Finding Optimal Strategies for Imperfect Information Games*

Ian Frank
Complex Games Lab
Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305
ianf@etl.go.jp

David Basin
Institut für Informatik
Universität Freiburg
Am Flughafen 17
Freiburg, Germany
basin@informatik.uni-freiburg.de

Hitoshi Matsubara
Complex Games Lab
Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305
matsubar@etl.go.jp

Abstract

We examine three heuristic algorithms for games with imperfect information: Monte-carlo sampling, and two new algorithms we call *vector minimaxing* and *payoff-reduction minimaxing*. We compare these algorithms theoretically and experimentally, using both simple game trees and a large database of problems from the game of Bridge. Our experiments show that the new algorithms both out-perform Monte-carlo sampling, with the superiority of payoff-reduction minimaxing being especially marked. On the Bridge problem set, for example, Monte-carlo sampling only solves 66% of the problems, whereas payoff-reduction minimaxing solves over 95%. This level of performance was even good enough to allow us to discover five errors in the expert text used to generate the test database.

Introduction

In games with imperfect information, the actual 'state of the world' may be unknown; for example, the position of some of the opponents' playing pieces may be hidden. Finding the optimal strategy in such games is NP-hard in the size of the game tree (see *e.g.*, (Blair, Mutchler, & Liu 1993)), and thus a heuristic approach is required to solve non-trivial games of this kind.

For any imperfect information game, we will call each possible outcome of the uncertainties (*e.g.*, where the hidden pieces might be) a possible *world state* or *world*. Figure 1 shows a game tree with five such possible worlds $\mathsf{w}_1, \cdots, \mathsf{w}_5$. The squares in this figure correspond to MAX nodes and the circles to MIN nodes. For a more general game with n possible worlds, each leaf node of the game tree would have n payoffs, each corresponding to the utility for MAX of reaching that node in each of the n worlds.[1]

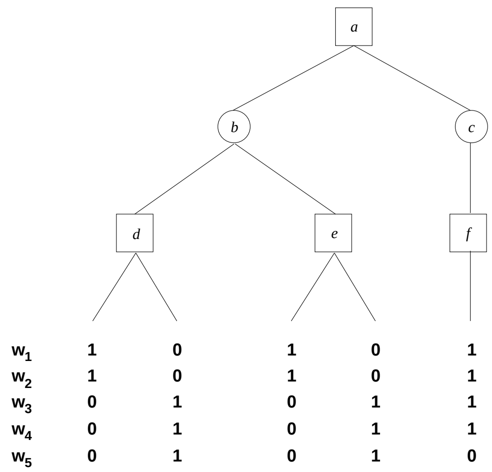

Figure 1: A game tree with five possible worlds

If both MAX and MIN know the world to be in some state w_i then all the payoffs corresponding to the other worlds can be ignored and the well-known minimax algorithm (Shannon 1950) used to find the optimal strategies. In this paper we will consider the more general case where the state of the world depends on information that MAX does not know, but to which he can attach a probability distribution (for example, the toss of a coin or the deal of a deck of cards). We examine this situation for various levels of MIN knowledge about the world state.

We follow the standard practice in game theory of assuming that the best strategy is one that would not change if it was made available to the opponent (von

[1]For the reader familiar with basic game-theory, (von Neumann & Morgenstern 1944; Luce & Raiffa 1957), Figure 1 is a compact representation of the extensive form of a particular kind of two-person, zero-sum game with imperfect information. Specifically, it represents a game tree with a single chance-move at the root and n identically shaped subtrees. Such a tree can be 'flattened', as in Figure 1, by assigning payoff n-tuples to each leaf node so that the ith component is the payoff for that leaf in the ith subtree. It is assumed that the only move with an unknown outcome is the chance move that starts the game. Thus, a single node in the flattened tree represents between 1 and n *information sets*: one if the player knows the exact outcome of the chance move, n if the player has no knowledge.

Neumann & Morgenstern 1944). For a MIN player with *no* knowledge of the world state the situation is very simple, as an expected value computation can be used to convert the multiple payoffs at each leaf node into a single value, and the standard minimax algorithm applied to the resulting tree. As MIN's knowledge increases, however, games with imperfect information have the property that it is in general important for MAX to prevent MIN from 'finding out' his strategy, by making his choices probabilistically. In this paper, however, we will restrict our consideration to *pure* strategies, which make no use of probabilities. In practice, this need not be a serious limitation, as we will see when we consider the game of Bridge.

We examine three algorithms in detail: Monte-carlo sampling (Corlett & Todd 1985) and two new algorithms we call *vector minimaxing* and *payoff-reduction minimaxing*. We compare these algorithms theoretically and experimentally, using both simple game trees and a large database of problems from the game of Bridge. Our experiments show that Monte-carlo sampling is out-performed by both of the new algorithms, with the superiority of payoff-reduction minimaxing being especially marked.

Monte-carlo Sampling

We begin by introducing the technique of Monte-carlo sampling. This approach to handling imperfect information has in fact been used in practical game-playing programs, such as the QUETZAL Scrabble program (written by Tony Guilfoyle and Richard Hooker, as described in (Frank 1989)) and also in the game of Bridge, where it was proposed by (Levy 1989) and recently implemented by (Ginsberg 1996b).

In the context of game trees like that of Figure 1, the technique consists of guessing a possible world and then finding a solution to the game tree for this complete information sub-problem. This is much easier than solving the original game, since (as we mentioned in the Introduction) if attention is restricted to just one world, the minimax algorithm can be used to find the best strategy. By guessing different worlds and repeating this process, it is hoped that an action that works well in a large number of worlds can be identified.

To make this description more concrete, let us consider a general MAX node with branches $M_1, M_2, \cdots$ in a game with n worlds. If e_{ij} represents the minimax value of the node under branch M_i in world w_j, we can construct a scoring function, f, such as:

$$f(M_i) = \sum_{j=1}^{n} e_{ij} \Pr(\mathsf{w}_j)\,, \tag{1}$$

where $\Pr(\mathsf{w}_j)$ represents MAX's assessment of the probability of the actual world being w_j. Monte-carlo sampling can then be viewed as selecting a move by using the minimax algorithm to generate values of the e_{ij}s, and determining the M_i for which the value of $f(M_i)$ is greatest. If there is sufficient time, all the e_{ij} can be generated, but in practice only some 'representative' sample of worlds is examined.

As an example, consider how the tree of Figure 1 is analysed by the above characterisation of Monte-carlo sampling. If we examine world w_1, the minimax values of the left-hand and the right-hand moves at node a are as shown in Figure 2 (these correspond to e_{11} and e_{21} for this tree). It is easy to check that the

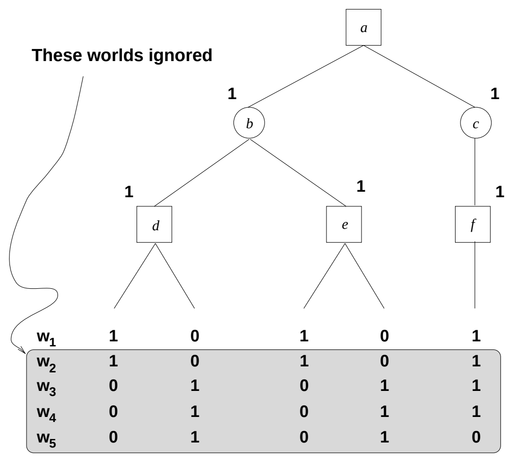

Figure 2: Finding the minimax value of world w_1

minimax value at node b is again 1 if we examine any of the remaining worlds, and that the value at node c is 1 in worlds w_2, w_3, and w_4, but 0 in world w_5. Thus, if we assume equally likely worlds, Monte-carlo sampling using (1) to make its branch selection will choose the left-hand branch at node a whenever world w_5 is included in its sample. Unfortunately, this is not the best strategy for this tree, as the right-hand branch at node a offers a payoff of 1 in four worlds. The best return that MAX can hope for when choosing the left-hand branch at node a is a payoff of 1 in just three worlds (for any reasonable assumptions about a rational MIN opponent).

Note that, as it stands, Monte-carlo sampling identifies pure strategies that make no use of probabilities. Furthermore, by repeatedly applying the minimax algorithm, Monte-carlo sampling models the situation where both MIN and MAX play optimally in each individual world. Thus, the algorithm carries the implicit assumption that both players *know* the state of the world.

Vector Minimaxing

That Monte-carlo sampling makes mistakes in situations like that of Figure 1 has been remarked on in the literature on computer game-playing (see, *e.g.*, (Frank 1989)). The primary reason for such errors has also recently been formalised as *strategy fusion* in (Frank &

Basin 1998). In the example of Figure 2, the essence of the problem with a sampling approach is that it allows *different choices* to be made at nodes d and e in *different worlds*. In reality, a MAX player who does not know the state of the world must make a *single choice* for all worlds at node d and another single choice for all worlds at node e. Combining the minimax values of separate choices results in an over-optimistic analysis of node b. In effect, the false assumption mentioned above that MAX knows the state of the world allows the results of different moves — or strategies — to be 'fused' together.

We present here an algorithm that removes the problem of strategy fusion from Monte-carlo sampling by ensuring that at any MAX node a single branch is chosen in all worlds. This algorithm requires the definition of a payoff vector, $\vec{K}(\nu)$, for leaf nodes of the game tree, ν, such that $\vec{K}[j](\nu)$ (where $\vec{K}[j]$ is the jth element of the vector $\vec{K}$) takes the value of the payoff at ν in world w_j ($1 \leq j \leq n$). Figure 3 defines our algorithm, which we call *vector minimaxing*. It uses payoff vectors to identify a strategy for a tree t, where $sub(t)$ computes the set of t's immediate subtrees.

Algorithm *vector-mm*(t):
Take the following actions, depending on t.

Condition	Result
t is leaf node	$\vec{K}(t)$
root of t is a MIN node	$\min\limits_{t_i \in sub(t)} \textit{vector-mm}(t_i)$
root of t is a MAX node	$\max\limits_{t_i \in sub(t)} \textit{vector-mm}(t_i)$

Figure 3: The vector minimaxing algorithm

In this algorithm, the normal min and max functions are extended so that they are defined over a set of payoff vectors. The max function returns the single vector $\vec{K}$, for which

$$\sum_{j=1}^{n} \Pr(\mathsf{w}_j)\vec{K}[j] \tag{2}$$

is maximum, resolving equal choices randomly. In this way, vector minimaxing commits to just *one* choice of branch at each MAX node, avoiding strategy fusion (the actual strategy selected by the algorithm is just the set of the choices made at the MAX nodes).

As for the min function, it is possible to define this as the dual of the max function, returning the single vector for which (2) is minimum. However, this would result in modelling the simple situation, described in the Introduction, where MIN, like MAX, has no knowledge of the state of the world. Instead, we therefore say that for a node with m branches and therefore m payoff vectors $\vec{K}_1, \cdots, \vec{K}_m$ to choose between, the min function is defined as:

$$\min_i \vec{K}_i = (\min_i \vec{K}_i[1], \min_i \vec{K}_i[2], \cdots, \min_i \vec{K}_i[n])\,. \tag{3}$$

That is, the min function returns a vector in which the payoff for each possible world is the lowest possible. This models a MIN player who has *complete* knowledge of the state of the world, and uses this to choose the best branch in each possible world. As we pointed out in the previous section, this is the same assumption that is implicitly made by Monte-carlo sampling. We use the assumption again as it represents the most conservative approach: modelling the strongest possible opponents provides a lower bound on the payoff that can be expected when the opponents are less informed. Also, we shall see later that this assumption is actually used by human experts when analysing some imperfect information games.

As an example of vector minimaxing in practice, Figure 4 shows how the algorithm would analyse the tree of Figure 1, using ovals to represent the vectors produced at each node. The branches selected at MAX nodes by the max operator (assuming equally likely worlds) are highlighted in bold, showing that the right-hand branch is correctly chosen at node a.

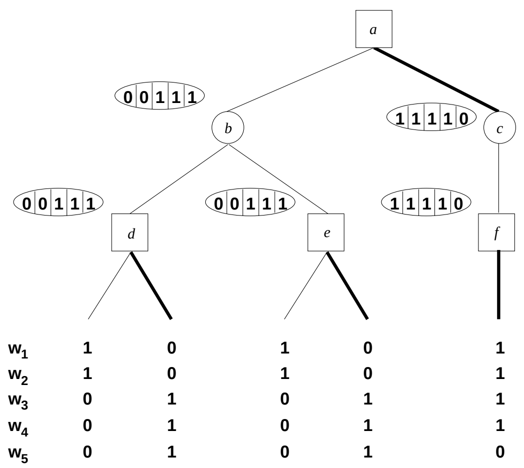

Figure 4: Vector minimaxing applied to example tree

Payoff-reduction Minimaxing

Consider Figure 5, which depicts a game tree with just three worlds. If we assume that MIN has complete knowledge of the world state in this game (as implicitly modelled by Monte-carlo sampling and vector minimaxing) the best strategy for MAX is to choose the left-hand branch at node d and the right-hand branch at node e. This guarantees a payoff of 1 in world w_1.

In the figure, however, we have annotated the tree to show how it is analysed by vector minimaxing. The branches in bold show that the algorithm would choose the right-hand branch at both node d and node e. The vector produced at node b correctly indicates that when MAX makes these selections, a MIN player who knows the world state will always be able to restrict MAX to a payoff of 0 (by choosing the left branch at node b in world w_1 and the right branch in worlds w_2 and w_3). Thus, at the root of the tree, both subtrees have the same analysis, and vector minimaxing never wins on this tree.

Applying Monte-carlo sampling to the same tree, in the limiting case where all possible worlds are examined, we see that node b has a minimax value of 1 in world w_1, so that the left-hand branch would be selected at the root of the tree. However, the same selections as vector minimaxing will then be made when subsequently playing at node d or node e. Thus, despite both modelling the situation where MIN has complete knowledge of the actual world state, neither Monte-carlo sampling nor vector minimaxing choose the best strategy against a MIN player with complete information on this tree.

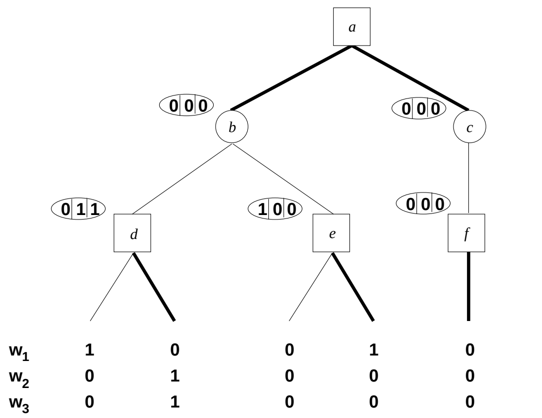

Figure 5: Example tree with three worlds

The difficulty here is that MIN can *always* restrict MAX to a payoff of 0 in worlds w_2 and w_3 by choosing the right-hand branch at node b. Thus, at node d the payoffs of 1 under the right-hand branch will never actually be realised, and should be ignored. Effectively, and perhaps counterintuitively, the analysis of node d is dependent on first correctly analysing node e. This is an example of how imperfect information games can not be solved by search algorithms that are 'compositional' (*i.e.*, algorithms that determine the best play at an internal node ν of a search space by analysing only the *local* subtree beneath ν). Such algorithms do not take into account information from other portions of the game tree. In particular, they do not recognise that under some worlds the play may never actually reach any given internal node, ν. At such nodes, they may therefore mistakenly select moves on the basis of high expected payoffs in world states that are in fact of no consequence at that position in the tree (as happens in Figure 5). This problem, which is more difficult to eliminate than strategy fusion, has been formalised as *non-locality* in (Frank & Basin 1998).

We propose here a new algorithm that lessens the impact of non-locality by reducing the payoffs at the frontier nodes of a search tree. As in the case of Monte-carlo sampling and vector minimaxing, the assumption in this algorithm is that MIN plays as well as possible in each individual world. However, this time we implement this assumption by reducing the payoff in any given world w_k to the maximum possible (minimax) return that can be produced when the game tree is examined as a single, complete information search tree in world w_k. The resulting algorithm, which we call *payoff-reduction minimaxing*, or *prm*, is shown in its simplest form in Figure 6 (it can be implemented more efficiently, for example by combining steps 2 and 3 together).

Algorithm $prm(t)$:
Identifies strategies for game trees, t

1. Conduct minimaxing of each world, w_k, finding for each MIN node its minimax value, m_k, in that world.
2. Examine the payoff vectors of each leaf node. Reduce the payoffs p_k in each world w_k to the minimum of p_k and all the m_k of the node's MIN ancestors.
3. Apply the *vector-mm* algorithm to the resulting tree.

Figure 6: Simple form of the *prm* algorithm

The reduction step in this algorithm addresses the problem of non-locality by, in effect, parameterising the payoffs at each leaf node with information on the results obtainable in other portions of the tree. By using minimax values for this reduction, the game-theoretic value of the tree in each individual world is also left unaltered, since no payoff is reduced to the extent that it would offer MIN a better branch selection at any node in any world.

As an example, let us consider how the algorithm would behave on the tree of Figure 5. The minimax value of node c is zero in every world, but all the payoffs at node f are also zero, so no reduction is possible. At node b, however, the minimax values in the three possible worlds are 1, 0, and 0, respectively. Thus, all the payoffs in each world at nodes d and e are reduced to at most these values. This leaves only the two payoffs of 1 in world w_1 as shown in Figure 7, where the strategy selection subsequently made by vector-minimaxing has also been highlighted in bold. In this tree, then, the *prm* algorithm results in the correct strategy being chosen. In the next section, we examine how often this holds in general.

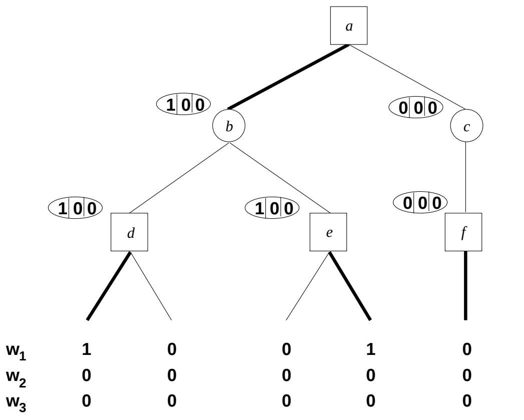

Figure 7: Applying *vector-mm* after payoff reduction

Experiments on Random Trees

We tested the performance of Monte-carlo sampling, vector minimaxing and payoff-reduction minimaxing on randomly generated trees. For simplicity, the trees we use in our tests are complete binary trees, with $n = 10$ worlds and payoffs of just one or zero. These payoffs are assigned by an application of the Last Player Theorem (Nau 1982), so that the probability of there being a forced win for MAX in the complete information game tree in any individual world is the same for all depths of tree.[2]

We assume that MAX has no information about the state of the world, so that each world appears to have the equally likely probability of $1/n$ (in our tests, $1/10$). For MIN's moves, on the other hand, we assume that for i ($0 \leq i < n$), the rules of the game allow MIN to identify the actual world state in i cases. In each of these (randomly selected) i worlds, MIN can therefore make branch selections based on the actual payoffs in that particular world, and will only require an expected value computation for the remaining $n - i$ worlds. We define the level of knowledge of such a MIN player as being $i/(n-1)$.

The graphs of Figure 8 show the performance of each of the three algorithms on these test game trees. These graphs were produced by carrying out the following

[2]The Last Player Theorem introduces a probability, p, that governs the assignment of leaf node payoffs as follows: if the last player is MAX, choose a 1 with probability p; if the last player is MIN, choose a 1 with probability $1-p$. For complete information binary trees with a MAX node at the root, (Nau 1982) shows that if $p = (3-\sqrt{5})/2 \approx 0.38197$ the probability of any node having a minimax value of 1 is constant for all sizes of tree ($1-p$ for MAX nodes, and p for MIN nodes). For lower values of p, the chance of the last player having a forced win quickly decreases to zero as the tree depth increases. For higher values, the chance of a forced win for the last player increases to unity.

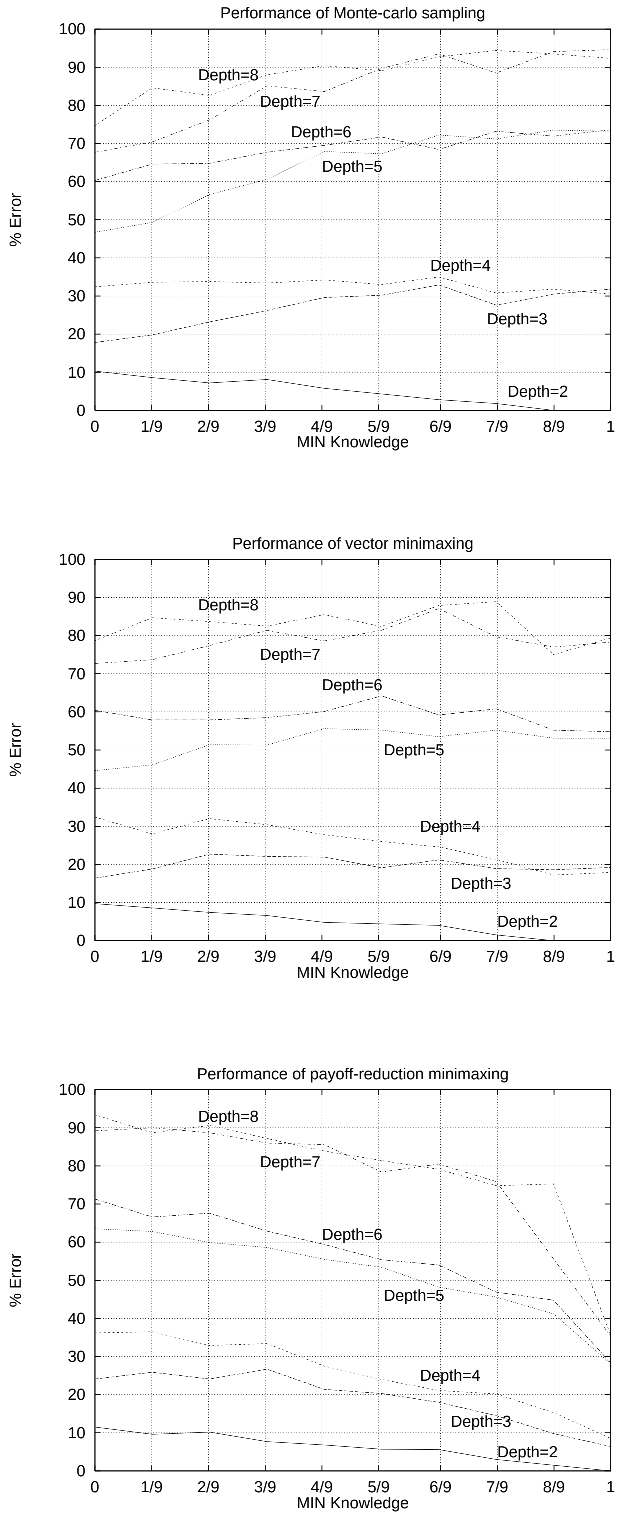

Figure 8: The error in Monte-carlo sampling, vector minimaxing and payoff-reduction minimaxing, for trees of depth 2 to 8

steps 1000 times for each data point of tree depth and opponent knowledge:

1. Generate a random test tree of the required depth.
2. Use each algorithm to identify a strategy. (For each algorithm, assume that the payoffs in *all* worlds can be examined.)[3]
3. Compute the payoff of the selected strategies, for an opponent with the level of knowledge specified.
4. Use an inefficient, but correct, algorithm (based on examining every strategy) to find an optimal strategy and payoff, for an opponent with the level of knowledge specified.
5. For each algorithm, check whether they are in error (*i.e.*, if any of the values of the strategies found in Step 3 are inferior to the value of the strategy found in step 4, assuming equally likely worlds).

Our results demonstrate that vector minimaxing out-performs Monte-carlo sampling by a small amount, for almost all levels of MIN knowledge and tree depths. This is due to the removal of strategy fusion. However, even when strategy fusion is removed, the problem of non-locality remains, to the extent that the performance of vector minimaxing is only slightly superior to Monte-carlo sampling. A far more dramatic improvement is therefore produced by the *prm* algorithm, which removes strategy fusion and further reduces the error caused by non-locality. When MIN has no knowledge on the state of the world, *prm* actually *introduces* errors through its improved modelling of the assumption that MIN will play as well as possible in each world. However, as MIN's knowledge increases, this assumption becomes more accurate, until for levels of knowledge of about 5/9 and above, the *prm* algorithm out-performs both Monte-carlo sampling and vector minimaxing. When MIN's knowledge of the world state is 1 the performance advantage of *prm* is particularly marked, with the error of *prm* for trees of depth 8 being just over a third of the error rate of Monte-carlo sampling.

To test the performance advantage of *prm* on larger trees, we extended the range of our tests to cover trees of depth up to 13 (the largest size that our algorithm for finding optimal solutions could handle), with the opponent knowledge fixed at 1. The results of this test are shown in Figure 9. When the trees reach depth 9, Monte-carlo sampling and vector minimaxing have error rates of 99.9% and 96%, whereas *prm* still identifies the optimal strategy in over 40% of the trials. For trees of depth 11 and over, where Monte-carlo sampling and vector minimaxing *never* find a correct solution, *prm* still performs at between 40% and 30%.

[3]Note that in general, examining all the payoffs may not be possible, but just as Monte-carlo sampling deals with this problem by selecting a subset of possible worlds, vector minimaxing and *prm* can also be applied with vectors of size less than n.

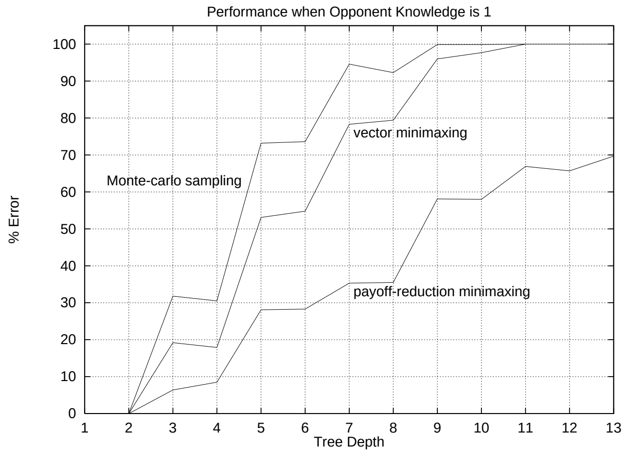

Figure 9: Superiority of payoff-reduction minimaxing, with opponent knowledge of 1, trees of depth up to 13

The ability to find optimal solutions when the opponents have full knowledge of the world state is highly significant in games with imperfect information. For instance, we have already pointed out that the payoff obtainable against the strongest possible opponents can be used as a lower bound on the expected payoff when the opponents are less informed. We have also noted that the other extreme, where MIN has *no* knowledge, is easily modelled (thus, it is not significant that all the algorithms in Figure 8 perform badly when MIN's knowledge is zero). Most significant of all, however, is that the assumption that the opponents know the state of the world is, in fact, made by human experts when analysing real games with imperfect information. We examine an example of this below.

Experiments on the Game of Bridge

As we mentioned earlier, Monte-carlo sampling has in fact been used in practical game-playing programs for games like Scrabble and Bridge. Bridge is of particular interest to us here as we have shown in previous work (Frank & Basin 1998) that expert analysis of single-suit Bridge problems is typically carried out under the *best defence* assumption that the opponents know the exact state of the world (*i.e.*, the layout of the hidden cards). Further, Bridge has been heavily analysed by human experts, who have produced texts that describe the optimal play in large numbers of situations. The availability of such references provides a natural way of assessing the performance of automated algorithms.

To construct a Bridge test set, we used as an expert reference the Official Encyclopedia of Bridge, published by the American Contract Bridge League (ACBL 1994). This book contains a 55-page section presenting optimal lines of play for a selection of 665 single-suit problems. Of these, we collected the 650 examples that gave *pure* strategies for obtaining the maximum possible payoff against best defence.[4] Using the FINESSE Bridge-playing system (Frank, Basin, & Bundy 1992; Frank 1996), we then tested Monte-carlo sampling, vector minimaxing and *prm* against the solutions from the Encyclopedia. In each case, the expected payoff of the strategy produced by the algorithms (for the maximum possible payoff) was compared to that of the Encyclopedia, producing the results summarised in Figure 10.

Algorithm	Correct	Incorrect
Monte-carlo sample	431 (66.3%)	219 (33.7%)
Vector minimaxing	462 (71.1%)	188 (28.9%)
The *prm* algorithm	623 (95.8%)	27 (4.2%)

Figure 10: Performance on the 650 single-suit Bridge problems from the Encyclopedia

As before, these results demonstrate that vector minimaxing is slightly superior to Monte-carlo sampling, and that the *prm* algorithm dramatically outperforms them both. In terms of the expected loss if the entire set of 650 problems were to be played once (against best defence and with a random choice among the possible holdings for the defence) the *prm* algorithm would be expected to lose just 0.83 times, compared to 16.97 and 12.78 times for Monte-carlo sampling and vector minimaxing, respectively.

The performance of *prm* was even good enough to enable us to identify five errors in the Encyclopedia (in fact, these errors could also have been found with the other two algorithms, but they were overlooked because the number of incorrect cases was too large to check manually). Space limitations prevent us from presenting more than the briefest summaries of one of these errors here, in Figure 11. In our tests, the line of play generated for this problem has a probability of success of 0.266 and starts by leading small to the Ten.

K T 8 x x

N
W E
S

J x x

For four tricks, run the Jack. If this is covered, finesse the eight next. Chance of success: 25%

Figure 11: Problem 543 from the Bridge Encyclopedia

More Experiments on Random Trees

To understand why the performance of all the algorithms is better on Bridge than on our random game trees, we conducted one further test. The aim of this experiment was to modify the payoffs of our game trees so that each algorithm could identify optimal strategies with the same success rate as in Bridge. We achieved this by the simple expedient of parameterising our trees with a probability, q, that determines how similar the possible worlds are. To generate a tree with n worlds and a given value of q:

- first generate the payoffs for n worlds randomly, as in the original experiment, then
- generate a set of payoffs for a dummy world w_{n+1},
- and finally, for each of the original n worlds, overwrite the complete set of payoffs with the payoffs from the dummy world, with probability q.

Trees with a higher value of q tend to be easier to solve, because an optimal strategy in one world is also more likely to be an optimal strategy in another. Correspondingly, we found that by modifying q it was possible to improve the performance of each algorithm. What was unexpected, however, was that the value of q for which each algorithm performed at the same level as in Bridge roughly coincided, at $q \approx 0.75$. For this value, the error rates obtained were approximately 34.1%, 31.5% and 6.1%, as shown in Figure 12. Thus, on two different types of game we have found the relative strengths of the three algorithms to be almost identical. With this observation, the conclusion that similar results will hold for other imperfect information games becomes more sustainable.

Efficiency Issues

All of the algorithms we have presented execute in time polynomial in the size of the game tree. In our tests, the *prm* algorithm achieves its performance gains with a small, constant factor, slowdown in execution time. For example, to select a strategy on 1000 trees of depth 13 takes the *prm* algorithm 571 seconds, compared to 333 seconds for vector minimaxing and 372 seconds for the Monte-carlo sampling algorithm (all timings obtained on a SUN UltraSparc II running at 300MHz). Over all depths of trees, the *prm* algorithm ranges between 1.08 and 1.39 times as expensive as our implementation of Monte-carlo sampling. Similarly for the Bridge test set, *prm* takes an average of 11.9 seconds to solve each problem, compared to 1.9 seconds for vector minimaxing and 4.1 seconds for Monte-carlo sampling.

However, the efficiency of the implementations was not our major concern for the current paper, where

[4]The remaining fifteen examples split into four categories: six problems that give no line of play for the maximum number of tricks, four problems involving the assumption of a *mixed* strategy defence, four for which the solution relies on assumptions about the defenders playing sub-optimally by not *false-carding*, and one where there are restrictions on the resources available.

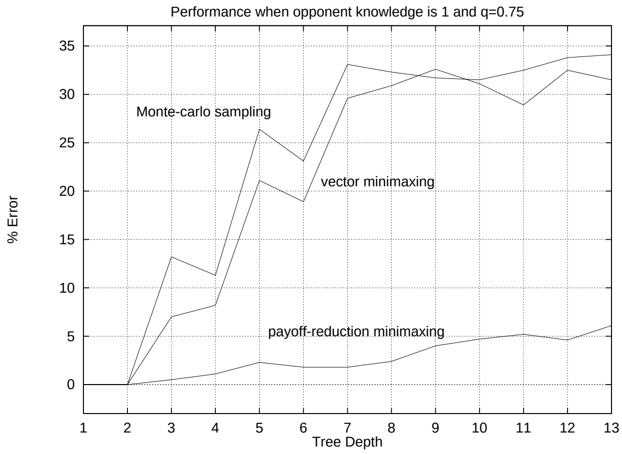

Figure 12: Superiority of payoff-reduction minimaxing on random game trees where the optimal strategy in one world is more likely to be optimal in another

we were interested instead in producing a qualitative characterisation of the relative strengths of the different algorithms. Thus, the data presented in this paper was obtained without employing any of the well-known search enhancement techniques such as alpha-beta pruning or partition search (Ginsberg 1996c).

Note, though, that it is possible to quite simply incorporate the alpha-beta algorithm into the vector minimaxing framework via a simple adaptation that prunes branches based on a pointwise $\geq$ (or $\leq$) comparison of vectors. Whether this kind of enhancement can improve the efficiency of *prm* to the point where it can tackle larger problems such as the full game of Bridge is a topic for further research. In this context, it is noteworthy that the 66.3% performance of Monte-carlo sampling in our single-suit tests correlates well with the results reported by (Ginsberg 1996a), where the technique was found to solve 64.4% of the problems from a hard test set of complete deals. Combined with the results of the previous sections, this extra data point strengthens the suggestion that the accuracy of *prm* will hold at 95% on larger Bridge problems.

Conclusions and Further Work

We have investigated the problem of finding optimal strategies for games with imperfect information. We formalised *vector minimaxing* and *payoff-reduction minimaxing* by discussing in turn how the problems of strategy fusion and non-locality affect the basic technique of Monte-carlo sampling. We tested these algorithms, and showed in particular that payoff-reduction minimaxing dramatically outperforms the other two, both on simple random game trees and for an extensive set of problems from the game of Bridge. For these single-suit Bridge problems, *prm*'s speed and level of performance was good enough to allow us to detect errors in the analysis of human experts.

The application of *prm* to larger, real-world games, as well as the further improvement of its accuracy, are important topics for further research. We are also investigating algorithms that solve weakened forms of the best defence model, for example taking advantage of mistakes made by less-than-perfect opponents.

References

ACBL. 1994. *The Official Encyclopedia of Bridge.* 2990 Airways Blvd, Memphis, Tennessee 38116-3875: American Contract Bridge League, Inc., 5th edition.

Blair, J.; Mutchler, D.; and Liu, C. 1993. Games with imperfect information. In *Games: Planning and Learning, 1993 AAAI Fall Symposium*, 59–67.

Corlett, R., and Todd, S. 1985. A Monte-carlo approach to uncertain inference. In Ross, P., ed., *Proceedings of the Conference on Artificial Intelligence and Simulation of Behaviour*, 28–34.

Frank, I., and Basin, D. 1998. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence.* To appear.

Frank, I.; Basin, D.; and Bundy, A. 1992. An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, 72–76.

Frank, A. 1989. Brute force search in games of imperfect information. In Levy, D., and Beal, D., eds., *Heuristic Programming in Artificial Intelligence 2.* Ellis Horwood. 204–209.

Frank, I. 1996. *Search and Planning under Incomplete Information: A Study using Bridge Card Play.* Ph.D. Dissertation, Department of Artificial Intelligence, Edinburgh. Also to be published by Springer Verlag in the Distinguished Dissertations series.

Ginsberg, M. 1996a. GIB vs Bridge Baron: results. Usenet newsgroup *rec.games.bridge.* Message-Id: <56cqmi$914@pith.uoregon.edu>.

Ginsberg, M. 1996b. How computers will play bridge. *The Bridge World.* Also available for anonymous ftp from dt.cirl.uoregon.edu as the file /papers/bridge.ps.

Ginsberg, M. 1996c. Partition search. In *Proceedings of AAAI-96*, 228–233.

Levy, D. 1989. The million pound bridge program. In Levy, D., and Beal, D., eds., *Heuristic Programming in Artificial Intelligence.* Ellis Horwood. 95–103.

Luce, R. D., and Raiffa, H. 1957. *Games and Decisions—Introduction and Critical Survey.* New York: Wiley.

Nau, D. S. 1982. The last player theorem. *Artificial Intelligence* 18:53–65.

Shannon, C. E. 1950. Programming a computer for playing chess. *Philosophical Magazine* 41:256–275.

von Neumann, J., and Morgenstern, O. 1944. *Theory of Games and Economic Behaviour.* Princeton University Press.