Theoretical Computer Science 252 (2001) 217–256

Theoretical Computer Science

www.elsevier.com/locate/tcs

A theoretical and empirical investigation of search in imperfect information games

Ian Frank[a,*], David Basin[b]

[a] *Complex Games Lab, ETL, Umezono 1-1-4, Tsukuba, Ibaraki, Japan*
[b] *Institut für Informatik, Universität Freiburg, D-79110 Freiburg, Germany*

Abstract

We examine search algorithms for games with imperfect information. We first investigate Monte Carlo sampling, showing that for very simple game trees the chance of finding an optimal strategy rapidly approaches zero as size of the tree increases. We identify the reasons for this sub-optimality, and show that the same problems occur in Bridge, a popular real-world imperfect information game. We then analyse the complexity of the underlying problem, proving it to be NP-complete and describing several polynomial time heuristics. We evaluate these heuristics theoretically and experimentally, demonstrating that they significantly out-perform Monte Carlo sampling. Indeed, on a set of Bridge problems drawn from a definitive expert text, our heuristics consistently identify strategies as good as, or superior to, the expert solutions – the first time a game-general tree search algorithm has been capable of such performance. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Game tree search; Imperfect information; Game theory; Bridge

1. Introduction

We examine the problem of finding optimal strategies for games with imperfect information. We show that finding the optimal strategy against *best defence* in such games is computationally intractable and hence requires heuristics to automate in general. We analyse the suitability of Monte Carlo sampling as a heuristic and propose several new alternatives. These new heuristics, when applied to the game of Bridge, can consistently solve single-suit play problems as well as, or better than, professional players.

Let us begin by motivating the problem in a simple, but general, setting, which provides a basis for our subsequent analysis. In games with imperfect information, the

* Corresponding author. Tel. +81-0298-61 3371; fax.: +81-0298-61 5919.
E-mail address: ianf@etl.go.jp (I. Frank).

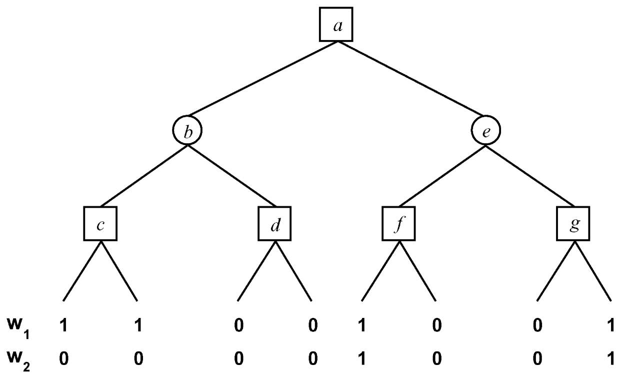


Fig. 1. A game tree with two possible worlds.

actual 'state of the world' may be unknown; for example, the position of some of the opponents' playing pieces may be hidden. We will call each possible outcome of the uncertainties (e.g., where the hidden pieces might be) a possible *world state* or *world*. Simple game trees can be used to represent such games and their associated worlds, as shown in Fig. 1. Here, the squares and circles correspond to MAX and MIN nodes, respectively, and there are just two possible worlds: w_1 and w_2. More generally, for a game with n possible worlds, each leaf node of the game tree would have n payoffs, each corresponding to the utility for MAX of reaching that node in each of the n worlds. In Fig. 1, the possible moves in each of the two world states are the same, but in Section 2 we show how trees with different branching patterns in different worlds can also be represented.

If both MAX and MIN know the world to be in some state w_i then all the payoffs corresponding to the other worlds can be ignored and the well-known minimax algorithm [21] used to find optimal strategies. In this paper, we will consider the more general case where the state of the world depends on information that MAX does not know, but to which he can attach a probability distribution (e.g., the toss of a coin or the deal of a deck of cards). We examine this situation for various levels of MIN knowledge about the world state.

The game of Bridge will serve as a running example throughout the paper, tying our general analysis to a well-known real-world game. In particular, we will test all the algorithms we present on a definitive set of play problems taken from an expert Bridge text. Since the Monte Carlo sampling algorithm has recently been applied to Bridge with some success [14], we begin by examining the performance of this algorithm. We first demonstrate the problems that occur when Monte Carlo sampling is applied to simple binary trees with ten worlds and randomly generated payoffs, and then show that the same problems arise in Bridge.

We introduce new heuristics that find optimal strategies more reliably than Monte Carlo sampling, but first demonstrate why heuristics are actually important by analysing the complexity of the underlying problem. Specifically, we prove that finding optimal strategies against *best defence* is NP-complete in the size of the game tree. That is, we prove that arbitrary (imperfect information) *trees* like that of Fig. 1 are hard to analyse. This differs from perfect information games, where arbitrary trees are easy to analyse (the minimax algorithm executes in time linear in the size of the tree), but arbitrary *games* may be hard to analyse when their definitions generate trees of exponential size. In contrast, finding optimal strategies for games such as that of Fig. 1 – even if the game tree is small enough to be exhaustively searched by computer – may be infeasible, and heuristics may be necessary.

The heuristics we propose to combat this intractability tackle the *non-local* nature [8] of imperfect information games by partially modelling the dependencies between the choices made at MAX nodes. We demonstrate the practical importance of these heuristics on random trees and on the Bridge database. For the Bridge problems, we find that combining all our heuristics into a single algorithm actually produces performance superior to the human experts that produced the model solutions. In the past, special-purpose Bridge programs for identifying complex positions such as *squeezes* have been developed [20], but our results represent the first general tree search algorithm capable of consistently performing either at, or above, expert level in actual card-play.

1.1. Outline

This paper gives a complete account of our work on search algorithms for imperfect information games and supersedes previous results reported in [7, 10, 11]. In Section 2 we introduce relevant background and provide definitions. We also show how Bridge card-play problems can be represented in the form of Fig. 1. We formalise Monte Carlo sampling in Section 3 and test its performance on random game trees in Section 4. Our analysis of these tests reveals why the algorithm performs sub-optimally, and we show in Section 5 that the same problems occur in Bridge. We then prove in Section 6 that the underlying problem is NP-complete in the size of the game tree and in Section 7 we present heuristics that tackle this complexity. In Section 8 we give experimental results that demonstrate the performance improvements produced by our heuristics and in Section 9 we draw conclusions.

2. Game theory background

Here, we introduce the relevant background for discussing games with imperfect information, concentrating in particular on the kind of game tree introduced in Fig. 1. We describe how such trees relate to the *extensive form* of a game, how they can be used to represent real-life games, and how to give a well-defined meaning to the *level of information* held by each player.

2.1. Extensive form

In terms of basic game theory [12, 18], trees like that of Fig. 1 are simply a compact way of representing the *extensive form* of two-person, zero-sum games with imperfect

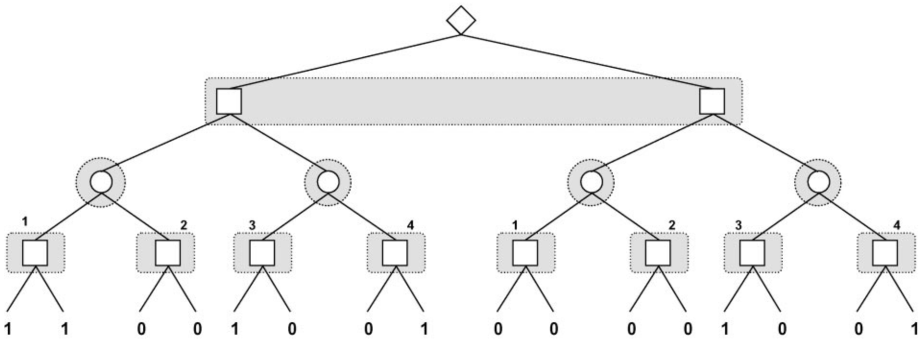


Fig. 2. Extensive form representation of a game with two possible worlds.

information. For example, an alternative rendering of the tree of Fig. 1 is shown in Fig. 2. In addition to MAX and MIN nodes, this tree includes a chance node, represented by the diamond at the root of the tree. This tree also depicts a further aspect of the extensive form of a game: the *information sets* of each player. These sets indicate what, within the rules of the game, each player can know when they make a move. Each information set groups together nodes that a player will find indistinguishable: when making any move, a player will have enough knowledge to identify the current information set, but not enough knowledge to identify the particular node within that set that play has reached.

Let us examine the information sets of Fig. 2. For the first MAX level of the tree, there is a single information set that includes both nodes. Since this means that MAX cannot distinguish between these nodes, MAX 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 allow MIN to distinguish between them (i.e., MIN must know 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, MAX is therefore unaware of the outcome of the first chance move.

The game trees we present throughout this paper will fit the pattern of Fig. 2, where the outcome of each player's moves is known to the other and there is one chance move, which occurs at the beginning of the game. In general, there may be n possible outcomes of this chance move, and we will say that each of these outcomes determines a possible *world state* or *world* in which the play takes place. Whenever the tree of possible moves is the same in each world (as in Fig. 2) we can *flatten* the extensive form game tree (as in Fig. 1) by representing 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. (Real games may not appear to fit the requirement that the tree of possible moves is the same in each world, but see the discussion in Section 2.3.)

A single node in a flattened tree represents between 1 and n information sets: one if the player whose turn it is to move knows nothing about the outcome of the chance move, and n if the player has exact knowledge. In this way, a single game tree in flattened form actually represents a number of extensive form games, each with different arrangements of information sets (for example, the information sets of Fig. 2 are just one of the possibilities described by Fig. 1). What determines the *actual* composition of the information sets is the level of knowledge of each player. We give this a precise definition below.

2.2. Level of knowledge

In our analysis throughout this paper, we will assume that MAX has no information about the world state so that, for each MAX move, the best that can be done is to consider the *expected* outcome over all worlds. Thus, any MAX node in a flattened tree corresponds to a single information set containing n nodes (as in Fig. 2).

For MIN's moves, however, we will examine different assumptions about the level of information available in the game. In particular, our tests on binary trees (presented in Section 4) will examine the consequences of gradually increasing MIN's knowledge from the same level as MAX up to perfect information.

We model MIN's level of knowledge by assuming that, for i $(0 \leqslant i < n)$, MIN's information sets contain $n - i$ nodes. That is, the value of i 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) i 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 $n - i$ worlds. We define the level of knowledge of such a MIN player as being

$$i/(n-1). \tag{1}$$

This takes a value between 0 (MIN has no knowledge and can only choose moves based on expected values) and 1 (MIN has perfect knowledge and can choose the best move at each node in each particular world).

2.3. A Bridge example

To show that flattened game trees like those given in Fig. 1 can be used to represent interesting games, consider the example of Fig. 3, which depicts a simple situation in a single suit of a game of Bridge. Such single-suit problems are common in the Bridge literature; the task is to find the optimal way to play just the cards in one suit, ignoring possible influences from other suits (such as *ruffing* or *entry* requirements), and assuming that the opponents do not initiate play in the suit.

In our example, a single MAX player controls both the North and the South cards (Ace, Queen, and 2) against the two defenders East and West (who are assumed to hold the King and the remaining nine low cards). The tree on the left-hand side of Fig. 3 shows the tree of optimal MAX moves for this situation. We make the simplifying

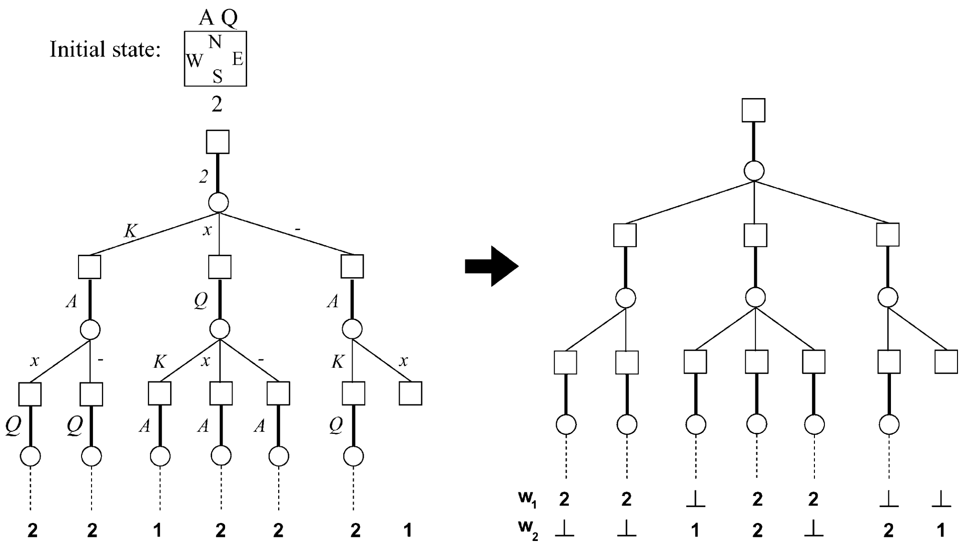


Fig. 3. The optimal MAX line of play for a single-suit Bridge problem represented (left) using constraints on the MIN branches, and (right) using multiple worlds.

assumption (which does not affect the analysis of the problem) that East and West have at most three distinct options at any point in the game: to play the King ('K'), a low card ('x'), or a card from a different suit ('-').[1] The total payoff for North–South (in terms of the number of *tricks* won) is indicated at the leaf nodes of the tree.

The game tree formed in this way is somewhat nonstandard in that all of the branches at MIN nodes can only be followed if certain conditions are true. For instance, East can only play the King if East actually holds the King. Applying the standard minimax algorithm to this tree without respecting these conditions results in a value of 1. However, we have some knowledge about the constraints on MIN's available moves, namely that they are the result of a chance move (the deal of the cards). If we do not distinguish between the nine low cards, this chance move can result in 20 distinct possibilities (for either East or West, they may hold between 0 and 9 low cards, and either hold the King or not). Rather than list the payoffs for each of these twenty worlds in the figure, we have simplified the presentation by instead considering the two mutually exclusive possibilities of "West holds the King" (w_1) and "East holds the King" (w_2). On the right-hand side of our figure, we have included a second game tree with payoffs at the leaf nodes for just these two worlds, using the symbol $\perp$ to

[1] With this restriction, the complete tree of MAX and MIN moves for this problem has 76 leaf nodes. In the pruned tree depicted here (actually a MAX *strategy*), MAX starts by playing the two from the South hand, and then playing the Ace from North hand if West plays a King or a card from a different suit. If West plays a low card, however, the Queen is played from the North hand (this is an example of a manœuvre called a *finesse*). After a play by the second opponent (East), the highest of the four cards played is said to win a *trick* for either North–South or East–West, and the next trick begins. In our example, North–South will then either be able to gain a trick with the Ace or the Queen, or will have no further options.

represent leafs that cannot be reached in one of the worlds. This tree shows that two tricks can always be made when West holds the King, but only one trick when East holds the King.

The second game tree of Fig. 3 is now in the same form as that of Fig. 1, with the only difference being that some of the payoffs take the undefined value $\perp$. So, the $\perp$ values allow flattened trees to represent games that have different branching patterns in different worlds. For the example of Fig. 1, all the payoff values are defined (either 1 or 0) so the extensive form tree has a single chance move at the root and $n=2$ identically shaped subtrees. In the Bridge example, there is also a single chance move (the deal of the cards), but the moves that are possible in each world are different, sometimes giving rise to $\perp$ payoffs. The algorithms we present throughout this paper will all manipulate payoffs in this general sense. To facilitate this, we extend the normal min and max functions so that $\min(\perp,\perp) = \max(\perp,\perp) = \perp$, and $\min(x,\perp) = \min(\perp,x) = \max(x,\perp) = \max(\perp,x) = x$, for all $x \neq \perp$.

3. Monte Carlo sampling

One well-known technique for handling imperfect information is Monte Carlo sampling [3]. This approach has been used in games such as Scrabble (see [5]) and also in Bridge, where it was proposed by Levy [17] and recently implemented by Ginsberg [14]. In the context of game trees like that of Fig. 1, Monte Carlo sampling consists of guessing a possible world and then ignoring the payoffs associated with the remaining worlds. This produces an easier problem than the original game because restricting attention to just one world creates a perfect information situation, and the minimax algorithm can thus be used to find the optimal moves. By guessing different worlds and repeating this process, it is hoped that an action that works well in a large number of worlds can be identified.

To make this description more concrete, let us consider a general MAX node with branches $M_1, M_2, \ldots$ in a game with n worlds. If e_{ij} represents the minimax value of the node under branch M_i in world w_j (see Fig. 4), we can construct a scoring function, f, such as

$$f(M_i) = \sum_{\substack{j=1 \\ e_{ij} \neq \perp}}^{n} \Pr(\mathsf{w}_j) e_{ij}, \tag{2}$$

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

As an example, consider how the tree of Fig. 1 is analysed by the above characterisation of Monte Carlo sampling. If we examine world w_1, the minimax values below

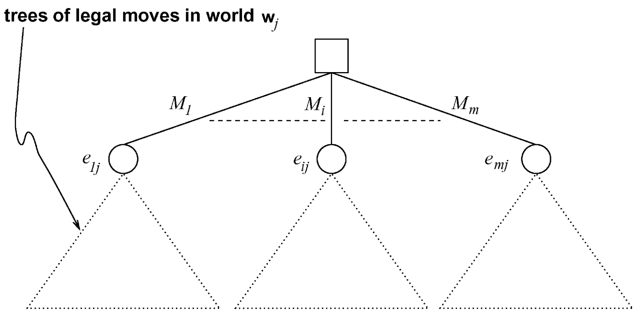


Fig. 4. Producing the minimax values, e_{ij}, of each move M_i under world w_j.

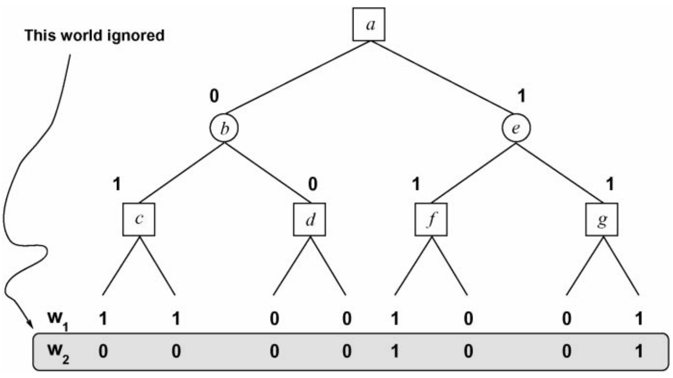


Fig. 5. Finding the minimax value in world w_1.

node a are as shown in Fig. 5 (these correspond to e_{11} and e_{21} for this tree). Furthermore, the minimax values at nodes b and c are unchanged if we instead examine w_2. Thus, Monte Carlo sampling will choose the right-hand branch at node a, which is the correct move for this tree.

Although Monte Carlo sampling consists of a sequence of guesses (for instance, made in accordance with the probability distribution of the chance move) and associated trials (i.e., passes through the game tree) it can in fact be implemented with an algorithm that analyses the game tree just once. We introduce this single-pass algorithm here since it will form the basis for the experiments and the generalisations that we present later in this paper.

For any flattened tree with payoffs for multiple worlds at the leaf nodes, we define the function *payoff-vector*. For any leaf-node, v, *payoff-vector*(v) returns an n-element vector $\boldsymbol{K}$ such that $\boldsymbol{K}[j]$ (the jth element of $\boldsymbol{K}$) takes the value of the payoff at v in world w_j ($1 \leqslant j \leqslant n$). Using this representation, we can define a minimax-like algorithm that analyses trees by manipulating payoff vectors rather than numbers. To do this, we extend the definition of the functions min and max to cover sets of m payoff vectors

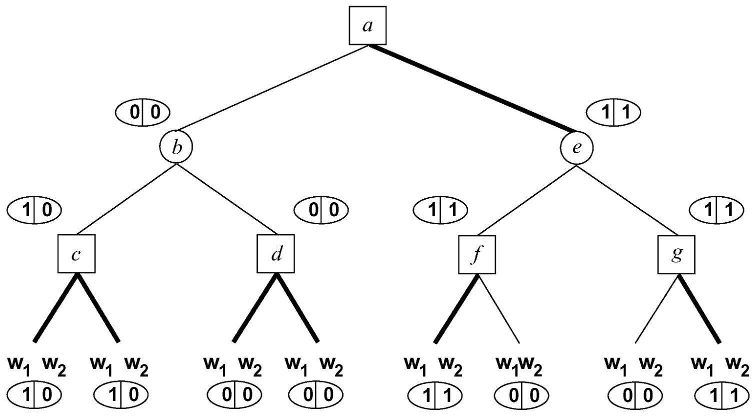


Fig. 6. Monte Carlo sampling using vectors.

$\boldsymbol{K}_1, \ldots, \boldsymbol{K}_m$ such that

$$\min_i \boldsymbol{K}_i = \left(\min_i \boldsymbol{K}_i[1], \min_i \boldsymbol{K}_i[2], \ldots, \min_i \boldsymbol{K}_i[n]\right), \tag{3}$$

$$\max_i \boldsymbol{K}_i = \left(\max_i \boldsymbol{K}_i[1], \max_i \boldsymbol{K}_i[2], \ldots, \max_i \boldsymbol{K}_i[n]\right), \tag{4}$$

where i always ranges from 1 to m. 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 simultaneously backed up the tree. For example, the tree of Fig. 1 is analysed as shown in Fig. 6.

In this figure, we have shown the vectors that are produced at each node. We have also indicated (in bold) the branches that would be selected by a Monte Carlo sampling algorithm examining every world. The choice of branch at MAX nodes is determined by an updated version of (2) that evaluates the payoff vector, $\boldsymbol{K}$, associated with each possible move, M_i, as follows:

$$f(M_i) = \sum_{\substack{j=1 \\ e_{ij} \neq \perp}}^{n} \Pr(\mathsf{w}_j)\boldsymbol{K}[j]. \tag{5}$$

The branch selections in Fig. 6 illustrate the main benefit of the single-pass approach: the simultaneous identification of the selections made by Monte Carlo sampling at *every* MAX node in the game tree. (Note that at nodes c and d, (5) gives the same score to each branch; such choices are resolved randomly.)

A specification of the branch selections at the MAX nodes in a tree is equivalent to the formal notion of a *strategy* in the normal form of a game (see, for example, [18]). In general, the number of possible strategies in a tree is doubly exponential in the number of MAX levels in the tree. We have implemented a correct, but computationally expensive, program to identify optimal strategies on simple game trees. Below, we examine, for different levels of MIN knowledge, how often the optimal strategy for a randomly generated game tree is superior to the strategy selected by Monte Carlo sampling.

Note that Monte Carlo sampling identifies *pure* strategies, which make no use of probabilities. In fact, imperfect information games have the property that it is in general important for players to prevent their opponents from 'finding out' their strategies, by making choices probabilistically. In this paper, however, we will restrict our consideration to pure strategies (such as those identified by Monte Carlo sampling). In practice, this need not be a serious limitation, as we will see when we consider the game of Bridge in Section 5.

4. Performance of Monte Carlo sampling on random game trees

To investigate the performance of Monte Carlo sampling, we carry out tests on complete binary trees with 10 randomly generated payoffs at the leaf nodes. For these trees, we assume that the players always alternate and that the player whose play we try to optimise (MAX) goes first, with the opponent (MIN) playing second.

The leaf node payoffs of our test trees are either 1 or 0, and are assigned so that the probability of there being a forced win for MAX in the perfect information game tree in any individual world is the same for all depths of tree. This is done by an application of the Last Player Theorem [19]. This theorem introduces a probability, p, that 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 (perfect information) binary trees with a MAX node at the root, [19] gives us that the probability of a forced MAX win remains constant over all tree depths iff $p=(3-\sqrt{5})/2 \approx 0.38197$.

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 Fig. 7. To create this figure, we carried out the following procedure 1000 times for each data point of tree depth and opponent knowledge:

(1) Generate a random test tree of the required depth.
(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 a correct (but computationally expensive) program 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 MIN's knowledge, as the depth of the game tree rises, the error in Monte Carlo

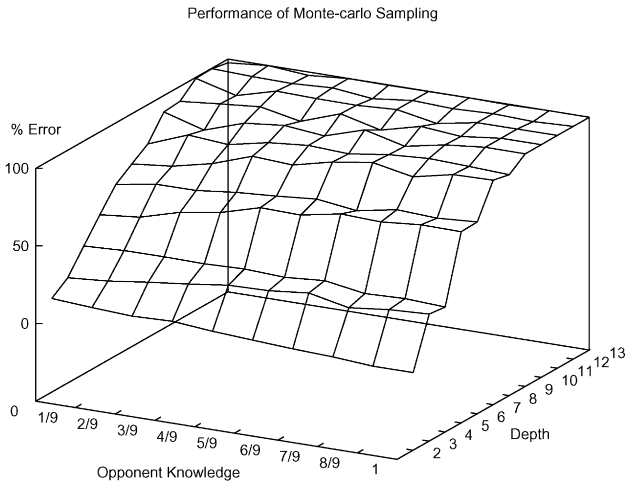


Fig. 7. The percentage of binary trees with 10 possible worlds for which Monte Carlo sampling selects a sub-optimal strategy, plotted for trees of depth between 2 and 13 (y-axis), and for varying levels of knowledge held by the opponent (x-axis, ranges from zero to perfect knowledge). One thousand tests per data point.

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 in Fig. 8 a simplified version of Fig. 7, which makes the finer detail of the error surface easier to appreciate. This new figure plots on a single plane just the curves for trees of depths 2–8, 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 answer each of these questions in the four subsections that follow.

4.1. A – MIN's knowledge

Here we explain the first annotated region of the graph in Fig. 8. To do this, we will appeal to the simple game tree shown in Fig. 9.

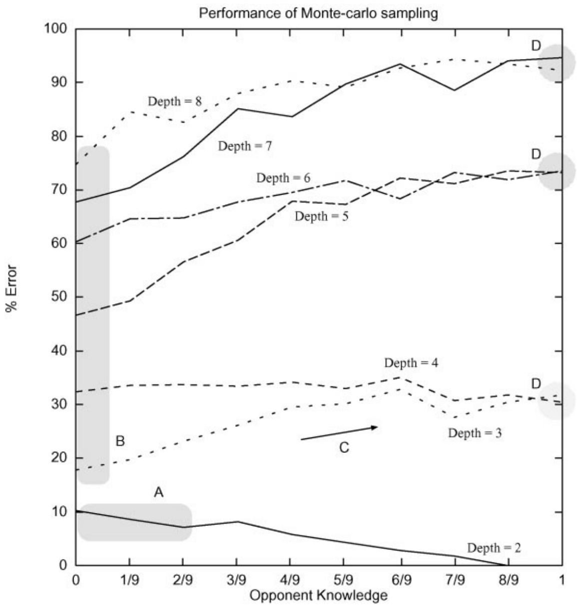


Fig. 8. A simplified graph showing the error in Monte Carlo sampling just for trees of depths 2–8. Specific features of this graph explained in the text are labelled A, B, C and D.

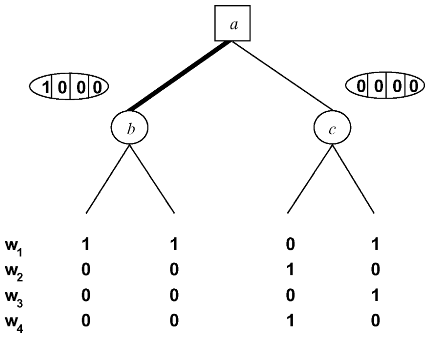


Fig. 9. Simple tree of depth 2.

Consider what happens when Monte Carlo sampling is applied to this tree. In world w_1, minimising the leaf node payoffs gives node b an evaluation of 1 and node c an evaluation of 0 (as shown by the vectors placed at the nodes). 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 0. Whenever Monte Carlo sampling includes w_1 in its sample, then, it will select the left-hand branch at node a. We indicate this selection in the figure by rendering the branch in bold.

Now let us go back the feature marked A in Fig. 8. At the point where the ‘opponent knowledge’ is zero, MIN’s knowledge about the actual world state 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 such a MIN player choose in the tree of Fig. 9? 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 Fig. 8 is due to the model of MIN's knowledge implicit in Monte Carlo sampling. Specifically, when Monte Carlo sampling uses minimisation (as represented by (3)) to find the value of a MIN node in some world w_i, a MIN player can only guarantee the same result if he knows that w_i is the actual world state. With less than perfect information, MIN will actually have to reason about the *expected* payoffs when making a move.

In terms of the information sets described in Section 2.1, for a MIN player with no knowledge of the world state, each of the nodes *b* and *c* constitute a single 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. This is therefore an example of *strategy fusion*, as formalised in [6, 8]. 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 Fig. 9, 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 increasingly accurate and strategy fusion at MIN nodes gradually disappears. This explains why the error rate for trees of depth 2 gradually decreases in Fig. 8.

4.2. B – MAX's knowledge

Next, consider the tree of Fig. 10, where we have again shown the vectors produced by 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 can only be 1 in the worlds w_1, w_2, and w_3, whereas the moves in the other subtree additionally produce a payoff of 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 player with imperfect information must make the *same* move in every world. Allowing MAX to choose a different strategy in each world is again strategy fusion.

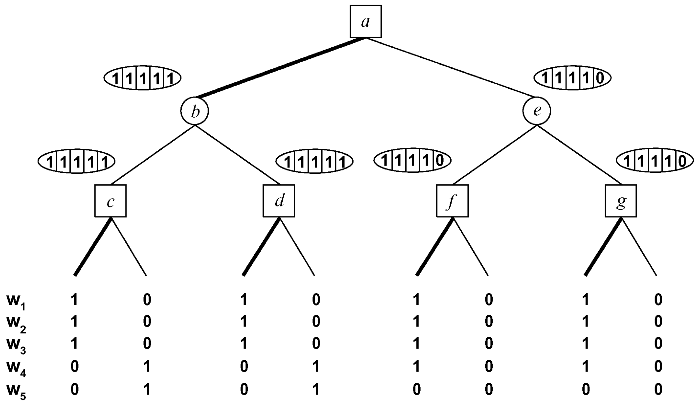


Fig. 10. Simple game tree of depth 3.

How does this relate to the graph of Fig. 8? 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 has. Now, we see how strategy fusion also affects MAX nodes, because of the assumption that MAX has more knowledge than he actually has. 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 nature of the region marked as B in Fig. 8.

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 Fig. 10 represent the MIN and MAX moves in two separate games, each starting with a chance move that selects one of the possible worlds $\mathsf{w}_1,\ldots,\mathsf{w}_5$. 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

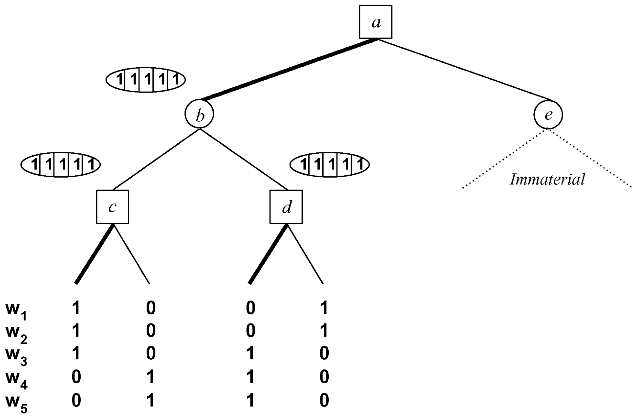


Fig. 11. Simple game tree illustrating non-locality.

this, we will consider the example tree of Fig. 11. This figure is a slightly modified version of Fig. 10 from the previous section, with the payoffs under node *d* altered.

As before, strategy fusion will cause Monte Carlo sampling to have a strong preference for selecting the left-hand branch at node *a*. 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 does not 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, MIN will select the right-hand branch, since MAX's choice of branch at node *d* will then lead to a payoff of 0. 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 perfect 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. For example, 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. 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

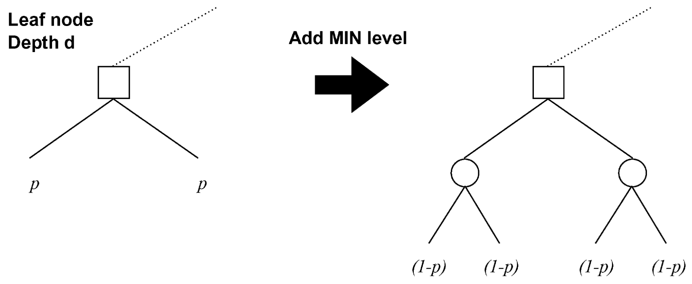


Fig. 12. The probability of assigning a ‘1’ at leaf nodes.

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 worlds w_4 and w_5. This affects the analysis of node d, since at any node below node b the maximum attainable payoff in worlds w_4 and w_5 is then 0. 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 that leads to a payoff of 1 in most worlds.

In general, the choice of a branch at a given MAX node v is not simply a function of the payoffs of the paths that contain v, but of the payoffs along *any* path in the tree. If MIN can choose a move at an ancestor of v that reduces the payoff (in any world) from what MAX would expect from examining v's direct subtree then the best branch at v may change. This problem of having to consider all other nodes in the tree is known as *non-locality* (for a precise formalisation, see [6, 8]). Non-locality clearly depends both on the number of MAX levels in the game tree and also on the level of MIN's knowledge. This explains why the trend for trees of depth greater than two is for the error to increase with the knowledge of the opponent (in our tests, trees of depth two do not suffer from non-locality because the problem only arises at MAX nodes that have MIN ancestors).

4.4. Region D

Finally, we look at the convergence in Fig. 8 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 previously explained, we use the Last Player Theorem [19] to set the probability of a 1 at the terminal nodes of such game trees at $p=(3-\sqrt{5})/2\approx 0.38197$. Consider now the effect of adding an extra MIN layer to such a tree (see Fig. 12). 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 a single world, we can calculate the probability that the evaluation of either of the MIN nodes will be 1. Since MIN can be expected 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. (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, and the chance that any branch at a MIN node has an evaluation of 1 remains at $1-p$.) 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. A MIN player with a level of knowledge less than 1 will in general be unable to make the best selections in every world.

The above observation allows us to round off our description of Fig. 8. So far, we have identified the following problems with Monte Carlo sampling:

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

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

C The problem of non-locality arises when playing against an opponent with knowledge of the world state. The errors caused by non-locality increase 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 trees of odd depth d and depth $d+1$ converge. At the far left of the graph, the extra layer of MIN nodes in the trees of depth $d+1$ leads to an increased incidence of strategy fusion errors at MIN nodes caused by the assumption of perfect 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 trees, 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.

5. Performance of Monte Carlo sampling on Bridge problems

Here we tie the results of the previous section to real games by examining how Monte Carlo sampling performs on single-suit Bridge problems, such as the one we discussed in Section 2.3. Bridge is of interest to us because it is a well-known example of an imperfect information game that has also been heavily analysed by human experts. These experts have produced texts that describe the optimal plays in large numbers of card-play situations. The availability of such references provides a natural way of assessing the performance of automated algorithms.

Unfortunately, a complete description of Bridge is beyond the scope of this paper. Readers interested in full details are referred to one of the many excellent books on the subject, such as [15]. However, most of the essential features of the game should already be apparent from the example in Sectoin 2.3. Recall that despite being played by four players, the game can be viewed as a two-player contest between the two teams, North/South and East/West. The further examples we give in this section will again follow the standard convention of specifying the cards to be played by North and South against the remaining (hidden) cards held by East and West.

5.1. Testing against the Encyclopedia of Bridge

To construct a Bridge test set, we used as an expert reference the Official Encyclopedia of Bridge published by the American Contract Bridge League [1]. This book contains a 55-page section presenting optimal lines of play for a selection of 665 single-suit problems, chosen for their coverage of the possible play situations that can arise in practice. All of the expert lines of play are in the form of pure strategies, so they form an ideal benchmark for evaluating the pure strategies produced by Monte Carlo sampling. We selected the 650 examples that gave pure strategies for obtaining the maximum number of tricks against opponents that also employ pure strategies. [2] Using the FINESSE Bridge-playing system [6, 9], we then tested Monte Carlo sampling against the solutions in the Encyclopedia.

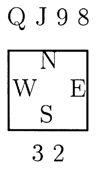
The results of our Bridge test showed that Monte Carlo sampling produced sub-optimal strategies in 220 cases (33.8%). These errors were all due to strategy fusion or non-locality. In terms of the *chance* of these sub-optimal strategies actually leading to a sub-optimal result, we found that the incorrect solutions produced the maximum possible payoff with a probability of 0.07 less than the model solution. Hence, if Monte Carlo sampling were used to play the entire Encyclopedia problem set with randomly distributed outstanding cards, the expected number of cases where the maximum possible payoff would be missed is $0.077 * 220 \approx 17$ cases (about 2.6%).

5.2. Bridge examples of strategy fusion and non-locality

To give a flavour of how the problems of strategy fusion and non-locality manifest themselves in Bridge, we present here two simple examples of problems from the Encyclopedia.

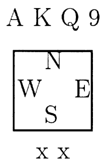
First, consider Fig. 13. For this problem, Monte Carlo sampling correctly selects the finesse of the nine, as suggested by the Encyclopedia. The first step in this play is to lead a low card from the South hand. North then plays the nine, unless West plays the ten, in which case North instead inserts the Queen. If both East and West play low cards on this first trick, the correct continuation on the second trick is to finesse the eight. Monte Carlo sampling, however, will continue by finessing the Jack.

[2] The remaining 15 Encyclopedia examples split into four categories: six problems that give no line of play for the maximum number of tricks, four problems involving the assumption of a *mixed* strategy defence, four for which the solution relies on assumptions about the defenders playing sub-optimally by not *false-carding*, and one where there are constraints on the cards that can be played.



For two tricks, finesse the nine. Chance of success: 0.51.

Fig. 13. Problem 583 from the Encyclopedia of Bridge, illustrating strategy fusion.



For four tricks, finess the nine; hope that West has both the Jack and Ten. Chance of success: 0.24.

Fig. 14. Problem 12 from the Encyclopedia of Bridge, illustrating non-locality.

This error is caused by (MAX) strategy fusion. To see this, consider what happens if the second-round finesse of the Jack loses to a high card played by East. The North hand will then have just two cards remaining: the Queen and the eight. Just one of these cards must be selected, and they will produce two tricks under different circumstances.[3] Strategy fusion, though, results in Monte Carlo sampling believing that it can gain two tricks in *both* sets of circumstances. In contrast, when the alternative (correct) second-round continuation of finessing the eight loses to a high card played by East, the cards remaining in the North hand are the Queen and Jack. Since these cards are essentially equivalent, no strategy fusion can occur. The second-round finesse of the Jack therefore *appears* more attractive than the finesse of the nine whereas in fact the reverse is true. The actual chance of success of the line of play chosen by Monte Carlo sampling (finesse the nine, then finesse the Jack) is just 0.289.

For an example where the correct strategy is obscured by non-locality, see Fig. 14. In this problem, the optimal first action is again to finesse the nine, and again Monte Carlo sampling discovers this. But, if West plays the Jack or the Ten on the first trick (and East plays a low card), Monte Carlo sampling again fails to find the correct continuation of repeating the finesse, this time choosing instead to cash a top card. The reason for this is that whilst repeating the finesse appears to succeed whenever West holds the Jack and Ten (a probability of 0.24) the cash appears to succeed not only when the cards are split JTx-xxxx or JT-xxxxx (a probability of 0.052) but also in the cases Txxxx-Jx, Jxxxx-Tx, Txxx-Jxx, and Jxxx-Txx (a probability of 0.2503). However, for these latter four cases, the promise of success is illusory; with these cards West can restrict North/South to just three tricks by playing low on the first trick. These worlds

[3] Playing the nine succeeds whenever the cards were originally split Txxx-AKx, Kxx-TAxx, or Axx-TKx (a probability of 0.142) whereas playing the Queen succeeds for the disjoint distributions Kxxx-TAx, Axxx-TKx, and Txx-AKxx (a probability of 0.124).

should therefore play no part in the decision of how to continue after West plays the Jack. Monte Carlo sampling doesn't account for this non-local effect, however, and chooses to cash the Ace in the expectation of an overall probability of success of over 0.3. In fact, the actual chance of success of the line of play is just 0.054 (succeeding under the two distributions JTx-xxxx or JT-xxxxx, and also when East has no cards at all in the suit).

5.3. *The best defence model*

A close examination of Bridge examples such as the ones above reveals that expert analysis involves some implicit assumptions about the way to analyse strategies. For example, to determine whether a strategy (such as "finesse the nine") succeeds under a particular distribution of the remaining cards, it is assumed that East/West play optimally in this perfect information sub-problem. This assumption is standard in expert analysis of Bridge, where the overall chance of success of a strategy is taken to be the sum of the probabilities of the distributions for which the strategy succeeds in this way. In previous work on Bridge, we have formalised the assumptions used by human experts to analyse Bridge problems [8], producing a *best defence model*, which we summarise here:

A-I MIN has perfect information.

A-II MIN chooses his strategy after MAX.

A-III The strategy adopted by MAX is a pure strategy.

This model is termed the *best defence model* because it formalises the strongest possible assumptions about the opponents: that MIN knows the actual state of the world (A-I) and chooses his strategy in the knowledge of MAX's strategy (A-II). The assumption that MAX chooses a pure strategy (A-III) also restricts the set of possible solutions to a finite (though possibly very large) set.

The best defence model is a useful basis for analysis not only because it is used by human players to analyse real games (e.g., all solutions in the Encyclopedia of Bridge were generated under these assumptions [4]), but also because modelling the strongest possible opponents provides a lower bound on the payoff that can be expected when the opponents are less informed. Also, the opposite extreme where MIN (like MAX) has no knowledge of the world state is easy to model, since an expected value computation can be used to convert the multiple payoffs at each leaf node into a single value, and the standard minimax algorithm can be applied to the resulting tree. In the remainder of this paper, we concentrate on the best defence model, examining its complexity and introducing heuristics that enable optimal solutions to be found.

Note that the best defence assumptions are also similar to those made implicitly by Monte Carlo sampling. As we have already pointed out, Monte Carlo sampling identifies pure MAX strategies that make no use of probabilities (A-III), and that the repeated application of minimaxing assumes that MIN can respond optimally to MAX's moves in each individual world, for which perfect knowledge of the world state is a prerequisite (A-I).

Given this, one might expect that Monte Carlo sampling would perform well on Bridge (and especially for the right-hand side of the graphs of Figs. 7 and 8, where opponent knowledge is 1). However, our analysis in the previous section demonstrated how the problems of strategy fusion and non-locality degrade performance. We saw how strategy fusion results from the implicit assumption that MAX can play optimally in each world, and non-locality results from the way that branch selections are based only on a node's direct subtree. We will later introduce heuristics that reduce the severity of these problems. Before doing this, however, we motivate why heuristics are actually needed by analysing the complexity of the underlying problem.

6. Complexity of play against best defence

Here we show that, given a game tree, finding optimal strategies under the best defence model is an NP-complete problem. Hence, unless $\mathrm{P} = \mathrm{NP}$, heuristics are required to tackle this problem in practice. As we noted in the Introduction, this is fundamentally different from perfect information games (such as n-by-n checkers) that, although PSPACE-hard in the size of the initial game configuration, can be solved in linear time in the size of the game tree [13]. In contrast, our results show that for the problems we are considering, even when a complete game tree is small enough to be searched by computer, finding optimal strategies may be infeasible.

Note that there have been previous analyses of the complexity of finding optimal strategies in imperfect information games [2, 16], but these proofs are not applicable when considering the best defence model. For example, [2] show that it is NP-complete to determine whether an n-player game has a pure strategy equilibrium point, but the proof uses a reduction (from the 3-Partition problem) that cannot be reproduced with trees such as that of Fig. 1. Moreover, the notion of a pure strategy point is also not helpful in the best defence model, since A-II introduces an asymmetry: MIN's strategy is chosen after MAX has made a decision, and thus a 'stable' strategy pair is always found by simply finding the optimal response to any MAX selection.

For our proof, we first formalise the relevant problems in the format of [13].
Best Defence:
Instance: A game tree t over n worlds and a positive integer $k \leqslant n$.
Question: Is there a MAX strategy that returns a payoff of 1 in at least k worlds under the best defence model?
Clique:
Instance: A graph $G = (V, E)$ and a positive integer $k \leqslant |V|$.
Question: Does G contain a clique of size k or more?
Note that here and later, we assume that the payoffs are bounded so that storage is possible in constant space, and we measure the size of a game tree t to be the number of nodes in t plus the number of payoffs listed (which is the product of the number of leaf-nodes and the number of worlds). Given that Clique is NP-complete [13], we now prove:

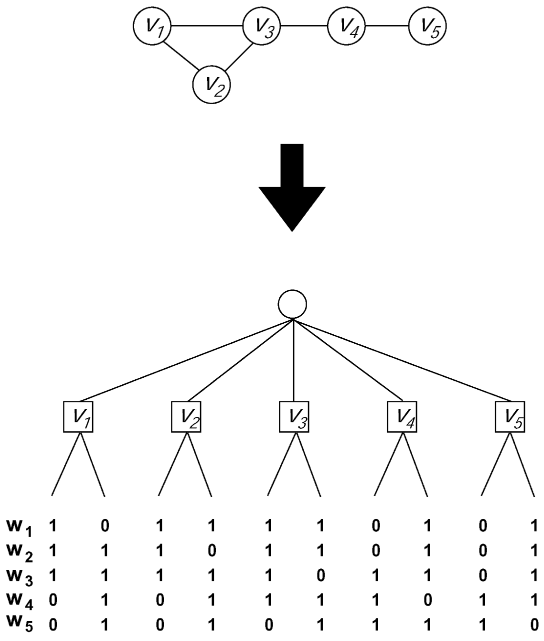


Fig. 15. Graph (top) with a corresponding game tree (bottom).

Theorem 1. Best Defence *is* NP-*complete.*

Proof. To see that Best Defence is in NP observe that given a game tree t we can guess a MAX strategy, s, (e.g., by specifying the branch to be chosen at each MAX node in the tree) and correctly determine the optimal payoff in time linear in the size of t. This can be done with an algorithm that is very similar to the single-pass Monte Carlo algorithm we described in Section 3. At MIN nodes, (3) is used to back up a vector of payoffs. But at MAX nodes, rather than using the max operator of (6), the payoff vector on the branch specified by s is returned. [8] have shown that this algorithm correctly computes the payoff of s, and the time taken is clearly linear in the size of the tree.

To show NP-hardness we reduce Clique to Best Defence. Let $G=(V,E)$ and k be given. We translate G to a tree t, with $n=|V|$ worlds, $\mathsf{w}_1,\dots,\mathsf{w}_n$, constructed to have a payoff of 1 in at least k worlds iff G has a k clique. The root of t is a MIN node. The next layer has n MAX nodes, which we label $v_1,\dots,v_n$, for $v_i \in V$. At each MAX node v_i there is a left and right branch, called l_i and r_i respectively. The payoff at the leaf node under each l_i is 1 in the jth world iff $i=j$ or $(v_i,v_j)\in E$. The payoff at the leaf node under each r_i is 1 in the jth world iff $i\neq j$. An example of a graph and its translation are given in Fig. 15. Note that the reduction is trivially computable in time polynomial in the size of G.

Let us call a vertex v_i *selected* if MAX chooses the left branch at v_i in his strategy. Suppose G has a k clique. The clique defines a subset $V' \subseteq V$ where $k=|V'|$ and for

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

Condition	Result
t is leaf node	*payoff-vector*(t)
root of t is a MIN node	$\min_{t_i \in sub(t)}$ *vector-mm*(t_i)
root of t is a MAX node	$\max_{t_i \in sub(t)}$ *vector-mm*(t_i)

Fig. 16. The vector minimaxing algorithm.

each $v_i, v_j \in V'$, where $i \neq j$, $(v_i, v_j) \in E$. It is easy to see that the MAX strategy that selects the vertices in V' has a payoff of 1 in each world w_i, where $v_i \in V'$. Hence this strategy has a payoff of at least k.

Conversely, suppose there is a strategy for MAX with a payoff of at least k. Let W be the set of worlds in which MAX's strategy yields a payoff of 1 and let $V = \{v_i \mid \mathsf{w}_i \in W\}$. Observe that V comprises a clique in G of size at least k since, for each world $\mathsf{w}_i \in W$, every selected vertex $v_j \in V$, $i \neq j$, must have a payoff of 1 in world w_i, which implies that $(v_i, v_j) \in E$. $\square$

In our example, we can select v_1, v_2, and v_3 and MIN's best strategy yields a 1 for MAX in worlds w_1, w_2 and w_3, and a 0 in the remaining two.

7. New heuristics for play against best defence

Given NP-completeness, we turn to the question of whether there are good heuristics for finding optimal strategies. We introduce four different heuristics that tackle either strategy fusion or non-locality by improving the modelling of the dependencies between the choices made at MAX nodes. The heuristics are all *conservative*, in that they never degrade the analysis, *general*, in that they improve the performance of a non-trivial number of examples, and *tractable* (i.e., they are polynomial time computable).

7.1. Vector minimaxing

In Section 3 we discussed how to carry out Monte Carlo sampling in a single pass of a tree by using the *payoff-vector* function to convert leaf node payoffs to vectors. A slight modification of this idea produces an algorithm that avoids the problem of strategy fusion altogether by ensuring that at any MAX node a single branch is chosen in all worlds. Fig. 16 defines this new algorithm, which we call *vector minimaxing*. The strategy selected by the algorithm is just the set of choices made at the MAX nodes.

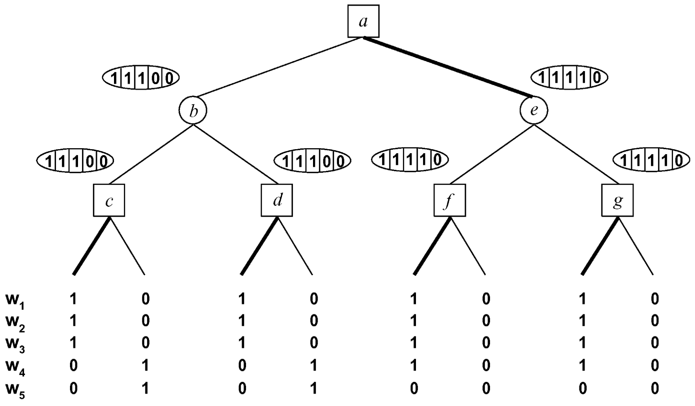


Fig. 17. Vector minimaxing applied to the example tree of Fig. 11.

In this algorithm, the function *sub*(t) computes the set of t's immediate subtrees. For the min function, we employ again the definition given by (3), which returns a vector in which the payoff for each possible world is the lowest possible. This correctly models a MIN player with perfect knowledge of the world state. For the max function, on the other hand, we assume that it returns the single vector $\boldsymbol{K}$, for which

$$\sum_{\substack{j=1 \\ \boldsymbol{K}[j]\neq\perp}}^{n} \Pr(\mathsf{w}_j)\boldsymbol{K}[j] \tag{6}$$

is maximum, resolving equal choices randomly. This equation is similar to the scoring function (5) we used to select branches for the Monte Carlo algorithm, but vector minimaxing now backs up the tree the single vector that achieves this maximum value, rather than using (4) to construct a new vector.

As an example of vector minimaxing in practice, Fig. 17 shows how the algorithm would analyse the tree of Fig. 11, introduced when first discussing strategy fusion in Section 4. The branches selected by (6) (assuming equally likely worlds) have been highlighted in bold. At the root of the tree, the right-hand branch is correctly chosen, in contrast to Monte Carlo sampling, which we saw would pick the left-hand branch.

7.2. *Payoff-reduction minimaxing*

Consider the game represented by the tree of Fig. 18. Against a MIN player with perfect knowledge of the world state in this game, the optimal strategy for MAX is to choose the left-hand branch at nodes a and e. This guarantees a payoff of 1 in world w_1. In the figure, however, we have annotated the tree to show how it is analysed by vector minimaxing. The branches in bold show that the algorithm would choose the right-hand branch at node e. The vector produced at node b correctly indicates that when MAX makes this selection, a MIN player who knows the world state will always be able to restrict MAX to a payoff of 0 (by choosing the right-hand

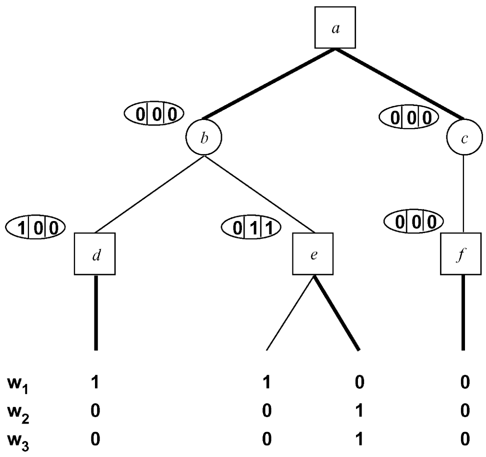


Fig. 18. Example tree with three worlds.

branch at node b in world w_1 and the left-hand branch in worlds w_2 and w_3). Thus, at the root of the tree, both subtrees have the same analysis, and vector minimaxing never wins on this tree. Applying Monte Carlo sampling to the same tree, in the limiting case where all possible worlds are examined, we see that node b has a minimax value of 1 in world w_1, so that the left-hand branch would be selected at the root of the tree. However, the same selection as vector minimaxing will then be made when subsequently playing at node d or node e. Thus, despite both algorithms modelling the situation where MIN has perfect knowledge of the actual world state, neither Monte Carlo sampling nor vector minimaxing choose the best strategy against a MIN player with perfect information on this tree. The choice made at node e is incorrect because the situation in a different (non-local) subtree rooted on node d makes it impossible to actually achieve some of the payoffs under node e.

We introduce here a new algorithm that lessens the impact of non-locality on vector minimaxing by reducing the payoffs at the frontier nodes of a search tree. As in the case of Monte Carlo sampling, the assumption in this algorithm is that MIN plays as well as possible in each individual world. However, this time we implement this assumption by reducing the payoff in any given world w_k to the maximum possible (minimax) return that can be produced when the game tree is examined as a single, perfect information search tree in world w_k. The resulting algorithm, which we call *payoff-reduction minimaxing*, or *prm*, is shown in its simplest form in Fig. 19 (it can be implemented more efficiently, for example by combining steps 2 and 3).

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

Algorithm *prm*(t):

(1) Use the standard minimax algorithm to conduct minimaxing of t in every world w_k. For every MIN node in t, record its minimax value in each world, m_k.
(2) Examine the payoff vectors at each leaf node of t. Reduce the (non-$\perp$) payoffs p_k in each world w_k to the minimum of p_k and all the m_k of the node's MIN ancestors.
(3) Apply the *vector-mm* algorithm to the resulting tree.

Fig. 19. Simple form of the *prm* algorithm.

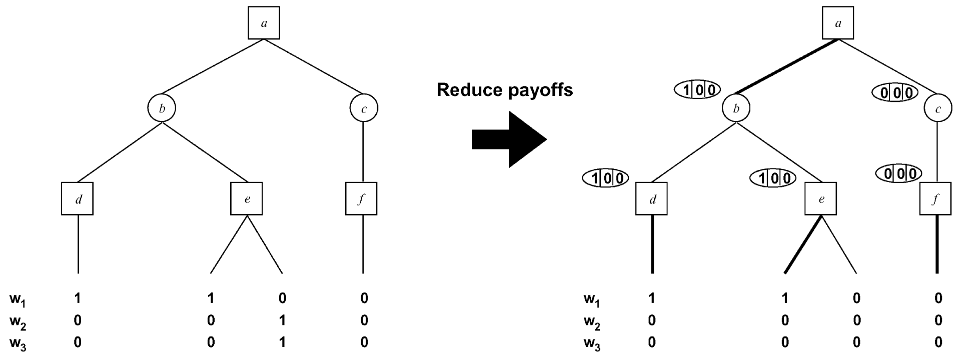


Fig. 20. How *prm* analyses a tree: payoffs are reduced and then *vector-mm* is applied.

As an example, consider how the *prm* algorithm behaves on the tree of Fig. 18. The minimax value of node c is 0 in every world, but all the payoffs at node f are also 0, so no reduction is possible. At node b, however, the minimax values in the three possible worlds are 1, 0, and 0, respectively. Thus, all the payoffs in each world at the leaf nodes under d and e are reduced to at most these values. This leaves only the two payoffs of 1 in world w_1 as shown in Fig. 20, where the strategy selection subsequently made by vector-minimaxing has also been highlighted in bold. In this tree, then, the *prm* algorithm identifies the correct strategy.

Of course, there are still problems that *prm* cannot solve. A simple example of this is the tree of Fig. 11, introduced when first discussing non-locality. For this tree, the minimax value in each world is 1, so no payoffs can be reduced, and *prm* will thus make the same (incorrect) branch choices at nodes c and d as Monte Carlo sampling and vector minimaxing. To solve problems such as this, further heuristics for tackling non-locality are required. We consider two such heuristics in the following sections.

7.3. Beta-reduction

This heuristic takes as its inspiration the well-known procedure of alpha–beta pruning. The alpha–beta technique is used to speed up the search of a perfect information game tree by maintaining cutoff values that are used to decide, based on the search so far, whether a new node can affect the root value of the tree. There are always

Algorithm $vm\text{-}\alpha\beta(t, \boldsymbol{\alpha}, \boldsymbol{\beta})$:

Take the following actions, depending on t.

Condition	Result
t is a leaf node	$payoff\text{-}vector(t)$

t's root is a MIN node	for each $t_i \in sub(t)$ do
	    $\boldsymbol{\beta} \leftarrow \min(\boldsymbol{\beta}, vm\text{-}\alpha\beta(t_i, \boldsymbol{\alpha}, \boldsymbol{\beta})$
	    if $\min(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\beta}$ then return $\boldsymbol{\alpha}$
	end
	return $\boldsymbol{\beta}$

t's root is a MAX node	for each $t_i \in sub(t)$ do
	    $\boldsymbol{\alpha} \leftarrow \max(\boldsymbol{\alpha}, vm\text{-}\alpha\beta(t_i, \boldsymbol{\alpha}, \boldsymbol{\beta})$
	    if $\min(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\beta}$ then return $\boldsymbol{\beta}$
	end
	return $\boldsymbol{\alpha}$

Fig. 21. Vector minimaxing with alpha–beta pruning.

two values: an *alpha* value that can never decrease, and a *beta* value that can never increase.

An extension of this technique to game trees with multiple payoffs at the leaf nodes is shown in Fig. 21. Here, the alpha and beta values are n-tuples and the max and min functions are again as defined in (6) and (3). The min function is also used to represent the pruning criterion, as $\min(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\beta}$. This is a simple expedient for dealing with the possibility that some payoff values may take the undefined value $\perp$.

For perfect information games, the alpha–beta algorithm represents a more efficient technique for computing the same value as standard minimax. With $vm\text{-}\alpha\beta$, however, it is not only efficiency that may be improved, but also *accuracy*. That is, $vm\text{-}\alpha\beta$ will not, in general, return the same value as *vector-mm*. For an illustration of this, consider again the tree of Fig. 18. Fig. 22 shows how this tree is analysed by $vm\text{-}\alpha\beta$. When node d is examined, it produces an alpha value of $(1, 0, 0)$, which then becomes the beta of node b. This beta value is then passed down to node b's next daughter. At node e, the first daughter is a leaf node and the alpha value of node e is therefore set to the leaf node values $(1, 0, 0)$. Now, this alpha value is at least as good as the beta value of b (that is, $\min(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is now equal to $\boldsymbol{\beta}$), so the remaining branches at node e can be pruned (beta pruning). Thus, $vm\text{-}\alpha\beta$ selects the correct strategy on this tree, whereas we saw in Section 7.2 that *vector-mm* is sub-optimal.

The explanation for $vm\text{-}\alpha\beta$'s superiority here is that the beta-pruning at node e in effect tackles the non-local nature of this game tree by preventing the second (sub-optimal) branch at node e from being examined. This is a simple example of a general effect. Non-locality occurs when choices are made at internal MAX nodes without reference to other subtrees. Since pruning decreases the number of MAX nodes that are actually examined, it also decreases the chance that non-local effects will lead to errors.

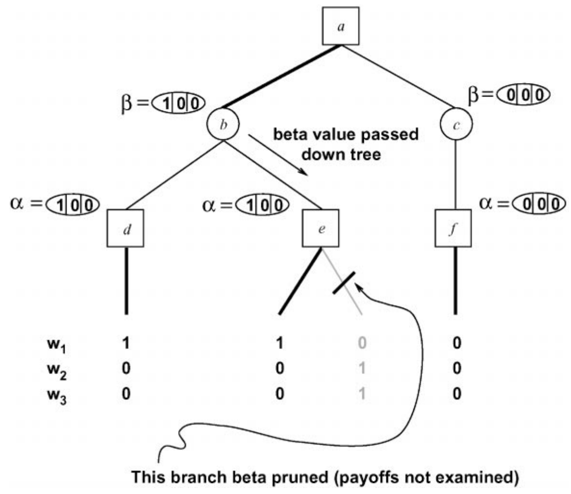


Fig. 22. A beta-pruning carried out by the *vm-αβ* algorithm on a simple tree.

Although *vm-αβ* improves upon *vector-mm*, it is not hard to produce a modified tree for which both algorithms find the same, incorrect solution. For instance, the small change of increasing by one the payoff under node *d* in world w_1 leaves the optimal strategy and its payoff unchanged. However, *vm-αβ* (and also *vector-mm*) will not be able to find this strategy. Even for this modified tree, though, there is an adaptation of the alpha–beta technique that does improve accuracy. To see this, we observe that, when using *vm-αβ*, the branch selections made during a search are constantly reflected in the alpha and beta values passed to any node. These values therefore offer a natural way of tackling non-locality: by ensuring that payoffs rendered unachievable by branch selections in the analysed portion of the tree do not adversely affect the selections in the remainder of the search.

In particular, any beta value, $\boldsymbol{\beta}$, generated at a MIN node, v, can be used to reduce non-locality at MAX nodes in any subtree of v that has yet to be examined. Since MIN chooses the best play in each individual world, each value $\boldsymbol{\beta}[j]$ imposes an upper bound on the value of the optimal payoff that can be obtained in world w_j. Thus, when making a new selection at a MAX node in a subtree of v, all the payoffs of each $\boldsymbol{K}_i$ in world w_j should be *reduced* to at most $\boldsymbol{\beta}[j]$. A simple way to implement this observation is to modify the result returned at leaf nodes in the algorithm of Fig. 21 to the following:

$$\min(\mathit{payoff\text{-}vector}(t), \boldsymbol{\beta}), \tag{7}$$

where min is again as defined in (3). Let us call the algorithm produced by this modification *vm-beta*, and the reductions of leaf node payoffs made by (7) *beta-reductions*. This new algorithm can correctly solve the tree of Fig. 22 even when the payoff in world w_1 at node d is greater than 1, since the payoffs of 1 in worlds w_2 and w_3 are beta-reduced to 0.

Since beta-reduction only utilises information about branches *already selected*, it is sensitive to branch ordering. For example, consider the effect of swapping the order of the branches at node b in Fig. 22. The choice between the vectors $(1,0,0)$ and $(0,1,1)$ at node e would then have to be made *before* realising that the payoffs of 1 in w_2 and w_3 could not be achieved. Thus, no beta-reductions (or, of course, beta prunings) would be possible, and the optimal strategy for this reordered tree would not be found. However, note that *vm-beta* is still correct (unlike *vm-*$\alpha\beta$) if we simply swap the two branches at node e.

In games with perfect information, alpha-beta pruning is also affected by branch ordering, at least in terms of efficiency. The optimal branch ordering is when the moves that are best for MAX come first at MAX nodes, and the moves that are best for MIN come first at MIN nodes. In fact, the same ordering is also the best when *vm-beta* is applied to imperfect information games, but, in terms of accuracy, it is the ordering at MIN nodes that is most important; the earlier in the search that the payoff vectors with relatively small values are encountered, the more likely that beta-reductions will become possible.

Note that the *prm* algorithm can find optimal solutions for the tree of Fig. 22 irrespective of branch ordering. Also, conversely, Fig. 11 demonstrates that *vm-beta* can find optimal solutions for some trees that *prm* cannot. That *prm* and *vm-beta* can find optimal strategies on different trees suggests the creation of a hybrid *prm-beta* algorithm. Such a hybrid is easily produced by simply replacing *vector-mm* with *vm-beta* in step three of the *prm* algorithm of Fig. 19. The insensitivity of the payoff reduction technique to branch ordering allows this hybrid algorithm to benefit fully from an ordering that favours beta-reduction.

7.4. Iterative biasing

In our complexity proof of Section 6, we indicated that when a MAX strategy is fixed, its payoff can be found in time linear in the size of the game tree. This suggests a heuristic for finding good strategies: simply *guess* a strategy (or even a partial strategy) and then check the strategy's payoff. This guessing can be repeated until an answer is demanded, at which point the guess with the best evaluation can be returned. However, given the form of our game trees, we know (see, e.g., [8]) that, when the non-terminal nodes have at least binary branching, the number of strategies for MAX is exponential in the size of tree and thus doubly exponential in its depth. Finding good guesses among so many possibilities is unlikely to be a practical proposition in general.

However, there is something other than strategies that can be guessed: payoffs. In fact, given an optimal payoff vector, $\boldsymbol{K}_{\max}$, we can efficiently find an optimal strategy for MAX. To see this, consider a game tree with payoff vectors $\boldsymbol{K}$ at the leaf nodes. Assume it is known that the optimal payoff vector for this game is $\boldsymbol{K}_{\max}$. We then compute an optimal strategy (which may not be unique, as there could be more than one optimal payoff vector and each such vector could also result from more than one strategy) with the following steps, which run in time linear in the size of the game tree:

(1) Compare the payoff vector $\boldsymbol{K}$ at each leaf with $\boldsymbol{K}_{\max}$. Replace the vector with the integer 1 if $\boldsymbol{K}$ is at least as good for MAX as $\boldsymbol{K}_{\max}$ (that is, if $\min(\boldsymbol{K}, \boldsymbol{K}_{\max}) = \boldsymbol{K}_{\max}$), and 0 otherwise.
(2) The optimal strategy is the one returned for MAX by applying standard minimax to the resulting tree.

That this procedure is correct can be shown by first observing that the minimax step must find a strategy with a payoff of 1. If this was not the case there would be no strategy in the original tree returning a payoff that is at least as good as $\boldsymbol{K}_{\max}$, contradicting that $\boldsymbol{K}_{\max}$ is an optimal payoff. Now observe that a payoff of 1 means that the strategy yields a payoff at least as good as $\boldsymbol{K}_{\max}$ on the original tree.

If we are playing a game where the leaf payoffs come from a finite m-element domain (e.g., natural numbers between 0 and $m-1$), the space of possible payoff vectors has size m^n. Like the total number of strategies, this is exponential, but now the exponent is different: it is the number of worlds n. Thus, whereas guessing strategies may not be practical, guessing payoff vectors may be more feasible. In single-suit Bridge problems, for example, redundancies in the domain often reduce the number of significant worlds to a manageable number (such as the twenty worlds of the problem in Fig. 3, produced by treating the low cards as indistinguishable).

We suggest the basic approach of guessing a *single* element of the optimal payoff vector to be some value v (i.e., guessing that $\boldsymbol{K}_{\max}[k] = v$ for a particular world w_k). This guess can then be passed to a modified version of vector minimaxing that uses it to *bias* the search. This biasing is achieved by defining a new function, $\max_{v,k}$, to replace the definition of (6) in the *vector-mm* algorithm. The $\max_{v,k}$ function returns from amongst a set of vectors the one that is best according to the relation $\underset{v,k}{>}$ defined below.

Definition 1 (*Biasing relation*). For any two payoff vectors, $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$, we say that $\boldsymbol{K}_1 \underset{v,k}{>} \boldsymbol{K}_2$ if and only if either of the following hold:

- the vector $\boldsymbol{K}_1$ offers a payoff of at least v in world w_k, but the vector $\boldsymbol{K}_2$ does not, or
- if *neither* of $\boldsymbol{K}_1$ or $\boldsymbol{K}_2$ offers a payoff of at least v in w_k, or if *both* $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ offer a payoff of at least v in w_k, then $\boldsymbol{K}_1$ must be superior to $\boldsymbol{K}_2$ based on an expected value computation on the remaining worlds. That is,

$$\sum_{\substack{i=1 \\ (i \neq k, \boldsymbol{K}_1[i] \neq \perp)}}^{n} \Pr(\mathsf{w}_i)\boldsymbol{K}_1[i] > \sum_{\substack{i=1 \\ (i \neq k, \boldsymbol{K}_2[i] \neq \perp)}}^{n} \Pr(\mathsf{w}_i)\boldsymbol{K}_2[i].$$

This definition is designed to bias a search so that, wherever possible, a branch with a payoff greater than or equal to v in world w_k is selected. Given some finite set, $\mathscr{S}$,

```
Algorithm ivm(t, $\mathcal{S}$)
Given $\mathcal{S} = \{\langle v_1, k_1\rangle, \langle v_2, k_2\rangle, \cdots\}$,
for each $\langle v_j, k_j\rangle \in \mathcal{S}$
  compute $s_j =$ biased-vm($t, v_j, k_j$)
end
return the $s_j$ that represents the best expected payoff
```

Here, *biased-vm(t,v,k)* takes the following actions, depending on t.

Condition	Result
t is leaf node	*payoff-vector*(t)
root of t is a MIN node	$\min_{t_i \in sub(t)}$ *biased-vm*(t_i, v, k)
root of t is a MAX node	$\max_{v,k}{}_{\, t_i \in sub(t)}$ *biased-vm*(t_i, v, k)

Fig. 23. Iterative biasing, as carried out by the iterative vector minimaxing algorithm.

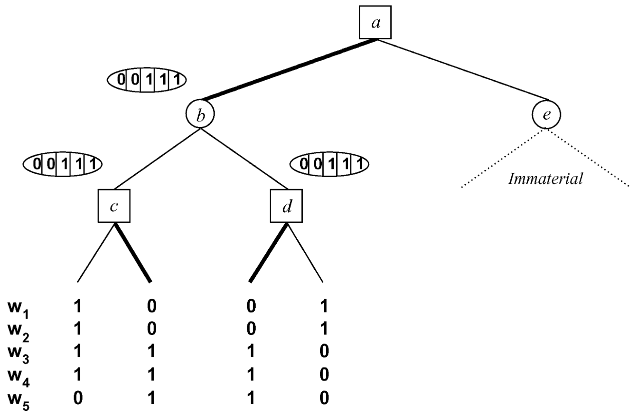


Fig. 24. Correct strategy chosen by iterative biasing with the guess $\langle 1, 5\rangle$.

of guesses for the pair of values $\langle v, k\rangle$, the search can then be repeated with different biases – a technique we call *iterative biasing*. We formalise this as the *iterative vector minimaxing* (or *ivm*) algorithm of Fig. 23.

Iterative biasing enables the *ivm* algorithm to tackle non-locality by, on each iteration, introducing a dependency between *all* MAX selections in a tree. To see that biasing can correctly analyse problems that payoff-reduction and beta-reduction cannot, simply consider the tree of Fig. 24. This tree is a slightly modified version of Fig. 11 produced by altering just two payoffs under node c. In the figure, we have indicated in bold the strategy chosen by *ivm* when using the guess $\langle 1,5\rangle$. This strategy gives a payoff of 1 in the three worlds w_3, w_4, and w_5. For the alternative guesses of $\langle 1,3\rangle$ and $\langle 1,4\rangle$, *ivm* will select the sub-optimal strategy of the left-hand branch at both nodes c and d (a payoff of 1 in just w_3 and w_4). This is also the strategy selected by Monte Carlo sampling, vector minimaxing, *prm* and *vm-beta*. For the guesses of $\langle 1,1\rangle$ and $\langle 1,2\rangle$, *ivm* selects the left-hand branch at c and the right-hand branch at d (a payoff of 1 in just w_1 and w_2).

For trees with binary payoffs, an obvious choice for the set of payoff guesses is $\mathscr{S}=\{\langle 1,1\rangle,\langle 1,2\rangle,\ldots,\langle 1,n\rangle\}$, which guesses the value $v=1$ for each of the n possible worlds. For games where the payoffs can take more than two values, however, we suggest the more general $\mathscr{S}=\{\langle v_{\max},1\rangle,\langle v_{\max},2\rangle,\ldots,\langle v_{\max},n\rangle\}$. Here, $v_{\max}$ is the largest of the (perfect information) minimax values of the root of the game tree in each individual world (such values can be efficiently calculated, as in the first step of the *prm* algorithm, for example). The value of $v_{\max}$ is also an upper bound on the value of any entry in the optimal payoff vector, $\boldsymbol{K}_{\max}$. Thus, such payoff guesses are appropriate for Bridge, where a common task is to identify the strategy with the best chance of producing the maximum possible number of tricks. In fact, a simple efficiency improvement can be made by omitting any guess $\langle v_{\max},k\rangle$ for which the (perfect information) minimax value of the game tree in world w_k is less than $v_{\max}$. This is justified by noting that the value of $\boldsymbol{K}_{\max}[k]$ will never be $v_{\max}$ if even the best possible play in w_k itself cannot produce a payoff of $v_{\max}$ tricks.

7.5. Summary of heuristics

We have shown how the problem of strategy fusion in Monte Carlo sampling can be avoided by the use of the vector minimaxing algorithm. We also introduced the heuristics of payoff-reduction, beta-reduction and iterative biasing, demonstrating how they further addressed the problem of non-locality by introducing dependencies between choices at MAX nodes.

We further noted that beta-reduction could be combined with payoff-reduction to produce the *prm-beta* algorithm. In fact, there are eight possible algorithms that can be produced by combinations of payoff-reduction, beta-reduction, and iteration, as shown in Fig. 25.

In the following section we present test results that demonstrate the practical use of these algorithms. First, however, we give a further intuition on their characteristics by examining how they perform on the tree used for our complexity proof in Fig. 15. The summary in Fig. 26 details the node selections made on this tree by each of the eight algorithms. The *vector-mm* algorithm (and also Monte Carlo sampling) can at best find a 1-clique, by making a fortunate guess at node v_3. Payoff-reduction cannot improve on this, as the minimax value of each individual world is 1. However, both beta-reduction and iterative biasing improve the result, and used together they find the optimal solution.

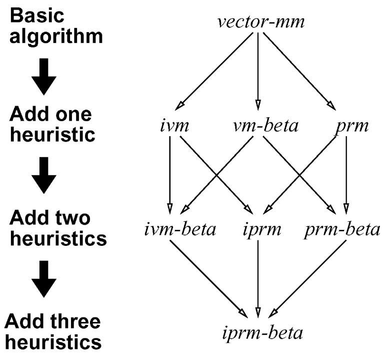


Fig. 25. Possible combinations of heuristics.

Algorithm	Nodes selected	Equivalent clique
vector-mm	v_3 selected with 50% probability.	1-clique or 0-clique
prm	Same as vector minimaxing, since minimax value is 1 in every world.	1-clique or 0-clique
vm-beta, *prm-beta*	If v_3 is selected (again, a 50% chance) v_4 is also selected. If v_3 is not selected, v_4 and v_5 are selected. (If tree is analysed right to left, v_1, v_2 and v_3 are selected.)	2-clique (3-clique)
ivm, *iprm*	For any payoff guess of 1 in w_k, the corresponding v_k will be selected. For the guesses $k=1$, $k=2$, and $k=4$ there is a 50% chance that v_3 is selected.	2-clique or 1-clique
ivm-beta, *iprm-beta*	v_1, v_2 and v_3 only selected if the payoff guess is a 1 in world w_1. All other guesses lead to the selection of just two nodes.	3-clique

Fig. 26. Performance comparison on tree used in complexity proof (see Fig. 15).

8. Further experiments

To demonstrate the practical performance of the algorithms presented in this paper, we carried out further tests on both random game trees and the game of Bridge. The results of these tests are described in this section.

8.1. More experiments on random trees

We first tested the algorithms introduced in the previous section on random trees with payoffs of either 1 or 0 at the leaf nodes. We did this by repeating the test procedure of Section 4, in which Monte Carlo sampling was tested on binary trees

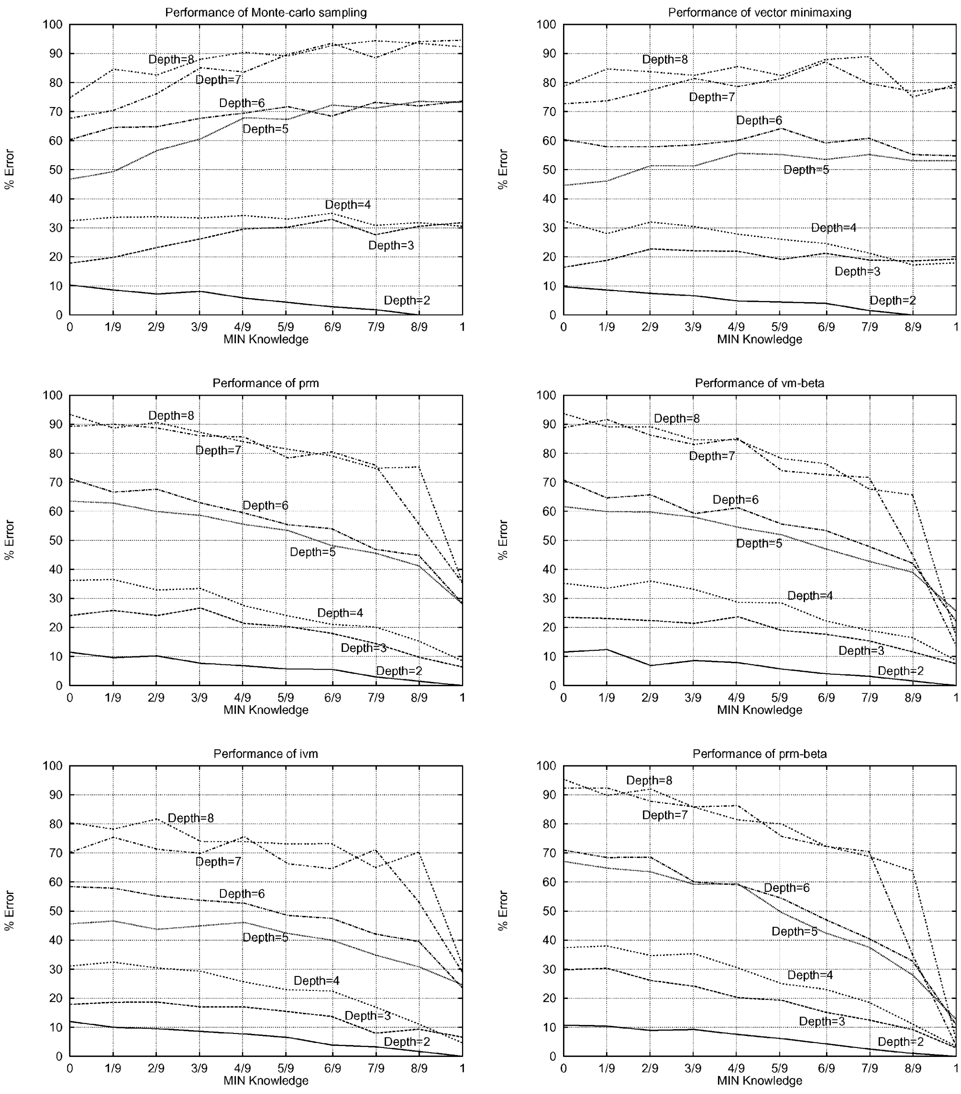


Fig. 27. Comparing the error in Monte Carlo sampling to the error in five algorithms that tackle non-locality and strategy fusion, for trees of depths 2 to 8.

with ten worlds and randomly generated payoffs. The graph of the error rate for Monte Carlo sampling is reproduced in Fig. 27, accompanied by further graphs showing the performance of the new algorithms when tested in the same way. For the iterative biasing heuristic, the set of payoff guesses consisted of the worlds for which the perfect information minimax value of the game tree was equal to 1, as described at the end of Section 7.4.

The results of Fig. 27 demonstrate that vector minimaxing out-performs Monte Carlo sampling by a small amount, for almost all levels of MIN knowledge and tree depths. This is due to the removal of strategy fusion. However, even without strategy fusion the problem of non-locality remains, to the extent that the performance of vector minimaxing is only slightly superior to Monte Carlo sampling. A far more dramatic improvement is therefore produced by the payoff-reduction, beta-reduction and iterative biasing heuristics. These heuristics (as embodied by the algorithms *prm*, *vm-beta* and *ivm*) remove strategy fusion and further reduce the error caused by non-locality.

As would be expected, when MIN has no knowledge of the state of the world (the left-hand side of the graphs), all the algorithms have significant error rates, because they all carry the assumption that MIN plays as well as possible in each world. However, since there is a simple algorithm that correctly models a MIN player with no knowledge (the minimaxing of expected values), this portion of the graphs is of relatively little interest. In fact, the *prm* and *vm-beta* algorithms are actually *worse* than Monte Carlo sampling at this point of the graphs because of their improved modelling of the assumption that MIN will play as well as possible in each world. However, as MIN's knowledge increases, this assumption becomes more accurate, until for levels of knowledge of about 5/9 and above, the algorithms out-perform both Monte Carlo sampling and vector minimaxing. The *ivm* algorithm, on the other hand, does not modify payoffs and so is uniformly equivalent to or better than vector minimaxing.

When MIN's knowledge of the world state is 1 the performance advantage of the non-locality heuristics is particularly marked, with the error of *prm* and *ivm* for trees of depth 8 being around a third of the error rate of Monte Carlo sampling. The performance of *vm-beta* and the composite algorithm *prm-beta* deserve special explanation, since trees of depths 7 and 8 are easier to solve than trees of depths 5 and 6. This is because, for our test trees and against a MIN player with a high level of knowledge of the world state, as the depth of the tree increases the optimal strategy becomes more and likely to be one that gives a payoff of 1 in just a single world. With beta-reduction, a constant update of the best possible current result is always reflected in the beta value passed down the tree. Once this beta value is reduced to a vector with a single 1, branch selection is then focused solely on the payoffs in the corresponding world, removing completely the possibility of non-locality.

8.2. More experiments on the game of Bridge

To test our new algorithms on a real imperfect information game, we returned to the Bridge test set described in Section 5. Just as we did for Monte Carlo sampling, we implemented each algorithm within the framework of the FINESSE Bridge-playing system [6, 9], and compared the expected payoff of the resulting strategies (for the maximum possible number of tricks) with the expected payoff of the solution given in the Encyclopedia. For the iterative algorithms, the set of payoff guesses was produced by finding the worlds where the perfect information minimax value of the game tree equalled the maximum possible number of tricks. The results are summarised in Fig. 28.

As in our tests on random trees, vector minimaxing is again slightly more accurate than Monte Carlo sampling, and correctness further improves as heuristics are added.

Algorithm	Optimal	Sub-optimal	Expected loss	Time
Monte Carlo	430 (66.2%)	220 (33.8%)	17.00	8.1
vector-mm	460 (71.8%)	190 (28.2%)	12.81	3.8
vm-beta	555 (85.4%)	95 (14.6%)	6.24	4.3
ivm	613 (94.3%)	37 (5.7%)	1.61	25.5
prm	622 (95.7%)	28 (4.3%)	0.86	15.5
prm-beta	638 (98.2%)	11 (1.8%)	0.34	19.6
ivm-beta	645 (99.2%)	5 (0.8%)	0.23	96.3
iprm	645 (99.2%)	5 (0.8%)	0.13	104
iprm-beta	648 (99.7%)	2 (0.3%)	0.06	101

Fig. 28. Performance on the 650-problem test set from the Encyclopedia of Bridge.

The most effective individual heuristic is payoff-reduction (*prm* outperforms both *ivm* and *vm-beta*).

When payoff-reduction, beta-reduction and iterative biasing are all combined in the *iprm-beta* algorithm, sub-optimal strategies are only generated for two problems. Given that our algorithms also produced *better* strategies than the Encyclopedia on six problems, however, this performance (and also that of *ivm-beta* and *iprm*) is actually above the level of the human experts that produced the model solutions.[4] In fact, we traced the cause of *iprm-beta*'s two errors to a problem with Finesse itself that resulted in the optimal strategies not actually being present in the search space. We intend to correct this design error in the near future.

The results of Fig. 28 also include an 'Expected Loss' column, which states how often the sub-optimal strategies produced by each algorithm can be expected to result in inferior performance. These figures measure the expected number of times that the Encyclopedia's strategies would out-perform each algorithm when playing the entire set of 650 problems once (against best defence and with a random choice among the possible holdings for the defence). The values are produced by summing, over every problem in the test set, the chance of success of the Encyclopedia's strategy minus the chance of success of the strategy produced by the algorithm in question. When measured by expected loss, the superiority of *iprm-beta* over Monte Carlo sampling or *vector-mm* is less marked. However, note that there is at least one task for which optimality is a crucial factor, namely the creation of tutoring systems where a computer

[4] The six problems for which our algorithms find strategies with a better chance of success than the Encyclopedia's are numbers 289, 477, 543, 568, 601, and 622. We also discovered two cases where the Encyclopedia's probability calculation is wrong (numbers 31 and 430), and one simple typo (number 609). Further, there were also seven problems where the Encyclopedia's given solution made unstated assumptions about the defence playing sub-optimally (numbers 437, 459, 541, 548, 590, 591, and 595). To produce the chance of success stated in the Encyclopedia for these problems, the lines of play have to be analysed under the assumption that the defenders will make some kind of mistake. If the defenders do not make a mistake (i.e., they play according to the best defence model), the line of play stated in the Encyclopedia actually has a chance of success lower than those produced by our algorithms.

must generate (and perhaps even explain) the best way to play a game. One natural application of *iprm-beta*, therefore, is as the basis for such a system.

The 'Time' column gives the average number of seconds required for a single problem (on a Sun SPARCstation running at 200 MHz). We have not paid particular attention to the efficiency of our implementations (for example, none of the beta-reduction algorithms actually incorporate pruning to speed up the search). Nevertheless the speeds are acceptable, with *prm-beta*, in particular, offering a good trade-off of accuracy against speed. The iterative algorithms may appear particularly slow, but note that they can all be used in 'any-time' fashion, returning the best result encountered so far when available time is exhausted.

8.3. Relating the results

The performance of all of our algorithms is better on Bridge than on random game trees. The main reason for this is that game trees resulting from Bridge are not random: payoffs in different worlds are often correlated and plays that work well in one world often work well in others. To investigate this, and to better understand the relationship between game structure and algorithm performance, we conducted one further experiment. The aim of this experiment was to modify the payoffs of our game trees so that each algorithm could identify optimal strategies with the same success rate as in Bridge. We achieved this by the simple expedient of parameterising the test trees with a probability, q, that determines how similar the possible worlds are. To generate a tree with n worlds and a given value of q:

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

Trees with a higher value of q tend to be easier to solve, because an optimal strategy in one world is also more likely to be an optimal strategy in another. Correspondingly, we found that by modifying q it was possible to improve the performance of each algorithm. For example, Fig. 29 shows the error surface of Monte Carlo sampling for different values of q.

What was unexpected, however, was that the value of q for which each algorithm performed at the same level as in Bridge roughly coincided, at $q \approx 0.75$. For this value, the error rates obtained were as shown in Fig. 30. Thus, on two different types of game we have found that the relative strengths of the algorithms coincide. With this observation, the conclusion that similar results will hold for other imperfect information games becomes more sustainable.

9. Conclusions and future work

We have investigated the problem of finding optimal strategies for games with imperfect information. We examined the performance of Monte Carlo sampling on simple

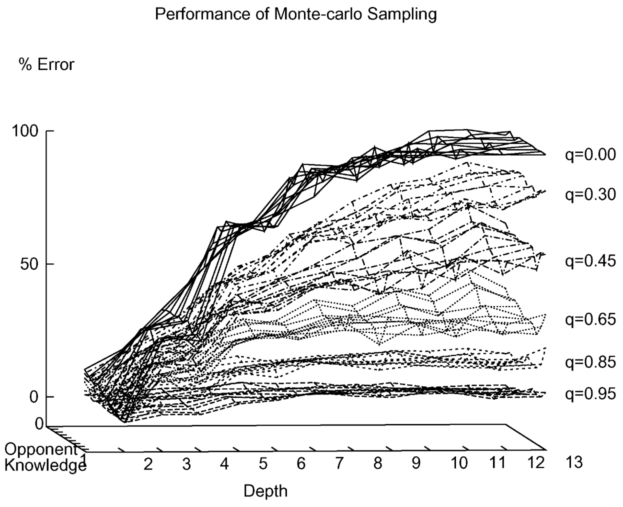


Fig. 29. The error surface of Monte Carlo sampling on binary trees with 10 possible worlds for different values of q. The plot is for trees of depth between 2 and 13 (y-axis), for varying levels of knowledge held by the opponent (x-axis, ranges from zero to perfect knowledge). One thousand tests per data point.

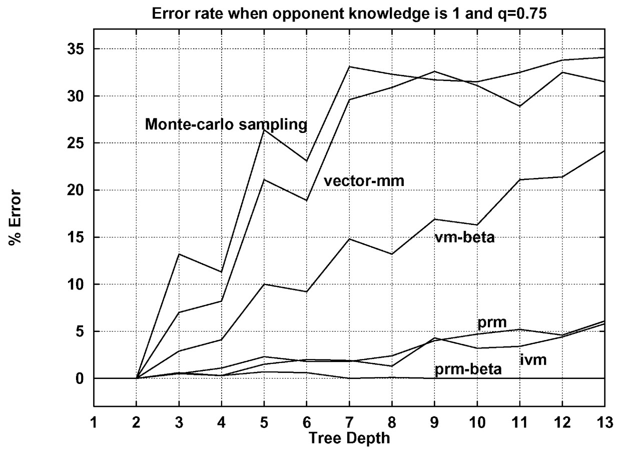


Fig. 30. Algorithm performance on random game trees where the optimal strategy in one world is more likely to be optimal in another.

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 and non-locality.

We showed that strategy fusion and non-locality also affect the analysis of Bridge, despite the way that Monte Carlo sampling largely reflects the *best defence* assumptions commonly made when analysing Bridge problems. We then demonstrated that finding optimal play against best defence is NP-complete in the size of the game tree, and introduced the new heuristics of *vector minimaxing*, *payoff-reduction minimaxing*, *beta-reduction* and *iterative biasing*. We presented test results that demonstrated the effectiveness of these heuristics, particularly when combined together to produce the *iprm-beta* algorithm. On our database of problems from the game of Bridge, *iprm-beta* actually makes fewer errors than the human experts that produced the model solutions. It thus represents the first general search algorithm capable of consistently performing either at, or above, expert level on a significant aspect of Bridge card-play.

In the long-term, we are looking at how our algorithms can be applied to larger, real-world games. We are also investigating algorithms that solve weakened forms of the best defence model, for example taking advantage of possible mistakes made by less-than-perfect opponents. A short-term goal, though, is to use the *iprm-beta* algorithm – as it stands – as the basis of a true expert-level tutor for single-suit Bridge play.

References

[1] ACBL, The Official Encyclopedia of Bridge, 5th Edition, American Contract Bridge League, Inc., 2990 Airways Boulevard, Memphis, TN 38116-3875, 1994.

[2] J.S. Blair, D. Mutchler, M. van Lent, Perfect recall and pruning in games with imperfect information, Comput. Intelligence (1996).

[3] R.A. Corlett, S.J. Todd, A Monte-Carlo approach to uncertain inference, in: P. Ross (Ed.), Proc. Confe. on Artificial Intelligence and Simulation of Behaviour, 1985, pp. 28–34.

[4] E. Crowhurst, Personal communication, May 17 1996.

[5] A. Frank, Brute force search in games of imperfect information, in: D.N.L. Levy, D.F. Beal (Eds.), Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad, Ellis Horwood, Chichester, UK, 1989,, pp. 204–209.

[6] I. Frank, Search and planning under incomplete information: a study using bridge card play, Ph.D. Thesis, Department of Artificial Intelligence, Edinburgh, 1996, also published by Springer-Verlag in the Distinguished Dissertations Series, ISBN 3-540-76257-4, 1998.

[7] I. Frank, D. Basin, Optimal play against best defence: complexity and heuristics, in: H.J. van den Herik, H. Iida (Eds.), Proc. 1st Internat. Conf. of Computers and Games, CG98, Lecture Notes in Computer Science, Vol. 1558, Tsukuba, Japan, Springer, Berlin, 1998, pp. 50–73, ISBN 3-540-65766-5.

[8] I. Frank, D. Basin, Search in games with incomplete information: a case study using bridge card play, Artifi. Intell. 100 (1–2) (1998) 87–123.

[9] I. Frank, D. Basin, A. Bundy, An adaptation of proof-planning to declarer play in bridge, Proc. ECAI-92, Vienna, Austria, 1992, pp. 72–76, longer version available from Edinburgh as DAI Research Paper No. 575.

[10] I. Frank, D. Basin, H. Matsubara, Monte-carlo sampling in games with incomplete information: empirical investigation and analysis, Workshop on Game Tree Search: Past, Present and Future, Princeton, NJ, 1997, also available from the Electrotechnical Laboratory, 1-1-4 Umezono, Japan, as Technical Report ETL-97-18.

[11] I. Frank, D. Basin, H. Matsubara, Finding optimal strategies for imperfect information games, Proc. AAAI-98, 1998, pp. 500–507.

[12] D. Fudenberg, J. Tirole, Game Theory, MIT Press, Cambridge, MA, 1995.

[13] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness, W H Freeman, New York, 1979.

[14] M. Ginsberg, Partition search, Proc. AAAI-96, 1996, pp. 228–233.

[15] C.H. Goren, Goren's New Bridge Complete, Century Hutchinson Limited, 1986.

[16] D. Koller, N. Meggido, B. von Stengel, Efficient computation of equilibria for extensive two-person games, Games Econom Behav 14 (2) (1996) 247–259.

[17] D.N.L. Levy, The million pound bridge program, in: D.N.L. Levy, D.F. Beal (Eds.), Heuristic Programming in Artificial Intelligence – The First Computer Olympiad, Ellis Horwood, Chichester, UK, 1989, pp. 95–103.

[18] R. Duncan Luce, Howard Raiffa, Games and Decisions – Introduction and Critical Survey, Wiley, New York, 1957.
[19] D.S. Nau, The last player theorem, Artif. Intell. 18 (1982) 53–65.
[20] Y. Nygate, L. Sterling, Python: an expert squeezer, J. Logic Programming 8 (1,2) (1990).
[21] C.E. Shannon, Programming a computer for playing chess, Philos. Mag. 41 (1950) 256–275.