# Optimal Play Against Best Defence: Complexity and Heuristics

Ian Frank[1] and David Basin[2]

[1] Complex Games Lab, ETL, Umezono 1-1-4, Tsukuba, Ibaraki, Japan 305
ianf@etl.go.jp
[2] Institut für Informatik, Universität Freiburg, Am Flughafen 17, Freiburg, Germany
basin@informatik.uni-freiburg.de

**Abstract.** We investigate the best defence model of an imperfect information game. In particular, we prove that finding optimal strategies for this model is NP-complete in the size of the game tree. We then introduce two new heuristics for this problem and show that they out-perform previous algorithms. We demonstrate the practical use and effectiveness of these heuristics by testing them on random game trees and on a hard set of problems from the game of Bridge. For the Bridge problem set, our heuristics actually out-perform the human experts that produced the model solutions.
**Keywords**: imperfect information, heuristics, complexity, Bridge

## 1 Introduction

We analyse the problem of finding optimal strategies for two player games of the form illustrated in Figure 1. In this game, played between MAX (square nodes) and MIN (circular nodes), the terminal values indicate the payoff to MAX of reaching each leaf node under any of five possible *worlds* $\mathtt{w}_1, \cdots, \mathtt{w}_5$. The state of the world that actually holds depends on information that MAX does not know, but to which he can attach a probability distribution (for example the toss of a coin or the deal of a deck of cards). For a more general game with $n$ possible worlds, every leaf node of the tree has $n$ payoffs, each corresponding to the utility for MAX of reaching that node in one of the $n$ worlds.

An example of a real-life game that can be fit to the model of Figure 1 is Bridge (for a concrete illustration, see §2). Recently, the assumptions used by human experts to analyse Bridge problems have been formalised in a *best defence model* [6], 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 described as 'best defence' because it represents the strongest possible assumptions about the opponent — that MIN knows the actual state of the world (A-I) and chooses his strategy in the knowledge of MAX's choices (A-II). It is used by human players because modelling the strongest possible

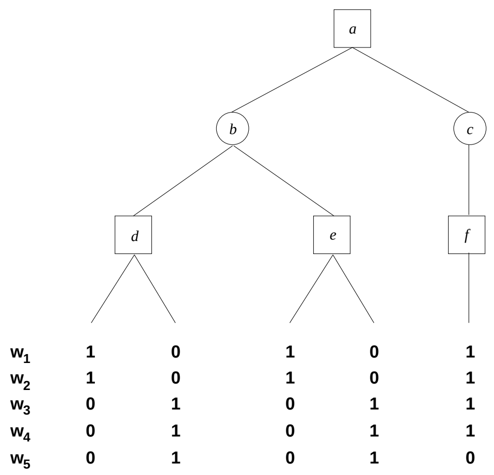


**Fig. 1.** A game tree with five possible worlds

opponents provides a lower bound on the payoff that can be expected when the opponents are less informed. 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.

In this paper we review existing algorithms for analysing trees such as that of Figure 1. These algorithms are all heuristic in nature. To show that such heuristics are the best that can realistically be hoped for, we examine the complexity of the problem and demonstrate that finding optimal strategies for the best defence model is NP-complete in the size of the game tree. This means that arbitrary (imperfect information) *trees* are hard to analyse. This is in contrast to perfect information games, where arbitrary trees are easy to analyse (the minimax algorithm returns results in time linear in the size of the tree), but arbitrary *games* are hard to analyse, at least when their definitions generate trees of exponential size.

To combat such intractability, we introduce two new heuristics, which we call *beta-reduction* and *iterative biasing*. Both of these tackle the *non-local* nature [6] of the types of game shown in Figure 1 by introducing dependencies between the choices made at MAX nodes. We demonstrate the practical importance of our new heuristics theoretically and experimentally, using both simple game trees and a large database of problems from the game of Bridge. For the Bridge database, we find that a combination of our heuristics actually outperforms 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 [16], but our heuristics represent the first general tree search algorithm capable of consistently performing at and above expert level in actual card play.

## 2 A Bridge Example

To show how problems arising in real-life games can be represented in the form of Figure 1, consider the example tree of Figure 2. This depicts a simple situation in a single suit of a game of Bridge.

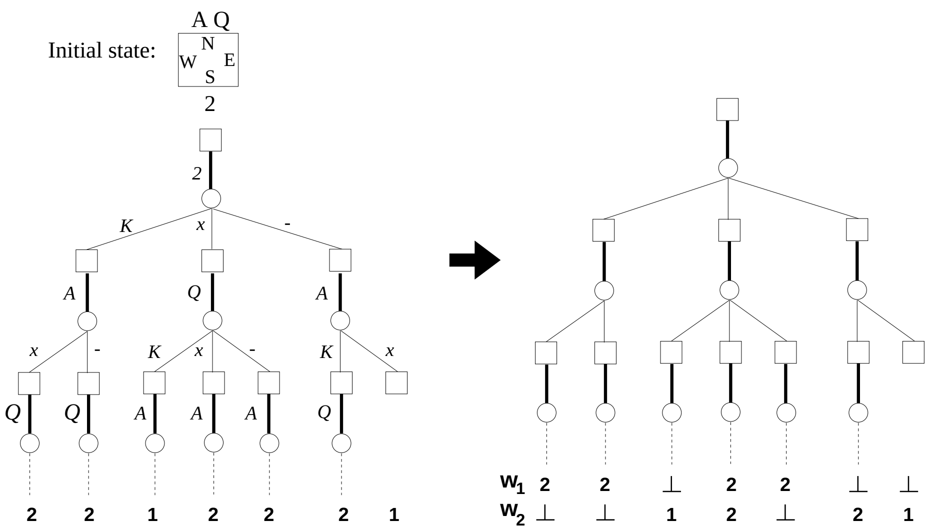


**Fig. 2.** The optimal MAX strategy for a single-suit Bridge problem represented (left) using constraints on the MIN branches, and (right) using possible worlds at leaf nodes

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). It is also assumed 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).

To produce the tree of possible moves in Figure 2, we have made the natural simplification (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 ('-'). Even then, however, the complete tree for this problem has 76 leaf nodes, so we have also made the further simplification of displaying only the MAX branches that form part of the optimal strategy. This optimal strategy 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 to date 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 by cashing the Ace or the Queen, or will have no further options. The total number of North-South tricks 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 the 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 again simplified the presentation by instead considering the two mutually exclusive possibilities of "West holds the King" ($\mathsf{w}_1$) and "East holds the King" ($\mathsf{w}_2$). On the right 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 represent branches that cannot be followed. This 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 Figure 2 is now in a format similar to that of Figure 1. Indeed, the only substantial difference is that some of the payoffs take the undefined value $\perp$. In terms of basic game theory [18, 14], both this tree and the tree of Figure 1 are just compact ways of representing the extensive form of a particular kind of two-person, zero-sum game with imperfect information. For the example of Figure 1, the extensive form tree would have a single chance move at the root, and $n = 5$ identically shaped subtrees (the payoff $n$-tuples can be assigned at each leaf node so that the $i$th component is the payoff for that leaf in the $i$th subtree). 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, giving rise to some $\perp$ values as payoffs.

So, game trees like those given in Figures 1 and 2 can be used to represent interesting and nontrivial games. The rest of this paper will therefore investigate the complexity of playing such games optimally, and introduce new algorithms for finding optimal strategies. Throughout our analysis of such games, we will assume that the only move with an unknown outcome is the chance move that starts the game. Thus, a single node in one of our trees represents between 1 and $n$ *information sets*: one if the player whose turn it is to move knows the exact outcome of the chance move, and $n$ if the player has no knowledge. Since our analysis will make repeated reference to algorithms such as minimaxing, we will make one further notational convention. Throughout the paper, the normal min and max functions will be extended 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 Review of Existing Algorithms

The play of imperfect information games has been analysed by a number of researchers [4, 17, 12]. Here, we review the well-established technique of Monte-carlo sampling, and then describe two algorithms that we have recently demonstrated [8] are better at finding optimal strategies. This section contains no new results, but since the developments in this field are relatively recent, we summarise them in some detail.

### 3.1 Monte-carlo Sampling

One technique for handling imperfect information is Monte-carlo sampling [3]. This approach has been used in games such as Scrabble (see [4]) and Bridge, where it was proposed by Levy [13] and recently implemented by Ginsberg [10]. In the context of game trees like that of Figure 1, Monte-carlo sampling consists of guessing a possible world and then finding a solution to the game tree for this complete information sub-problem. This is much easier than solving the original game, since if attention is restricted to just one world, the minimax algorithm can be used to find the best strategy. By guessing different worlds and repeating this process, it is hoped that an action that works well in a large number of worlds can be identified.

To make this description more concrete, let us consider a general MAX node with branches $M_1, M_2, \cdots$ in a game with $n$ worlds. Now, let us say that we can find the minimax value, $e_{ij}$, of the node under branch $M_i$ in world $\mathsf{w}_j$ with the minimax algorithm. We can then construct a scoring function, $f$, such as:

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

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

As an example, consider how the tree of Figure 1 is analysed by the above characterisation of Monte-carlo sampling. If we examine world $\mathsf{w}_1$, the minimax values below node $a$ are as shown in Figure 3 (these correspond to $e_{11}$ and $e_{21}$ for this tree). It is easy to check that the minimax value at node $b$ is again 1 if we examine any of the remaining worlds, and that the value at node $c$ is 1 in worlds $\mathsf{w}_2$, $\mathsf{w}_3$, and $\mathsf{w}_4$, but 0 in world $\mathsf{w}_5$. Thus, if we assume equally likely worlds (that is, the function Pr returns 1/5 for each world), Monte-carlo sampling using (1) to make its branch selection will choose the left-hand branch at node $a$ whenever world $\mathsf{w}_5$ is included in its sample.

Unfortunately, this is not the best strategy for this tree. The best return that MAX can hope for when making this choice of the left-hand branch at

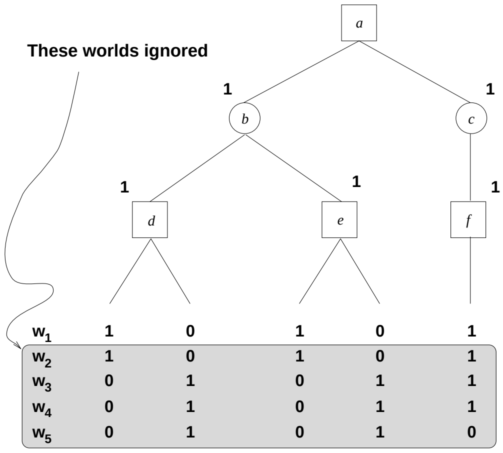


**Fig. 3.** Finding the minimax value of world $w_1$

node $a$ is a payoff of 1 in just three worlds (under the best defence model, and indeed for any reasonable assumptions about a rational opponent). Choosing the right-hand branch at node $a$, however, offers a payoff of 1 in four worlds.

The reason for this error can be understood by examining the implicit assumptions made by a Monte-carlo sampling approach. At first sight, the algorithm appears to model very closely the best defence assumptions given in the Introduction. For instance, it identifies pure MAX strategies that make no use of probabilities (A-III). Moreover, by repeatedly applying minimaxing it is assumed 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).

However, as well as perfect knowledge for MIN, the repeated application of minimaxing also makes the assumption that MAX will be able to play optimally in each individual world. It is this false assumption that leads to incorrect analyses such as that in Figure 3. At nodes $d$ and $e$ in this example, MAX makes *different choices* in *different worlds*. In reality, however, MAX can only attach probabilities to the state of the world and must make a *single choice* for all worlds at node $d$ and another single choice for all worlds at node $e$. Combining the minimax values of separate choices results in an over-optimistic analysis of node $b$. This is an example of the problem of *strategy fusion* formalised recently in [6]: the false assumption that MAX can play optimally in each world allows the results of different moves — or strategies — to be 'fused' together.

### 3.2 Vector Minimaxing

In previous work [8], we have shown how to remove strategy fusion from Monte-carlo sampling with a *vector minimaxing* algorithm. This algorithm ensures that at any MAX node a single branch is chosen in all worlds. To do this, it requires a function *payoff-vector*, defined over the leaf nodes of the game tree. For any leaf-node, $\nu$, *payoff-vector*$(\nu)$ returns an $n$-element vector $\mathbf{K}$ such that $\mathbf{K}[j]$ (the $j$th element of $\mathbf{K}$) takes the value of the payoff at $\nu$ in world $\mathsf{w}_j$ $(1 \leq j \leq n)$. Vector minimaxing then uses these payoff vectors as shown in Figure 4 to identify a strategy for a tree $t$, where $sub(t)$ computes the set of $t$'s immediate subtrees.

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

| Condition | Result |
|---|---|
| $t$ is leaf node | *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. 4.** The vector minimaxing algorithm

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

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

is maximum, resolving equal choices randomly. In this way, vector minimaxing commits to just *one* choice of branch at each MAX node, avoiding strategy fusion (the actual strategy selected by the algorithm is just the set of the choices made at the MAX nodes). As for the min function, for a node with $m$ branches and therefore $m$ payoff vectors $\mathbf{K}_1, \cdots, \mathbf{K}_m$ to choose between, a player with perfect knowledge of the state of the world is modelled as follows:

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

That is, the min function returns a vector in which the payoff for each possible world is the lowest possible.

As an example of vector minimaxing in practice, Figure 5 shows how the algorithm would analyse the tree of Figure 1, using ovals to represent the vectors produced at each node. The branches selected by (2) (assuming equally likely worlds) have been highlighted in bold. At node $d$, for example, the right-hand branch is chosen because its evaluation of 3/5 is higher than the 2/5 of the left-hand branch. Eventually, at the root of the tree, the right-hand branch is correctly chosen.

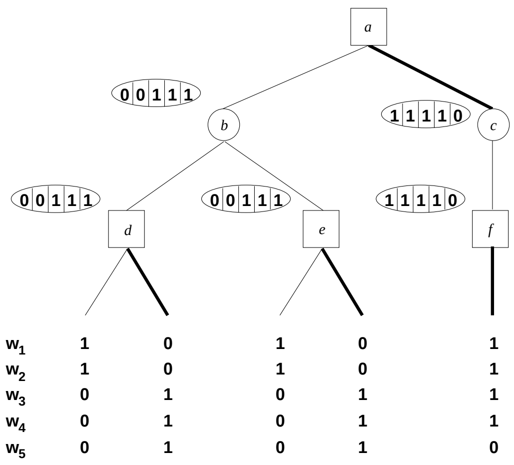


**Fig. 5.** Vector minimaxing applied to example tree

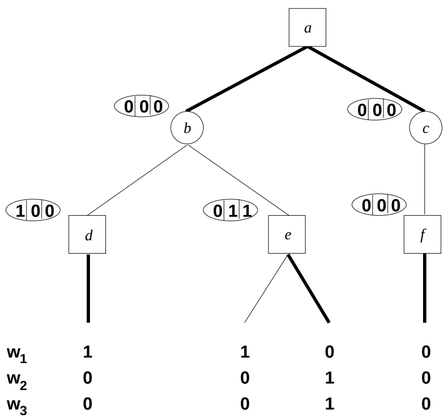


**Fig. 6.** Example tree with three worlds

### 3.3 Payoff-reduction Minimaxing

Vector minimaxing offers slight accuracy improvements over Monte-carlo sampling (again, see [8]). However, both Monte-carlo sampling and vector minimaxing suffer from the problem of *non-locality* [6]. This problem is illustrated in Figure 6, which depicts a game tree with just 3 worlds.

Against best defence, the optimal strategy for MAX is to choose the left-hand branch at both node $a$ and node $e$. This guarantees a payoff of 1 in world $\mathsf{w}_1$. 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 branch at node $b$ in world $\mathsf{w}_1$ and the left-hand branch in worlds $\mathsf{w}_2$ and $\mathsf{w}_3$). 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 $\mathsf{w}_1$, 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, neither Monte-carlo sampling nor vector minimaxing choose the best strategy against best defence 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 $e$.

To tackle the problem of non-locality, we have previously described [8] an algorithm called *payoff-reduction minimaxing*, or *prm*. This algorithm is shown in its simplest form in Figure 7. 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.

As an example, consider how the *prm* algorithm would behave on the tree of Figure 6. The minimax value of node $c$ is zero in every world, but all the payoffs at node $f$ are also zero, so no reduction is possible. At node $b$, however, the minimax values in the three possible worlds are 1, 0, and 0, respectively. Thus, all the payoffs in each world at the leaf nodes under $d$ and $e$ are reduced to at most these values. This leaves only the two payoffs of 1 in world $\mathsf{w}_1$ as shown in Figure 8, where the strategy selection subsequently made by vector-minimaxing has also been highlighted in bold. In this tree, then, the *prm* algorithm results in the correct strategy being chosen.

**Algorithm** *prm*$(t)$:

1. Use the standard minimax algorithm to conduct minimaxing of $t$ in every world $\mathsf{w}_k$. 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 $\mathsf{w}_k$ 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. 7.** Simple form of the *prm* algorithm

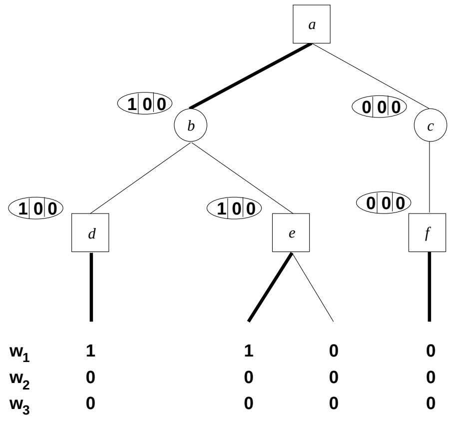


**Fig. 8.** Applying *vector-mm* after payoff reduction

## 4 Complexity Analysis of Optimal Play

Here we show that, given a game tree, finding optimal strategies under the best defence model is an NP-complete problem. Hence, unless P = NP, heuristics are required to tackle this problem in practice. As explained in the Introduction, this is fundamentally different from perfect information games (such as $n$-by-$n$ checkers) which, although PSPACE-hard in the size of the initial game configuration, can be solved in linear time in the size of the game tree [9]. In contrast, our results show that for the problems we are considering, even if the complete game tree of a game 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 [11, 2], but these proofs are not applicable when considering the best defence model. For example, [2] show that it is NP-complete to determine whether or not 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 Figure 1. Moreover, the notion of a pure strategy equilibrium 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 [9].

Best Defence:
*Instance:* A game tree $t$ over $n$ worlds and a positive integer $k \leq n$.
*Question:* Is there is 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 $k \leq |V|$.
*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 [9], we now prove:

**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 *vector-mm*. The only modification required is at MAX nodes, where, rather than using the max operator of (2), the payoff on the branch specified by $s$ is returned. [6] have shown that this algorithm correctly computes the payoff of $s$, and the time taken is clearly linear in the size of the tree.

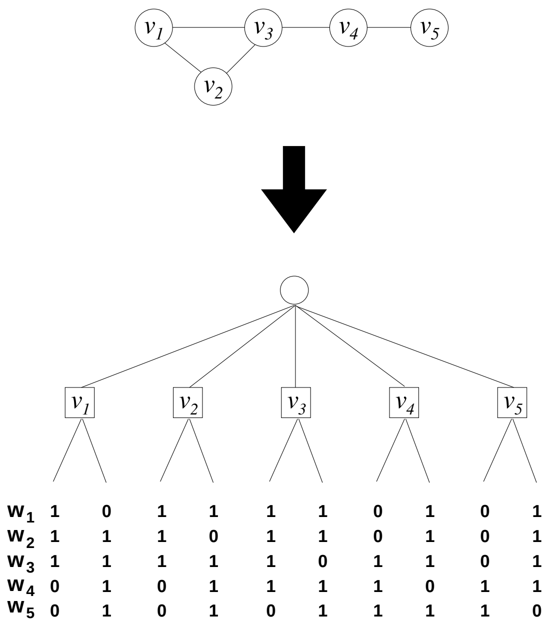


**Fig. 9.** Graph (top) and corresponding game tree (bottom)

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, $\mathsf{w}_1, \cdots, \mathsf{w}_n$, 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 $v_1$, ..., $v_n$, 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 $j$th 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 $j$th world iff $i \neq j$. An example of a graph and its translation are given in Figure 9. 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 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 $\mathsf{w}_i$, 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 $\mathsf{w}_i$, which implies that $(v_i, v_j) \in E$. *QED*

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 $\mathsf{w}_1$, $\mathsf{w}_2$ and $\mathsf{w}_3$, and a 0 in the remaining two.

## 5 New Heuristics

We have shown that the problem of finding optimal strategies for the best defence model is NP-complete in the size of the game tree. To date, the most consistently accurate heuristic solution has been the *prm* algorithm, reviewed in §3.3. Here, we introduce two new heuristics, both with similar motivations to that of *prm*: the reduction of non-locality through the introduction of dependencies between the choices at MAX nodes.

### 5.1 Beta-reduction (and Branch Ordering)

Our first new 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 two values: an *alpha* value that can never decrease, and a *beta* value that can never increase. A simple extension of this technique to game trees with multiple payoffs at the leaf nodes is shown in Figure 10. Here, the alpha and beta values are $n$-tuples and the max and min functions are as defined in (2) 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 $\perp$.

**Algorithm** *vm*-$\alpha\beta(t, \boldsymbol{\alpha}, \boldsymbol{\beta})$:
Take the following actions, depending on $t$.

| Condition | Result |
|---|---|
| $t$ is a leaf node | *payoff-vector*$(t)$ |
| root of $t$ is a MIN node | for each $t_i \in sub(t)$ do<br>$\boldsymbol{\beta} \leftarrow \min(\boldsymbol{\beta}, vm\text{-}\alpha\beta(t_i, \boldsymbol{\alpha}, \boldsymbol{\beta})$<br>if $\min(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\beta}$ then return $\boldsymbol{\alpha}$<br>end<br>return $\boldsymbol{\beta}$ |
| root of $t$ is a MAX node | for each $t_i \in sub(t)$ do<br>$\boldsymbol{\alpha} \leftarrow \max(\boldsymbol{\alpha}, vm\text{-}\alpha\beta(t_i, \boldsymbol{\alpha}, \boldsymbol{\beta})$<br>if $\min(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\beta}$ then return $\boldsymbol{\beta}$<br>end<br>return $\boldsymbol{\alpha}$ |

**Fig. 10.** Vector minimaxing with alpha-beta pruning

For perfect information games, the alpha-beta algorithm represents a more efficient technique for computing the same value as standard minimax. With *vm*-$\alpha\beta$, however, it is not only efficiency that may be improved, but also *accuracy*. That is, *vm*-$\alpha\beta$ will not, in general, return the same value as *vector-mm*. For an illustration of this, let us look again at the tree of Figure 6, introduced in §3.3. Figure 11 shows how this tree is analysed by *vm*-$\alpha\beta$. 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*-$\alpha\beta$ selects the correct strategy on this tree, whereas we saw in §3.3 that *vector-mm* is sub-optimal.

The explanation for *vm*-$\alpha\beta$'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.

Although *vm*-$\alpha\beta$ represents an improvement over *vector-mm* on this particular example, 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 $\mathsf{w}_1$ leaves the optimal strategy and its payoff unchanged. However, *vm*-$\alpha\beta$ (and also *vector-mm*) will not be able to find this strategy. Even for this modified tree, though, there is an adaptation of the

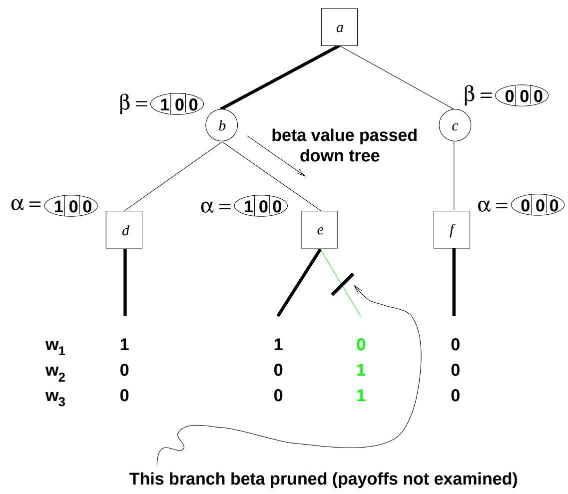


**Fig. 11.** A beta-pruning carried out by the *vm-*$\alpha\beta$ algorithm on a simple tree

alpha-beta technique that does improve accuracy. To see this, we simply need to realise that, when using *vm-*$\alpha\beta$, 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, $\nu$, can be used to reduce non-locality at MAX nodes in any subtree of $\nu$ that has yet to be examined. Since MIN chooses the best play in each individual world, each value $\boldsymbol{\beta}[j]$ imposes a limit (that cannot increase) on the value of the optimal payoff that can now be obtained in world $\mathsf{w}_j$. Thus, when making a new selection at a MAX node in a subtree of $\nu$, all the payoffs of each $\mathbf{K}_i$ in world $\mathsf{w}_j$ 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 Figure 10 to the following:

$$\min(\mathit{payoff\text{-}vector}(t), \boldsymbol{\beta})\,, \tag{4}$$

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 (4) *beta-reductions*. This new algorithm can correctly solve the tree of Figure 11 even when the payoff in world $\mathsf{w}_1$ at node $d$ is greater than one, since the payoffs of 1 in worlds $\mathsf{w}_2$ and $\mathsf{w}_3$ are beta-reduced to zero.

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 Figure 11. 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 $\mathsf{w}_2$ and $\mathsf{w}_3$ 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 $e$.

Of course, standard (perfect information) alpha beta pruning is also affected by branch ordering, at least in terms of efficiency. It is well known that the optimal branch ordering for the algorithm is for MAX's better moves to come first at MAX nodes, and for MIN's better moves to come first at MIN nodes. In fact, the same holds for *vm-beta*, 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 reviewed in §3.3 can find optimal solutions for the tree of Figure 11 irrespective of branch ordering. To show that other trees exist for which *vm-beta* actually out-performs *prm*, then, consider the example of Figure 12. Here, the single MIN node has a minimax value of one in every world. Thus, *prm* cannot reduce any leaf node payoffs, and will therefore produce the same strategy (shown in the figure) as vector minimaxing. It is easy to see that this strategy has a payoff of zero in every world. Employing beta-reduction, on the other hand, will guarantee a payoff of 1 in either world $\mathsf{w}_2$ or $\mathsf{w}_3$ (depending on the random choice made at the second MAX node). That *prm* and *vm-beta* can find optimal strategies on different trees suggests the creation of a hybrid *prm-beta* algorithm. This is easily done by simply replacing *vector-mm* with *vm-*

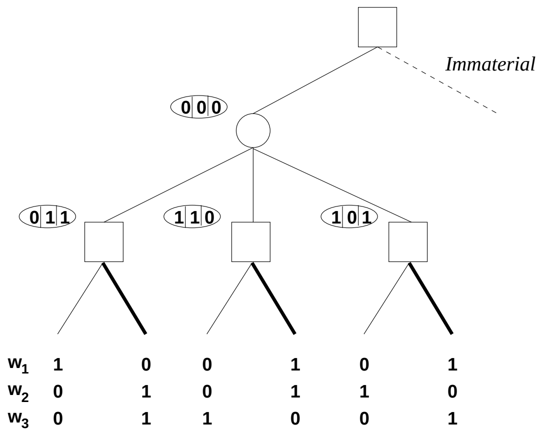


**Fig. 12.** An incorrect strategy selected by *vector-mm* (and by *prm*) on a tree correctly analysed by *vm-beta*

*beta* in step three of the *prm* algorithm of Figure 7. The insensitivity of the payoff reduction technique to branch ordering allows this hybrid algorithm to benefit fully from an ordering that favours beta-reduction.

### 5.2 Iterative Biasing

In our complexity proof of §4, we showed 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 our extensive form game tree, we know (see, *e.g.*, [6]) that when the nodes have at least binary branching the number of strategies for MAX is exponential in the size of tree (doubly exponential in its depth). Finding good guesses among such a large number of possibilities is unlikely to be a practical proposition in general.

However, there is something else besides strategies that can be guessed: payoffs. In fact, given an optimal payoff vector, $\mathbf{K}_{max}$, we can efficiently find an optimal strategy for MAX. To see this, consider a game tree with payoff vectors $\mathbf{K}$ at the leaf nodes. Assume it is known that the optimal payoff vector for this game is $\mathbf{K}_{max}$. 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 $\mathbf{K}$ at each leaf with $\mathbf{K}_{max}$. Replace the vector with the integer 1 if $\mathbf{K}$ is at least as good for MAX as $\mathbf{K}_{max}$ (that is, if $\min(\mathbf{K}, \mathbf{K}_{max}) = \mathbf{K}_{max}$), 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 one. 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 $\mathbf{K}_{max}$, contradicting that $\mathbf{K}_{max}$ is an optimal payoff. Now observe that a payoff of one means that the strategy yields a payoff at least as good as $\mathbf{K}_{max}$ on the original tree. QED.

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 Figure 2, 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 $\mathbf{K}_{max}[k] = v$ for a particular world $\mathsf{w}_k$). 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 (2) 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, $\mathbf{K}_1$ and $\mathbf{K}_2$, we say that $\mathbf{K}_1 \underset{v,k}{>} \mathbf{K}_2$ if and only if either of the following hold:

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

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

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 $\mathsf{w}_k$ is selected. Given some finite set, $\mathcal{S}$, of guesses for the pair of values $\{v, k\}$, we can then repeat the search with different biases — a technique we call *iterative biasing*. Specifically, we can create the *iterative vector minimaxing* (or *ivm*) algorithm of Figure 13.

**Algorithm** *ivm*$(t, \mathcal{S})$
Given $\mathcal{S} = \{\{v_1, k_1\}, \{v_2, k_2\}, \cdots\}$,
for each $\{v_j, k_j\} \in \mathcal{S}$
    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. 13.** Iterative biasing, as carried out by the iterative vector minimaxing algorithm

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 *ivm* can correctly analyse problems that *vector-mm* cannot, simply consider again the tree of Figure 12. For any of the guesses $\{1,1\}$, $\{1,2\}$, or $\{1,3\}$, *ivm* will return an optimal strategy. Each guess results in a different optimal strategy, however, that wins in just one world ($\mathsf{w}_1$, $\mathsf{w}_2$, or $\mathsf{w}_3$, respectively).

For trees with binary payoffs, it is always possible to construct the simple set $\mathcal{S} = \{\{1,1\},\{1,2\},\cdots,\{1,n\}\}$, 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 $\mathcal{S} = \{\{v_{max},1\},\{v_{max},2\},\cdots,\{v_{max},n\}\}$. 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, $\mathbf{K}_{max}$. 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 $\{v_{max},k\}$ for which the (perfect information) minimax value of the game tree in world $\mathsf{w}_k$ is less than $v_{max}$. This is justified by noting that the value of $\mathbf{K}_{max}[k]$ will never be $v_{max}$ if even the best possible play in $\mathsf{w}_k$ itself cannot produce $v_{max}$ tricks.

### 5.3 Summary of Heuristics

We have introduced the heuristics of beta-reduction and iterative biasing, demonstrating how they address the problem of non-locality by introducing dependencies between choices at MAX nodes. We described how each heuristic represents an improvement over the basic *vector-mm* algorithm, and also noted that beta-reduction could be combined with payoff-reduction to produce the *prm-beta* algorithm. In fact, there are a total of eight possible algorithms that can be produced by combinations of payoff-reduction, beta-reduction, and iterative biasing, as shown in Figure 14.

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 Figure 9. The summary in Figure 15 details the node selections made on this tree by each of the eight algorithms. The original *vector-mm* algorithm can at best find a 1-clique, if it makes a fortunate guess at node $v_3$. Payoff-reduction cannot improve on this, as the minimax value of each individual world is one. However, both beta-reduction and iterative biasing improve the result, and when they are used together the optimal solution is found.

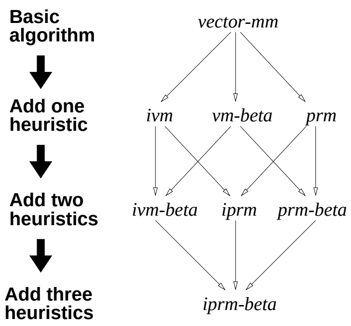


**Fig. 14.** Possible combinations of heuristics

| 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 one 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 $\mathsf{w}_k$, the corresponding $v_k$ will be selected. For the guesses $k=1$, $k=2$, and $k=4$ there is a 50% chance that $v_3$ may also be 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 $\mathsf{w}_1$. All other guesses lead to the selection of just two nodes. | 3-clique |

**Fig. 15.** Performance comparison on tree generated by complexity proof (see Figure 9)

# 6 Test Results

We tested the algorithms in Figure 14 on random game trees and on a hard set of problems from the game of Bridge. The results of these tests are presented below.

## 6.1 Experiments on Random Trees

We follow the approach of [8] in conducting tests using complete binary trees, with $n$ = 10 worlds and payoffs of just one or zero. These payoffs are assigned by an application of the Last Player Theorem [15], so that the probability of a forced win for MAX in the complete information game tree in any individual world is the same for trees of all depths. The game trees are further modified by 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 randomly generate payoffs for $n$ worlds, then
- generate a set of payoffs for a dummy world $\mathsf{w}_{n+1}$,
- 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. In Figure 16 we show the results obtained when $q = 0.75$ — a value we chose because it produces similar results to the game of Bridge. This graph was produced by carrying out the following steps 1000 times for each data point of tree depth and opponent knowledge:

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

The graph for *vm-beta* shows that whilst it improves on simple vector minimaxing, the improvement is not as large as that produced by *ivm* or *prm*. However, a combination of heuristics performs better, with *prm-beta* performing at 100%.

### 6.2 Experiments on the Game of Bridge

Bridge has been heavily analysed by human experts, who have produced texts that describe the optimal play in large numbers of situations. The availability of such references provides a natural way of assessing the performance of automated algorithms. In fact, it was careful scrutiny of such expert analyses that led to the formalisation of the best defence model; thus, the game provides an excellent test of the algorithms in this paper.

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. Of these, we collected the 650 examples that gave *pure* strategies for obtaining the maximum possible payoff against best

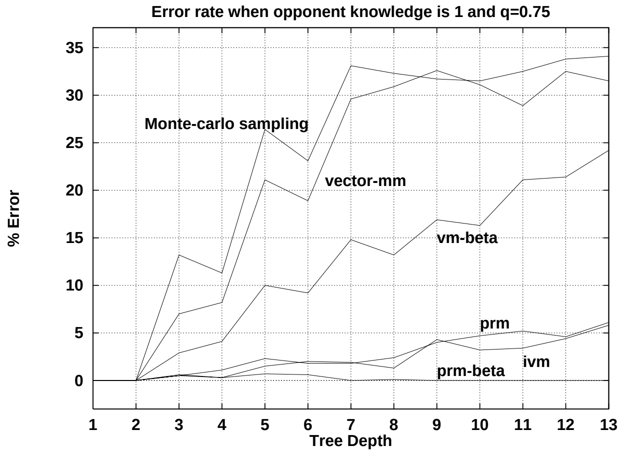


**Fig. 16.** Algorithm performance on random game trees where the optimal strategy in one world is more likely to be optimal in another

defence.[1] Using the FINESSE Bridge-playing system [7, 5], we then tested Monte-carlo sampling, and the eight algorithms summarised in the previous section, against the solutions from the Encyclopedia. (For the iterative algorithms, the set of payoff guesses was produced by finding the worlds for which the perfect information minimax value of the game tree was equal to the maximum possible number of tricks, as described at the end of §5.2.)

We compared the expected payoff of the strategies produced by each algorithm (for the maximum possible number of tricks) with the expected payoff of the solution given in the Encyclopedia. The results summarised in Figure 17 show how often each algorithm was optimal. 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. 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 revealed nine errors in the test set itself, however, this performance (and also that of *ivm-beta* and *iprm*) is actually better than the human experts that produced the model solutions. 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.

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

We intend to correct this design error in the near future.

| Algorithm | Optimal | Sub-optimal | Expected Loss | Time (s) |
|---|---|---|---|---|
| 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. 17.** Performance on the 650 single-suit problems from the Encyclopedia of Bridge

Our results table also includes an 'Expected Loss' column, which gives the number of times that the sub-optimal strategies produced by each algorithm can be expected to result in inferior performance. This figure measures 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 value is produced by simply 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 the crucial factor, namely the creation of tutoring systems where a computer 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 Ultra2 running at 200MHz). 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.

## 7 Conclusions

We have investigated the problem of finding optimal strategies under the best defence model of an imperfect information game. We demonstrated that this problem is NP-complete in the size of the game tree, and introduced the new heuristics of *beta-reduction* and *iterative biasing*. We presented test results that demonstrated the effectiveness of these heuristics, particularly when combined with payoff-reduction minimaxing to produce the *iprm-beta* algorithm. On our database of problems from the game of Bridge, *iprm-beta* actually makes less errors than the human experts that produced the model solutions. It thus represents the first general search algorithm capable of consistently performing at and above expert level on a significant aspect of Bridge card play.

## References


1. ACBL. *The Official Encyclopedia of Bridge.* American Contract Bridge League, Inc., 2990 Airways Boulevard, Memphis, Tennessee 38116-3875, fifth edition, 1994.
2. J.S. Blair, D. Mutchler, and M. van Lent. Perfect recall and pruning in games with imperfect information. *Computational Intelligence*, February 1996.
3. R.A. Corlett and S.J. Todd. A Monte-carlo approach to uncertain inference. In P. Ross, editor, *Proceedings of the Conference on Artificial Intelligence and Simulation of Behaviour*, pages 28–34, 1985.
4. A. Frank. Brute force search in games of imperfect information. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 204–209. Ellis Horwood, 1989.
5. I. Frank. *Search and Planning under Incomplete Information: A Study using Bridge Card Play.* PhD thesis, University of Edinburgh, 1996. Also published by Springer in the Distinguished Dissertations Series, ISBN 3-540-76257-4, 1998.
6. I. Frank and D. Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1–2):87–123, 1998.
7. I. Frank, D. Basin, and A. Bundy. An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, pages 72–76, Vienna, Austria, 1992.
8. I. Frank, D. Basin, and H. Matsubara. Finding optimal strategies for imperfect information games. In *Proceedings of AAAI-98*, pages 500–507, 1998.
9. M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W H Freeman, 1979.
10. M. Ginsberg. Partition search. In *Proceedings of AAAI-96*, pages 228–233, 1996.
11. D. Koller, N. Meggido, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behaviour*, 14(2):247–259, June 1996.
12. D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, July 1997.
13. D.N.L. Levy. The million pound bridge program. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 95–103. Ellis Horwood, 1989.
14. R. D. Luce and H. Raiffa. *Games and Decisions—Introduction and Critical Survey.* Wiley, New York, 1957.
15. D. S. Nau. The last player theorem. *Artificial Intelligence*, 18:53–65, 1982.

16. Y. Nygate and L. Sterling. Python: An expert squeezer. *The Journal of Logic Programming*, 8(1 and 2), January/March 1990.
17. S.J.J. Smith and D.S. Nau. Strategic planning for imperfect information games. In *Games: Planning and Learning, Papers from the 1993 Fall Symposium*, pages 84–91, AAAI Press, 1993.
18. J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behaviour.* Princeton University Press, 1944.