Submitted to the Fourth Game Programming Workshop in Japan. Accepted with minor revisions.

TR-97-21
Received 25 Jul 97

Payoff-reduction Minimaxing

Ian Frank

Electrotechnical Laboratory
Machine Inference Group
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305

David Basin

Institut für Informatik
Universität Freiburg
Am Flughafen 17
Freiburg, Germany

Hitoshi Matsubara

Electrotechnical Laboratory
Machine Inference Group
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305

Abstract

We describe *payoff-reduction minimaxing*, a new search algorithm for games with incomplete information. We formalise this algorithm and present results that show it dramatically outperforms the major competing algorithm: Monte-carlo sampling. On simple game trees where Monte-carlo sampling is *never* able to identify an optimal strategy, payoff-reduction minimaxing has a success rate of up to 40%. Payoff-reduction minimaxing is designed for incomplete information games in which the opponent is assumed to know the state of the world. This is exactly the situation found in expert analysis of the game of Bridge. We analyse further simple game trees on which Monte-carlo sampling has same success rate as it does for Bridge (about 65%). On these trees, payoff-reduction minimaxing has the even better success rate of 93%.

Payoff-reduction Minimaxing

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 describe *payoff-reduction minimaxing*, a new search algorithm for games with incomplete information. We formalise this algorithm and present results that show it dramatically outperforms the major competing algorithm: Monte-carlo sampling. On simple game trees where Monte-carlo sampling is *never* able to identify an optimal strategy, payoff-reduction minimaxing has a success rate of up to 40%. Payoff-reduction minimaxing is designed for incomplete information games in which the opponent is assumed to know the state of the world. This is exactly the situation found in expert analysis of the game of Bridge. We analyse further simple game trees on which Monte-carlo sampling has same success rate as it does for Bridge (about 65%). On these trees, payoff-reduction minimaxing has the even better success rate of 93%.

1 Introduction

In games with incomplete information, the actual 'state of the world' 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. Finding the optimal strategy in such games is NP-hard [Blair *et al.*, 1993]. Thus, a heuristic approach will be required to efficiently solve any practical game with incomplete information. In previous work [Frank *et al.*, 1997a] we have examined the heuristic of Monte-carlo sampling [Corlett and Todd, 1985], showing that, for very simple game trees, the chance of selecting the correct strategy rapidly approaches zero as the size of the tree increases. Here, we introduce a new algorithm, *payoff-reduction minimaxing*, that produces better results than Monte-carlo sampling without sacrificing efficiency. On simple game trees where Monte-carlo sampling is *never* able to identify an optimal strategy, payoff-reduction minimaxing has a success rate of up to 40%.

This research was motivated by our own work on a Bridge-playing program [Frank *et al.*, 1992; Frank, 1996]. Payoff-reduction minimaxing is therefore designed for incomplete information games in which, as in expert analysis of Bridge, the opponent is assumed to know the state of the world. However, our test results also show that payoff-reduction minimaxing out-performs Monte-carlo sampling when the opponent is less well-informed. The full test results are presented in §5. Before this, §2 describes the incomplete information games we use as our examples, §3 describes the difficulties of identifying optimal strategies in such games, and §4 shows how these difficulties are addressed by the payoff-reduction minimaxing algorithm.

2 Game Theory Background

Here we introduce some background necessary to make the paper self-contained. In particular, we describe how we represent two-player games with incomplete information. Such games will form the basis of the examples and the test problems in the remainder of the paper.

2.1 Extensive Form

We consider two-player games between a player whose actions we try to optimise (MAX) and an opponent (MIN). Our modelling of incomplete information in this setting 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 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. The higher the payoff, the better the result for MAX. The lower the payoff, the better the result for MIN.

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 briefly 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 described by the information sets of Figure 1. 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 therefore 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 actual path taken to reach these nodes (*i.e.*, both

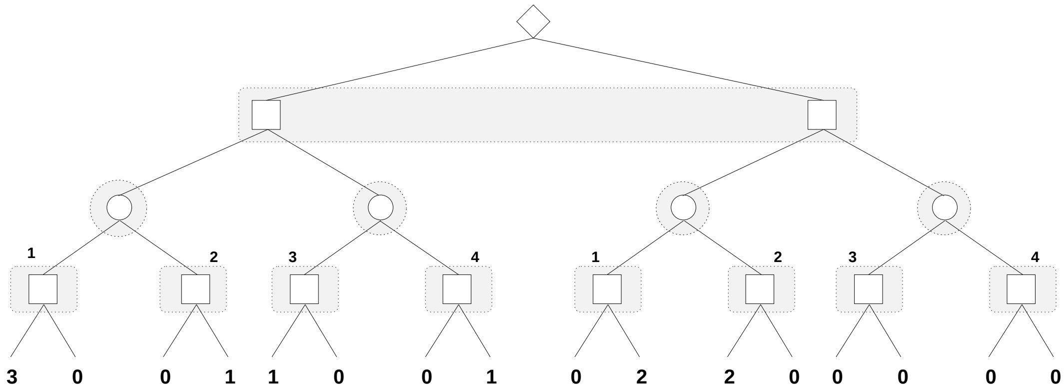

Figure 1: Game in extensive form

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

In the rest of this paper, 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.

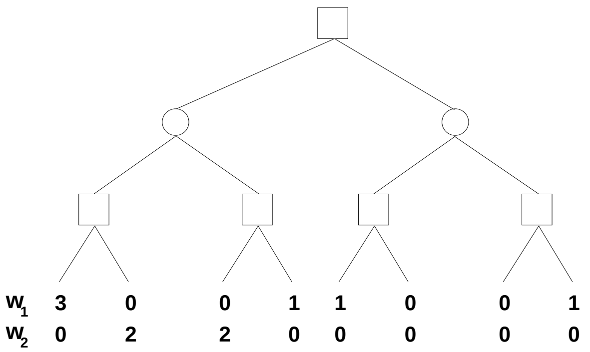

Figure 2: A simplified game tree with 2 worlds

For a more general game with N possible worlds, each leaf node of the game tree will have N payoffs 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, we are particularly interested in the situation where MIN always knows the exact state of the world. In our tests in §5, however, we will also consider the case where MIN has less knowledge. 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 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 $n/(N-1)$.

3 Identifying a Strategy

In this section we examine how an optimal *strategy* can be selected for the trees described above. We explain the actual notion of a strategy, and show the difficulties in identifying an optimal strategy in games with incomplete information. Since our original motivation for this research comes from the game of Bridge, we measure the strength of each strategy under the assumption that MIN has complete knowledge of the state of the world. We have shown elsewhere [Frank, 1996] that this assumption is commonplace in the expert analysis of Bridge problems.

3.1 What is a Strategy?

A (pure) *strategy* for a player in a game is a specification of a decision for each possible situation that might arise in the play of the game (see, for example [Luce and Raiffa, 1957, Page 51]).

If we consider again the tree of Figure 2, we can represent the possible MAX strategies as 5-tuples that determine the choice to be made at each of the five MAX nodes. For example, in Figure 3, we have numbered the MAX nodes in the order in which they would be encountered during a pre-order traversal of the tree. A 5-tuple in which the ith element represents the choice to be made at the node numbered i (effectively, the ith information set) can then be used to represent a MAX strategy.

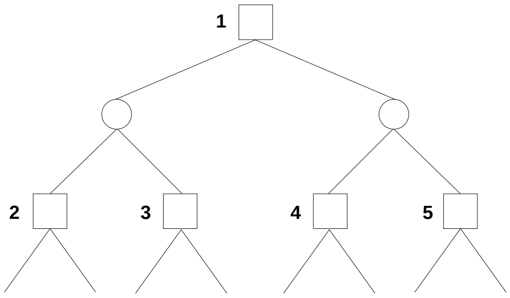

Figure 3: Nodes labelled in order of pre-order traversal

However, note that the number of *different* strategies is not 2^5, but 8. 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 our tree of Figure 2 if we choose the right-hand branch at the root, it is not necessary to specify what actions must be taken at the MAX nodes in the left-hand subtree of the root. If we indicate such superfluous branch selections by an underscore, and use '1' and '2' to denote the selection of a 'left' and a 'right' branch respectively, the set of possible strategies is:

Left branch at root	Right branch at root
(1,1,1,_,_)	(2,_,_,1,1)
(1,2,1,_,_)	(2,_,_,2,1)
(1,1,2,_,_)	(2,_,_,1,2)
(1,2,2,_,_)	(2,_,_,2,2)

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} (see [Frank, 1996]).

3.2 Assessing Strategies

In Figure 4 we have repeated a portion of the example tree of Figure 2, displaying just the left subtree at the root. There are four possible strategies for this tree, which we will call $(1,1)$, $(1,2)$, $(2,1)$ and $(2,2)$. The strategy $(1,1)$ has been highlighted in bold.

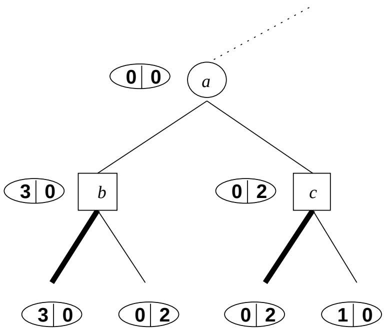

Figure 4: Subtree of Figure 2, showing strategy $(1,1)$

To make it easier to represent the payoffs offered by the strategy $(1,1)$, we have also labelled the nodes with vectors, $\vec{K}$, consisting of one element for the payoff in each of the two possible worlds. In the figure, these vectors are represented by ovals which contain the possible payoffs in worlds w_1 and w_2 respectively. By selecting the left-hand branch at node b, MAX will get a payoff of 3 in w_1 and a payoff of 0 in w_2 whenever the play reaches this node. Similarly, at node c MAX gets payoffs of 0 and 2. To assess the overall strength of the strategy $(1,1)$, we now need to examine what happens at node a. A MIN player with knowledge of the actual world state (as we are assuming) will choose the right-hand branch if the world is in state w_1, to give MAX a payoff of zero. If the world is in state w_2, he will choose the left-hand branch, to again give MAX a payoff of zero. The strategy $(1,1)$ therefore has a payoff of zero in both the possible worlds (against a MIN player with complete knowledge of the world state).

The remaining strategies on this tree can be assessed in the same way as above. For instance, Figure 5 shows the strategy $(2,1)$. For this strategy, MAX will get a payoff of 2 in w_2 no matter which branch MIN chooses. This strategy is therefore superior to the strategy $(1,1)$. In fact, this strategy optimal for this game tree (given

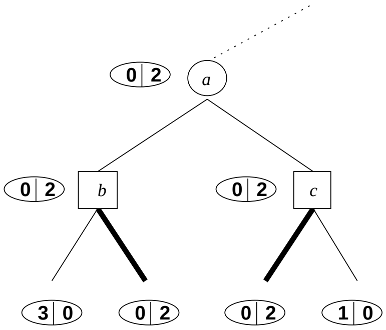

Figure 5: Subtree of Figure 2, showing strategy $(2,1)$

that w_1 and w_2 are equally likely), since the strategy $(1,2)$ produces a 1 in w_1 and a 0 in w_2 and the strategy $(2,2)$ produces a 0 in both possible worlds.

3.3 Non-locality

Consider analysing the game tree of Figure 4 by examining in turn the nodes b and c. If we assume that each of the two possible worlds is equally likely, what is the best choice of branch to make at node b? Since the left-hand branch offers a payoff of 3 in just w_1 (an expected payoff of $3/2$) and the right-hand branch offers a payoff of 2 in just w_2 (an expected payoff of $2/2$), the left-hand branch appears to be the best choice. Similarly, at node c the left-hand branch appears to be the best choice. Thus, by analysing the MAX nodes in turn we have selected the strategy $(1,1)$.

However, recall from the previous section that the optimal strategy for the game tree of Figure 4 is not $(1,1)$, but $(1,2)$. Thus, analysing each node in turn fails to find the optimal strategy. This is because when we examine node b, we are misled by the payoff of 3 in w_1 into thinking that the left-hand branch is a good selection for MAX. However, in w_1, MIN can reduce MAX's payoff by choosing the right-hand branch at node a. The *actual* payoff that can be achieved at node b in w_1 is therefore less than 3. Realising this, however, calls for an examination of node c. In [Frank and Basin, 1997; Frank, 1996] we have formalised this problem, naming it *non-locality* to reflect the way that the best choice at any (internal) node in the game tree depends not only on the node's local subtree, but also on all the other nodes in the tree. In [Frank *et al.*, 1997b], we have shown that Monte-carlo sampling suffers from non-locality. It is to tackle the effects of non-locality that we now develop the payoff-reduction minimaxing algorithm.

4 Payoff-reduction Minimaxing

In the previous section we showed how non-locality makes it impossible to choose the correct action at any internal node of (an incomplete information) game tree by just analysing the node's local subtree. Here, we show how this problem can be ameliorated by supplementing a local branch selection function with extra information about the situation in the remainder of the tree. These considerations lead directly to the formalisation of the payoff-reduction minimaxing algorithm. Payoff-reduction minimaxing is developed under the continued assumption of an opponent with complete information, but our test results (§5) show that in fact the algorithm also performs well when the opponent is less well-informed.

4.1 Naïve Minimaxing

As a starting point for our development, Consider the algorithm of Figure 6. This algorithm formalises the strategy selection process we used informally when discussing non-locality in §3.3. It therefore suffers from non-locality, and for reasons we describe below we call it *naïve minimaxing*. It identifies strategies for a tree t, when $node(t)$ returns the root node of t and $sub(t)$ computes the set of immediate subtrees of $node(t)$.

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

Condition	Result
t is leaf node	$\vec{K}(t)$
$node(t)$ is a MIN node	$\min_{t_i \in sub(t)}$ *naïve-mm*(t_i)
$node(t)$ is a MAX node	$\max_{t_i \in sub(t)}$ *naïve-mm*(t_i)

Figure 6: Naïve form of a minimax algorithm for trees with incomplete information

Here, the normal min and max functions are extended so that they are defined over a set of payoff vectors. At 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:

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

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 possible world is the lowest possible. This is equivalent to allowing MIN to choose any branch under any possible world.

The max function, on the other hand, returns the single vector with the highest expected payoff. That is, in a game with N possible worlds, it returns the vector, $\vec{K}_i$, for which

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

is maximum, where $\Pr(\mathsf{w}_j)$ represents MAX's assessment of the probability of the actual world being w_j.

Thus, *naïve minimaxing* commits to just *one* choice of branch at each MAX node, while allowing MIN to choose different branches in different worlds (as in our example of §3.3). In fact, we can see from this formalisation exactly how the problem of non-locality arises. In (1) MIN can choose different branches in each world. However, since the algorithm is recursive, the MIN choices at any node will always be made *after* MAX's choices in the subtrees of that node. When analysing any MAX node, then, the choices to be made at its MIN ancestors will not yet have been decided, and the possible worlds under which the play will reach the node will therefore be unknown. MAX's evaluation in (2) ignores this problem, simply taking the *naïve* approach of assuming that *every* world is possible. Below, we therefore look at how MAX can use some knowledge of MIN's likely moves to improve on these naïve branch selections.

4.2 Doublethink

One way to take account of MIN's choices is as follows: each time the min equation (1) selects a particular branch as the best MIN move in world w_j, 'annotate' the alternative branches at that node which are *not* selected (*i.e.*, those with higher evaluations in this world) with the label $\neg\mathsf{w}_j$. For any internal MAX node, ν, the subset of possible worlds under which ν will be reached can then be found by excluding all the worlds annotated on the MIN branches which occur in the path from ν to the root.

This suggests a two-phase algorithm which attempts to use such *inferences* to improve MAX's move selections. Whilst carrying out an initial naïve minimaxing of the game tree, each MIN branch is annotated with the worlds under which MIN would *not* select it, as described above. Then, a second pass through the tree is made, in which these annotations are taken into account when making selections at MAX nodes. MAX's new formula for choosing a vector at a node where $\mathcal{W}$ represents the *exclusion set* of the indices of the possible world annotations in the path connecting it to the root node is:

$$\sum_{\substack{j=1 \\ j \notin \mathcal{W}}}^{N} \Pr(\mathsf{w}_j)\vec{K}[j] \,. \quad (3)$$

The indices in the exclusion set address the problem of non-locality by allowing MAX to infer some information about the possible situations under which any node may be reached. The process can thought of as conducting an extra reasoning step which is akin to *doublethinking* MIN.

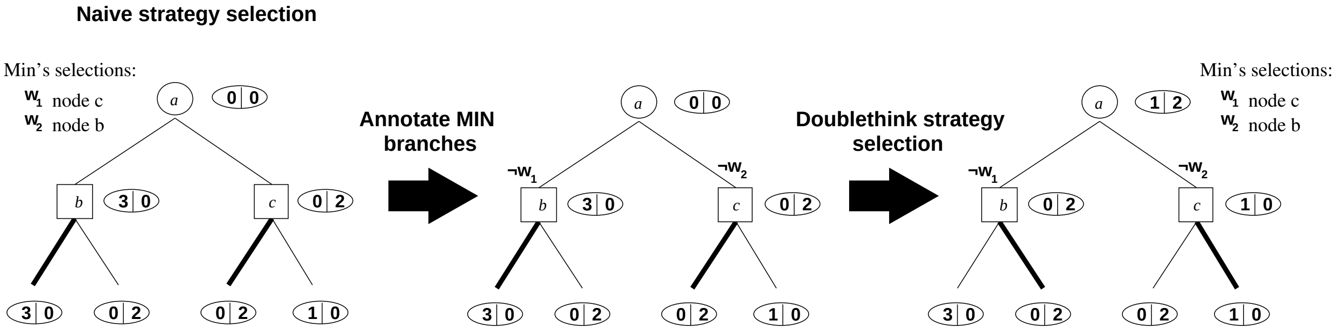

Figure 7: Carrying out doublethink

As an example, consider again the tree of Figure 4. As we have seen already, the naïve algorithm of Figure 6 selects the left-hand branches at nodes b and c, which produces a payoff of zero in both worlds. Allowing MAX a doublethink reasoning step incorporating the branch annotations representing MIN's plays leads MAX to change his selection at both nodes as shown in Figure 7. If MIN is not allowed to change his branch selections after this doublethink stage, MAX will expect a payoff of 1 in world w_1 and a payoff of 2 in world w_2.

4.3 N-think

Of course, if we allow MAX to doublethink MIN, we have in fact made MAX's strategy selection conditional on both the local evaluation of a node and the normal strategy that MIN would select. Another layer of reasoning can then be added, taking the MAX strategy resulting from the doublethink step and allowing MIN to re-evaluate the best choices at each MIN node — a step which could be described as *triplethink*.

To see that MIN's choices may change when applying triplethink, consider again a MAX node at which, in the naïve algorithm, MAX finds the best choice to be branch a. If this is the case, then

$$\forall i.\ i \neq a \Rightarrow \sum_{j=1}^{N} \vec{K}_a[j]\,\Pr(\mathsf{w}_j) \quad > \quad \sum_{j=1}^{N} \vec{K}_i[j]\,\Pr(\mathsf{w}_j)\,. \quad (4)$$

Let us say that when conducting doublethink the exclusion set for the node is found to be $\mathcal{W}$, and the move which achieves the maximum evaluation (using the modified assessment produced by (3)) changes to branch b, *i.e.*:

$$\sum_{\substack{j=1\\ j\notin\mathcal{W}}}^{N} \vec{K}_a[j]\,\Pr(\mathsf{w}_j) \quad < \quad \sum_{\substack{j=1\\ j\notin\mathcal{W}}}^{N} \vec{K}_b[j]\,\Pr(\mathsf{w}_j)\,. \qquad (5)$$

For this to be true, $\vec{K}_a[j] < \vec{K}_b[j]$ must hold for at least one value of $j \notin \mathcal{W}$. And, if this and (4) are both to be true, $\vec{K}_a[j] > \vec{K}_b[j]$ must be true for at least one value of $j \in \mathcal{W}$. Therefore, MAX's doublethink has the effect of reducing the payoff in at least one of the worlds represented by the exclusion set, and increasing the payoff in one of the remaining worlds.

Now, if it happens that the reduction of the payoff for a world in the exclusion set takes its value below the previous minimum to which MIN could restrict MAX at that node, allowing MIN an extra step of reasoning *will* lead him to go down the branch in this world, despite MAX's inference that he will not. MAX's doublethink inferences, therefore, can be used by MIN to actually *reduce* his payoff in certain worlds, if triplethink is allowed.

Similarly, the increase in payoff for a world not in the exclusion set may be a false improvement for MAX, since if the increase takes the payoff above the minimum payoff of its siblings, MIN will no longer select the node in this world, preferring instead to restrict MAX's payoff by selecting one of the other branches. Thus, if MIN is allowed to conduct another level of reasoning, the worlds present in the exclusion set of a node may no longer be correct. Also, worlds which should be present may be absent. To remedy this, and to be sure of making the best moves, MAX would need to conduct another level of reasoning about MIN's strategy.

In general, however, it is not easy to see where this chain of reasoning will terminate. In fact, it is relatively simple to construct situations in which successive levels of reasoning will loop forever through a finite set of choices. For instance, let us return to our example of Figure 7. In this example, we allowed MAX to use doublethink to change his strategy. But advantage for MAX is also disadvantage for MIN. If we allow a further, triplethink step (see Figure 8), the payoff which MAX was expecting from his new strategy is revealed to be over-optimistic; instead of achieving a 1 in w_1 and a 2 in w_2, he achieves zero in both. Two more stages of reasoning then bring us back to the same situation as in the original naïve scheme.

An n-think algorithm could be constructed using this approach, conducting some number of 'thinking' steps (possibly with a check for looping) and then selecting the best strategy from amongst those encountered. However, notice that our example demonstrates that the process of adding extra reasoning steps may terminate *before* the optimal strategy is examined; the two strategies encountered are $(1,1)$ and $(2,2)$ which, as we saw in §3.2, are both inferior to the two strategies that remain.

4.4 Unconditional Inferences

There are in fact two types of inferences involved when producing the exclusion set of a node as described above. The annotations which were made in Figure 8 were found to *change* each time MIN was allowed an extra reasoning step. However, there are some inferences which MAX can make which will *never* be invalidated by future rea-

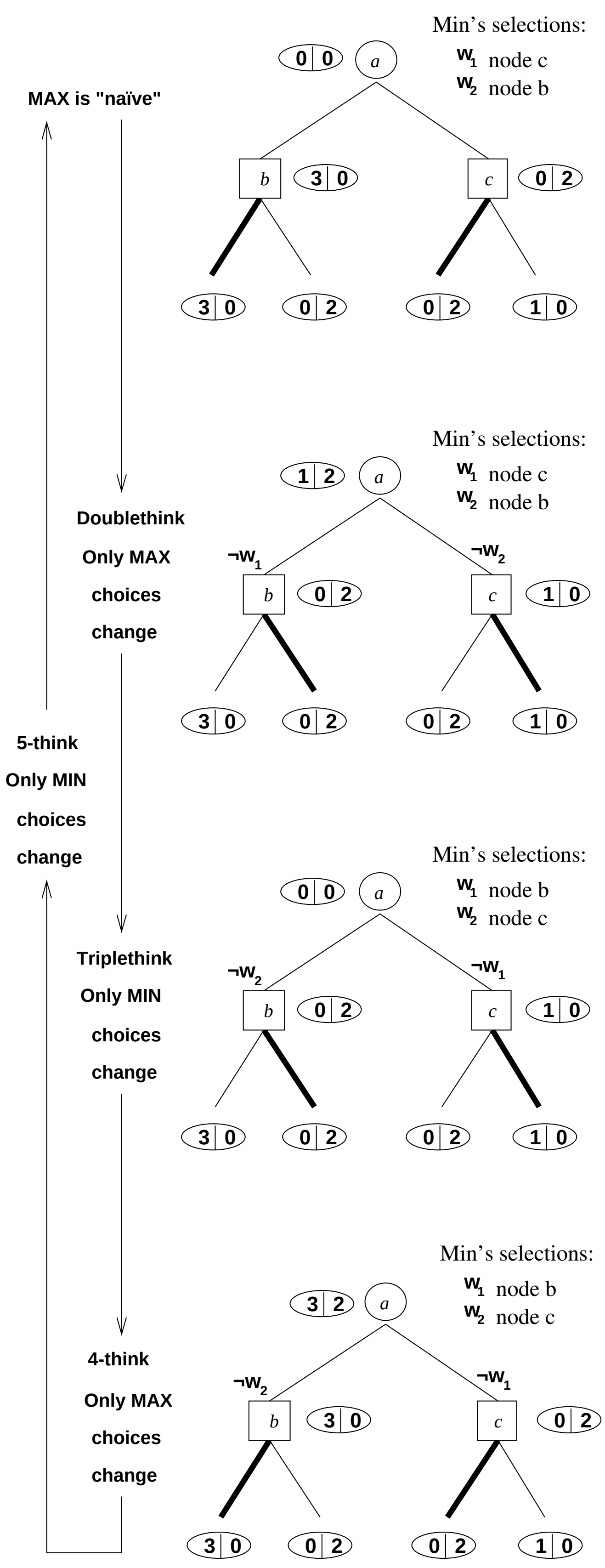

Figure 8: Multiple levels of reasoning gives rise to loops

soning carried out by MIN. For example, consider the tree of Figure 9, in which we have added one extra possible world to the tree of Figure 4.

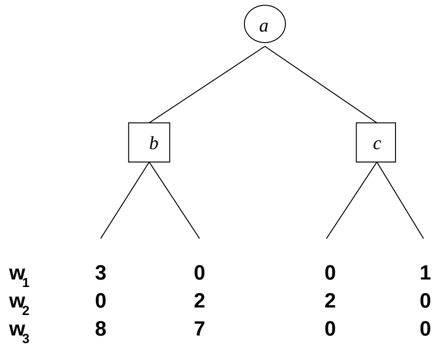

Figure 9: Example tree with one extra world

In this tree, the payoffs of 8 and 7 in world w_3 will *never* be realised, because the two zeros under node c will always enable MIN to restrict MAX to a zero in this world. Thus, the addition of the extra possible world has no effect on the assessment of the best strategy, since node b will *never* be reached under world w_3, whatever strategy is selected by MAX. We can therefore annotate node b with the term $\neg w_3$, in the knowledge that this inference will never be incorrect. This *unconditional* inference is different in nature to the ones that we saw in the previous section, whose truth depended on the actual strategy chosen by MAX. For instance if MAX selects the left-hand branch at node b, MIN's evaluation function will always back up the value from node c in world w_1. Thus, node b should be given the *conditional* inference that 'MIN will not direct play down this branch in world w_1 *if* the left-hand branch is selected'. The $\neg w_2$ annotations in Figure 8 are also conditional on the strategy selected by MAX.

4.5 Payoff-reduction Minimaxing

It is possible to envisage a two-step algorithm which first identifies all the *unconditional* inferences which are correct independent of strategy selection and then uses these to supply extra information to a local evaluation function. The use of inferences that are always correct would make it unnecessary to allow MIN a further level of reasoning after MAX's doublethink step. However, we can do better than this if, instead of focussing on the inferences themselves, we consider how the game tree itself can be manipulated. Specifically, we can try to modify the payoffs at the leaf nodes of a tree so that the payoffs of each strategy are unaffected but the number of possible inferences is reduced.

To see this, we can apply some reasoning about the nature of incomplete information games. In any game tree with a *single* world (where both players know the payoffs at the leaf nodes and can see the outcomes of each other's moves), the optimal strategy can be found with the standard minimax algorithm, and the best payoff that MAX can achieve is the minimax value of the game tree. For an incomplete information game with N possible worlds, let us use p^k_{max} to denote this mini-

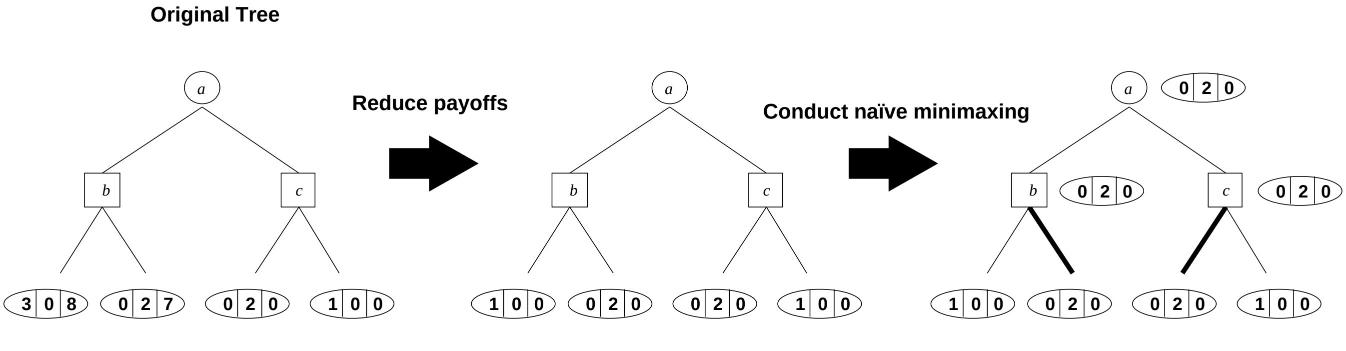

Figure 10: Applying the *prm* algorithm

max value of the game tree formed by deleting all worlds from consideration except world w_k. Even when playing a game with multiple worlds (under our assumption that MAX does *not* know the state of the world but MIN does), the value of p^k_{max}, forms an upper bound on MAX's payoff in world w_k. When making a branch selection at a node, then, any branch that offers a payoff in world w_k that is higher than p^k_{max} will never be reached in that world. Such payoffs may adversely affect branch selection at the node. Rather than making conditional inferences by checking payoffs against the payoffs of sibling MIN nodes (which change with MAX's strategy), we can instead utilise the values of the p^k_{max}, which, like unconditional inferences, do not depend on the strategy selected by MAX.

In fact, this argument holds not just for the minimax values of the entire game tree, but for every MIN node ancestor; any branch selection that MAX makes in anticipation of a payoff higher than the p^k_{max} of the tree rooted on *any* MIN ancestor is bound to based on false information, since on the way to this MAX node, play must pass through a node at which MIN will always be able to direct the play towards a lower payoff.

We therefore propose a new algorithm, *payoff-reduction minimaxing* (or *prm* for short) which improves on *naïve* minimaxing by modifying the payoffs of any given game so that *none* of them exceed the minimax value of a MIN ancestor in any world. This algorithm first finds, for each world w_k, the values of the p^k_{max} of every MIN node in the game tree. It then examines the payoff vectors at each leaf node; any payoff in some world w_k that exceeds the smallest p^k_{max} of a MIN ancestor is reduced to this value, and those that do not are left unchanged. Finally, the *naïve-mm* algorithm is applied to the resulting tree.

Figure 10 shows how *prm* behaves on the tree of Figure 9. For the tree of Figure 9, the minimax values in each of the three possible worlds are 1, 2, and 0, respectively. Thus, *prm* reduces all the payoffs in each world to at most these values (there is only one MIN node in the tree, so each leaf node has only one MIN ancestor). The *prm* algorithm then applies *naïve-mm* to the resulting tree, producing the strategy selection highlighted in bold. For this example, *prm* results in the correct strategy being chosen. Note that in general, the reductions of the payoffs made by *prm* do not affect the minimax value of the tree in each world, since no payoff is reduced to the extent that it would offer MIN a better branch selection at any node in any world.

It may seem that *prm* is inefficient, since it has to conduct minimaxing once in each world, then reduce the payoffs, and finally carry out naïve minimaxing. However, it is possible to make some of these steps more efficient and to combine others together. For example, the minimax value of each MIN node in each possible world can be calculated in one pass of the tree. This can be done with a modified form of the *naïve-mm* algorithm in which MAX has the same luxury as MIN to choose different branches in different worlds. That is, we can rewrite the definition of the max operator over a set of payoff vectors to be:

$$\max_i \vec{K}_i = (\max_i \vec{K}_i[1], \max_i \vec{K}_i[2], \cdots, \max_i \vec{K}_i[m])\,. \quad (6)$$

In Figure 11 we apply this modified algorithm to the example tree of Figure 9. At node b, MAX uses (6) to choose the payoff in each world and finds that payoffs of 3, 2, and 8 can be achieved. Similarly, a vector containing payoffs of 1, 2, and 0 is produced at node c. Then, MIN applies (1) to build the vector of minimax values at the root. A *prm* algorithm incorporating this approach is given in Figure 12.

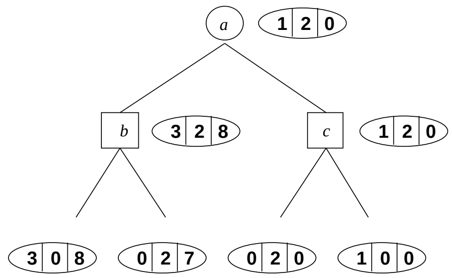

Figure 11: Minimaxing all worlds in one pass of the tree

Notice that the *prm* algorithm makes all the unconditional inferences we discussed in §4.4. These inferences occur when a node will *never* be visited in a particular world; that is, when the payoff in that world on *every* branch at the node is higher than the minimax value of a MIN ancestor. For example, consider world w_3 in our example tree. At node b, both the payoffs in this world are higher than the minimax value of the tree (which is

Algorithm $prm(t)$:
Returns strategies for game trees t:

1. Conduct (in a single pass) minimaxing of each world, w_k, storing at each MIN node its minimax value in each world, p^k_{max}.
2. Apply a modified form of *naïve-mm* (see Figure 6) to the annotated tree. The only difference from *naïve-mm* is at the leaf nodes, where the payoff vectors are reduced before being returned. Since the *naïve-mm* can be implemented depth-first, keep a record, for each world, w_k, of the annotations encountered at the MIN nodes leading to the current position in the tree — when encountering leaf nodes, reduce the payoff in each world w_k to the smallest p^k_{max} encountered.

Figure 12: Modified form of the payoff-reduction minimaxing algorithm

zero). The reduction of these payoffs to this value is effectively the same as ignoring the world altogether, given equally likely worlds.

However, *prm* still does not make *all* the inferences which are possible, since it is probable that MAX will not be able to achieve the actual minimax value of the tree in all worlds simultaneously. When this is the case, it is possible that some MAX node selections will be erroneously influenced by a payoff in some world which — although not higher than the minimax value of any MIN ancestor in that world — is higher than the actual value which is achieved as a result of MAX's other branch selections. In the next section, though, we show that the inferences that *are* made by *prm* lead to a significant performance advantage.

5 Practical Experiments

We tested the performance of payoff-reduction minimaxing on randomly generated trees. We compare the results of these tests to the results produced by Monte-carlo sampling on the same trees, showing that payoff-reduction minimaxing delivers a significant increase in performance at the cost of only a small decrease in speed.

5.1 Game Trees used for Testing

For simplicity, the trees we use in our tests are complete binary trees, with ten worlds that have payoffs of just one or zero. These payoffs are assigned by an application of the Last Player Theorem [Nau, 1982], so that the complete information game tree in any individual world can be won by MAX with a probability of 1/2. The Last Player Theorem introduces a probability, p, which determines the chance of selecting a 1 at the leaf nodes of a tree as follows: if the last player is MAX, choose a 1 with probability p, but if the last player is MIN, 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 $p = (3-\sqrt{5})/2 \approx 0.38197$. This value of p has the property that, in an individual world, 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 $1-p$. 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.

5.2 Tests on Monte-carlo Sampling

In previous work [Frank *et al.*, 1997a] we have examined the performance of Monte-carlo sampling on the game trees of §5.1. In this context, Monte-carlo sampling makes a selection at a MAX node by finding the minimax value of each of the direct subtrees in each world (or a statistically significant subset of worlds), and choosing the branch with has the best expected payoff. [1]

For comparison with the *prm* algorithm, we replicate the main results of our previous research in Figure 13.

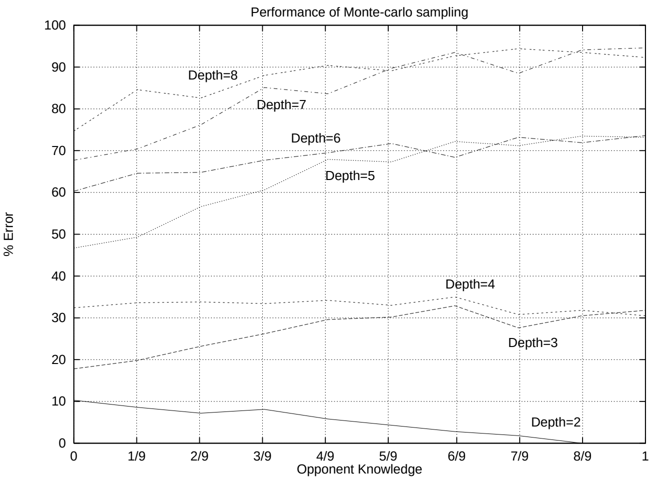

Figure 13: The error in Monte-carlo sampling for trees of depth 2 to 8 (first appears in [Frank *et al.*, 1997a])

This figure was produced by carrying 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 §5.1.
2. Use Monte-carlo sampling to identify a strategy (examining all possible worlds).
3. Check the payoff of this strategy, for an opponent with the level of knowledge specified.
4. Use an inefficient, but correct algorithm (described in [Frank *et al.*, 1997a]) to find an optimal strategy, for an opponent 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

[1] In fact, Monte-carlo sampling can be implemented using a very similar approach to the algorithms discussed already in this paper. For example, in §4.5 we described how to compute the minimax value of every world simultaneously by replacing (2) with (6) in the naïve minimaxing algorithm. The strategy found by Monte-carlo sampling is actually the same as that found by a naïve minimaxing algorithm that uses (6) to back up values at MAX nodes, but (2) to make it's actual branch selection at each node.

is inferior to the value of the strategy found in step 4, under the assumption of equally likely worlds).

In this graph the general increase in error rates as the opponent knowledge increases is due to the problem of non-locality. A further source of error is *strategy fusion*, which we formalise in [Frank, 1996; Frank and Basin, 1997]. For a more detailed explanation of the form of Figure 13 readers are referred to [Frank *et al.*, 1997a].

5.3 Tests on Payoff-reduction Minimaxing

We repeated the tests of the previous section substituting the *prm* algorithm for Monte-carlo sampling, producing the graph of Figure 14.

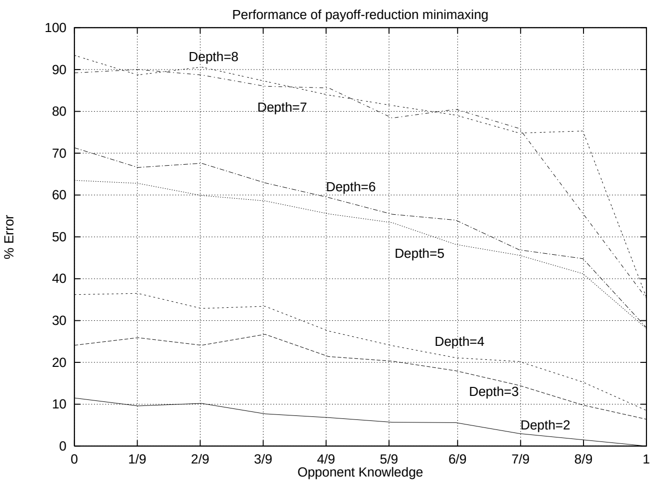

Figure 14: The error in payoff-reduction minimaxing for trees of depth 2 to 8

Figure 14 clearly demonstrates that the *prm* algorithm is more successful than Monte-carlo sampling when the knowledge of the opponent is 1. For trees of depth 8, the error of *prm* is just over a third of the error rate of Monte-carlo sampling. Note further that — although we designed *prm* for the situation where the opponent has complete knowledge — Figure 14 suggests that *prm* also out-performs Monte-carlo sampling for levels of opponent knowledge down to about 5/9.

To test the *prm* algorithm on larger trees, we extended the range of these tests to cover trees of depth up to 13, with the opponent knowledge fixed at 1. The results of this test are shown in Figure 15. When the trees reach depth 9, Monte-carlo sampling has an error rate of 99.9%, whereas *prm* still identifies the optimal strategy in over 40% of the trials. For trees of depth 11 and over, where Monte-carlo sampling *never* finds a correct solution, *prm* is still performing at between 40% and 30%.

Since others have suggested using Monte-carlo sampling in the game of Bridge, (where it was proposed by [Levy, 1989] and recently implemented by [Ginsberg, 1996b; 1996c]), we modified our game trees so that Monte-carlo sampling could identify optimal strategies with the same success rate as it does for Bridge problems (about 65% [Ginsberg, 1996a; Frank, 1996]). We achieved this by the simple expedient of parameterising

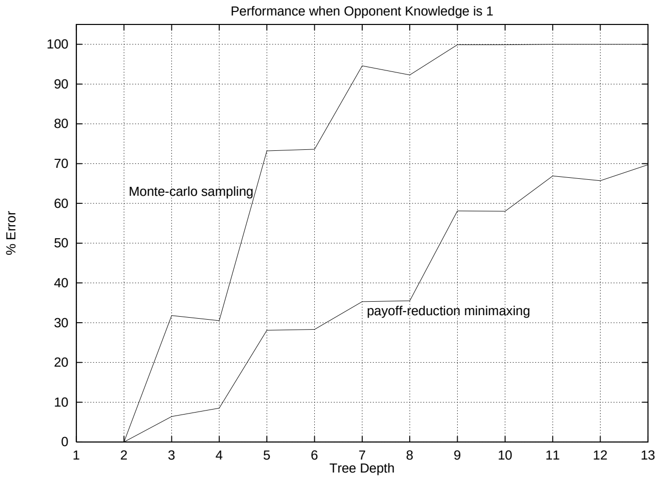

Figure 15: Superiority of payoff-reduction minimaxing, opponent knowledge set at 1, on trees of depth up to 13

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 described in §5.1, 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. For a value of $q = 0.7$, Monte-carlo sampling performed at around the 65% level, whereas *prm* identified correct strategies in about 93% of the test trees (see Figure 16).

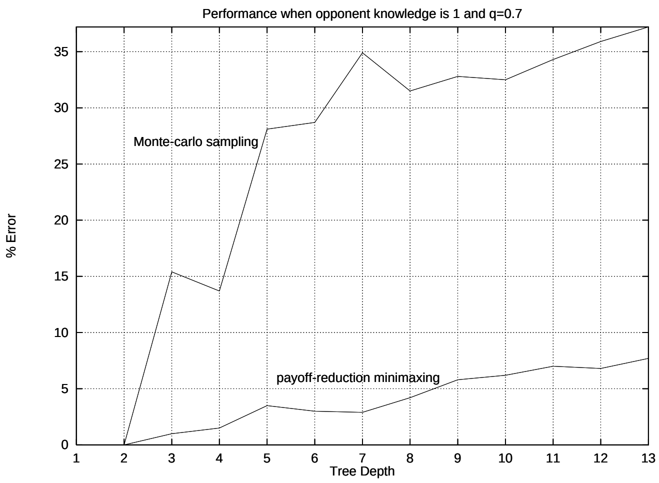

Figure 16: Superiority of payoff-reduction minimaxing on trees where the optimal strategy in one world is more likely to be optimal in another

The *prm* algorithm achieves these performance gains without sacrificing efficiency. In Figure 17, we have plotted the total computation time required for Monte-carlo sampling and the *prm* algorithm to select a strategy on 1000 trees of depths between 1 and 12 (so that a point with a y-value of 10sec indicates that each individual run took about 0.01 seconds). The *prm* algorithm is between

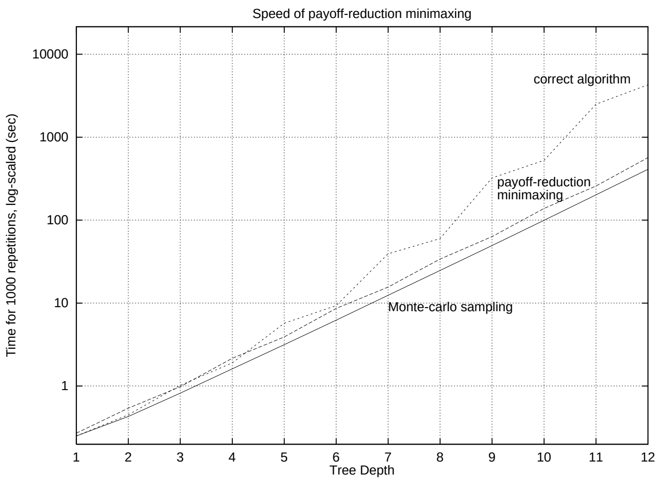

Figure 17: Algorithm speed. Timings produced on a Sun SPARCstation Ultra2 running at 200MHz.

1.08 and 1.39 times as expensive as our implementation of Monte-carlo sampling on these trees. For comparison, the correct but inefficient algorithm that we mentioned earlier is up to over 10 times as expensive to run.

6 Conclusions

We have introduced *payoff-reduction minimaxing*, and shown that it is an effective algorithm for finding optimal strategies in games with incomplete information. By using simple test game tress, we demonstrated that payoff-reduction minimaxing dramatically outperforms the major competing algorithm, Monte-carlo sampling. We showed how the process of making inferences address the problem of non-locality, which arises in a game with incomplete information when an opponent has superior knowledge about the state of the world. Payoff-reduction minimaxing is designed to model an opponent with complete knowledge of the actual world state, but our test results suggest that it also performs well in situations where the opponent is less well-informed.

References

[Blair *et al.*, 1993] J.R.S. Blair, D. Mutchler, and C. Liu. Games with imperfect information. In *Games: Planning and Learning, Papers from the 1993 Fall Symposium*, pages 59–67, AAAI Press, 1993.

[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, 1997] 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. Longer Version available from Edinburgh as DAI Research Paper No. 575.

[Frank *et al.*, 1997a] I. Frank, D. Basin, and H. Matsubara. Monte-carlo sampling in games with incompltete information: emprical investigation and analysis. Technical Report ETL-97-18, Electrotechnical Laboratory, 1997. Submitted to Workshop on Game Tree Search in the Past, Today, and in the Future, Princeton, NJ.

[Frank *et al.*, 1997b] I. Frank, D. Basin, and H. Matsubara. Monte-carlo sampling in games with incompltete information: emprical investigation and analysis. In *Proceedings of the Workshop on Game Tree Search: past, present and future*, Princeton, NJ, 1997. Also avalaible from the Electrotechnical Laboratory as Technical Report ETL-97-18.

[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.