HyperPlay: A Solution to General Game Playing with Imperfect Information

Michael Schofield and **Timothy Cerexhe** and **Michael Thielscher**
School of Computer Science and Engineering
The University of New South Wales
{mschofield,timothyc,mit}@cse.unsw.edu.au

Abstract

General Game Playing is the design of AI systems able to understand the rules of new games and to use such descriptions to play those games effectively. Games with imperfect information have recently been added as a new challenge for existing general game-playing systems. The HyperPlay technique presents a solution to this challenge by maintaining a collection of models of the true game as a foundation for reasoning, and move selection. The technique provides existing game players with a bolt-on solution to convert from perfect-information games to imperfect-information games. In this paper we describe the HyperPlay technique, show how it was adapted for use with a Monte Carlo decision making process and give experimental results for its performance.

Introduction

General Game Playing (GGP) is concerned with the design of AI systems able to understand the rules of new games and to use such descriptions to play those games effectively. GGP with perfect information has made advances, thanks largely to the standardisation of the Game Description Language (Love et al. 2006) and its widespread adoption, particularly in the AAAI GGP Competition (Genesereth, Love, and Pell 2005). Successful players typically employ either automatically generated evaluation functions (Clune 2007; Schiffel and Thielscher 2007) or some form of Monte Carlo technique such as the modern UCT (Björnsson and Finnsson 2009). The relative ease of Monte Carlo parallelisation has also rewarded distributed and cluster-based hardware in this domain (Méhat and Cazenave 2011). Games with *imperfect information* have recently been added as a new challenge for existing general game-playing systems (Thielscher 2010). However little progress has been made on these types of games beyond a specification of what their rules should look like (Quenault and Cazenave 2007; Thielscher 2011).

Frank and Basin (2001) have investigated imperfect-information games with a focus on Bridge, presenting a 'game-general' tree search algorithm that exploits a number of imperfect-information heuristics. This may effectively complement existing work applying theorem-proving techniques (Schiffel and Thielscher 2009). The Alberta Computer Poker Research Group has developed systems at the forefront of computer Poker players (Billings et al. 2006)—a challenging domain combining incomplete and misleading information, opponent modelling, and a large state space. While not explicitly interested in GGP, they do describe several techniques that could generalise to this field, including *miximix*, fast opponent modelling, and Nash equilibrium solutions over an abstracted state space. Meanwhile, our work is motivated by set sampling (Richards and Amir 2009) and by particle system techniques (Silver and Veness 2010). Similar special-case applications of sampling to reduce imperfect- to perfect-information can be found in (Ginsberg 2011; Kupferschmid and Helmert 2007).

Despite these advances, we are unaware of any explicit attempts to model and play *general* imperfect-information games. In this paper we introduce the *HyperPlay* technique, which presents a solution to this challenge by maintaining a collection of models of the true game as a foundation for reasoning and move selection. The technique provides existing game players with a bolt-on solution to convert from perfect- to imperfect-information games. We demonstrate this by integrating the technique with a Monte Carlo decision making process and providing experimental results of its performance in standard benchmark games.

The remainder of the paper is organised as follows. In the next section, we recapitulate syntax and operational semantics of the Game Description Language GDL-II, which provides the formal basis for General Game Playing (Genesereth, Love, and Pell 2005; Thielscher 2010). In the section that follows, we describe the general HyperPlay technique. Thereafter, we report on experiments with a player that uses the HyperPlay technique in conjunction with a standard Monte Carlo decision making process. We conclude with a short discussion.

Background: Game Description Language

The science of General Game Playing requires a formal language that allows an arbitrary game to be specified by a complete set of rules. The declarative Game Description Language (GDL) serves this purpose (Genesereth, Love, and Pell 2005). It uses a logic programming-like syntax and is characterised by the special keywords listed in Table 1.

Originally designed for games with complete information (Genesereth, Love, and Pell 2005), GDL has recently been extended to GDL-II (for: *GDL with incomplete/imperfect information*) by the last two keywords (`sees`, `random`)

```
 1 role(candidate).  role(random).
 2
 3 init(closed(1)).  init(closed(2)).  init(closed(3)).
 4 init(step(1)).
 5
 6 legal(random,hide_car(?d))  <= true(step(1)),
 7                                true(closed(?d)).
 8 legal(random,open_door(?d)) <= true(step(2)),
 9                                true(closed(?d)),
10                                not true(car(?d)),
11                                not true(chosen(?d)).
12 legal(random,noop)           <= true(step(3)).
13 legal(candidate,choose(?d)) <= true(step(1)),
14                                true(closed(?d)).
15 legal(candidate,noop)       <= true(step(2)).
16 legal(candidate,noop)       <= true(step(3)).
17 legal(candidate,switch)     <= true(step(3)).
18
19 sees(candidate,?d) <= does(random,open_door(?d)).
20 sees(candidate,?d) <= true(step(3)), true(car(?d)).
21 next(car(?d))     <= does(random,hide_car(?d)).
22 next(car(?d))     <= true(car(?d)).
23 next(closed(?d)) <= true(closed(?d)),
24                     not does(random,open_door(?d)).
25 next(chosen(?d)) <= does(candidate,choose(?d)).
26 next(chosen(?d)) <= true(chosen(?d)),
27                     not does(candidate,switch).
28 next(chosen(?d)) <= does(candidate,switch),
29                     true(closed(?d)),
30                     not true(chosen(?d)).
31
32 next(step(2)) <= true(step(1)).
33 next(step(3)) <= true(step(2)).
34 next(step(4)) <= true(step(3)).
35
36 terminal <= true(step(4)).
37
38 goal(candidate,100) <= true(chosen(?d)), true(car(?d)).
39 goal(candidate,  0) <= true(chosen(?d)), not true(car(?d)).
40 goal(random,0).
```

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

`role(?r)`	`?r` is a player
`init(?f)`	`?f` holds in the initial position
`true(?f)`	`?f` holds in the current position
`legal(?r,?m)`	`?r` can do `?m` in the current position
`does(?r,?m)`	player `?r` does move `?m`
`next(?f)`	`?f` holds in the next position
`terminal`	the current position is terminal
`goal(?r,?v)`	`?r` gets payoff `?v`
`sees(?r,?p)`	`?r` perceives `?p` in the next position
`random`	the random player (aka. Nature)

Table 1: GDL-II keywords

to describe arbitrary (finite) games with randomised moves and imperfect information (Thielscher 2010).

Example 1 (Monty Hall)**.** The GDL-II rules in Fig. 1 formalise a simple but famous game where a car prize is hidden behind one of three doors, a goat behind the others, and where a candidate is given two chances to pick a door. The intuition behind the rules is as follows.[1] Line 1 introduces the players' names (the game host is modelled by `random`). Lines 3–4 define the four features that comprise the initial game state. The possible moves are specified by the rules for `legal`: in step 1, the `random` player must decide where to place the car (lines 6–7) and, simultaneously, the candidate chooses a door (lines 13–14);[2] in step 2, `random` opens one of the other doors (lines 8–11) to reveal a goat; finally, the candidate can either stick to their earlier choice (`noop`) or switch to the other, yet unopened door (lines 16, 17). The candidate's only percepts are: the door opened by the host (line 19) and the location of the car at the end of the game (line 20). 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`).

GDL-II comes with some syntactic restrictions—for details we must refer to (Love et al. 2006; Thielscher 2010) due to lack of space—that ensure that every valid game description has a unique interpretation as a state transition system as follows. The **players** in a game are determined by the derivable instances of `role(?r)`. The **initial state** is the set of derivable instances of `init(?f)`. For any state S, the **legal moves** of a player `?r` are determined by the instances of `legal(?r,?m)` that follow from the game rules *augmented by an encoding of the facts in* S using the keyword `true`. Since game play is synchronous in the Game Description Language,[3] states are updated by *joint* moves (containing one move by each player). The **next position** after joint move $\vec{m}$ is taken in state S is determined by the instances of `next(?f)` that follow from the game rules *augmented by an encoding of* $\vec{m}$ *and* S using the keywords `does` and `true`, respectively. The **percepts** (aka. information) a player `?r` gets as a result of joint move $\vec{m}$ being taken in state S is likewise determined by the derivable instances of `sees(?r,?p)` after encoding $\vec{m}$ and S using `true` and `does`. Finally, the rules for `terminal` and `goal` determine whether a given state is **terminal** and what the players' **goal values** are in this case. In the remainder of the paper, we will use the following mathematical notation for some core elements of the semantics of a given (syntactically valid) GDL-II description.

- $\mathcal{L}(r, S)$ is the set of legal moves for player r in state S;
- $u(\vec{m}, S)$ is the updated state for a joint move $\vec{m}$ in state S;
- $\mathcal{I}(r, \vec{m}, S)$ is the set of percepts of player r after joint move $\vec{m}$ is taken in state S.

[1] A word on the syntax: We use infix notation for GDL-II rules as we find this more readable than the usual prefix notation.

[2] The precondition `true(closed(?d))` for the two moves `hide_car(?d)` and `choose(?d)` serves to restrict the domain of the variable argument `?d` in both cases.

[3] Synchronous means that all players move simultaneously. Turn-taking games are modelled by allowing players only one legal move without effect (such as `noop`) if it is not their turn.

This definition provides the basis for the operational semantics of GDL-II according to the following execution model:

1. Starting with the initial state, which is completely known to all players, in each state S each player r selects one of their legal moves from $\mathcal{L}(r, S)$. By definition `random` must choose a legal move with uniform probability.
2. The update function (synchronously) applies the joint move $\vec{m}$ to the current position, resulting in the new position $S' \leftarrow u(\vec{m}, S)$. Furthermore, each role r receives their individual information $\mathcal{I}(r, \vec{m}, S)$.
3. This continues until a terminal state is reached, and then the goal relation determines the result for all players.

The HyperPlay Technique

The HyperPlay technique provides a general approach that can be used by any general game player to convert from perfect-information games to imperfect-information games. The intuition is to translate imperfect-information games into a format suitable for simpler, perfect-information players. This translation is achieved by sampling the state space of an imperfect-information game, playing individual rounds in each of these perfect-information samples, and collating the separate results back together. This HyperPlay algorithm is summarised in Fig. 2 and will be explained in detail next, starting with the following points and notation:

- the state space (at a given step k) is sampled by *models* of the form $M = \langle B_1, \vec{m}_1, \ldots, B_{k-1}, \vec{m}_{k-1}, B_k \rangle$, which are sequences of joint moves interleaved with 'bad' moves (joint moves known to lead to an inconsistent model);
- the HyperPlay technique maintains a *bag* of these models (a.k.a. "HyperGames"): $\mathcal{H} = \{\!|M_1, \ldots, M_p|\!\}$;
- the *percepts* when taking a given joint move $\vec{m}$ in (the last state of) a given model M are denoted by $\mathcal{I}(\vec{m}, M)$ (as determined by the game rules);[4]
- similarly, we denote the *legal joint moves* in (the last state of) model M by $\mathcal{L}(M)$ (as determined by the game rules);
- *our* move in a joint move $\vec{m}$ is indexed by $\vec{m}|_r$;
- the *extension* of a model M by joint next move $\vec{m}_k$:

$$M \oplus \vec{m}_k = \langle B_1, \vec{m}_1, \ldots, B_{k-1}, \vec{m}_{k-1}, B_k, \vec{m}_k, \emptyset \rangle$$

  (that is, bad move sets are initialised to the empty set);
- **choose** $\phi \in \Phi$ **with** Θ is a pseudocode construct that attempts to choose a ϕ from the set Φ that satisfies additional constraints Θ; it returns true (and binds ϕ as a side-effect) if a suitable ϕ is found;
- global variables are:
  - the bag $\mathcal{H}$ of HyperGames;
  - our moves $a_1, a_2, a_3, \ldots$; and
  - our collected percepts $I_1, I_2, I_3, \ldots$

[4] We abuse notation here to make the pseudocode clearer. $\mathcal{I}(\vec{m}, M)$ is really shorthand for $\mathcal{I}(r, \vec{m}, state(M))$, where we are role r, and $state(M)$ represents the resulting state from the sequence of joint moves in M.

```
1  procedure main()
2  begin
3    H := {|⟨∅⟩, ..., ⟨∅⟩|}
4    n := 1
5    repeat
6      a_n := select_move(H)
7      I_n := submit_move(a_n)
8      for all M ∈ H do
9        forward(M, n + 1)
10     n := n + 1
11   until end_of_game
12 end
13
14 procedure forward(M = ⟨B_1, m_1, ..., B_{k-1}, m_{k-1}, B_k⟩, n)
15 begin
16   if k < n then
17     if choose m ∈ L(M) \ B_k
18           with m|_r = a_k && I(m, M) = I_k then
19       M := M ⊕ m
20       forward(M, n)
21     else
22       backtrack(M, n)
23 end
24
25 procedure backtrack(⟨B_1, m_1, ..., B_{k-1}, m_{k-1}, B_k⟩, n)
26 begin
27   B_{k-1} := B_{k-1} ∪ {m_{k-1}}
28   forward(⟨B_1, m_1, ..., B_{k-1}⟩, n)
29 end
```

Figure 2: Hyperplay

More explicitly, we maintain a bag $\mathcal{H}$ of HyperGames—random samples or 'guesses' of the current true game state. Each of these samples is in fact a model M—a hypothetical game history consistent with all percepts seen so far, along with sets of 'bad' moves in each round that were found to be inconsistent. Thus our samples are always drawn from the game's *information set* (Rasmusen 2007). In each round n, a perfect-information player can select a next move a_n suitable for each of these isolated models; see line 6. This move selection is arbitrary from the perspective of our algorithm provided the selection is a legal move for the player with respect to the perfect-information model (further discussion is provided in the following subsection). Our move selection is then submitted to the game controller and a new set of percepts I_n for this round is received (line 7). Each model M in our bag of samples $\mathcal{H}$ is then propagated *forward* (line 9) to reflect the deeper game tree. This process randomly **chooses** a joint move $\vec{m}$ for each player from the set of non-bad, joint legal moves in that state (line 17) **with** the additional constraints that our move is the one we submitted and that the expected percepts $\mathcal{I}(\vec{m}, M)$ are identical to the set of percepts I_n that we actually received (line 18). Notice that if we select a path that the last percepts reveal to be impossible, then we reach a state where no consistent joint move can be found. In this case (line 22) we *backtrack* by adding the guilty move vector to a set of bad moves for that state (line 27), then calling *forward* on this earlier game node, effectively undoing the move and attempting to push forward again. This process repeats until a consistent model is found for the current round ($k < n$ fails, on line 16).

Move Selection

The HyperPlay algorithm is agnostic of the move selection process, which should be designed with the following considerations in mind:

- the expected value of a move in a HyperGame;
- the probability that the HyperGame is the true game.

Each HyperGame represents a path of move choices made from the initial game state. These move choices are recorded, along with all of the possible choices for each step along the path. From this information we are able to determine the likelihood that a HyperGame is the true game, relative to the other HyperGames in the bag. We begin with the conditional probability that a particular HyperGame HG_i is the true game, given the *Percepts* it has collected:

$$P(HG_i|\textit{Percepts}) = \frac{P(\textit{Percepts}|HG_i) \cdot P(HG_i)}{P(\textit{Percepts})}$$

Because the percepts are fully determined by the game rules, $P(\textit{Percepts}|HG_i)$ equals 1 in the case that the game is consistent with the percepts, and 0 otherwise. $P(\textit{Percepts})$ is a normalising factor across all HyperGames, so the above equation simplifies to:

$$P(HG_i|\textit{Percepts}) \sim P(HG_i)$$

The probability of a HyperGame being the true game, $P(HG_i)$, will in general depend on the preferences of (biased) opponents. A detailed discussion of opponent modelling in this context goes beyond the scope of this paper, which is why we only consider the approximation of, as far as our player knows, a uniform distribution over all possible moves by the other players.

To this end, let the *choice factor* of a HyperGame HG_i be the product of the number of *choices* at each node along the path from the initial state to the current state:

$$\textit{ChoiceFactor}_i = \prod_j \textit{Choices}_{i,j}$$

Where $\textit{Choices}_{i,j}$ is the number of choices at the *jth* node in the path of the HyperGame HG_i.

Then the probability of a HyperGame being the true game, relative to the other HyperGames in the bag and assuming uniform distribution over the move choices, is:

$$P(HG_i) = \frac{1/\textit{ChoiceFactor}_i}{\sum_n 1/\textit{ChoiceFactor}_n}$$

where the sum of all the relative probabilities is unity.

The *expected payoff* for a move can then be expressed as:

$$E(\textit{Move}_j) = \sum_i E(\textit{Move}_{i,j}) \cdot P(HG_i)$$

where $E(\textit{Move}_{i,j})$ refers to the expected value (as determined by the embedded perfect-information player) of $\textit{Move}_j$ in HyperGame HG_i.

The HyperPlayer then submits the move with the greatest expected payoff to the game server.[5]

[5]The reader will note that this approach does not prevent the HyperPlayer from making a move that is illegal in the true game state, but does minimise such an event if the expected value is set to zero in all HG_i in which the move is illegal.

Example 2 (Monty Hall)**.** The effectiveness of the move selection calculations detailed above can be seen by an examination of the Monty Hall game. Note that this game has low computational complexity, yet the validation of the expected payoff calculation is far from trivial.

If the HyperPlayer gives equal weighting to each HyperGame then it will switch doors 50% of the time and hence only win the car 50% of the time. However using the choice factor to calculate the relative probability of each HyperGame, the expected payoff for switching is correctly identified and the car is won 67% of the time.

An inspection of the following game tree illuminates the impact of the choice factor on the decision making process.

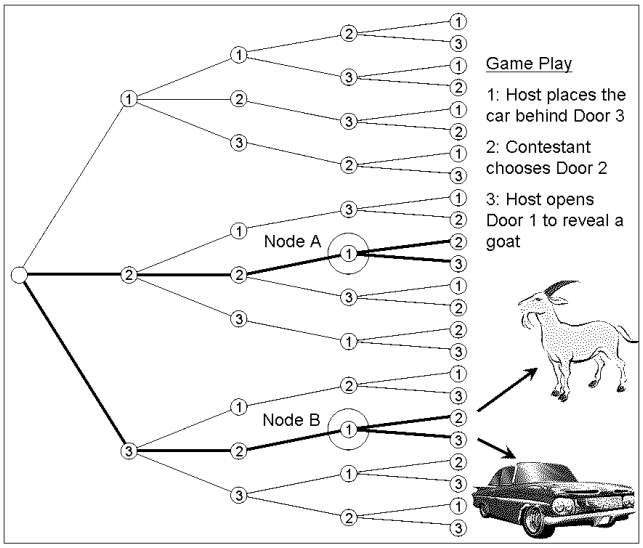

Figure 3: Sample game tree (where moves have been serialised, for the sake of clarity)

At the last round there are only two possible states for a model of the game (Node A and Node B); reflected by equal numbers of HyperGames in the bag. However one of these models is twice as likely to be the true game:

$$\text{Node A: } ChoiceFactor = 3 \cdot 3 \cdot 2 = 18$$
$$\text{Node B: } ChoiceFactor = 3 \cdot 3 \cdot 1 = 9$$
$$P(HG_A) = \frac{1/18}{1/18 + 1/9} = 33\%$$
$$P(HG_B) = \frac{1/9}{1/18 + 1/9} = 67\%$$

The simulation results will give an average payoff of 50% for each of the remaining doors in each HyperGame, but the relative probabilities of the HyperGames will result in an expected payoff of 67% if the HyperPlayer switches from Door 2 to Door 3. This is verified in the experimental results in the following section.

Experiments

To validate our claim that HyperPlay provides a solution to GGP with imperfect information, we implemented a version of the HyperPlayer designed to facilitate high throughput computing. We modelled the game server and both players in a single thread so it could be parallelised across many CPUs. The computing array consisted of (approx) 200 dual core Intel PCs situated throughout the School of Computer Science and Engineering. The experiments were broken up into small batch runs and scheduled across all of the PCs simultaneously. The HyperPlayer was given sufficient resources to complete each move in less than 15 seconds. Moves were played as soon as the HyperPlayer had completed its calculations in order to compress the required play time.

Each data point plotted in the charts below is supported by 1000 games. Blind Breakthrough played on a 7x7 board is complex enough to require 432 hours of CPU time. Testing larger boards would have been prohibitive.

We utilised simple, standard Monte Carlo simulations where the expected value of a move in a HyperGame is given by the average payoff from simulations, then computed the expected payoff using the technique detailed above. Several games were selected for these experiments: Monty Hall, Krieg-TicTacToe, and Blind Breakthrough, all taken from the standard source for benchmark games.[6]

Monty Hall In addition to the rules already presented, we varied the number of initial doors between three, four, and five doors. The host always opens all but two doors.

Krieg-TicTacToe A variant of traditional TicTacToe where the opponent's pieces are hidden. Players move simultaneously and the winning length was fixed at four in a row. We varied the board size between 4x4, 5x5, and 6x6 squares. Players are given feedback via percepts as to whether their move, and their opponent's move, was successful (they placed a new mark) or not (due to collision or selecting a square that had already been marked).

Blind Breakthrough A variant of chess where each player starts with two rows of pawns on their side of the board. Players take turns, trying to 'break through' their opponent's ranks to reach the other side first. Opponent's pieces are hidden, as with Krieg-TicTacToe. Pawn moves are as per normal chess, including taking moves, with the exception of *en passant* and opening double-square moves, which were removed. The board size was varied between 5x5, 6x6, and 7x7 squares and players were given equal opportunities to go first. In each round, each player has one chance to make a true move; if they fail then the game server makes a random legal move on their behalf. Players are given feedback via percepts on the actual move made in the game and about the capturing moves available to them.

[6] *http://ggpserver.general-game-playing.de*

Standardised Opponent

In each of the two player games the HyperPlayer opposed a Cheat—a HyperPlayer with access to the true game, who maintains HyperGames that were the true game (instead of models of the true game). The Cheat was fully resourced so that it made the best move choices within the limitations of the move selection process, where the key resources are the number of HyperGames maintained and the number of simulations performed in each HyperGame. The method for calculating the Cheat's resources was to play one Cheat against another Cheat with different resources. The following figure shows the results for a game of Krieg-TicTacToe played on a 5x5 board with a win length of 4. This method served two

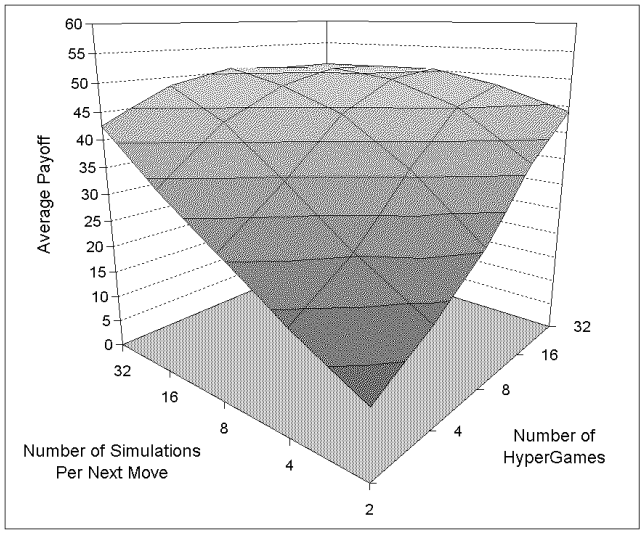

Figure 4: Calculating the resources for the Cheat

purposes; it validated the software implementation by giving an expected result, and it identified the minimum resources needed for the Cheat to be considered fully resourced. In this case the Fully Resourced Cheat was initially given one HyperGame (the true game) and ran 1024 simulations for each next move. The results showed that this could be reduced to one HyperGame and 256 simulations without any degradation in the Cheat's performance. This was then used as the standardised opponent for 5x5 Krieg-TicTacToe.

Confidence Level

Each game variant and HyperPlayer configuration was played one thousand times and the average results (payoff) reported. For the two player games a result of zero indicated that the Cheat scored the maximum payoff in every game, and a result of 100 indicated that the HyperPlayer scored the maximum payoff in every game.

In order to prove our claim, the data should show that the results tend towards a limiting value for fully resourced players. That is, some large number of models of the true game can be substituted for perfect information about the true game.

A confidence level of 99% was used in a two-tailed calculation using the standard deviation for a binomial distribution. That is to say that we were 99% confident that the HyperPlayer would score within the range reported.

Resource Limitation

To show a trend towards a limiting value, experiments were conducted with varying resources—the HyperPlayer was limited in the number of HyperGames and the simulations per next move. These limitations were combined into a single factor for our graphs:

$$Resources = NoOfHyperGames \cdot NoOfSimsPerNextMove$$

Results

Monty Hall The results for the Monty Hall experiments validated the use of the choice factor in making the move selection. As expected the adequately resourced HyperPlayer was able to achieve a performance of:

$$AveragePayoff = (NoOfDoors - 1)/NoOfDoors$$

while the inadequately resourced HyperPlayer achieved an average payoff of 50%.

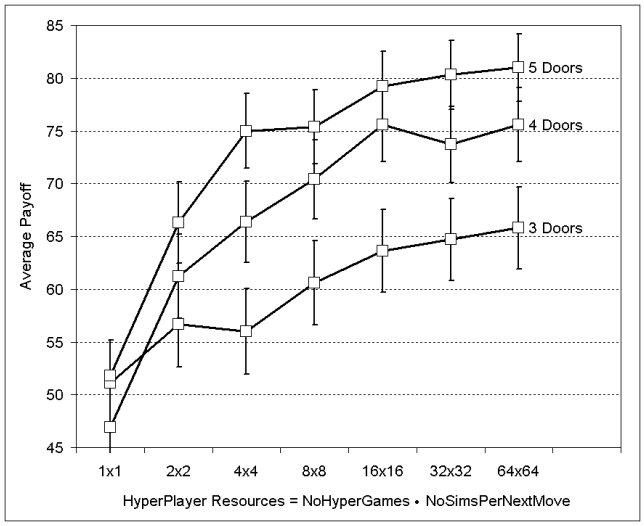

Figure 5: Monty Hall results

Krieg-TicTacToe These experiments showed steady improvement in performance as HyperPlayer resources increased. The limiting values appear to be well short of the 50% mark, especially on the larger board. This is due to the reduced number of percepts relative to the game duration—the Cheat often wins before the HyperPlayer can establish an accurate model.

Blind Breakthrough The results of the Blind Breakthrough experiments show clear improvement in performance as resources increase; approaching a limiting value.

As the number of HyperGames increases and the number of simulations per next move increases the HyperPlayer is able to match the Cheat.

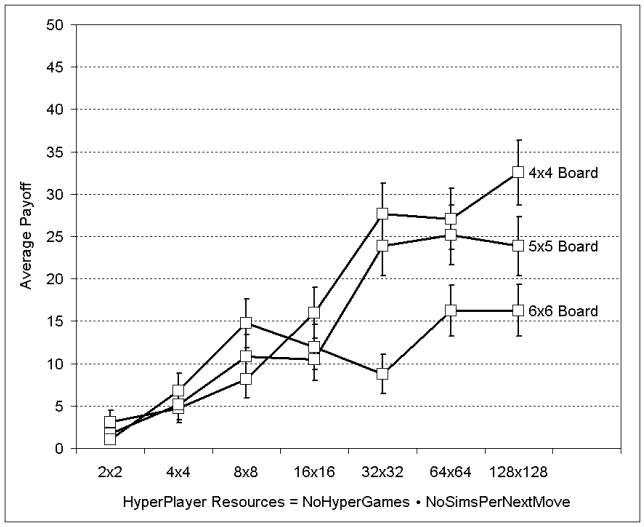

Figure 6: Krieg-TicTacToe results

Special note should be paid to the 5x5 results where the percepts were sufficient for the HyperPlayer to maintain models that were very close to the true game. This allowed the HyperPlayer to perform as if it had perfect information.

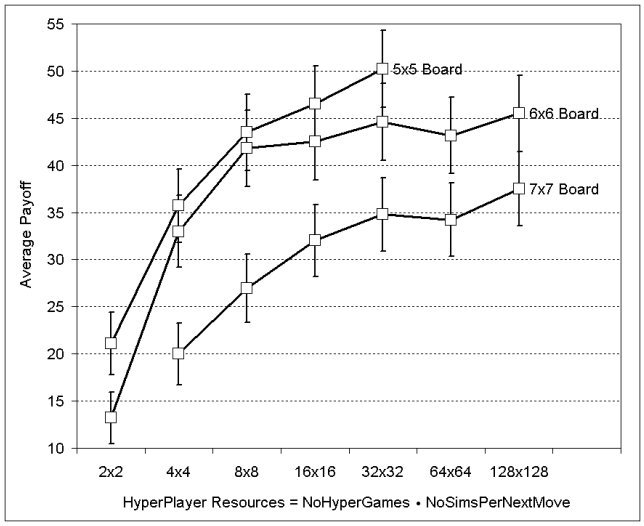

Figure 7: Blind Breakthrough results

GGP Competition

We have also implemented a version of our HyperPlay algorithm that understands the new Game Description Language GDL-II using an extension of the reasoning module that underlies FluxPlayer (Schiffel and Thielscher 2007). To test this player in the absence of competitors we appealed to the same test framework as above—we played against a cheat, for which we chose CadiaPlayer, a former champion of the complete-information track (Björnsson and Finnsson 2009).

The setup gives CadiaPlayer a huge advantage—it can see the true game state but we have to guess at it. Naturally this

takes the fun out of some games. However the imperfect information found in simultaneous play is often enough for us to approach or even outperform our challenger. Space limitations dictate only a short qualitative summary of these results, see Table 2.[7]

Game	Results
Monty Hall	Our move selection identifies the optimal 'always switch' strategy. CadiaPlayer sees the host's placement.
Krieg-TicTacToe	Our HyperPlayer needs only a few samples to identify highly accurate models. We frequently beat CadiaPlayer due to the complexity of simultaneous play.
Blind Breakthrough	The huge state space makes synchronisation expensive for the HyperPlayer. This restricts the number of simulations possible in the short time limits tested, giving CadiaPlayer an easy lead.

Table 2: GGP-compatible tests against CadiaPlayer

Discussion

The experimental results show a successful implementation of the HyperPlay technique for playing imperfect-information games. The collection of models (HyperGames) can be very accurate, and is a credible substitute for perfect information about the true game, especially in games with more percepts (compare Blind Breakthrough to Krieg-TicTacToe)—we can be competitive even against a Cheat.

Our choice factor for each model is adequate in calculating the relative probability for the game within the information set, assuming uniform distribution, and hence is able to correctly identify choices influenced by conditional probability.

We intend to explore additional factors, e.g. with a view towards more cautious/aggressive play and for opponent modelling. HyperPlay also has application as a reasoning engine in conditions of uncertainty, and we will be exploring its use in artificially intelligent systems using abductive reasoning to identify and implement goal-oriented behaviour.

Our HyperPlayer implementation only uses Monte Carlo simulation for move selection, yet the case is demonstrated for the use of the HyperPlay technique as a bolt-on solution for all types of perfect-information game players. That is, it should work for knowledge-based players too. Our immediate future work is to demonstrate this by facilitating an easy-integration interface and testing with previous champions. However the advantages of this flexibility come with a price—backtracking can become a bottleneck in very large search spaces. We are developing strategies to overcome this problem and hope to bring these to light in future works.

[7]For completeness: play and start clocks were varied between 15 and 60 seconds. Both players and the game controller were run on a single 2.4 GHz dual-core iMac8,1.

Acknowledgements. We thank the anonymous reviewers for their constructive comments on an earlier version of the paper. This research was supported under Australian Research Council's (ARC) *Discovery Projects* funding scheme (project DP 120102023). Michael Thielscher is the recipient of an ARC Future Fellowship (project FT 0991348). He is also affiliated with the University of Western Sydney.

References

Billings, D.; Davidson, A.; Schauenberg, T.; Burch, N.; Bowling, M.; Holte, R.; Schaeffer, J.; and Szafron, D. 2006. Game-tree search with adaptation in stochastic imperfect-information games. In *Proc. Computers and Games*, 21–34.

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

Clune, J. 2007. Heuristic evaluation functions for general game playing. In *Proc. AAAI*, 1134–1139.

Frank, I., and 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.; and Pell, B. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26(2):62–72.

Ginsberg, M. 2011. GIB: Imperfect information in a computationally challenging game. *CoRR*.

Kupferschmid, S., and Helmert, M. 2007. A Skat player based on Monte-Carlo simulation. In *Proc. Computers and Games*, 135–147.

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2006. General game playing: Game description language specification. Technical Report LG–2006–01, Stanford Logic Group.

Méhat, J., and Cazenave, T. 2011. A parallel general game player. *KI Journal* 25(1):43–47.

Quenault, M., and Cazenave, T. 2007. Extended general gaming model. In *Computer Games Workshop*, 195–204.

Rasmusen, E. 2007. *Games and Information: an Introduction to Game Theory*. Blackwell Publishing, fourth edition.

Richards, M., and Amir, E. 2009. Information set sampling for general imperfect information positional games. In *Proc. IJCAI-09 Workshop on GGP (GIGA'09)*, 59–66.

Rosenhouse, J. 2009. *The Monty Hall Problem*. Oxford University Press.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In *Proc. AAAI*, 1191–1196.

Schiffel, S., and Thielscher, M. 2009. Automated theorem proving for general game playing. In *Proc. IJCAI*, 911–916.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *Proc. NIPS*, 2164–2172.

Thielscher, M. 2010. A general game description language for incomplete information games. In *Proc. AAAI*, 994–999.

Thielscher, M. 2011. The general game playing description language is universal. In *Proc. IJCAI*, 1107–1112.