Monte-Carlo Sampling in Games with Incomplete Information: empirical investigation and analysis

Ian Frank
Complex Games Lab
Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305

David Basin
Institut für Informatik
Universität Freiburg
Am Flughafen 17
Freiburg, Germany

Hitoshi Matsubara
Complex Games Lab
Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305

Abstract

We investigate Monte-carlo sampling in games with incomplete information. We show that for very simple game trees the chance of finding the optimal strategy with Monte-carlo sampling rapidly approaches zero as the number of moves in the game increases. We explain this sub-optimality by identifying the different kinds of errors that can arise and by analysing their interplay. We also relate our test results to real games, suggesting why the error rates observed in practice may not be so high.

1 Introduction

We examine the use of Monte-carlo sampling [Corlett and Todd, 1985] in games with incomplete information. In such games, the actual state of the game may be unknown; for example, some playing pieces may be hidden, some of the playing area may not be visible, or the outcome of some moves may not be known. Monte-carlo sampling of such games involves guessing a 'likely' state of the game (*e.g.*, what the hidden pieces might be, where the hidden pieces might be, or what the opponent's previous moves might have been) and then finding a solution to this complete information sub-problem. The hope is that by examining a representative sample of the set of possible game states (called *possible worlds* below) an action that works well in a large percentage of them can be identified. A Monte-carlo sampling approach has been used in 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].

This paper describes some empirical tests carried out on simple games which take the form of complete binary trees with payoffs of either '1' or '0' at the leaf nodes. In complete information games, the game-theoretic value of such trees can be simply found using the minimax algorithm or its more efficient extensions. To model incomplete information, we introduce not one but N payoffs at each leaf node, to correspond to the value of the leaf node in each of N possible worlds. Monte-carlo sampling on such trees is then conducted by carrying out minimax searches on individual worlds. We examine the performance of this technique under different assumptions about the amount of knowledge held by the opponent on the actual state of the world. In particular, we answer the question 'How often will such algorithms play a game *perfectly*', by examining how often the optimal strategy is overlooked.

This research was motivated by our own work on a Bridge-playing program [Frank *et al.*, 1992; Frank, 1996], and in particular in trying to understand why our algorithms gave results that differed from the analyses presented in expert Bridge texts. Our empirical work here is an examination of the kinds of problems that can arise in games like Bridge, reduced to a simplified setting that is more amenable to analysis and requires no Bridge experience to appreciate. However, to make the link back to more complex 'real' games, we show that our simple test trees can be parameterised in a way that enables the observed rate of error to be manipulated to the levels found in other games.

2 Game Trees used for Testing

We begin by describing in detail the games we use for our tests. Our approach is to represent two-player games as complete binary trees whose leaf nodes record information on the outcomes of the game under different possible worlds. In this setting, the level of information held by each player can be given a well-defined meaning. The leaf node payoffs are chosen to offer each player an even chance of winning.

2.1 Extensive Form

A test game tree is produced by first generating a complete binary tree. It is assumed that this tree represents a 2-player game in which the players always alternate. The player whose play we try to optimise (MAX) goes first, and the opponent (MIN) goes second.

Our modelling of incomplete information is very simple, and is best explained with the help of an example. Consider the tree of Figure 1, which is in *extensive form* (our terminology in this section is standard in game theory: see, for example, [Luce and Raiffa, 1957; Fudenberg and Tirole, 1995]). This tree represents a

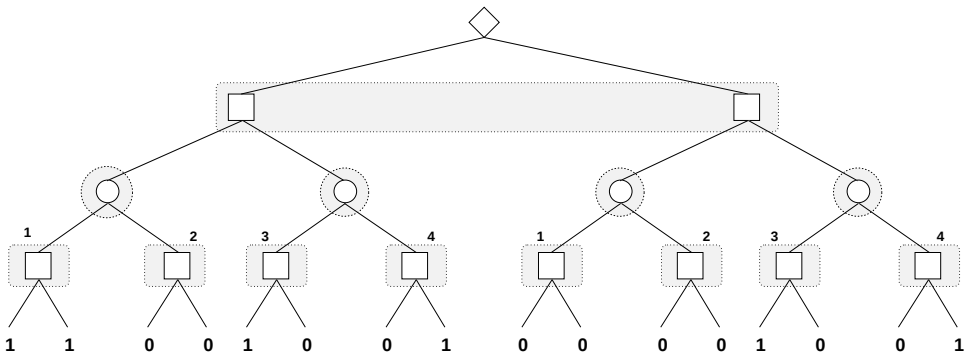

Figure 1: Game in extensive form

game between MAX (square nodes) and MIN (circular nodes) in which a chance node is also present (diamond node). At the leaf nodes of the tree are payoffs. A payoff of 1 corresponds to MAX achieving some goal, such as winning the game or receiving a certain return, whereas a payoff of 0 corresponds to MAX failing in that goal.

Also portrayed in this figure are the *information sets* of each player. The purpose of these sets is to indicate what, within the rules of the game, each player can know when they make a choice at a node. As we mentioned in the Introduction, it may be that the rules of the game hide information on the pieces in the game or on the previous moves of the opponent. The information sets capture this feature of the game by grouping together nodes which will be indistinguishable for each player. For example, consider the kind of game that the information sets of Figure 1 describe. The first level of MAX nodes in the tree are contained in the same information set. Since MAX cannot distinguish between these nodes, he must be unaware of the actual branch followed at the chance node at the root of the tree. Each of the MIN nodes, however, are in *separate* information sets, so the game rules must give MIN knowledge of the actual path taken to reach these nodes (*i.e.*, both the outcome of the initial chance move and the move selected by MAX). Finally, the moves at the next MAX layer are grouped into four information sets, (numbered 1, 2, 3 and 4). The only element of the path to the nodes in each of these sets that differs is the choice of the branch at the root of the tree. Again, it is therefore the outcome of the first chance move of which MAX is unaware.

2.2 Possible Worlds

We will use game trees that fit the pattern of Figure 1, where the outcome of each player's moves is known to the other and there is one chance move, that occurs at the beginning of the game. Let us say that there are N possible outcomes of this chance move. We will say that each of these outcomes determines a possible *world state* or *world* in which the play takes place. We will then represent the possible worlds in the vertical dimension as differing payoffs at the leaf nodes rather than in the horizontal dimension as different subtrees for each world. For example, the tree of Figure 1 can be represented instead as in Figure 2.

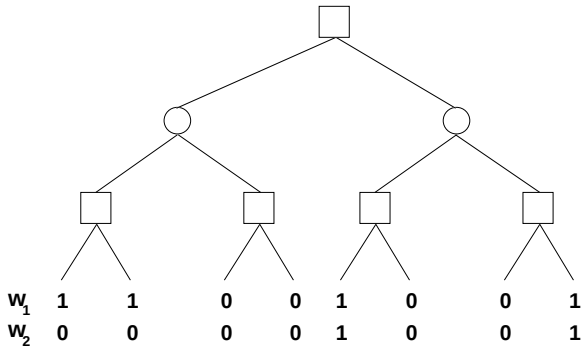

Figure 2: A simplified game tree with 2 worlds

For a more general game with N possible worlds, each leaf node of the game tree will have N payoffs (either 1 or 0) which correspond to the utility for MAX of reaching that node in each of the N worlds. We assume that MAX has no information about the state of the world so that for each MAX move the best that can be done is to consider the best *expected* outcome over all worlds. For MIN's moves, however, our tests will look at different assumptions about the level of knowledge of the opponent. Specifically, we will examine the consequences of gradually increasing MIN's knowledge from the same level as MAX up to complete information[1]. We will do this by assuming that for n ($0 \leq n < N$), MIN's information sets contain $N - n$ nodes. The value of n determines the number of outcomes of the chance move for which the rules of the game allow MIN to see the actual result. In each of these (randomly selected) n worlds, MIN can

[1]An opponent with complete information may seem unrealistic, but this assumption is in fact common in expert analysis of Bridge, where it forms part of the model of 'best defence' (see, *e.g.*, [Frank, 1996]).

therefore make branch selections based on the best payoffs in that particular world, and will only require an expected value computation for the remaining $N - n$ worlds. We define the level of knowledge of such a MIN player as being $n/(N-1)$.

2.3 Assignment of Payoffs

The 1s and 0s at the leaf nodes of our test trees are assigned so that the complete information game tree in any individual world can be won by MAX with a probability of 1/2. This is done by an application of the Last Player Theorem [Nau, 1982]. This theorem introduces a probability, p, which determines the chance of selecting a 1 at the leaf nodes of a tree as follows: if the tree is of odd depth, choose a 1 with probability p, but if the tree is of even depth, choose a 1 with probability $1 - p$. For binary trees with a MAX node at the root, [Nau, 1982] gives us that MAX has a 50% chance of a win if $p = (3 - \sqrt{5})/2 \approx 0.38197$.

3 Monte-carlo Sampling

How can Monte-carlo sampling be used to select MAX's next move in the incomplete information games described in §2? Let us consider an arbitrary MAX node in a tree with N possible worlds, and assume that it is possible to choose an $n \leq N$ such that when we randomly generate n worlds, $\mathsf{w}_1, \cdots, \mathsf{w}_n$, we have sufficient computing resources to find the minimax value of the complete information game tree rooted on the MAX node under each of these worlds. If there are m possible moves, $M_1, \cdots, M_m$, to choose between and we use e_{ij} to denote the minimax value of the ith possible move under world w_j, the situation in this world will be as depicted in Figure 3.

Each move, M_i, can then be given a score based on its expected payoff. For example, in the context of Figure 3, the scoring function, f, could be defined:

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

Selecting a move is achieved by actually using the minimax algorithm to generate the values of the e_{ij}s, and determining the M_i for which the value of $f(M_i)$ is greatest. Other scoring functions f can be imagined, but we will assume this simple form, which corresponds to selecting the move that has the largest expected return. Note that a Monte-carlo sampling approach with this scoring function was the technique suggested by [Levy, 1989] for the game of Bridge.

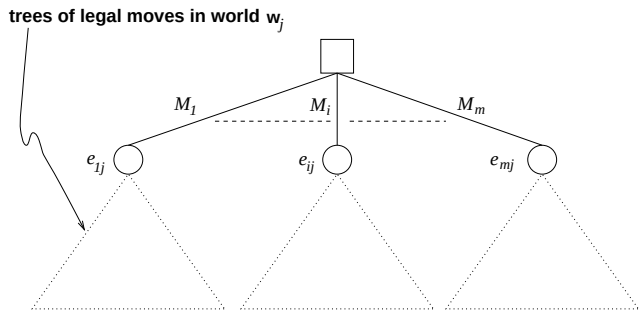

Figure 3: Producing the minimax values, e_{ij}, of each move M_i under world w_j

3.1 A Single-pass Monte-carlo Algorithm

Rather than simply selecting individual moves, we are interested in how the Monte-carlo sampling approach performs on an *entire* game. This could be investigated by playing a game from start to finish using Monte-carlo sampling to select MAX moves, and random selection to generate MIN moves. However, measuring the performance against the *optimal* MIN play would then require a large number of trials. Instead, we make use of the following simple observation: when Monte-carlo sampling is employed to find a move at the root of a game tree, all the information necessary to select moves at the other nodes in the tree is also (indirectly) computed. To utilise this information, we construct an algorithm that can analyse a game tree in a single pass.

To illustrate this algorithm, let us refer back to the game tree of Figure 2. Rather than labelling the leaf nodes with payoffs in each world separately as we did before, let us use a single vector, $\vec{K}$, consisting of one element for the payoff in each of the two possible worlds. In Figure 4 these vectors are represented by ovals which contain the possible payoffs in worlds w_1 and w_2.

By analysing this tree with a minimax-like algorithm that manipulates vectors of numbers (rather than individual numbers) we can carry out Monte-carlo sampling in a single pass. To do this, the definition of the functions min and max is first extended to cover sets of m payoff vectors $\vec{K}_1, \cdots, \vec{K}_m$:

$$\min_i \vec{K}_i = (\min_i \vec{K}_i[1], \min_i \vec{K}_i[2], \cdots, \min_i \vec{K}_i[m])\,, \qquad (2)$$

$$\max_i \vec{K}_i = (\max_i \vec{K}_i[1], \max_i \vec{K}_i[2], \cdots, \max_i \vec{K}_i[m])\,, \qquad (3)$$

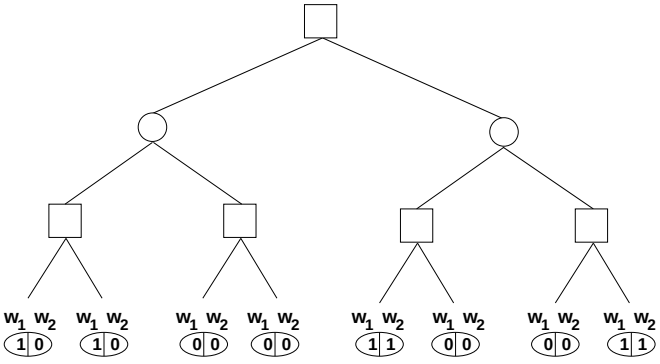

Figure 4: Payoffs represented as 'vectors' of outcomes

where $\vec{K}_i[j]$ is the jth element of the vector $\vec{K}_i$. That is, the min function returns a vector in which the payoff for each world is the lowest possible, and the max function returns a vector in which the payoff for each world is the highest possible.

If the min function is used at MIN nodes, and the max function is used at MAX nodes, the minimax value of each world is backed up the tree simultaneously. For example, the tree of Figure 4 would be analysed as shown in Figure 5.

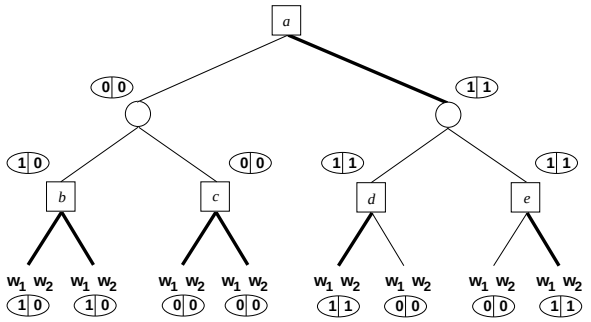

Figure 5: Monte-carlo sampling using vectors

In this figure, we have shown the vectors that are produced at each node. At the root, we have also indicated (in bold) the branch that would be selected by Monte-carlo sampling. In general, the branch selection at MAX nodes can be determined by an updated version of (1) that evaluates the payoff vector, $\vec{K}$, associated with each possible move, M_i. For example:

$$f(M_i) = \sum_{j=1}^{N} \Pr(\mathsf{w}_j)\vec{K}[j]\,, \qquad (4)$$

where $\Pr(\mathsf{w}_j)$ represents MAX's assessment of the probability of the actual world state being w_j. (Note that in all our experiments, we will assume that when a player is unaware of the actual outcome of the chance move, he assigns an equal probability to each possible world.)

3.2 Identifying a Strategy

The real gain of the single-pass approach is that it can be used to identify the MAX selections made by Monte-carlo sampling at *every* MAX node in the game tree. We have illustrated this in Figure 5 by marking the branches that would be selected using (4) at nodes b, c, d, and e (at nodes b and c, (4) gives the same score to each branch, so the choice between them is made at random.)

For a binary tree with a MAX node at the root and l levels of MAX nodes in total, there is a total of $2^0 + 2^2 + 2^4 + \cdots + 2^{2(l-1)} = (4^l - 1)/3$ MAX nodes in the tree. However, the number of *different* combinations of branch selections in the tree is actually lower than 2 raised to this power. This is because specifying a given branch selection at some node, ν, means that nodes beneath any remaining branches from ν will never be reached when the game is played. For example, in the tree of Figure 5 it is not necessary to specify what actions must be taken at nodes b and c once the right-hand branch has been chosen at node a.

A specification of just the branch selections at nodes that can be reached during the play of a game is equivalent to the formal notion of a *strategy* (see, for example [Luce and Raiffa, 1957]). The formula for the number of strategies in a binary tree with a MAX node at the root and l levels of MAX nodes in total is 2^{2^l-1}. This is smaller than the set of possible branch selections, but it still very quickly becomes large. We have therefore written an algorithm (briefly described in the Appendix) that identifies the optimal strategy from among these possibilities without enumerating them exhaustively. Below, we examine, for different levels of the opponent's knowledge, how often the optimal strategy for a randomly generated game tree is superior to the strategy selection made by Monte-carlo sampling.

4 Results and Interpretation

The best possible performance of Monte-carlo sampling (*i.e.*, when *all* the possible worlds are examined) on our binary game trees is illustrated in Figure 6. To create this figure, we carried out the following loop 1000 times for each data point of tree depth and opponent knowledge:

1. Generate a random test tree of the required depth, as described in §2.
2. Use Monte-carlo sampling to identify a strategy (examining all possible worlds).
3. Check the payoff of this strategy, for opponents with the level of knowledge specified.
4. Use the correct but computationally expensive algorithm (described in the Appendix) to find an optimal strategy, for opponents with the level of knowledge specified.
5. Check whether Monte-carlo sampling is in error (*i.e.*, if the value of the strategy found in Step 3 is inferior to the value of the strategy found in step 4, under the assumption of equally likely worlds).

The basic conclusion to be drawn from this graph is inescapable: whatever the level of knowledge of the opponent, as the depth of the game tree rises, the error in Monte-carlo sampling rapidly approaches 100%. In our tests, the difference between the expected return of the optimal strategy and the expected return of the strategy selected by Monte-carlo sampling approaches 0.1 as the trees get larger. Thus, if Monte-carlo sampling were to be used to repeatedly play random games, it would have a success rate of about 90%.

To help identify the reasons for this sub-optimality, we present a simplified version of Figure 6 in which the actual structure of the plots is easier to appreciate. In the new graph, shown in Figure 7, we have plotted on a single 2D plane just the curves for trees of depths 2 to

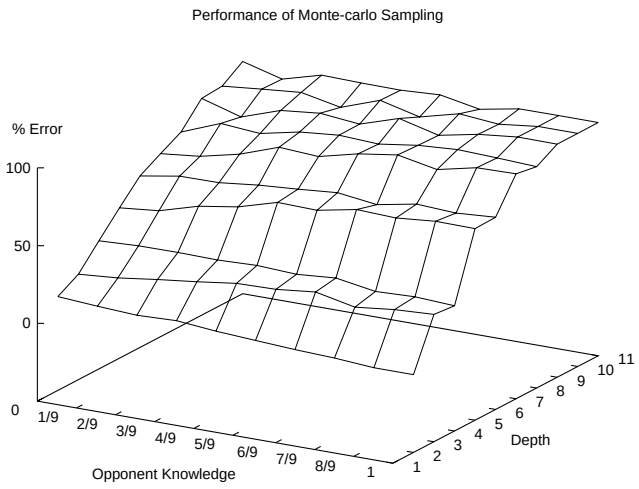

Figure 6: The percentage of binary trees with 10 possible worlds for which Monte-carlo sampling selects a suboptimal strategy, plotted for trees of depth between 2 and 11 (y-axis), and for varying levels of knowledge held by the opponent (x-axis, ranges from zero to complete knowledge). One thousand tests per data point.

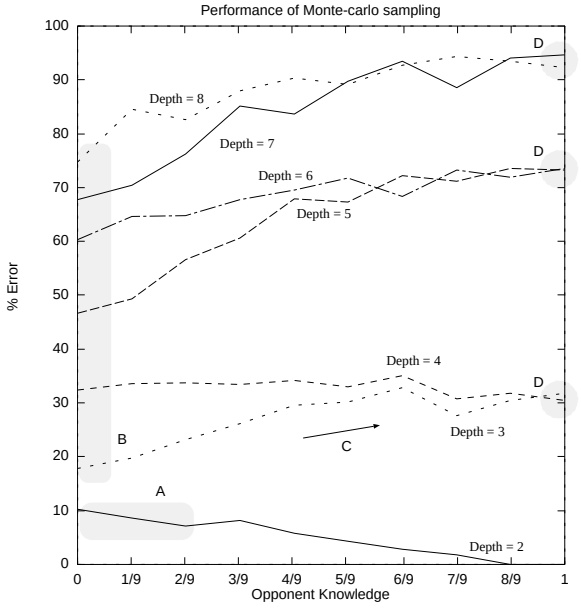

Figure 7: A simplified 2D graph showing the error in Monte-carlo sampling just for trees of depth 2 to 8. Specific features of this graph explained in the text are labelled A, B, C and D.

8, omitting the higher values for the sake of clarity. We have also annotated four distinct features of the graph that require explanation:

A For trees of depth 2, why does Monte-Carlo sampling select incorrect strategies when the opponent has zero knowledge of the world state (and then improve as the opponent becomes more informed)?

B Why does the error gradually increase as the game tree gets larger (at least for an opponent with zero knowledge of the world state)?

C For trees of every size other than 2, why does an increase in the opponent's knowledge result in an increase in error?

D Why does it appear that for odd d, the error rates for trees of depth d and depth $d+1$ converge towards the same answer as the opponent's knowledge increases?

We examine each of these questions in the four subsections that follow.

4.1 A — MIN's Knowledge

Here we explain the first annotated region on the graph of Figure 7. To do this, we will appeal to the simple example game tree shown in Figure 8.

Consider what happens when Monte-carlo sampling examines each world in this game tree. In w_1, minimising the leaf node payoffs gives node b an evaluation of 1 and node c an evaluation of 0. For w_2, however, minimising the payoffs gives both nodes a payoff of 0. Similarly, for w_3 and w_4, both nodes have a payoff of zero. Full Monte-carlo sampling will therefore lead to the selection of the left-hand branch at node a. We have indicated this selection in the figure by drawing the branch in bold, and have also depicted the vectors (representing the minimax values in each world) produced at each MIN node.

Now let us go back the feature marked A in Figure 7. At the point where the 'opponent knowledge' is zero, MIN's knowledge about the actual state of the world is the same as that of MAX, *i.e.*, he only knows that there are four equally likely possibilities and his information sets contain four nodes. MIN must therefore make the same move selection in every world. What moves will

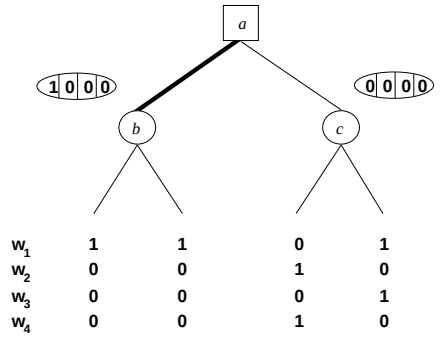

Figure 8: Simple tree of depth 2

such a MIN player choose in the tree of Figure 8? At node b his choice is immaterial since MAX's payoff is the same for each branch in every world. At node c, however, MIN can choose between giving MAX a payoff of 1 in just w_2 and w_4, or a payoff of 1 in just w_1 and w_3. In either case, the expected payoff for MAX at node c is 0.5 whereas the expected payoff at node b is just 0.25. MAX should therefore take advantage of the lack of information held by such a MIN player and select the right-hand branch at the root of the tree, instead of the branch selected by Monte-carlo sampling.

This example shows clearly that the form of the plot for trees of depth 2 in Figure 7 is due to the model of the opponent's knowledge implicit in Monte-carlo sampling. Specifically, when Monte-carlo sampling uses minimisation to find the value of a MIN node in some world w_i, an actual MIN player will only be able to guarantee producing the same result if he knows that w_i is the actual state of the world. With less than complete information, MIN will actually base his move on the *expected* payoff.

In terms of the information sets described in §2.1, a MIN player with no knowledge of the world state must view each of the nodes b and c as an information set of four nodes (one for each world) at which the same move must be made. Effectively, allowing a different move in each world is equivalent to allowing MIN to choose a different *strategy* in each world. We therefore call this problem *strategy fusion* (for a precise formalisation, see [Frank, 1996; Frank and Basin, 1997b]). When strategy fusion affects the analysis of MIN moves they will appear stronger for MIN than they actually are. Thus, as in the example of Figure 8, MAX may be misled into choosing sub-optimal moves. Of course, as MIN's actual knowledge about the world state increases, the model represented by the Monte-carlo method becomes more and more accurate and strategy fusion gradually disappears. This explains why the error rate for trees of depth 2 gradually decreases in Figure 7.

4.2 B — MAX's Knowledge

Next, consider the tree of Figure 9, where we have shown the vectors produced by full Monte-carlo sampling, and marked the branches that would be selected in bold. Although the left-hand branch has been selected at the root, it should be clear that the right-hand branch is the superior move: the payoff produced by the selected moves in the left-hand subtree is only 1 in the worlds w_1, w_2, and w_3, whereas the payoff produced by the moves in the other subtree is also 1 in world w_4.

Again, the source of difficulty here is strategy fusion, but this time it is MAX's moves that are affected instead of MIN's. When Monte-carlo sampling backs up the best payoff in each world at a MAX node, it effectively assumes that *different* moves can be chosen in *different* worlds. Collecting the minimax values of these moves and assuming that they represent the actual payoffs that can be expected ignores the fact that a MAX with incomplete information must make the *same* move in every world. Allowing MAX to choose a different strategy in each world is again strategy fusion.

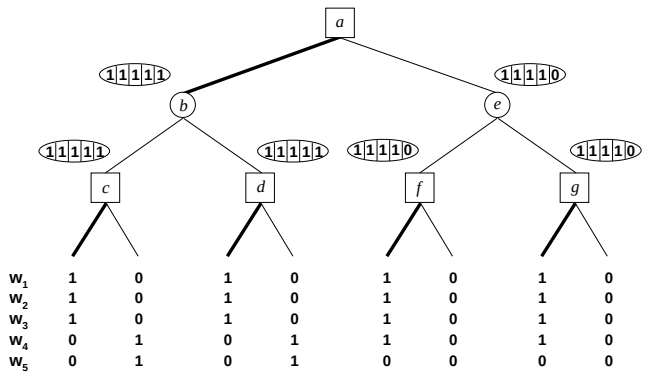

Figure 9: Simple game tree of depth 3

How does this relate to our graph of Figure 7? We have already seen in the previous section how strategy fusion leads Monte-carlo sampling to make errors at MIN nodes because it assumes that MIN has more knowledge than he sometimes does. Now, we have seen how strategy fusion also affects MAX nodes, because of the assumption that MAX has more knowledge than he actually does. Thus, the effect of strategy fusion grows with both the number of MIN levels (at least at the left-hand side of the graph), and also with the number of MAX levels. This explains the form of the region marked as B in Figure 7.

Note that another way to visualise the problem of strategy fusion is to think of Monte-carlo sampling as actually modelling the task of selecting between some number of *games*, each starting with the same chance move, in which *both* players have perfect information. For example, imagine that the subtrees rooted on nodes b and e in Figure 9 represent the MIN and MAX moves in two separate games, each starting with a chance move that selects one of the possible worlds $\mathsf{w}_1, \cdots, \mathsf{w}_5$. Which of these games would we rather play if the outcome of the chance move (and all other moves) is known to both players? It should be clear that the answer to this question is that we can always win a game based on node b, but that we will expect to lose the game based on node e one in five times (whenever the chance move selects w_5). However, it should also be clear that the situation modelled by this question is different from the original game, in which MAX (who actually has no knowledge of the chance move) and MIN (who is uncertain about the outcome of the chance move whenever his knowledge is less than 1) must try to find moves that work well across a number of worlds, rather than working well in just one.

4.3 C — Non-locality

Here, we tackle the question of why an increase in the opponent's knowledge increases the error in Monte-carlo sampling (for trees of depth greater than 2). To do this, we will consider the example tree of Figure 10. This figure is a slightly modified version of Figure 9 from the previous section, with the payoffs under node d altered.

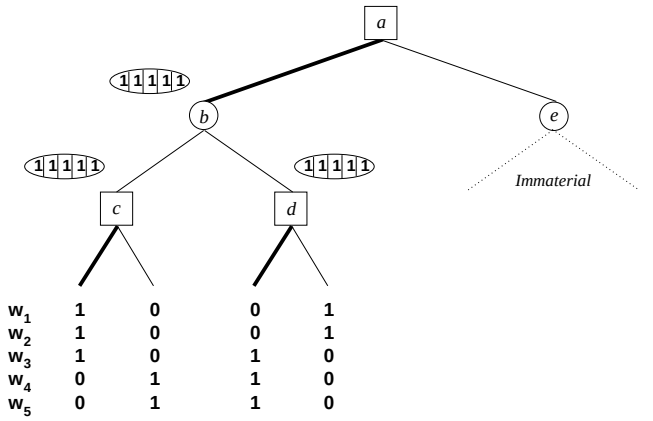

Figure 10: Simple game tree illustrating 'non-locality'

As before, strategy fusion will cause Monte-carlo sampling to have a strong preference for selecting the left-hand branch at the root of the tree. However, as well as the chance that the optimal move might actually be to direct the play to node e, there is now the new problem that even when node b is chosen Monte-carlo sampling doesn't guarantee making the correct branch choices when playing at the subsequent nodes c and d.

To see this, consider how a MIN player with perfect information will play at node b. In world w_1, he will select the right-hand branch, since MAX's choice of branch at node d will then lead to a payoff of zero. Similarly in w_2, if MIN selects the right-hand branch at node b, MAX will get a payoff of 0. In world w_3, MAX will get a payoff of 1 no matter which branch MIN chooses, but in worlds w_4 and w_5 MIN can again restrict MAX's payoff to 0, this time by picking the left-hand branch at node b. Thus, against a MIN player with complete knowledge, MAX's branch selections produce a payoff of 1 in just one world: world w_3. It is easy to check that there are better alternatives. Choosing the left-hand branch at node c and the right-hand branch at node d produces a payoff of 1 in both w_1 and w_2. Also, choosing the right-hand branch at node c and the left-hand branch at node d produces a payoff of 1 in both w_4 and w_5.

The problem here is distinct from that of strategy fusion, and can be traced to a different cause: the way in which a branch selection is made at a node on the basis of an evaluation only of its direct subtree. The inherent assumption in making a branch selection in this way is that the correct move is a function only of the possible continuations of the game. In perfect information situations (*i.e.*, where the position in the game tree is known), this assumption is justified and the minimax algorithm, with its compositional evaluation function, finds optimal strategies for such games. With more than one possible world, however, this assumption is no longer valid. For instance, at node c in our example, the left-hand branch appears to be the best choice because it produces a payoff of 1 in three out of the five possible worlds. However, as we described above, making this selection at node c allows a MIN with knowledge of the actual world state to restrict MAX to a payoff of 0 in w_4 and w_5. This affects the analysis of node d, since at any node below node b the maximum attainable payoff in world w_4 and w_5 is then zero. Under this circumstance, it is the right-hand branch that is the best choice at node d, since it offers a payoff of 1 in two worlds (w_1 and w_2), compared to the single payoff of 1 (in world w_3) offered by the left-hand branch. If we consider making a branch selection at node c after choosing a branch at node d, we find similarly that the best selection is no longer the one which leads to a payoff of 1 in most worlds.

In general, the choice of a branch at a given MAX node ν is not simply a function of the payoffs of the paths that contain ν, but of the payoffs along *any* path in the tree. If MIN can choose a move at an ancestor of ν that reduces the payoff (in any world) from what MAX would expect from examining ν's direct subtree then the best branch at ν may change. We call this problem of having to consider all other nodes in the tree *non-locality* (again, for a precise formalisation, see [Frank, 1996; Frank and Basin, 1997b]). Non-locality clearly depends both on the number of MAX levels in the game tree and also on the level of knowledge of the opponent. This explains why the trend for trees of depth greater than two is for the error to increase with the knowledge of the opponent (trees of depth two do not suffer from non-locality in our tests, because the problem only arises at MAX nodes which have MIN ancestors).

4.4 D — The Probability of Winning

Finally, we look at the convergence in Figure 7 of the error rates for trees of odd depth d and depth $d+1$. For trees of odd depth, the final layer of non-terminal nodes are MAX nodes. As we explained in §2.3, we use the Last Player Theorem [Nau, 1982], to set the probability of a 1 at the terminal nodes this game tree at $p = (3-\sqrt{5})/2 \approx 0.38197$. Consider now the effect of adding an extra MIN layer to such a tree (see Figure 11). Again, by the Last Player theorem, the chance of assigning a 1 at the new terminal nodes is $1-p$.

For this new tree, and considering just this world, we can calculate the probability that the evaluation of either of the MIN nodes will be 1. Since MIN can be expected

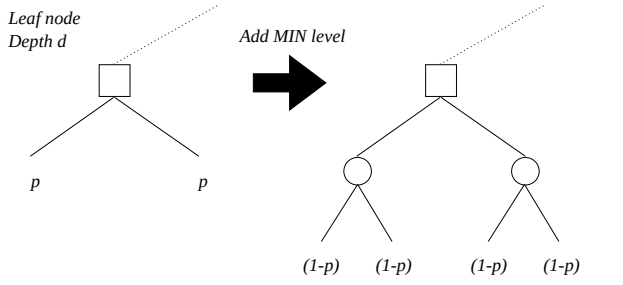

Figure 11: The probability of assigning a '1' at leaf nodes

to back up the smallest value from the leaf nodes of the tree, a 1 will only be produced if *both* payoffs at the leaf nodes are 1. This has a probability of $(1-p)^2$. It is not hard to verify that for the value of p given by the Last Player Theorem, this is equal to p.[2]

Thus, in terms of an individual world, adding an extra MIN level does not change much about the character of the game. However, with multiple worlds, there is a critical difference. The probabilities in the game tree are only preserved if MIN can actually choose the best branch in each world. As we saw in §4.1, if MIN has incomplete information on the actual world state, then choosing a branch based on the expected overall return may well turn out to be sub-optimal in individual worlds.

The above observations allow us to tie up our description of the graph of Figure 7 very neatly. So far, we have identified the following problems with Monte-carlo sampling:

A There is an implicit assumption that MIN has complete knowledge. The strategy fusion errors caused by this assumption increase with the number of MIN levels in the tree, but decrease as the knowledge of the opponent increases.

B There is an implicit assumption that MAX has complete knowledge. The strategy fusion errors caused by this assumption increase with the number of MAX levels in the tree, but are unaffected by the knowledge of the opponent.

C The problem of non-locality arises when playing against an opponent with knowledge of the world state. The error caused by non-locality increases with the number of MAX levels in the tree and also with the knowledge of the opponent.

It is now easy to see why the plots for a tree of odd depth d and the tree of depth $d+1$ converge. At the far left of the graph, the extra layer of MIN nodes in the tree of depth $d+1$ leads to an increased incidence of strategy fusion errors at MIN nodes caused by the assumption of complete MIN knowledge (A). Both the problems of strategy fusion at MAX nodes (B) and non-locality (C), however, depend on the number of MAX levels in the tree, so from this perspective the trees are identical. Thus, as we move right along the x-axis and the assumption about the MIN knowledge becomes more accurate, the stability of the probabilities discussed above leads the plots for depth d and depth $d+1$ to converge.

[2]This is the essence of the Last Player Theorem; as values are backed up the tree, the chance that any branch at a MAX node has an evaluation of 1 remains at p. Similarly, at any MIN node the chance that a branch has an evaluation of 1 remains at $1-p$. To see this, consider what happens when backing up values at a MAX node. The evaluation of the MAX node is 1 with probability $1-Pr(\text{both branches have an evaluation of 0}) = 1-(1-p)^2$. For the value of p given to us by the Last Player Theorem, this is equivalent to $1-p$. For lower values of p, the chance of a 1 decreases quickly to zero as the tree depth increases. For higher values, the chance of a 1 increases to unity.

5 Comparison with Real Games

The figures in the previous section are slightly at odds with experimental results in real games. Notably, in the incomplete information game of Bridge, [Frank, 1996; Frank and Basin, 1997b] have found that for the subproblem of single suits non-locality leads to just 33.5% sub-optimality. More recently, Ginsberg has reported that a fast Monte-carlo approach [Ginsberg, 1996c] is sub-optimal on 35.6% of complete Bridge deals [Ginsberg, 1996a]. Thus, there is evidence that the 100% sub-optimality we have witnessed on our binary trees does not occur in practice.

The experience with Bridge suggests that a measure is needed for incomplete information games that describes game properties that have a bearing on the difficulty of solving the game with a Monte-carlo approach. Designing such a measure will be a topic of our future research, but here we describe briefly a first step in this direction, which suggests the plausibility of the task.

Parameterising Similarity

Consider a measurement based on 'similarities' between worlds. For our binary trees, we modify the way that the payoffs of 1 and 0 are assigned at the leaf nodes so that winning strategies in one world are also more likely to win in another. We achieve 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 normally, as described in §2.1, then
- generate a set of payoffs for a dummy world w_{N+1}, then
- for each of the original N worlds, overwrite the complete set of payoffs with the payoffs from the dummy world, with probability q.

For parameterised trees with 10 worlds, we repeated the original test of §4 for values of q ranging from $q = 0.00$ to $q = 0.95$, in steps of 0.05. The results of this experiment are shown in the graph of Figure 12. It should be clear from this graph that as q increases, the performance of Monte-carlo sampling also improves. To illustrate more clearly the effects of changing the parameter q, we have also plotted in Figure 13 the error rate of Monte-carlo sampling just for trees of depth 11.

The graphs of Figure 12 and Figure 13 illustrate a remarkably uniform decrease in error as q increases. From Figure 13 we can read off a value of q for which the error rate is approximately 35% as $q \approx 0.70$. A natural next step for this testing would involve constructing a general metric for incomplete information games that can be measured in practice and used to predict how hard such games will be to solve with a Monte-carlo approach. As a test for accuracy, this metric should produce the same value for Bridge as for our binary game trees with a value of $q = 0.7$.

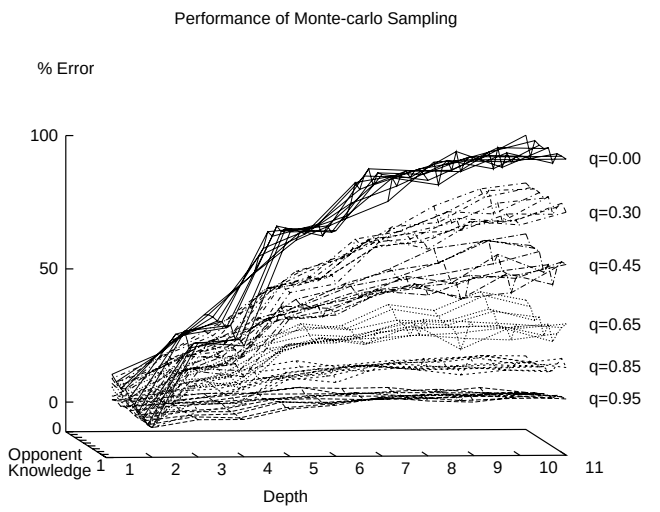

Figure 12: The error surface of Monte-carlo sampling on binary trees with 10 possible worlds for different values of q. The plot is for trees of depth between 2 and 11 (y-axis), for varying levels of knowledge held by the opponent (x-axis, ranges from zero to complete knowledge). One thousand tests per data point.

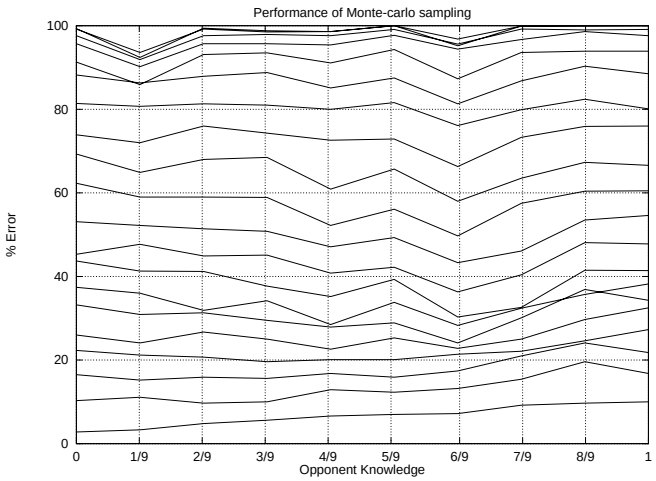

Figure 13: A cross-section through the plane of the graph of Figure 12 where depth=11. Each line represents a different value of q, ranging from 0.95 at the bottom to 0.0 at the top, in steps of 0.05.

6 Conclusions

We have examined the performance of Monte-carlo sampling on simple binary game trees, demonstrating that as the depth of the game tree increases, the observed error rapidly approaches 100%. We explained the sources of these errors in terms of strategy fusion (caused by the implicit assumptions of complete MIN and MAX knowledge made by Monte-carlo sampling), and non-locality. We also related our results to real games such as Bridge, showing that it is possible to adjust our test game trees to produce error rates comparable to that seen in practice.

One natural continuation for this research would be to continue the program sketched in §5 and design a metric that predicts the difficulty of producing the correct analysis of an incomplete information game using a Monte-carlo approach. Another continuation that we have already pursued is the design of a tractable algorithm that out-performs Monte-carlo sampling. Using the insights gained from the analysis of §4, we have formulated a new algorithm, *payoff-reduction minimaxing*, which does not suffer from strategy fusion at MAX nodes and also significantly reduces the incidence of non-locality. This research is reported in [Frank and Basin, 1997a].

Appendix: Finding the Best Strategy

Here we describe the algorithm we use to find the optimal strategy on our test game trees. The essence of this algorithm is that instead of selecting a single branch at any MAX node, the result of *each* possible selection is calculated and backed up the tree. Since all combinations of branch selections (that form strategies) are considered, the best expected payoff at the root identifies an optimal strategy.

We will illustrate this algorithm by considering again the example game tree of Figure 10, originally used to illustrate the effects of non-locality. Let us consider just the subtree rooted on node b, and label the payoff vector at each leaf node with a template to represent a *partial strategy* which will become instantiated as the tree is processed (see Figure 14). The entries in these templates represent the MAX choices at the nodes c and d respectively, as either '1' (left branch), '2' (right branch) or '_' (immaterial).[3]

At node c, MAX has two choices (either the left or the right-hand branch). We therefore raise the two vectors which are at the leaf nodes and store them at node c. Also, the two partial strategy templates are filled in to indicate which vector is produced by selecting the left-hand branch, and which is produced by selecting the right. Similarly, there are two possible vectors for MAX to select between at node d.

At MIN nodes, we have to analyse MIN's possible actions in response to each of the possible combinations of choices that MAX may make beneath that node. In our example, this can be done by looking at each of the possible combinations of the vector/partial strategy pairs that label the MAX nodes c and d. For example, let us consider how to combine together the choices represented by the partial strategies $(1, \_)$ and $(\_, 1)$ (corresponding to choosing the left-hand branch at nodes c and d). MIN's

[3]Note that strategies for general trees can be constructed by, for example, forming larger n-tuples which correspond to the possible strategies in the game by virtue of the ith element representing the choice to be made at the ith MAX encountered during a pre-order traversal of the tree.

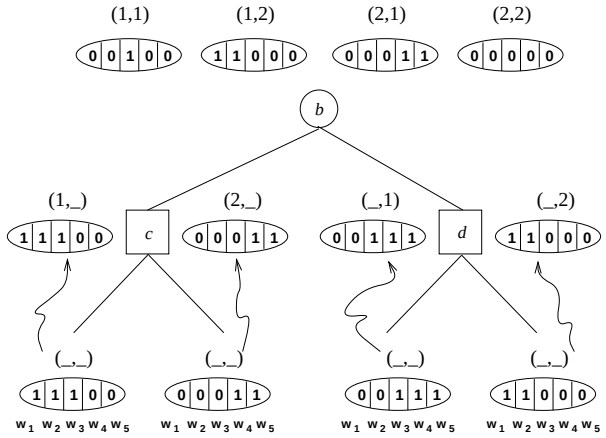

Figure 14: Annotating each node with a 'vector' describing the outcomes of the possible strategies

desire to minimise MAX's payoff is modelled by selecting the smallest payoff in each possible world for which MIN is aware of the outcome of the chance move. In the current example, we will assume that MIN's knowledge is complete, so that the equation (2) we introduced in §3.1 can be used to find the smallest value in every world (in general, MIN may have to use an expected value computation for worlds where he does not know the outcome of the chance move). This results in a new vector with a single 1 in world w_3. The identification of the strategy represented by this vector is found by combining together the partial strategies to produce $(1,1)$.

Thus, we can label node b with an oval representing the payoff for the strategy $(1,1)$. The combination procedure is then repeated for the other possible choices of vectors at nodes c and d, producing annotations at node b for the strategies $(1,2)$, $(2,1)$ and $(2,2)$, as shown in Figure 14. Strategy $(1,1)$ (the strategy chosen by Monte-carlo sampling) is found to give a payoff of 1 in just world w_3, strategy $(1,2)$ is found to give a payoff of 1 in just worlds w_1 and w_2, *etc.*

This example illustrates the basic nature of our algorithm. At MAX nodes, each vector/partial strategy pair is backed up. At MIN nodes, each possible combination of vector/partial strategy pairs is backed up. The exact formalisation will depend on the method used to represent and build the strategies, which we omit here.

For a general tree, the number of the vectors present at any MAX or MIN node is the same as the number of strategies in the subtree rooted on that node. As we noted in §3.2, this number can get very large very quickly (it is doubly exponential in the number of MAX levels in the tree). However, note that if the collection of vectors at a node contains members which are pointwise less than or equal to any other member at that node, these vectors may be ignored as inevitably giving rise to inferior strategies. For example, the vector containing entirely zeros at node b may be omitted from further consideration, as there are other vectors at the same node that offer at least as good a payoff in every world. Prunings of this sort lead to an improvement in performance that is sufficient to allow us to carry out our experiments.

References

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

[Frank and Basin, 1997a] I. Frank and D. Basin. Payoff-reduction minimaxing. In *Proceedings of the Fourth Game Programming Workshop in Japan (GPW-97)*, Hakone, Japan, 1997. To appear.

[Frank and Basin, 1997b] I. Frank and D. Basin. Search in games with incomplete information: A case study using bridge card play. Technical Report ETL-97-7, Electrotechnical Laboratory, 1997.

[Frank *et al.*, 1992] I. Frank, D. Basin, and A. Bundy. An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, pages 72–76, Vienna, Austria, 1992.

[Frank, 1989] A. Frank. Brute force search in games of imperfect information. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 204–209. Ellis Horwood, 1989.

[Frank, 1996] I. Frank. *Search and Planning under Incomplete Information: A Study using Bridge Card Play.* PhD thesis, Department of Artificial Intelligence, Edinburgh, 1996.

[Fudenberg and Tirole, 1995] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, 1995.

[Ginsberg, 1996a] M. Ginsberg. GIB vs Bridge Baron: results. Usenet newsgroup *rec.games.bridge*, 31 October 1996. Message-Id: <56cqmi$914@-pith.uoregon.edu>.

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

[Ginsberg, 1996c] M. Ginsberg. Partition search. In *AAAI-96*, pages 228–233, 1996.

[Levy, 1989] D.N.L. Levy. The million pound bridge program. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 95–103. Ellis Horwood, 1989.

[Luce and Raiffa, 1957] R. Duncan Luce and Howard Raiffa. *Games and Decisions—Introduction and Critical Survey.* Wiley, New York, 1957.

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