), the conditions for the game to end (rule for `terminal`), and the payoff for the player depending on whether they got the door right in the end (rules for `goal`). ¶

2.2 Game Formalization

Here we present the basic mathematics of a game described in the GDL-II. While there have been many variations presented in the literature the differences are mostly superficial in regard to the nomenclature. We follow the most common conventions but standardize our nomenclature, with sets being upper-case Roman characters, elements being the corresponding lower-case character, and functions being lower-case Greek characters or short words giving some intuition of the functions purpose. We follow the formalisation presented by Schofield (2018) which builds upon the semantics of GDL-II presented by Schiffel and Thielscher (2014) where they show that a set of clauses forming a valid GDL-II description[4] is a representation of a state transition system as follows.

Definition 1. Let $G = \langle S, s_0, R, A, \lambda, P, \rho, \upsilon, \delta \rangle$ be an imperfect-information game given by a valid GDL-II description, then:

1. S is a set of states, disjoint decision and terminal states $S = D \uplus T$;
2. $s_0 \in S$ is the initial state of the game;
3. R is a set of roles in the game;
4. A is the set of all moves (actions) in the game;
5. $\lambda : D \times R \to 2^A$ is a function giving a set of legal moves for $r \in R$ in $d \in D$;
6. P is a set of all percepts in the game;
7. $\rho : D \times A^{|R|} \times R \to P$ is a function giving the percept (with multiple percepts conjoined and *null* as the empty percept) for role $r \in R$ resulting from enacting a joint move in a decision state $d \in D$;
8. $\upsilon : T \times R \to \mathbb{R}$ is the payoff function on termination; and
9. $\delta : D \times A^{|R|} \to S$ is the successor function. □

   Supplementary nomenclature used hereafter are detailed in Appendix B.

Example 1 continued. In the worked example shown in the GDL of Figure 1. Let s_1 result from the random player placing the car behind door 1 and the contestant choosing door 2 in the initial state s_0. In round 2, *random* will open one of the closed, unchosen doors that does not contain the car, defined at line 12. The contestant has a forced *noop*, defined by at line 23. The percepts for the contestant are defined at line 30, and *random* has no percepts. From Definition 1 we get:

$$\begin{aligned}
\vec{a} &= \langle (\mathit{open\_door}\ 3), \mathit{noop} \rangle \\
s_2 &= \delta(s_1, \vec{a}) = \delta(s_1, \langle (\mathit{open\_door}\ 3), \mathit{noop} \rangle) \\
\vec{p} &= \rho(s_1, \langle (\mathit{open\_door}\ 3), \mathit{noop} \rangle) = \langle \mathit{null}, 3 \rangle
\end{aligned}$$

¶

4. Refer to section 2.3 Valid Game Descriptions of Schiffel and Thielscher (2014).

3. The HyperPlay Technique

In this section, we introduce the HyperPlay technique, both as a function that completes imperfect play messages and as an implementation in a GGP player. We show that a uniform sample of a player's information set fails in some games and introduce a sampling technique based on Ockham's Razor. There is a proof sketch for soundness and completeness of the technique by examining properties of the game tree. We also present sufficient detail for the reader to implement the technique as a "bolt-on" to an existing perfect-information General Game Playing agent.

3.1 Overview

The technique maintains a bag of models of the true game, each model being a perfect-information game. In this respect the technique is a form of determinism, that is, it creates perfect-information samples of the players information set. The differentiating feature of the technique is that it is able to take samples of very large information sets. A case study in section 5.7 shows many samples being taken in real time where the upper bound for the information set is 6.0 E+23 and the probability of a valid sample is $< 0.01\%$ with an efficiency 50 times greater than random sampling.

The technique is not a move planning tool as it has no forward-looking capabilities and so it must be added onto an existing player. We choose to use a simple Monte Carlo player because we want our experimental results to focus on the sampling of the information set, not the cleverness of the move selection tool in finding the optimal solutions. We want the perfect-information player to cover a large portion of the game tree, by making mediocre move choices that will eventually converge to an optimal solution, so that we can validate the sampling technique in every circumstance.

3.2 Game Tree

We have defined a game as a multi-agent state transition system, described by a set of clauses in the GDL. We use the terminology of the GDL to describe agents as roles, actions as moves, and signals as percepts. The natural progression of the game from one state to its successor, beginning with the initial state, traces out paths in the *induced game tree*.

Definition 2. A game $G = \langle S, s_0, R, A, \lambda, P, \rho, \upsilon, \delta \rangle$ given by a valid GDL-II description induces a *game tree*, which is a connected, acyclic graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ with a single root node for the initial game state and where the edges are determined by the joint legal moves in non-leaf nodes while leaves correspond to terminal game states. We will use the following definitions:

1. $\mathit{state} : \mathbb{V} \rightarrow S$ is a function that maps from a node (vertex) $\mathrm{v} \in \mathbb{V}$ in the game tree to the corresponding game state $s \in S$; and
2. $\mathit{moves} : \mathbb{E} \rightarrow A^{|R|}$ is a function that maps from an edge $\mathrm{e} \in \mathbb{E}$ in the game tree to the corresponding joint move $\vec{a}$;
3. $\vec{\mathrm{e}}^n$ is a unique path of edges $\langle \mathrm{e}_1, \mathrm{e}_2, ...\mathrm{e}_n \rangle$ beginning at the single root node v_0, corresponding to the joint moves enacted in a game; and
4. $\mathit{node}^i : \mathbb{E}^{\mathbb{N}} \times \mathbb{N} \rightarrow \mathbb{V}$ is a function that returns the i^{th} node along a path e^n. $\square$

As the game progresses, there is a state s_t of the game described by a set of fluents as well as an ordered list of play messages forming a history of joint moves and percepts. This history can be traced out as a path from the root node v_0 to the current node v_t, such that, $s_t = \mathit{state}(\mathrm{v}_t)$ and forms the basis of the model maintained by the technique.

3.3 Information Set

Schiffel and Thielscher (2014) show that a valid GDL-II game can be understood as a partially-observable, extensive-form game and that there will be sets of legal play sequences (histories) that a player cannot distinguish. Therefore, we define information sets for a player in terms of indistinguishable histories.

Definition 3. Let $G = \langle S, s_0, R, A, \lambda, P, \rho, \upsilon, \delta \rangle$ be an imperfect-information game given by a valid GDL-II description and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the induced game tree, and a play message be a move and percept tuple associated with an edge on the induced game tree and a history be a sequence of such edges. Then:

1. M is the set of all play messages, $m = \langle a, p \rangle \in M$, for each role $m_r = \langle a_r, p_r \rangle$ where a_r is the move enacted in decision state d and $p_r = \rho(d, \vec{a}, r)$;
2. $\mathcal{H}$ is the set of all histories in the game with $h \in \mathcal{H}$ being a history of $\vec{m}^k$, which departs from the convention of a history of actions a^k or edges e^k so as to deal with imperfect histories, where the percepts provide additional information that is not in the state but aids in partitioning nodes into information sets;
3. $\xi : \mathcal{H} \times R \rightarrow \mathcal{H}$ gives the imperfect-information history as seen by role $r \in R$, and
4. $I_{r,n} \subseteq \mathcal{H}$ is the general form for an information set for role $r \in R$ in round n. □
   Supplementary nomenclature used hereafter are detailed in Appendix B.

Now we can extract information from any node in the game tree about the moves and percepts along its history including the imperfect-information history that represents the view of one of the roles and defines that role's information set.

3.4 Move Selection Policy

For a game to be played out to termination we require a move selection policy for each role. This policy represents the role's strategy for choosing a legal move in any decision state.

Definition 4. Let $G = \langle S, s_0, R, A, \lambda, P, \rho, \upsilon, \delta \rangle$ be an imperfect-information game given by a valid GDL-II description and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the induced game tree and each role have a move selection policy expressed as a probability distribution across all moves in all decision states. We use the following definitions:

1. Π is the set of all move selection policies $\pi \in \Pi$,
2. $\mathit{select} : \Pi \times D \times R \rightarrow \phi(A)$ is a move selection function, as a probability distribution across all moves, and
3. $\mathit{play} : S \times \Pi^{|R|} \rightarrow \phi(T)$ is the playout of a game from any state to termination according to the given move selection policies, expressed as a probability distribution across all terminal states. □
   Supplementary nomenclature used hereafter are detailed in Appendix B.

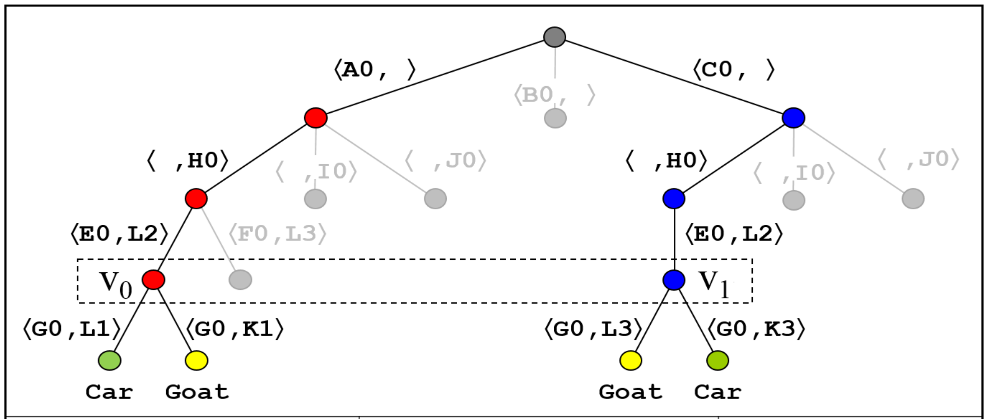
We define the *play*() function from any state including a terminal state. In such a case the distribution $\phi(T)$ would have one certain value. From this definition, we formulate the probability distribution across the set of terminal states (1) given an information set and move selection policies. We also calculate the expected outcome of the game (2).

$$\phi(T) = \sum_{h_i \in I_{r,n}} P(h_i) \cdot play(state(\mathrm{v}_{h_i}), \vec{\pi}) \qquad \textit{prima facie}\ P(h_i) = 1/|I_r| \tag{1}$$

$$E(I_r) = \sum_{t_i \in T} P(t_i) \times \upsilon(t_i, r) \tag{2}$$

Note that the playout results in a probability distribution across all the terminal nodes, those terminals that are unreachable from an information set would have zero probability.

3.5 Choice Factor



A	(does random (hide_car 1))	H	(does contestant (choose 1))	0	null
B	(does random (hide_car 2))	I	(does contestant (choose 2))	1	(sees contestant 1)
C	(does random (hide_car 3))	J	(does contestant (choose 3))	2	(sees contestant 2)
D	(does random (open_door 1))	K	(does contestant switch)	3	(sees contestant 3)
E	(does random (open_door 2))	L	(does contestant noop)		
F	(does random (open_door 3))				
G	(does random noop)				

Figure 2: Monty Hall game tree, as seen by the contestant. Moves are serialized and (does role noop) removed, for the sake of clarity. The moves in the game were (does random (hide_car 3)), (does contestant (choose 1)), (does random (open_door 2)).

Example 1 continued. Consider the situation shown in Figure 2:

$I_c = \{\mathrm{v}_0, \mathrm{v}_1\}$ is the contestant's information set
$\phi(T) = \sum_{\mathrm{v}_i \in I_c} P(\mathrm{v}_i) \times play(state(\mathrm{v}_i), \vec{mc})$ is the result of random playouts **assuming** a uniform probability distribution across I_c
$\phi(T) = \langle ..., 0.25, 0.25, ..., 0.25, 0.25, ... \rangle$

The reader will note that $\phi(T)$ suggests an even distribution of the outcomes which is **incorrect** for the real Monty Hall game, therefore, the assumed probability distribution across I_c must also be incorrect. ¶

The probability distribution across an information set[5], calculated in (1) and our example will depend on the move selection policy π of the other roles. Prima facie, the move selection policy for other roles is random, but that does not mean the probability distribution across an information set is uniform. We use Ockham's Razor[6] and rules in the GDL to formulate the probability distribution across an information set.

The original definition of *ChoiceFactor* (Schofield et al., 2012) focused on the choices being made by a player as the game was being played. It let the *ChoiceFactor* of a node v_i (3) be the product of the size of the sets of legal moves (choices) in each decision node along the path from the root node to the current round as defined by the history $\xi(\mathrm{v}_i)$.

Finally, the likelihood of the node v_i (4) being the true node v_t was expressed in proportion to the likelihood of all of the other samples of an information set.

$$ChoiceFactor(\mathrm{v}_i) = \prod_{j=0}^{n-1} |\lambda(state(node^i(\xi(\mathrm{v}_i), j)))| \tag{3}$$

$$P(\mathrm{v}_i = \mathrm{v}_t) = \frac{1/ChoiceFactor(\mathrm{v}_i)}{\sum_{\mathrm{v}_j \in I_r} 1/ChoiceFactor(\mathrm{v}_j)} \tag{4}$$

This approach has appeal as an information set may be intractable and we may only be able to take a sample. Therefore, our only indication of the likelihood of a sampled node being the true node must be expressed in terms of the likelihood of the other samples. However, we may have information about the move selection policy of other roles. Therefore, we redefine (4) by including the move selection policy π for the choices being made.

$$P(\vec{a} \mid d_j) = \prod_{r \in R} P(a_r \mid select(\pi_r, d_j, r)) \quad where \;\; d_j = state(node^i(\xi(\mathrm{v}_i), j)) \tag{5}$$

$$P(\mathrm{v}_i = \mathrm{v}_t) = \prod_{j=0}^{n-1} P(\vec{a} \mid d_j) \tag{6}$$

The probability expressed in (6) is only correct if we can sample the whole information set and the game tree has a constant branching factor. Therefore, as in the previous case (4), we must treat this a partial probability and use a normalizing factor to get an estimate of the true probabilities.

$$P(\mathrm{v}_i = \mathrm{v}_t) = \frac{1}{k} \prod_{j=0}^{n-1} P(\vec{a} \mid d_j) \tag{7}$$

$$normalisation\ factor \qquad k = \sum_{\mathrm{v}_i \in I_r} \Big(\prod_{j=0}^{n-1} P(\vec{a} \mid d_j) \Big) \tag{8}$$

Now we have an expression for the probability that an element of the sample of an information set is in fact the true game, relative to the other samples in our bag[7].

5. That is, the probability that an element in an information set is in fact the true node representing the current game: $P(\mathrm{v}_t = \mathrm{v}_i)$
6. We look for the simplest expression of likelihood that a path would be selected at random.
7. We use a bag of samples instead of a set of samples, as there may be duplication.

Example 1 continued. Reworking our example of the situation shown in Figure 2:
$I_c = \{\mathrm{v}_0, \mathrm{v}_1\}$ is the contestant's information set
$\phi(T) = \sum_{\mathrm{v}_i \in I_c} P(\mathrm{v}_i) \times \mathit{play}(\mathit{state}(\mathrm{v}_i), \vec{mc})$ is the result of random playouts
$\phi(I_c) = \langle 0.33, 0.67 \rangle$ (from equations (5)(7)(8))
$\phi(T) = \langle ..., 0.17, 0.17, ..., 0.33, 0.33, ... \rangle$

Which is now consistent with the probable outcomes of the real game. The expected outcome for the two move choices are:
$E(\mathit{noop}) = 0.33 \times \mathit{car} + 0.67 \times \mathit{goat}$
$E(\mathit{switch}) = 0.67 \times \mathit{car} + 0.33 \times \mathit{goat}$

Suggesting that the Contestant should (switch), unless the goat has more utility than the car. The distribution $\phi(T)$ is consistent with the known outcomes of the real game. ¶

3.6 HyperPlay Description

This technique (Schofield et al., 2012) maintains a bag of models of the real game, each model being a perfect-information game. The technique updates the models from one move to the next, replacing any unreachable models with new ones by backtracking along their history until a reachable model can be instantiated. The models are used as a weighted particle filter on an information set and provide a basis for perfect-information evaluation. We define a function that maintains each model by completing an imperfect history of play messages using the legal move choices available in each decision state. If the full history of the game is known, then all of the models converge to the game. The function must choose randomly from the set of legal moves and so we use a random seed x as an argument to the function. The same random seed will always return the same choice.

Definition 5. Let $G = (S, s_0, R, A, \lambda, P, \rho, \upsilon, \delta)$ be an imperfect-information game given by a valid GDL-II description and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be the induced game tree. The HyperPlay function is defined as follows.

1. $\mathit{hp} : M^n \times \mathbb{R} \to M^n$ is the function HyperPlay that completes an imperfect-information play history h_r by grounding the missing elements in the play messages consistent with the legal moves $a_{ri} \in \lambda(\mathit{state}(\mathit{node}^i(h_r, i), r)) \ \forall\ i : 0 \leq i < n$ known to the role r. The function takes a random seed x to randomize the grounding choices, the same seed will produce the same grounding choices.
2. Bad Move: $\vec{a}$ is labeled as "bad" in decision state d_k if the percepts do not match for the round $k+1$, ie. $\rho(d_k, \vec{a}, r) \neq h_t[k+1, r, i_p]$. □

We can now formulate an expression to sample an element in an information set (9) by completing an imperfect history and then obtaining the last node in the play sequence.

$$\mathrm{v}_i = \mathit{node}^i(\mathit{hp}(\xi(\mathrm{v}_t, r), x), n) \tag{9}$$

Example 1 continued. The HyperPlay function takes the imperfect history for the Contestant and constructs a perfect history to one of the nodes in an information set.

$$\xi(\mathrm{v}_t, \mathit{Contestant}) = \langle\langle\langle (\ ?\ ), (\ ?\ )\rangle, \langle(\mathit{noop}), ()\rangle\rangle, \langle(\ ?\ ), (\ ?\ )\rangle, \langle\langle(\mathit{choose}\ 1), ()\rangle\rangle\rangle$$
$$\mathit{hp}(\xi(\mathrm{v}_t, \mathit{Contestant}), x) = \langle\langle\langle(\mathit{hide\_car}\ 1), ()\rangle, \langle(\mathit{noop}), ()\rangle\rangle, \langle(\mathit{noop}), ()\rangle, \langle\langle(\mathit{choose}\ 1), ()\rangle\rangle\rangle$$

¶

3.7 Implementing HyperPlay

The technique maintains a bag of models of the true game, each model being a perfect-information game. The models are given the name HyperGames[8] as a reminder that they are more than just a grounding of the state variables, but a full instantiation of the game currently being played with agents in every role.

The HyperGame also contains the ability to re-instantiate the model if new percepts makes the model invalid, referred to as "backtracking". So, a HyperGame is more than a model. It is a data structure that contains a model that represents one possible state of the true game. When new percepts arrive the current model of the HyperGame becomes invalid and a new model is selected that is consistent with the newly received percepts and existing information. This also allows the calculation of the likelihood that the new model is the true game.

The term HyperGame will be used to mean a self-correcting data structure that contains a model of the current game instantiated in code for experimental purposes. So, the HyperGame is one of many self-correcting instantiations of the game containing a model that is utilized by a HyperPlayer to make move selection in the game. The "bag of models" refers to the collection of models contained within the HyperPlayer. A bag is used as some models may be identical.

The logic for the technique is as follows.

1. Models are updated from one move to the next using legal moves to replace the missing moves for all of the roles.
2. Replacement moves are made randomly using a random seed. The same random seed will produce the same move choices.
3. Bad[9] move choices are recorded in memory by each HyperGame and not repeated. Subtrees with no good move choices are discarded.
4. Invalid models that do not have any move choices consistent with the play messages received are replaced with new models by backtracking along the play history until a valid model can be instantiated in the current round.
5. Choice factors are calculated by each HyperGame along the models play history, and hence a weighting factor is determined for calculation of expected outcomes.
6. The models are used as a weighted particle filter on an information set and provide a basis for perfect-information evaluation.

Additionally, it is possible to place a counter into the update process so that any HyperGame taking too long to update its model can be taken off line. This prevents the player from stalling when one HyperGame becomes hopelessly lost. This merely postpones the update process to a later round; it does not permanently invalidate the model. Time permitting, the model can be brought back on line especially if the game is a turn taking game.

A function is defined, below, that maintains each model by completing a history of imperfect-information play messages using the legal move choices available in each decision

8. In the original analysis the term HypoGame was used to mean a hypothetical game, but the term was not popular and it morphed into HyperGame, and hence HyperPlay.
9. The term "bad" was coined in the original paper to mean legal, at the time, but inconsistent with later information.

state.[10] The function must choose randomly from the set of legal moves and so a random seed x is used.

$$h_i = hp(\xi(h_t, r), x) \tag{10}$$

Put in words, the player receives an imperfect-information play history $\xi(h_t, r)$ from the Game Controller and uses a random seed to construct a deterministic sample in the form of a perfect-information play history.

Example 1 continued. The HyperPlay function takes the imperfect-information play history for the Contestant and constructs a complete history to one of the nodes in the current information set.

$$\xi(h_t, Contestant) = \langle\langle\langle\ \ \rangle, \langle L0\rangle\rangle, \langle\langle\ \ \rangle, \langle H0\rangle\rangle, \langle\langle\ \ \rangle, \langle L2\rangle\rangle\rangle$$
$$hp(\xi(h_t, Contestant), x) = \langle\langle\langle A0\rangle, \langle L0\rangle\rangle, \langle\langle G0\rangle, \langle H0\rangle\rangle, \langle\langle E2\rangle, \langle L2\rangle\rangle\rangle$$

A different random seed x would produce a different sample.

The union of all such samples, in the limit, is an expression of an information set:

$$I_{r,n} = \bigcup_{lim} hp(\xi(h_t, r), x) \tag{11}$$

as the HyperPlay function can return any and every element of an information set.

3.8 Pseudo Code

Below is a presentation of the original process with some alterations to the nomenclature. Previously the procedure was described using mathematical notation, but here it is described using object-oriented pseudo code.

The procedures `forward()`, and `backward()` have been combined into `Update()`. Previously, the procedure `forward()` would replace missing information moving forward to the current round, while the procedure `backward()` would backtrack invalid paths until a new, untested branch could be found.

The HyperPlay algorithm is summarized in Figure 4 as part of an imperfect-information game player in Figure 3. In the code for the player:

- classes are declared for a Game, Step and HyperGame for the operation of the player,
- line 15 shows the initialization of the bag of models (HyperGames), each being equal to the initial node of the game,
- line 18 uses the bag of models as a weighted particle filter to calculate the move with the highest utility,
- line 19 submits the move to the game controller and receives a percept, and
- line 22 updates each model to agree with the most recent move and percept.

Each HyperGame randomly completes the imperfect-information history to provide a statistically valid sample of an information set.

10. If the full history of the game is known then all of the models converge to the true game.

```
class Game
  Node
  Round
class Step
  MyMove
  MyPercept
  Legal<Move>
  Bad<Move>
class HyperGame
  Node
  Round
  Path<Step>
  RandomSeed
procedure Player(GameController)
  Bag = <HyperGame, ... , HyperGame>
  Round = 0
  repeat
    MyMove = SelectMove(Bag)
    MyPercept = GameController.SubmitMove(MyMove)
    Round = Round +1
    for each HyperGame in Bag
      HyperGame.Update(MyMove, MyPercept, Round)
    next HyperGame
  until IsTerminal(GameController)
end
```

Figure 3: An imperfect-information player using the HyperPlay technique.

```
procedure HyperGame.Update(MyMove, MyPercept, Round)
  NewStep = Step.New(MyMove, MyPercept)
  HyperGame.Path.Add(NewStep)
  // Advance the HyperGame to the current round
  while HyperGame.Round < Round
    CurrentStep = HyperGame.Step[HyperGame.Round]
    // Find a move that is consistent with the play messages
    for each Move in CurrentStep.Legal and not in CurrentStep.Bad
      if IsCongruent(Move) then
        HyperGame.DoMove(Move)
        NextStep = HyperGame.Step[HyperGame.Round]
        NextStep.ResetLegalAndBad(RandomSeed)
        continue while
      end if
      CurrentStep.Bad.Add(Move)
    next Move
    // Backtrack the previous move as all of its children are bad
    BadMove = HyperGame.UndoLastMove()
    PreviousStep = HyperGame.Step[HyperGame.Round]
    PreviousStep.Bad.Add(BadMove)
  end while
end
```

Figure 4: The HyperPlay technique used to maintain a model of the game.

Looking at the `Update()` code in Figure 4 in more detail:

- lines 2 and 3 add a new step to the imperfect-information history,
- at first call, the HyperGame will be one round behind the game,
- lines 12 clears the array of bad moves and randomizes the array of legal moves,
- line 8 enumerates the legal, as yet untested, moves for the current search round,
- lines 9 - 14 advance the HyperGame with a move that is consistent with the known moves and percepts,
- line 15 records ‘bad’ moves that are inconsistent with the known moves and percepts, and
- if there are no ‘good’ moves then line 18 - 20 backtracks the HyperGame to the previous round, records the ‘bad’ move and continues the search, preserving all information so backtracking can pick up where the previous search left off.

3.9 Soundness and Completeness

The HyperPlay technique ”maintains” a collection of HyperGames which update their models from one round to the next. This proof sketch treats the `Update()` procedure shown in Figure 4 as a logical system and shows that it is both sound and complete, that is:

- Everything that the `Update()` procedure says is a valid model, is in fact valid, and
- Every valid model can be obtained by using the `Update()` procedure.

In this context a valid model corresponds to a play history from an information set for the current round.

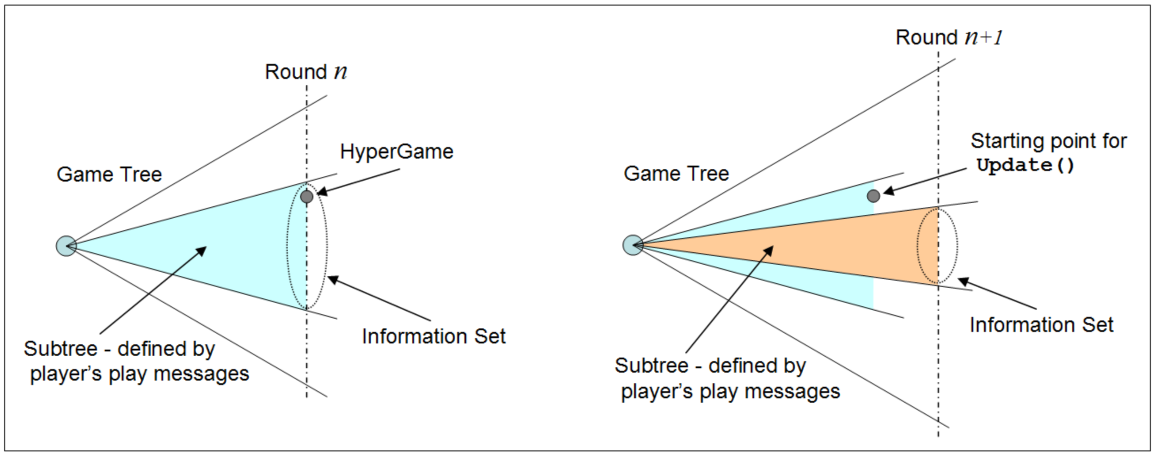

Round n
HyperGame
Game Tree
Information Set
Subtree - defined by player’s play messages
Round $n+1$
Starting point for Update()
Game Tree
Information Set
Subtree - defined by player’s play messages

Figure 5: The Game Tree for a GDL-II game at rounds n and $n+1$, showing the subtree defined by an information set.

As there is a bijective relationship between the play histories and the simple paths in the induced game tree and it is easier to visualize a tree than a set of histories then the following discussion will make use of the game tree in Figure 5 to illustrate an analysis of the maintenance of a sample of an information set.

The HyperPlay software was originally developed using the context of the game tree for inspiration and so memory structures and functions were developed using nodes and paths. In this context the soundness and completeness of the `Update()` procedure is modified by considering the return value of the procedure as being the game tree node corresponding to the model in the HyperGame and so the term 'HyperGame.Node' is used to mean the output of the logical system. So, the test for soundness and completeness is modified to be:

- Every HyperGame.Node returned by the `Update()` procedure corresponds to a play history in an information set of the current round; and
- Every node corresponding to a play history in an information set of the current round can be returned by the `Update()` procedure.

Consider the game tree for an imperfect-information game and identify the subtree $\mathbb{G}_{I_{r,n}}$ defined by the player's information set, which is induced by the players imperfect-information play histories received from the Game Controller. In Figure 5 there is a stylized representation of a game in progress. In round n the player's play messages define a subtree $\mathbb{G}_{I_{r,n}}$ of all of the paths representing possible histories identified within an information set, and that no paths pass outside the subtree. The converse being that any path outside the subtree leads to a node that is not in an information set, remembering that the tree is acyclic in its undirected form.

In round $n+1$ the new play messages define a similar subtree. From Definition 3 it can be seen that the play messages for round $n+1$ are built upon the play messages for round n and hence the new subtree may not include any nodes outside the original subtree, other than those nodes in round $n+1$.

Definition 6. Let $G = (S, R, A, P, \upsilon, \delta)$ be an imperfect-information game that satisfies the restrictions of the GDL-II and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a game tree induced by the game. The following definitions apply.

1. $\cong$: $\xi(h_1, r) \cong \xi(h_2) \iff \xi(h_1, r) = \xi(h_2, r)$ defines the congruence of a history of imperfect-information play messages with a history of complete play messages.
2. $\mathbb{G}_{I_{r,n}}$ is the subtree given by all of the paths $\vec{\mathrm{e}}^{\,n}$ from the initial node v_0 to a node $\mathrm{v}_{h_i} : h_i \in I_{r,n}$ induced by an information set. □

Theorem 1. *All nodes corresponding to an information set $I_{r,n+1}$ succeed nodes corresponding to an information set $I_{r,n}$.*

Sketch. by construction, showing that the simple paths in the subtree $\mathbb{G}_{I_{r,n+1}}$ can only be built upon simple paths in the subtree $\mathbb{G}_{I_{r,n}}$.

Let $G = \langle S, R, A, P, \upsilon, \delta \rangle$ be an imperfect-information game that satisfies the restrictions of the GDL-II and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a game tree induced by the game, then:

Base case $n = 0$:
$h_0 = \varnothing, \mathrm{v}_0 = path(h_0)$
the empty history inducing the root node of the game tree
General case for round $n+1$:
$\forall h_i \in I_{r,n+1} \;\; path(hp(\xi(h_i, r), x)) \in \mathbb{G}_{I_{r,n+1}}$ *from definitions 3, 5 and 6*
an information set subtree is a set of paths corresponding to a set of histories
$\xi(h_{n+1}, r) \cong \xi(h_{t,n+1})$ *from equation 10 and definition 6*

a history in an information set subtree is congruent with the true game history

$\xi(h_{n+1}) = \langle m_0, ...m_n, m_{n+1} \rangle = \langle \xi(h_n), m_{n+1} \rangle$ *from definition 3*

each history can be split into an existing history and an extension

$\xi(h_n, r) \cong \xi(h_{t,n})$ *from equation 10 and definition 6*

as before this existing history is congruent with the true game history

$\forall h_i \in I_{r,n}$ $path(hp(\xi(h_i, r), x)) \in \mathbb{G}_{I_{r,n}}$ *from definitions 3, 5 and 6*

each existing history corresponds to the previous information set subtree

□

Corollary 2. *All paths in an information set subtree for a role must pass through all previous information sets for that role.*

Corollary 3. *Not all paths in the previous information set subtree for a role are in the information set subtree for the current round.*

These corollaries underpin the backtracking process in the HyperPlay algorithm as they facilitate the finding of a "good" path without the need to start from the initial state of the game, and the pruning of "bad" paths at the earliest opportunity. Otherwise, the bad paths could not be pruned until the subtree they initiate was completely checked.

The tools are in place to show the soundness of the HyperPlay `Update()` procedure. From Corollary 2 any move can be safely tested at any round from the initial state of the game to the current round. If the imperfect play messages are not congruent with the game[11], then this move will never lead to a history in the current information set. In the context of the `Update()` procedure these moves are labeled as "bad".

Sketch. Let $G = (S, R, A, P, \upsilon, \delta)$ be an imperfect-information game that satisfies the restrictions of the GDL-II and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a game tree induced by the game, then:

`Update()` labels all moves outside the subtree $\mathbb{G}_{I_{r,n}}$ as bad
line 15 labels incongruent moves as `Bad`
line 20 labels moves as `Bad` *if all subsequent moves are* `Bad`
`Update()` backtracks all `Bad` moves
line 18 backtracks `Bad` *moves*
`Update()` only returns nodes in the current round
line 22 only exits when `HyperGame.Round = CurrentRound`

□

The HyperPlay `Update()` procedure randomly selects from all of the states in an information set of the current round when returning a value. The random choice does not have a uniform probability distribution. The selected node will be the one that required the least backtracking as all shallow options are exhausted before backtracking more deeply. This also speak to the efficiency of the process.

Sketch. Let $G = (S, R, A, P, \upsilon, \delta)$ be an imperfect-information game that satisfies the restrictions of the GDL-II and $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a game tree induced by the game, then:

11. Up to and including the round in question.

`Update()` can return any of $v_{h_i} : h_i \in I_{r,n}$

line 22 only exits with a valid sample in the current round

every path in the game tree originates the initial node

every path is accessible from every other path

`Update()` randomizes the evaluation of moves

line 12 resets the forward search using a `RandomSeed`

□

4. Implementing an Imperfect-Information Player

In this section, we present a simple imperfect-information player incorporating two basic elements, the HyperPlay algorithm and Monte Carlo sampling as the perfect-information reasoner. A formalism for the move selection policy is presented which allows for the aggregation of move utilities across a player's information set. We design and conduct some experiments to validate the player and identify its strengths, weaknesses and limitations.

4.1 Formalism for Our Player

We take a simple Monte Carlo player that runs a number of random simulations for each move in the set of move choices then averages the terminal values as a measure of utility. And thus, we define an evaluation function based on a number of "playouts" of the game after making a specific move.

Definition 7. Let $G = \langle S, R, A, P, \upsilon, \delta \rangle$ be an imperfect-information game given by a valid GDL-II description, and let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a game tree induced by the game, then:

1. B_r is a bag of models of G, each being an element of an information set I_r, and
2. $eval : S \times \Pi^{|R|} \times R \times \mathbb{N} \rightarrow \mathbb{R}$ is the evaluation function defined as $eval(s, \vec{\pi}, r, n) = \frac{1}{n} \sum_1^n \upsilon(play(s, \vec{\pi}), r)$.[12] □

From this definition, we can formulate an expression for the evaluation of a specific move in a specific decision state (12) for the Monte Carlo Player. Equation (13) sums the utility of the move class across all of the samples in the bag, testing to make sure the move class is legal each of the states[13] and applying the probability that the node is the true game node. Finally, the move selection policy (14) for our simple player, across the bag of models B_r of the game and using the weighted particle filter on an information set.

$$utility(d, a_{rj}) = eval(\delta(d, \langle \vec{a}_{-r}, a_j \rangle), \vec{mc}), r, n) \tag{12}$$

$$classUtility(a_{rj}) = \sum_{v_i \in B_r} \begin{cases} P(v_i) \times utility(state(v_i), a_{rj}), & a_{rj} \in \lambda(state(v_i)) \\ 0 \end{cases} \tag{13}$$

$$a_r = argmax_{a_{rj}} \big[\forall\ v_i \in B_r \big[\forall\ a_{rj} \in \lambda(state(v_i)) \big[classUtility(a_{rj})\big]\big]\big] \tag{14}$$

12. The variable n is not the round number, but some number of playouts used for an average $eval()$.
13. There is no guarantee that each move class is legal in each sample of an information set.

4.2 Testing the Player

Three games are chosen to test the move selection policy outlined in Equation (14). The Monty Hall game is used to validate the use of a weighted particle filter, Krieg-TicTacToe to represent two player simultaneous move games, and Blind Breakthrough as an example of a turn taking game with a very large search space. Each game is played with a variety of configurations and the results reported in Appendix D.

Strengths The maintenance of the imperfect-information path, the lists of legal moves and bad moves clearly works. It also facilitates the calculation of the probability $P(\mathrm{v}_i = \mathrm{v}_t)$ of the sample being the true game. This was demonstrated in the experimental results.

The player operated under a time budget with the ability for the `Update()` process for each model (HyperGame) to be taken off line so that slow updates would not slow down the player's move selection. So that the large search space in Blind Breakthrough did not cause the player to stall due to excessive backtracking. The HyperGame in question was simply taken off-line until it had finished its calculations and then returned on line. The efficiency of the technique in very large search spaces is the topic of the next section.

Weaknesses The primary weakness is that the search space may be extremely large and the enumeration of the possible imperfect-information histories given by $\xi(\mathrm{v}_t, MyRole)$ may take so long as to make the `Update()` process appear to be never ending.

In practical terms, there will always be a few HyperGames in the bag that have randomly chosen a path that is close to the true path. However, the size of the search space is a genuine concern for any implementation of this technique.

Limitations of the Player The problem that comes from elevating sample to fact (Frank & Basin, 1998) can be clearly demonstrated with this player, resulting in all information-gathering moves valued at zero utility; as all the information has already been gathered.

The HyperPlay-based player is unable to correctly value information-gathering moves. This is the HyperPlay's Achilles' heel and it motivates the technique in section 6.

5. The Efficiency of HyperPlay Over Random Sampling

This section tests the efficiency and effectiveness of the HyperPlay technique as a method of taking a deterministic sample of an information set (Schofield & Thielscher, 2017). The test is a comparison with a random sample taken by tracing out a play history from the initial state to the current round. Games were chosen as a representative sample of General Game Playing with Imperfect Information. Experiments focus on: efficiency over a random sampling approach, games where random sampling is impossible, samples being uniformly distributed across an information set, and rectifying a biased sample.

5.1 Random Sampling

The random sampling process starts at the root node and makes substitutions in the imperfect-information play history using the legal move choices in the state corresponding to the node. As the joint move vector is selected, the successor function is applied, and percepts received. If there are no legal move choices, then the sample is invalid and the process is started again. The probability equations use h for history, $1 \leq i \leq n$ for round,

$\vec{a}$ for joint move vector, s for state, G for game, $\rho()$ for percepts arising from actions, and $\delta()$ as the successor function.

At each round a valid sample is determined by the equality of the play histories:

$$Valid(\vec{a}_i) = [\rho(\delta(s_i, \vec{a}_i), r) = \rho(G_{i+1}, r)] \tag{15}$$

if the percepts produced, for role r, from the enacting of the randomly chosen joint move vector are equal to the actual percepts received from the Game Controller in round R then this is a valid grounding in that round.

The probability of randomly selecting a valid joint move vector is expressed as the ratio of valid/total joint move vectors.

$$P(Valid(\vec{a}_i)) = |\{Valid(\vec{a}_i)\}|/|\{\vec{a}_i\}| \tag{16}$$

The overall probability of randomly constructing a valid play history is:

$$P(Valid(h)) \leq \prod_{i=1}^{n} P(Valid(\vec{a}_i)) \tag{17}$$

less than or equal to the product of all of the probabilities along the path. The inequality comes from the possibility of making a valid choice in an early round that produces a dead end for all subsequent joint moves.

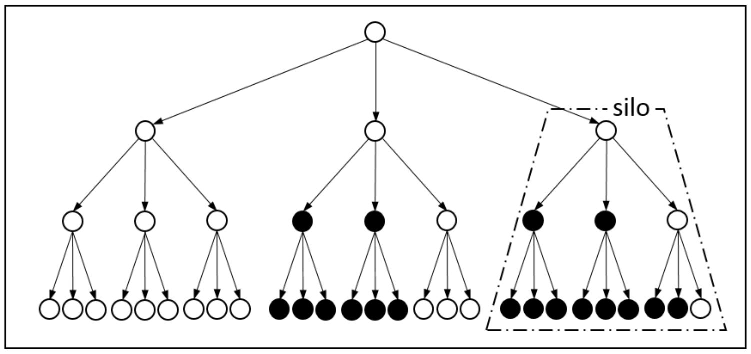


Figure 6: An example of a silo defined by the first move in a game. The black nodes are marked "bad" by the HyperPlay technique.

Experimentally it is prohibitive to measure $P(Valid(\vec{a}_i))$ for every node on the tree, but it is possible to find an estimate of this by measuring the average probability of such a sample being made in each round of the game. When such an average probability is taken across many experimental runs a reasonable value for $P(Valid(h))$ can be obtained.

5.2 Biased Samples

The HyperPlay technique advances each model by randomly substituting a legal move for missing information. If the move is invalid, then it searches the local sub tree for a valid combination. In extreme cases the subtree is expanded beyond the local region until a new model is found. This gives rise to a shortcoming of this technique. That is, the game tree can be divided into a small number of subtrees based on the first legal move substitution. These subtrees are called "silos", as shown in Figure 6. Initially there will be an equal number of models in each silo. As the game progresses one silo may have only one viable play history resulting in an over sampling as all of the models converge to that play history.

Experimentally it is possible to compare sample histories and look for duplicate histories, thereby identifying biased samples.

5.3 Uniformly Distributed Samples

A weighted particle filter is used[14], and so, some samples are more likely than others. However, *a priori* it must be assumed a uniform distribution across an information set.

When the sample size is smaller than the size of an information set $\mid B_r \mid < \mid I_r \mid$ then it is difficult to measure the uniformity of the distribution. But when the sample size is much larger than the size of an information set $\mid B_r \mid \gg \mid I_r \mid$ then the task becomes much easier.

By counting the number of times each element of an information set is sampled it is possible to use Pearson's χ^2 measure for a uniform distribution as a measure of the probability that the observed distribution matches the expected distribution:

$$E_{\text{v}} = \mid B_{r,i} \mid / \mid I_{r,i} \mid \tag{18}$$

as the ration of the size of the bag of models $B_{r,i}$ to the size of an information set $I_{r,i}$ for role r in round i.

The Pearson's chi squared statistic is then calculated:

$$\chi^2 = \sum_{\text{v} \in I_{r,i}} (O_{\text{v}} - E_{\text{v}})^2 / E_{\text{v}} \tag{19}$$

from the sum of the squares of the differences between observed O_{v} and expected E_{v} sampling frequencies. The resulting statistic is converted to a probability via pre-calculated tables.

5.4 Experimental Results

This section presents and interprets the experimental results in Appendix E. A summary of the experimental results is given below.

A batch of games is played while recording the states visited in each round when updating each of the models. The resources for each role are set so that it plays at well below the optimal level to ensures good variety in the game-play and a broad base for the calculation of the statistics. The basket of games chosen for experiments was drawn from the games available within the GGP community, and from the newly converted security games. A variety of information imperfections are represented in the games. Cut down versions of the game are used, when possible, without loss of generality. For example, the Blind Breakthrough would normally be played on an 8x8 board, but a 5x5 version is used to examine sampling efficiency. The roles were chosen to give meaningful results. In two

14. Weighted particle filters are described in Section 3.5.

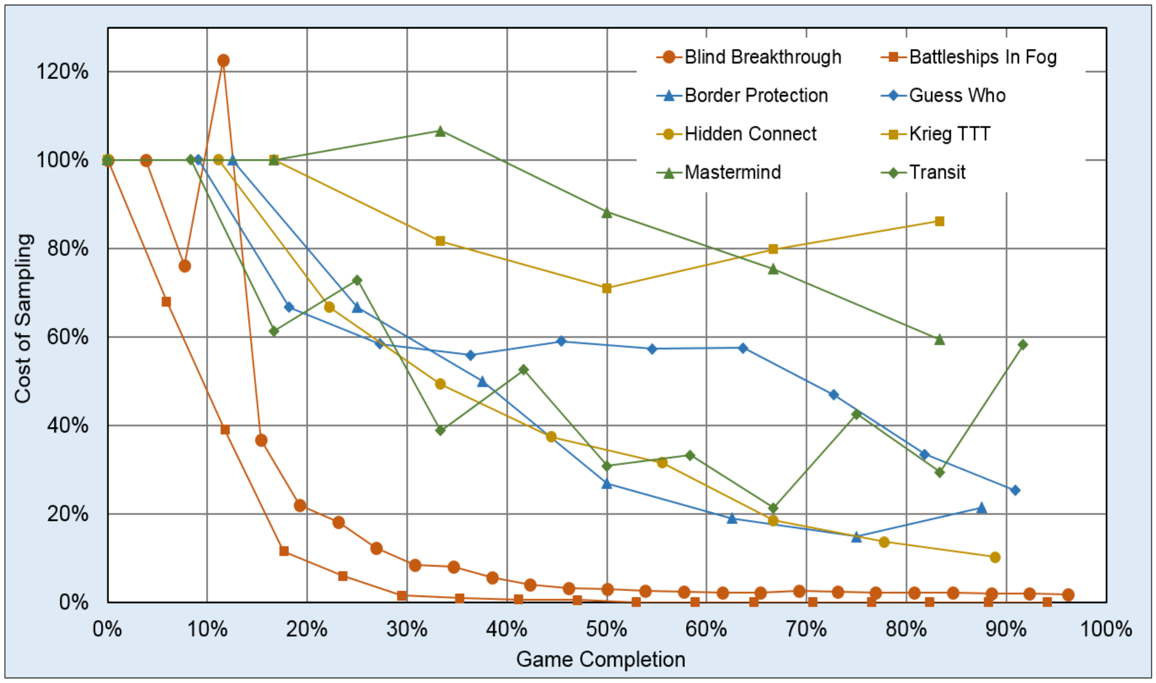


Figure 7: Cost of sampling using HyperPlay compared to performing a random sample, as measured by states visited.

player turn-taking games the second player is used for the statistic as they receive imperfect information first.

The HyperPlay process tests each move substitution in a random order; thus it is possible to gain an accurate estimate of the probability in equation 16 by calculating the “first time” successes in that round.

5.5 Sampling Efficiency

Figure 7 shows the HyperPlay cost of sampling compared to taking a random sample. The lines on the chart do not represent any continuous function, they just connect results from the same game. The horizontal axis is a measure of completion of the game. When a game is 100% complete then it is terminal, and no sampling is required. The vertical axis is the ration of states visited for each technique, a value of 50% means that HyperPlay visits only half the number of states per sample and is twice as efficient as random sampling.

The game-play is different for each game with some being turn-taking others not, some games have watershed rounds where percepts collapse an information set. The results show a significant reduction of the cost of sampling by using HyperPlay. The intuition here is that the cost of backtracking the local subtree will always be cheaper than starting each new sample attempt from the root node. See Appendix E for more comments.

5.6 Uniform Distribution

Bias is measured by playing a batch of games and logging the history footprint of each model for each round. The footprints are examined for repetition and a frequency chart is created. A Pearson's χ^2 test is then performed on the distribution and a probability value is calculated.

Game	Round	Q1	Median	Q3
Battleships In Fog	6	0.030	0.158	0.454
Blind Breakthrough	6	0.006	0.156	0.663
Border Protection	4	0.005	0.040	0.262
Guess Who	5	0.001	0.001	0.094
Hidden Connect	4	0.001	0.001	0.001
Krieg TTT	3	0.001	0.001	0.001
Mastermind	3	0.027	0.211	0.520
Transit	5	0.001	0.001	0.116

Table 1: Probability of a uniformly distributed sample of an information set created by HyperPlay in mid game.

The first remedy is to inversely **weight** the results from each model based on its sample frequency. If an element was represented by 5 models, then each model contributed only 20% of its outcomes to the evaluation process. The second remedy is to **re-balance** the sample by replacing a more frequently sampled history with a less frequently sampled history so that each element was sampled the same number of times.

The results in Appendix E show that in every game tested the sample became biased, with least mid-game bias in Mastermind with a 21% probability of an unbiased sample. However, by the end-game three of the games had given the agent enough percepts to allow it to re-sample in an unbiased way. As one playout of a game is not the same as another it is not possible to average the results so a median and upper and lower quartile readings of the probability value from a Pearson's χ^2 test are shown. The median value for Battleships in Fog of 0.158 infers that there is a 15.8% probability the sample is uniformly distributed. Some median values are extremely low at 0.001, or 0.1% probability of a uniform distribution.

The biased sample is a genuine concern as many of the search techniques are mathematically predicated on a uniform random sample of an information set.

It is worth noting that some samples are more uniform at the end of the game than they were in the middle of the game as the information set shrinks under certainty. Blind Breakthrough becomes a pawn swapping exercise towards the end game and nears certainty. Mastermind becomes certain as the binary search nears completion and the Transit game almost always becomes certain at the end when the evader is caught.

The table in Appendix E.6 show the results of a batch of games played with different player configurations. The base case is two evenly matched player with no attempt to correct biased samples. The **weight** remedy reduce the weighting of a sample proportional to its repetition so that each unique sample is given equal consideration before the application of the particle filter weighting, and the **balance** remedy re-balances the sample every round by replacing the oversampled play history with an under-sampled play history.

This was the hardest aspect of this research. That is, to find a repeatable, reproducible, realistic, in-game situation where the bias needs to be corrected in order to improve the agent's performance. While it was possible to manipulate the game-play to create scenarios where the agent's choices were significantly compromised by biased samples, these scenarios were so improbable as to have little impact on the average game. Generally, the remedies for biased samples do not improve the agent's performance. However, the cost of both remedies is so small that it is prudent and mathematically reassuring to implement them.

5.7 Case Study

In previous experiments the game variants and sizes have been chosen to prove (or disprove) the experimental objectives with a minimum of computational resources. However, there is value in extending the experimentation using one of the full-scale versions of a common game in the form of a case study. Blind Breakthrough in the full 8x8 format is used to show how impractical random sampling can be. The intention is to show that random sampling would become impossible within the normal time constraints, yet the HyperPlayer could successfully maintain a bag of models throughout the entire game. It should be noted that the HyperPlayer is capable of taking a model off-line if the backtracking process consumes too many resources. This is one of its design features for managing large search spaces.

Table 2 shows the results from the full-sized version of Blind Breakthrough. Both players were resourced just enough so as to exhibit a variety of game plays without making "stupid" moves. The results show a probability that a randomly chosen play history will be valid in the game being played, an estimated upper bound on the set of play histories and the number of active HyperPlay models.

This game is popular in GGP competitions, and so the results are very relevant to this work. In the 32nd round of a game the HyperPlayer could expect 1.3 models of a bag of 100 models to become inactive after each backtracking 100,000 states[15]. The successful models took an average of 745 states to update, giving a total cost of $166,000$ states to take the sample. Each valid random sample would cost approximately $32/2/0.01\% = 160,000$ states[16]. In this context, the random sampler would be completely ineffective taking only one sample for every 48 samples taken by the HyperPlayer. This result is totally consistent with the cut down version of the game reported in Figure 7 which shows a long-term cost of 2.2% for HyperPlay over random.

The conclusion is that HyperPlay is generally more efficient than a random search, and in some cases an order of magnitude more efficient. Clearly HyperPlay samples become biased, but with easy remedy. In some cases, random sampling is impossible within the time constraints of the game.

15. In turn taking games HyperPlay can use the idle turn to bring inactive models back on-line.
16. Each random sample will average 32/2 states before failure when the game is in the 32nd round

Round	$P(Valid(h_R))$	$sup\|\{h_R\}\|$	Active Models
1	100%	22	100%
2	100%	22	100%
-	-	-	-
16	1.19%	4.2 E+11	71.9%
17	0.73%	1.5 E+13	70.3%
-	-	-	-
32	$< 0.01\%$	6.0 E+23	50.0%
33	$< 0.01\%$	1.7 E+25	48.7%

Table 2: Full sized Blind Breakthrough with the probability of randomly choosing a valid play history, an upper bound on the set size and the models still active.

This technique is expected to be applicable in Artificial General Intelligence applications wherever an information set of indistinguishable action histories exists. Any search that can be described using a connected, directed graph with a single root node that is acyclic in its undirected form will benefit from this technique.

6. Lifting Model Sampling to Imperfect-Information Models

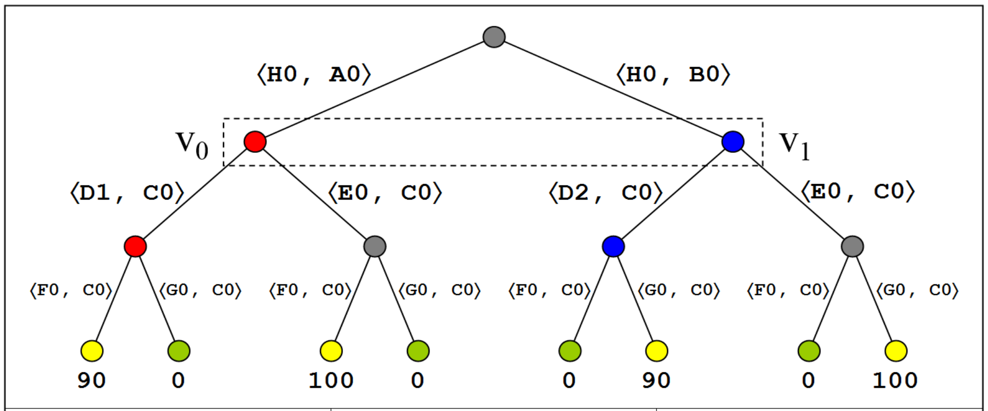
In this section, we offer a new improved technique that reasons with imperfect information. We present a formalism for a simple player based on the new technique then conduct experiments to validate the technique and to identify its strengths, weaknesses and failures.

Example 2. In the ExplodingBomb game in Appendix C and Figure 8 we see the following sequence of events:

- A spy secretly arms a bomb using either the Red wire or the Blue wire,
- A second spy must disarm the bomb, or both will die,
- The second spy may ask “Which Color?” for a small cost, and
- The second spy cuts one wire.

Using the HyperPlay algorithm, the second spy never asks the question as the samples of the players information set have perfect information. ¶

Example 2 continued. If we evaluate the legal moves in the Exploding Bomb game by reasoning on perfect information with Monte Carlo-based playouts we always (wait), we



A does (random arm(red))	D does (agent ask)	0 null
B does (random arm(blue))	E does (agent wait)	1 sees(agent red)
C does (random noop)	F does (agent cut(red))	2 sees(agent blue)
	G does (agent cut(blue))	
	H does (agent noop)	

Figure 8: The game tree for the Exploding Bomb. At Round 1 the agent has an information set of two nodes; that is, $I_{agent} = \{\mathrm{v}_0, \mathrm{v}_1\}$.

never (ask):

$$\begin{aligned}
&eval(do(h_i, ask), \vec{mc}, agent, 4)\\
&\quad = 0.25 \times (90 + 0 + 90 + 0) = 45\\
&eval(do(h_i, wait), \vec{mc}, agent, 4)\\
&\quad = 0.25 \times (100 + 0 + 100 + 0) = 50 \quad (\textit{selected})
\end{aligned}$$

¶

6.1 The Intuition Behind Imperfect Information Models

The original techniques updates models by grounding the missing information in the imperfect-information play histories. It does so with a randomly selected valid choice, then evaluates that model with a playout using a random move selection policy (aka. a random playout).

But what if it were possible to evaluate the model with an intelligent playout using move selection policies for all roles that were based on some experience of the game. This is the intuition behind an Imperfect-Information Model.

To implement this new technique, we are faced with three problems. Firstly, we must duplicate the game for each role. Secondly, we must replay the game from the start, not playout the game from the current round, as this will impact the valid choices in our sample. Thirdly, we must play out the game at every round as if it were the current round to calculate the move selection policies.

The solution to each of these problems leads to a multiplication of the number of states visited and characterizes the new techniques as a nested player. For simplicity we nest multiple HyperPlayers (one for each role) inside the simulations of another HyperPlayer reusing the same code for both parent and child. In Figure 3, line 18, the call to `SelectMove()` being the only difference between the parent HyperPlayer and the child HyperPlayer(s). This nested pair is the new player and the higher level simulation is described below.[17]

6.2 Imperfect-Information Simulations

The extended technique includes an Imperfect-Information Simulation (IIS) in the decision-making process. The IIS reasons directly with imperfect information, exploring the consequences of every action in the correct imperfect-information context. The new technique reasons across larger subsets of the information partition encompassing the upper bound of the union of information sets of all roles in the game. The result is that it places the correct value on information and will choose information-gathering moves and information protecting moves when it is cost effective to do so.

We use the term HyperPlay-II to indicate that this is the HyperPlay technique reasoning with imperfect information instead of reasoning with perfect-information models. Everything about the original technique remains in place, with an additional layer added in the form of the Imperfect-Information Simulation (IIS).

As before, the new technique requires a bag of models of an information set, representing a weighted sample. These models are updated, as before, based on moves and percepts from the game. But unlike before they are not evaluated directly using perfect-information evaluations but are the **mid-point** of an imperfect-information playout that starts at the original starting point of the game and passes **through** the model as if the model were the true game.

6.3 Formalism for HyperPlay-II

We recapitulate the formalism from (Schofield & Thielscher, 2015) adopting notation for finite games in extensive form; we refer to the original technique (Schofield et al., 2012) and we refer to the definitions in the previous sections.

For the correct valuation of information-gathering moves, the player must be able to evaluate a move using some type imperfect-information reasoning. In this case, we conduct a playout with imperfect information. To do this we take the path for a state corresponding to a sample of an information set and use it for an IIS with a HyperPlayer in each role, starting from the initial state of the game. The IIS will pass through the state sampled from the current information set on its way to termination. The terminal value is then used as a measure of utility in an imperfect-information context.

Definition 8. Let $G = \langle S, R, A, P, \upsilon, \delta \rangle$ be an imperfect-information game given by a valid GDL-II description, and let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a game tree induced by the game, then:

1. $replay : S \times \mathcal{H} \times \Pi \to S^{|R|}$ is the replay of a game consistent with the history of a state corresponding to a sample of a ninformation set,
2. $replay(s_0, h_r, \vec{\pi})$ is the replay of a game, as if $h_r = hp(\xi(s_t, r), x)$ and generating information sets for all roles, such that, $hp(\xi(h_r, i), x) \in H_i$ where r is our role and i is any other role,

17. This nested player could, itself, be nested inside another HyperPlayer such that the `SelectMove()` function could call on a full game played between multiple HyperPlayer-II players. This would be very resource intensive and does not appear, *prima facie*, to offer any advantage.

3. $IIS : S \times \Pi \times R \times \mathbb{N} \to \mathbb{R}$ is the imperfect-information simulation, where
4. $IIS(h_r, \vec{\pi}_{hp}, r, n)$ is an evaluation using n imperfect-information simulations, and is defined as $eval(replay(s_0, h_r, \vec{\pi}_{hp}), \vec{\pi}_{hp}, r, n)$ and $\vec{\pi}_{hp}$ is the move selection policy determined by the embedded HyperPlayer. □

Example 2 continued. The IIS generates multiple paths. Row one is what the agent knows, row two is a model in the agent's information set, row three shows two imperfect paths created by the IIS (one for each role), and row four shows models from an information sets of each of the IIS roles.

1	$\xi(s_t, agent)$		$\langle H0, \quad \rangle$ $\langle E0, \quad \rangle$	
2	$hp(\xi(s_t, agent), x) \to h_a$		$\langle H0, A0 \rangle$ $\langle E0, C0 \rangle$	
3	$\xi(h_a, role)$	$\langle H0, \quad \rangle$ $\langle E0, \quad \rangle$		$\langle \quad , A0 \rangle$ $\langle \quad , C0 \rangle$
4	$hp(\xi(h_a, role), x)$	$\langle H0, B0 \rangle$ $\langle E0, C0 \rangle$		$\langle H0, A0 \rangle$ $\langle D1, C0 \rangle$

It is worth noting that one of the models in row four may represent a state known to be outside *our* agent's information set. This is to be expected when the agent is considering the basis for its opponent's reasoning[18]. ¶

6.4 Move Selection Policy

We now show an abbreviated formalism for the HyperPlay-II technique, by only introducing the new aspects and assuming the reader will fill in the gaps from the original technique.

$$utility(d, a_{rj}) = IIS(\delta(d, \langle \vec{a}_{-r}, a_{rj} \rangle), \vec{\pi_{hp}}, r, n) \tag{20}$$

$$classUtility(a_{rj}) = \sum_{\mathrm{v}_i \in B_r} \begin{cases} P(\mathrm{v}_i) \times utility(state(\mathrm{v}_i), a_{rj}), & a_{rj} \in \lambda(state(\mathrm{v}_i)) \\ 0 \end{cases} \tag{21}$$

$$a_r = argmax_{a_{rj}} \left[\forall\ \mathrm{v}_i \in B_r \left[\forall\ a_{rj} \in \lambda(state(\mathrm{v}_i)) \left[classUtility(a_{rj}) \right] \right] \right] \tag{22}$$

Equation 20 gives the utility of a move for a role in a particular decision state (model) based on the terminal value of a complete path that passes through the decision state and chooses that particular move for that role. All other moves a chosen intelligently by a HyperPlayer performing its own evaluation on every move in every round.

Equation 21 collects the utility values for syntactically identical moves using a weighting factor calculated using Equation 4. While equation 22 finds the maximum utility across all equivalence classes across all models in the bag.

Note the similarity with the move selection policy $\vec{\pi}_{hp}$ given in Equation 12. In this respect, we characterize the new technique as a nested player.

18. The agent also considers the question "What might my opponent think about me?" based on what the agent knows about its opponent.

Example 2 continued. Reasoning on imperfect information using the move selection policy $\vec{\pi}_{hp}$ gives the following,

$$\begin{aligned}
&IIS(do(h_i, ask), \vec{\pi}_{hp}, agent, 4) \\
&\quad = 0.25 \times (90 + 90 + 90 + 90) = 90 \quad (selected) \\
&IIS(do(h_i, wait), \vec{\pi}_{hp}, agent, 4) \\
&\quad = 0.25 \times (100 + 0 + 100 + 0) = 50
\end{aligned}$$

¶

The use of the IIS extends the domain of reasoning to the least upper bound of the information partition $sup\mathcal{I} \subseteq D$. As the $hp()$ function generates paths across an information domain that is closed with respect to what the other roles **can** know, based on an information set of our role:

$$I_r = C_n(hp(\xi(s_t, r), x)) \tag{23}$$

$$sup\mathcal{I}_r = \bigcup_{i \in R} C_n(hp(\xi(I_r, i), x)) \tag{24}$$

It is both the expanded domain and the use of imperfect-information reasoning that gives the new technique an advantage over its predecessor. That is to say, there is an improvement in both the quantitative and qualitative aspects of the player.

6.5 Testing the Player

To validate our claim that HyperPlay-II correctly values information-gathering moves we implemented a version of the new technique using the move selection policy outlined in Equation (14). Games were selected from a variety of game topologies to cover different aspect of imperfect information. Games played at the Australasian Joint Conference on Artificial Intelligence 2013 were used as inspiration for the experiments.

Each game was played with a variety of configurations and the results reported in Appendix F.

Strengths The experimental results show the value that the new technique places on information, and how it correctly values information-gathering moves by itself and its opponents. It is able to collect information when appropriate, withhold information from its opponents, and keep its goals secret. The use of the Imperfect-Information Simulations is an effective tool for reasoning with imperfect information. A HyperPlayer-II was easily able to outperform an equally resourced HyperPlayer in all of the experiments.

Weaknesses We observe that the new technique is resource intensive as it uses nested playouts to evaluate move selections (Schofield & Thielscher, 2016) and have genuine concerns about its ability to scale up for larger games. These concerns motivate the next section of this article. Also, the IIS is effectively a search and can be influenced by the type of search. We observed that a simple search is susceptible to shallow traps and will follow this up with future work.

Limitations of HyperPlay-II There is an interesting type of games requiring what is known as "coordination without communication" (Fenster, Kraus, & Rosenschein, 1995) that goes beyond what our technique can achieve.

Consider the following *cooperative* variant of the Spy vs. Spy game. Spy1 sees which wire is used to arm a bomb. They then signal the name of a color to Spy2, who must try to disarm the bomb. *Both* win if Spy2 cuts the right wire and lose otherwise. Clearly Spy1 has an incentive to help Spy2, and there is one obvious way to do this: signal the color of the wire. The crux, however, is that the game rules can be designed such that the color being signaled is logically independent of the color of the armed wire. Whilst a human spy would see the syntactic similarity in the colors and hence the semantic link, the logic of the AI sees them as merely labels and does not make the connection.

7. Scalability of HyperPlay

Our motivation for this section is that the HyperPlay-II technique is resource intensive as it uses nested playouts for move selections. We focus on the consumption of computational resources for a particular level of performance. We make comparisons between HyperPlay and HyperPlay-II for a variety of games as well as measuring the increase in resources used by HyperPlay-II when a game is scaled up. We test several pruning techniques that have had some success in nested perfect-information players and measure the resources consumed.

The experiments examine the cost of the imperfect-information aspects of a player, not the embedded perfect-information search techniques. While the latter is fertile ground for improvement, we focus on the resource used by the imperfect-information algorithms.

7.1 States Visited

The new technique utilizes a nested playout for evaluating move choices, which causes a significant increase in the number of states visited during the analysis. However, because we are dealing with imperfect information the nested playout must start from the initial state of the game, not the current round. This doubles the number of states visited in a game compared to the perfect-information version of a nested playout.

There is a significant increase in computational resources required to play a game; from $O(bf \cdot d^2)$ for the HyperPlay-based player to $O(bf^2 \cdot d^4)$ for the HyperPlay-II player (Schofield & Thielscher, 2016), where bf is the Branching Factor and d is the depth of the game.

7.2 Imperfect-Information Game Topology

In the General Game Playing domain for imperfect-information games, the rules of the game and the reward structure is fully known to each player. What is not automatically known are the moves made by other players in the game. Player receive percepts from the game controller according to the rules of the game expressed in the GDL-II. And so, we look at the variations that can occur in the structure of a game. These topologies are detailed in Appendix G and one of each type of game is used in the experiments.

7.3 Heuristics and Pruning

Using a heuristic to improve the search and/or pruning the search space are effective way to improve the computational efficiency of the move selection process. We examine several techniques implemented in a Nested Monte Carlo player (Cazenave, Saffidine, Schofield, &

Thielscher, 2016) as there is a high degree of similarity between the nesting in this player and the nesting in the new technique.

Discounting Discounting is a way of improving the information extracted from a playout to give a rich set of terminal results instead of the usual 0 or 100. Discounting based on the depth of the playout has been demonstrated (Cazenave et al., 2016) to improve the search performance and to facilitate search pruning.

Discounting can only be effective in games with variable playout depth. For this reason our player's performance will not be improved when playing any of the games with fixed depth.

Cut on Win This technique works well with a Nested Monte Carlo player in turn taking two-player win/loss games, but has problems being implemented in games where players purchase information. The Cut on Win (CoW) technique requires a strict win/loss reward structure to be effective. We explore a variation where the player "knows" the maximum achievable score under optimal play conditions and uses that as a cut-off-point for the CoW pruning.

Pruning on Depth This technique also works well with a Nested Monte Carlo player in turn taking two-player win/loss games. Pruning on Depth (PoD) is ineffective when the playout depth is fixed.

7.4 Design of Experiments

We design experiments to answer two basic questions:

- Does HyperPlay-II perform better than HyperPlay at this type of game, and at what computational cost, and
- What is the impact of up-sizing the game on the computational cost for HyperPlay-II to achieve the same level of performance?

7.5 Experimental Results

The experimental results are shown in Appendix H.

HP versus HP-II When the topology is favorable the HP player performs as well as the HP-II player, improving its score as resources increase and reaching the same level of optimal play. Therefore, we would conclude that the HP player is an acceptable choice, except where the game topology makes it ineffective.

Computation Cost of HP-II The HP-II player requires significantly more resources to instantiate than the HP player. In each of the games tested, the number of states visited increased by an order of magnitude. The only benefit in using the HP-II player is that it correctly values information. Therefore, we conclude that the HP player should be the first choice, except where the game topology makes it ineffective.

Up-sizing the Game In all of the games tested we saw a significant impact when the game was up-sized. This was consistent with the theoretical analysis that stated the HP player as being $O(bf \cdot d^2)$ and the HP-II player as being $O(bf^2 \cdot d^4)$ (Schofield & Thielscher, 2016).

Discounting In all of the games, discounting had little impact on the outcome. In games with fixed depth, discounting is known to have no impact. In the other games, discounting did not hasten the win, or prolong the loss in any real way.

Pruning There was only one game out of five where pruning had a positive impact. Cut on Win and Pruning on Depth are known to be safe (Cazenave et al., 2016) for Nested Monte Carlo players with perfect information. The results from Banker and Thief, and Battleships in Fog suggest they may not be safe in Imperfect-Information Simulations, but the reason is not clear[19].

General The HP-II player will always play as well as the HP player, and will correctly value information in the context of the reward structure and the expected outcome of the game. Whereas, the HP player falls into the trap of elevating sample to fact and consequently values information at zero.

The player of choice should be the HP player, only utilizing the information valuing properties of the HP-II player when the game topology dictates.

8. Conclusions

We present a summary along with some comments and projections as to future direction of this research.

A formalism for the implementation of a "bolt-on" technique that converts a perfect-information player into an imperfect-information player was presented with sufficient detail to implement the technique is an existing player. A weighted particle filter was used on the player's information set. Whilst this is not a new idea, the confirmation of the need for weightings in GGP and the use of partial probabilities to calculate the weightings is useful.

The technique for maintaining samples is both sound and complete. That is, the `Update()` procedure will only return a state in the new information set and will select from all of the states. This provides a reliable sampling method for the weighted particle filter, even in a very large search space and a tight time constraint.

The question of the efficiency and the effectiveness of the `Update()` procedure compared to another sampling method has not been explored. The intuition is that the backtracking process will always be more efficient than forward tracking from the start of the game for each new sample.

The limitation of elevating sample to fact when correctly valuing information-gathering moves is identified with a specific game that is not played correctly. The player "believes" it has all of the information and so values any information-gathering (or withholding) moves at zero utility.

An improved technique that correctly values information-gathering moves is formalized. It is based on an Imperfect-Information Simulation that plays out an entire game with imperfect information and uses the terminal value as a measure of imperfect-information utility. The domain for reasoning is expanded considerably to the least upper bound of the information partition. Experimental results validating the improved technique are presented showing its strength at correctly value information in all of its forms.

19. Samples of an information set may not contain the same legal moves, but to offer this as a reason would be speculation.

An analysis showing the original technique as $O(bf \cdot d^2)$ and the improved technique as $O(bf^2 \cdot d^4)$ is investigated. Additional experimental results are presented along with the conclusion that the new technique was always effective, whereas the old technique was only effective when information-valuing was not required. In the case of non information-valuing games, the new technique was indeed resource intensive, consuming resources consistent with the theoretical analysis. Attempts to improve efficiency using pruning techniques borrow from nested perfect-information players were unsuccessful, but no real insight was afforded from the experimental data. This was a disappointment as these techniques were successful in perfect-information nested players.

9. Acknowledgments

This article recapitulates work previously submitted, and accepted, as a Ph.D. thesis (Schofield, 2018); the thesis expands on a collection of published conference papers and unpublished workshop papers.

The original research presented in this article was supported by the Australian Research Council under grant no. DP120102023.

References

Billings, D., Davidson, A., Schauenberg, T., Burch, N., Bowling, M., Holte, R., Schaeffer, J., & Szafron, D. (2006). Game-tree search with adaptation in stochastic imperfect-information games. In *Proceedings of the International Conference on Computers and Games (CG)*, pp. 21–34.

Bjornsson, Y., & Finnsson, H. (2009). CadiaPlayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, *1*, 4–15.

Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Cazenave, T., Saffidine, A., Schofield, M., & Thielscher, M. (2016). Discounting and pruning for nested playouts in general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 687–693.

Ciancarini, P., & Favini, G. P. (2010). Monte Carlo tree search in Kriegspiel. *Artificial Intelligence*, *174*, 670–684.

Clune, J. (2007). Heuristic evaluation functions for general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1134–1139.

Cowling, P. I., Powley, E. J., & Whitehouse, D. (2012). Information set Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(2), 120–143.

Edelkamp, S., Federholzner, T., & Kissmann, P. (2012). Searching with partial belief states in general games with incomplete information. In *Annual Conference on Artificial Intelligence*, pp. 25–36. Springer.

Engesser, T., Mattmüller, R., Thielscher, M., & Nebel, B. (2018). Epistemic Game Description Language and Dynamic Epistemic Logic compared. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1795–1802.

Fenster, M., Kraus, S., & Rosenschein, J. S. (1995). Coordination without communication: Experimental validation of focal point techniques. In *Proceedings of the International Conference on Multiagent Systems (ICMAS)*, pp. 102–108.

Frank, I., & Basin, D. (1998). Search in games with incomplete information: A case study in using Bridge card play. *Artificial Intelligence*, *100*(1–2), 87–123.

Frank, I., & Basin, D. (2001). A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, *252*(1-2), 217–256.

Genesereth, M. R., Love, N., & Pell, B. (2005). General game playing: Overview of the AAAI competition. *AI Magazine*, *26*(2), 62–72.

Ginsberg, M. L. (2001). GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, *14*, 303–358.

Hufschmitt, A. (2014). Parallelisation d'un general game player sur une architecture MPPA. Master's thesis, Universite Paris 8 de Vincennes a Saint-Deni.

Kupferschmid, S., & Helmert, M. (2007). A Skat player based on Monte-Carlo simulation. In *Proceedings of the International Conference on Computers and Games (CG)*, pp. 135–147.

Long, J., Sturtevant, N., Buro, M., & Furtak, T. (2010). Understanding the success of perfect information M Carlo sampling in game tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 134–140, Atlanta.

Love, N., Hinrichs, T., Schkufza, D. H. E., & Genesereth, M. (2006). General game playing: Game description language specification. Tech. rep. LG–2006–01, Stanford Logic Group.

Mehat, J., & Cazenave, T. (2011). A parallel general game player. *KI – Kunstliche Intelligenz*, *25*, 43–47. Springer.

Quenault, M., & Cazenave, T. (2007). Extended general gaming model. In *Computer Games Workshop*, pp. 195–204.

Richards, M., & Amir, E. (2012). Information set generation in partially observable games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Toronto.

Rosenhouse, J. (2009). *The Monty Hall Problem: The Remarkable Story of Math's Most Contentious Brain Teaser*. Oxford University Press.

Schiffel, S., & Thielscher, M. (2007). Fluxplayer: A successful general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1191–1196.

Schiffel, S., & Thielscher, M. (2014). Representing and reasoning about the rules of general games with imperfect information. *Journal of Artificial Intelligence Research*, *49*, 171–206.

Schofield, M. (2018). *Playing Imperfect-Information Games in General Game Playing by Maintaining Information Set Samples*. Ph.D. thesis, School of Computer Science & Engineering, UNSW Australia.

Schofield, M., Cerexhe, T., & Thielscher, M. (2012). HyperPlay: A solution to general game playing with imperfect information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1606–1612.

Schofield, M., & Thielscher, M. (2015). Lifting HyperPlay for general game playing to incomplete-information models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3585–3591.

Schofield, M., & Thielscher, M. (2016). The scalability of the HyperPlay technique for imperfect-information games. In *Proceedings of the AAAI Workshop on Computer Poker and Imperfect Information Games.*

Schofield, M., & Thielscher, M. (2017). The efficiency of the HyperPlay technique over random sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 282–290.

Silver, D., & Veness, J. (2010). Monte-Carlo planning in large POMDPs. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, pp. 2164–2172.

Thielscher, M. (2010). A general game description language for incomplete information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 994–999.

Thielscher, M. (2011). The general game playing description language is universal. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1107–1112.

Wisser, F. (2015). An expert-level card playing agent based on a variant of perfect information Monte Carlo sampling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 125–131, Buenos Aires.