Solving Single-suit Bridge Play: Winning and Knowing Why

Ian Frank [1]

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

David Basin [2]

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

Alan Bundy [3]

Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN

Abstract

In problem solving, it is often important not only to find a solution but also to be able to explain it. We use the game of Bridge to illustrate how tactics, which formalise domain specific expertise, can be used for both these tasks.

Our ideas are implemented in a system called FINESSE. To search the imperfect information game trees that occur in Bridge, FINESSE uses a new algorithm that always identifies optimal strategies. The use of tactics speeds up this algorithm by heuristically constraining the space of possible moves, and also provides the key to generating human understandable explanations of the resulting strategies. Using a canonical set of single-suit Bridge problems from a definitive expert text, we demonstrate that the tactics significantly prune the search space without sacrificing optimality. FINESSE finds optimal strategies for all problems in an average of 0.6sec, and also finds errors in the expert solutions at a rate of over 3%. Moreover, we show that FINESSE automatically generates English explanations of its solutions comparable to those in the expert text — a capability we are using to produce a comprehensive computer-authored reference on single-suit Bridge.

[1] Corresponding author. Email: ianf@etl.go.jp.
[2] Email: basin@informatik.uni-freiburg.de.
[3] Email: A.Bundy@ed.ac.uk

1 Introduction

In problem solving, we can distinguish between finding a solution to a problem and being able to communicate that solution to others. For instance, a medical diagnosis system is more valuable when it can explain its decisions; if RoboDoctor says you have cancer, you will want to ask it for some evidence.

In computer games, this distinction has been referred to as the difference between *cracking* and *solving* a game [2]. To crack a game, a program needs to play optimally. But to solve it, the program's play must be explainable in human understandable terms. Solving a game is much harder than cracking it. Considerable advances in search algorithms and specialised search hardware have led to strong computer play in games like chess, but have provided little help in explaining the resulting moves. Ask a typical chess program why it made that pawn capture and you are lucky to get back a screenful of numbers.

In this paper, we show how a difficult and significant part of the game of Bridge can be 'solved' in the technical sense. Bridge is a game with imperfect information and enormous search spaces. Nevertheless, it is played remarkably well by humans. In this paper, we consider a sub-problem of Bridge that has been heavily analysed by experts: finding the optimal way to play the cards in a single suit. Analysis of single-suit Bridge problems (also called 'suit combinations') is challenging even for master-level players and there are a number of Bridge books that cover the problem in depth [1,25,4,30].

Cognitive studies [14,11,12] have shown that human performance in Bridge can be attributed to the acquisition of high-level patterns and chunks of knowledge gained through experience. Our contribution is to show that a computer can combine search with a set of such patterns to find and explain optimal strategies for single-suit play. To accomplish this, we first formalise expert knowledge about Bridge as *tactics*; these distill Bridge knowledge down to seven distinct and simple manœuvres, such as playing winners and finessing (manœuvering against) missing cards. To solve a Bridge problem, we then search the space of possible tactic applications with a new imperfect information search algorithm that quickly finds optimal solutions, despite the task being NP-complete in the size of the game tree. The use of tactics has two benefits: it speeds up search, and it results in strategies that are trees of tactics, instead of trees of individual moves. We show that a natural consequence of the latter is that it allows FINESSE's strategies to be explained in plain text.

Our ideas are implemented in a system called FINESSE. Although the use of tactics significantly prunes the search space, all optimal strategies remain. We demonstrate this empirically by testing FINESSE on a collection of over 1500 problems from the authoritative Encyclopedia of Bridge [1]. For all these problems, the strategies found by searching the space of tactics are the same as those found by searching the space of possible card plays. Yet searching the tactic space is an order of magnitude faster, taking an average of 0.6sec for one problem. This is easily fast enough for real-time use, for example as part of a program for playing complete hands. FINESSE even reveals 58 errors in the Encyclopedia solutions (a rate of over 3%) and discovers strategies new to the Bridge community. Two of these new strategies were the subject of a recent article in a leading Bridge magazine [17].

The Encyclopedia also allows us to demonstrate that FINESSE can explain its strategies in text comparable to that produced by human experts. Since its strategies are also always optimal, this demonstrates that FINESSE satisfies the technical criteria for solving single-suit Bridge play — the first time that analysis of a significant part of the game has been settled in this way. We are in the process of using FINESSE's explanations to generate an error-free reference on single-suit Bridge. Another promising application would be to use FINESSE as the basis of a real-time Bridge tutor.

Although the above contributions are centred around game-playing and Bridge, we see our work as contributing to the larger question of methodologies for formalising and understanding reasoning. In fact, our original motivation comes from the domain of theorem proving where, like Bridge, humans have considerable success in navigating very large (often infinite) search spaces. One successful theorem-proving technique, developed over the last decade, is *proof planning* [5,7]. In this approach, tactics for proof construction are specified and proofs are searched for in the space of tactics, which is typically orders of magnitude smaller than the space of inference rules. This has proven successful for certain problem domains, like inductive theorem proving, where the intuitions and standard techniques used by mathematicians can be codified. There is an analogous situation in Bridge: beginners are taught standard techniques for analysing and playing hands, but often only by example, and through practice. Our work demonstrates that tactics can also be used to formalise the possible actions in a domain with opposition and imperfect information. Just as in proof planning, the tactics are used both to heuristically constrain the search and to explain the results.

This paper supersedes our previous work on solving Bridge, reported in [19]. We begin (§2) by discussing related work on computer game-playing, computer Bridge, and theorem proving. Next (§3), we describe the particular tactics we have developed for single-suit Bridge play. Given a play situation, FINESSE produces game trees containing all the possible sequences of tactic applications. In §4, we describe how FINESSE builds and searches these tactic trees, and in §5 we show how the tactic-level representation leads directly to textual explanations. Finally, in §6 we describe experimental results that validate our approach, and in §7 we draw conclusions and discuss future work.

2 Related Work

2.1 Computer Game Playing and Explanations

Games have been a popular topic for AI research since the 1950s, when acceptance of the 'Turing Test' focused attention on mimicking human behaviour as a test of machine intelligence. Fifty years later, the best human players have been equalled or bettered in games such as chess, checkers, backgammon, Othello, and Scrabble. These successes have largely been the result of capitalising on what computers do well: fast search, and fast storage and access of huge amounts of data.

Although computers play some games better than humans, they still trail far behind in their ability to account for their actions; strong computer play has seldom translated into an ability to explain the computer's moves. A good example here is the early computer chess research of Komissarchik and Futer, who were described as being 'at their wits' end when trying to explicate rules fit for humans to apply' from their endgame databases [2]. Or consider the Othello program LOGISTELLO, which uses self-play and linear regression to learn an evaluation function with 1.5 million weights [9]. There is little that a human can learn by looking at so many numbers. Yet, using this evaluation function, LOGISTELLO beat the human World Champion, Takeshi Murakami, 6-0 in August 1997 [8]. Afterwards, the defeated champion used LOGISTELLO to review some of the games, and reported that even the moves suggested by LOGISTELLO as alternatives to his 'mistakes' had no recognisable pattern.

To our knowledge, there are few instances of the generation of explanations from programs that are strong at game-playing. One simple example is provided by Davey's PROTEUS system [13], which produces a commentary on a game of noughts and crosses. However, this system investigates the natural language generation issues involved in explaining just the moves made in a game, rather than strategies for playing the game. A computer program also 'solved' Connect-4 [34] by using high-level concepts to search the space of moves. But although these concepts were understandable to humans (albeit still rather complicated), the program itself had no method of using them to generate explanations of optimal play.

2.2 Computer Bridge

Historically, Bridge has attracted less interest from AI researchers than many other games. One possible reason for this is that Bridge has *imperfect information.* In games such as chess or backgammon, the exact state of the game is always clear because the pieces on the board are always visible. However, at the start of a Bridge game, each player can only see his own cards and, as a result, standard approaches to search based on the minimax principle and its extensions are not directly applicable. Another difficulty of Bridge is that it has two stages: *bidding* and *card play*. Bidding increases the skill-level of Bridge by forcing the players to decide a specific goal for the subsequent card play. These stages require very different modelling and algorithms.

In this paper, we consider only the card play stage of Bridge. Early work on this problem used simple rule-bases to select a single move for any possible game situation [10,27]. However, all current approaches rely on lookahead search or planning. As noted in the Introduction, cognitive research has shown that humans play Bridge by utilising their knowledge of high-level patterns [14,11,12]. An analogous approach has been used by a number of Bridge researchers to restrict the number of options considered when carrying out lookahead with a computer. Indeed, we make use of this technique in FINESSE, where we call the possible actions *tactics*. Other examples are the 'schemas' of Stanier's [33] planner, the ASPEN system [28], which uses the same terminology of 'tactics', the 'Thematic Actions' of Khemani [24], and the 'multi-agent methods' in the TIGNUM system developed by Smith and Nau [31].

Of all the systems mentioned above, FINESSE is the only one to provide either exact probabilities or precise explanations, *i.e.*, the first system to crack or solve a significant aspect of the game. Stanier's system was an early attempt that constructed plans for the complete card play stage (not just single suits) based on only six schema — far too few for consistent performance, but enough to find answers to a small number of example problems. ASPEN included some recognition of the chance of success of its plans, but only distinguished between the outcomes of `certain`, `usual`, and `possible`. Khemani's Thematic Actions, on the other hand, were actually extreme examples of tactics where complete plans were retrieved by the planner from memory, but again with no exact probabilities attached. None of these planning systems were tested on large sets of play problems or under real play conditions. However, Smith and Nau's TIGNUM was incorporated as part of a commercial Bridge program called BRIDGE BARON that won the 1997 computer Bridge championships. The use of multi-agent methods was combined with the technique of forward-pruning [32], and produced a significant improvement over previous versions of the BRIDGE BARON software. However, forward-pruning is a search technique that sacrifices optimality by discarding branches at some nodes without a proof that they cannot affect the value of the search.

The best commercial programs today work on a different basis that avoids reasoning about high-level actions. Instead, card play is simplified by repeatedly solving the simpler problem where all the players reveal their cards to each other. This removes any uncertainty and creates a perfect information problem that can be tackled by minimax-based search. By sampling a statistically significant number of the possible distributions of the outstanding cards and finding the best move in each, the best overall move can be predicted. This approach is a version of Monte Carlo sampling, and was first suggested for Bridge by Levy [26]. The strongest program to use this approach is Gib, which Ginsberg [22] estimates is halfway between Bridge Baron and world class. However, whilst Monte Carlo sampling produces good overall play in Bridge, its faults are well documented. We have shown in [16] that the assumption of perfect knowledge on each individual sample inevitably leads to large numbers of small errors (for example, a 35% error rate on the Encyclopedia database we use in §6). This is one reason that computers still trail the best human players, as demonstrated by the most recent match pitching a human against computers (held at the Andrew Robson Bridge club in London), where the former world champion Zia Mahmood fended off seven computer challengers [3].

2.3 Tactic-Based Theorem Proving

The automated reasoning community has developed considerable experience in using formal tactics to prove theorems. The paradigm originated with the LCF project, where tactics are user programmable proof steps [23] that construct proofs by applying inference rules. As originally conceived, tactics are applied interactively by users, however it is possible to use planning techniques to automate their application. In *proof planning* [5], tactics are declaratively specified in a formal *meta-language*. Each tactic has an accompanying *method*, which formalises its preconditions (the conditions necessary for the tactic's application) and effects (the results of a successful tactic application). Proof planning attempts to prove conjectures by recursively applying methods to reason about the effect of tactics on the proof state. Especially for inductive theorem proving, this approach has had considerable success [7,6] and can drastically reduce the search space for proofs.

As noted in the Introduction, it was the success of proof planning that originally motivated our study of computer Bridge. We were interested in the new challenges that Bridge presented: a domain with opponents and imperfect information. We were also interested in exploring whether the high-level nature of a proof plan could provide advantages in Bridge analogous to those in theorem proving.

In the case of theorem proving, a successful proof plan produces both a composite method (a tree of methods that describes the overall plan) and a composite tactic (built from the tactics associated with each method in the tree). The tactic generates a low-level step-by-step derivation in the underlying logic, and the method provides a high-level explanation of this derivation. Both contribute to showing that the conjecture is true. Alan Robinson [29, Page 56] has expressed the requirement that a convincing proof should combine both these parts with the slogan 'Proof = Guarantee + Explanation'. Proof planning is capable of producing proofs in this general sense [5].

The same slogan can be applied to Bridge, where a 'guarantee' becomes a strategy in the game-theoretic sense [35], namely a recipe for playing a hand that specifies a single legal response for every possible play of the opponents. Given such a strategy, an 'explanation' equates to a high-level account of the strategy and maybe also of its chance for success. In theorem proving it is usually enough to find *any* proof of a theorem. In Bridge, however, every strategy leads to a complete game, and the important task is to identify a *particular* (often unique) optimal strategy. So, we can think of a 'proof' in Bridge as a strategy that guarantees optimality, accompanied by an explanation that would convince a Bridge player that this is the case. It is possible that (good but) suboptimal play is the necessary price of search pruning heuristics. Our empirical results show that for our Bridge tactics this is not the case.

3 Bridge Knowledge: Tactics

To solve single-suit Bridge, FINESSE combines knowledge and search. Here, we describe the first part of this formula: FINESSE's tactics. Since some Bridge background is necessary to explain these tactics, we begin by briefly introducing relevant aspects of single-suit play. Readers already familiar with the game may skip to §3.2.

3.1 Bridge and Single-suit Play

Bridge is a card game played by four players, traditionally named North, South, East and West. These players form two teams: North/South against East/West. In the full game, the deck contains 52 cards, split into 4 suits of 13 cards each. In this paper we consider just the simplified problem of suit combinations, where the goal is to play a single suit. Although this is a substantial restriction, the resulting problems are still very hard. As we noted in §2.2, cognitive studies have shown that knowledge of suit combinations is fundamental to human play of the complete game. There are entire books written about suit combinations [30,25,4,1] and as the author of one of these, J. M. Roudinesco (a world life master), notes [30, P.9]:

> 'How should I have handled this suit?' Seek advice from twenty experts,

> and you will get twenty different answers, among which at least nineteen will be wrong. Clearly the subject is very difficult and complicated, and there are not many bridge players able to cope correctly with all the suit combinations they meet.

In a suit combination, the 13 cards of a single suit are distributed between the four players. In order of strength, the cards are the Ace, King, Queen, Jack, 10, ..., 2 (we will sometimes abbreviate the first four of these to A, K, Q, and J). The task is then to analyse the best way for the players to play these cards. A simple example of such a problem is shown in Figure 1.

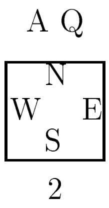

Fig. 1. A simple suit combination

In a full game of Bridge, one player places all his cards on the table once card play has started. This player takes no further part in the deal and is called the *dummy*. The dummy's cards are played by his partner, the *declarer*. Since suit combinations are just subsets of genuine Bridge positions, they are also always presented with a dummy and a declarer, conventionally, North and South. Thus, in the example of Figure 1, the dummy's cards are the Ace and Queen and the declarer's card is the two. The dummy's cards are assumed to be visible to all, whereas the remaining cards are initially only known to the players that received them. Solving suit combinations involves finding a strategy that gives the declarer the maximum chance of achieving a specified goal, given that East and West will do their best to prevent this.

Typically, declarer's goal is to win a certain number of *tricks*, or rounds of play. In each round the declarer lays a card on the table, either from his own hand, or from the dummy's. Progressing clockwise around the table, the other players then take turns to play a card from their own hand, if they have one. The highest ranked card of the round wins the trick for the team that played it. For example, in Figure 1, declarer could start by playing the Queen from the North hand. If East then plays the King, South plays the two, and West plays the six, then East/West win one trick. In real Bridge, the winner of one trick must then play the first card in the next trick. However, in single-suit problems it is always assumed that each trick is started by the declarer playing a card from the North or the South hand. Thus, to continue the play after East wins with the King, declarer plays the Ace from the North hand on the next trick. As the Ace is the highest-ranked card, this wins the trick for the declarer (given the standard assumption, made throughout this paper, that there is no *trump* suit).

Starting with the Queen in the situation of Figure 1 will always produce just one trick, but the declarer can improve on this. A second option is to begin with the Ace. This wins one trick, but also has a small chance of making two tricks. A basic rule of Bridge is that whenever a player holds cards in the suit that started a trick, they *must* play one of these on their turn. So, if East or West holds just the King (referred to in Bridge as a *singleton* King) they must give it up when the Ace is played. Thus, the play of *cashing* the Ace succeeds in winning two tricks for two of the 2^{10} possible ways that the remaining 10 cards in the suit can be split between East and West.

The other alternative for declarer in this situation is to start with the two. In fact, this play offers the highest chance of success on this problem, because if West holds the King it places him in a dilemma. If West plays the King, it is beaten by the Ace and the declarer then wins a second trick with the Queen. But if West plays a low card instead of the King, declarer can still win two tricks by playing the Queen from the North hand. Of course, the Queen will lose if East holds the King, but declarer will win two tricks whenever the King is with West — a 50% chance. This solution is based on the elementary principle of card play that the best results can be obtained by forcing an opponent to make a decision before your own high cards are committed. It is a typical example of a standard manœuvre called a *finesse*, which we will discuss in more detail in §3.3.

3.2 The Best Defence Model

In designing a program to play Bridge, we must first clarify what it means to do this well, i.e., answer the question 'what is an optimal strategy?' There are different possible answers to this question, depending on the capabilities and knowledge of the opponents. Our goal in designing the FINESSE system was to be able to generate analysis comparable to that given by Bridge professionals when they analyse single-suit problems. Hence our starting point was to take their model of the opposition.

A close examination of expert analysis reveals that they consistently make certain implicit assumptions when analysing problems. For example, to determine whether a strategy (such as 'finesse the Queen') succeeds under a particular distribution of the remaining cards, it is assumed that East/West play optimally in this perfect information sub-problem. The overall chance of success of the strategy is then taken to be the sum of the probabilities of the distributions for which it succeeds. In previous work on Bridge, we have formalised the assumptions used by human experts to analyse Bridge problems

[16,18], producing a *best defence model*, which we summarise here:

A–I The opponents have perfect information.
A–II The opponents choose their strategy after the declarer.
A–III The declarer adopts a pure strategy (making no use of probabilistic choices such as '80% of the time do this, 20% of the time do that').

This model is called 'best defence' because it formalises the strongest possible assumptions about the opponents: that they know how the cards are distributed (A-I) and choose their strategy in the knowledge of the declarer's strategy (A-II). The assumption that the declarer 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.*, the vast majority of the solutions in the texts [30,25,4,1] were generated under these assumptions), but also because modelling the strongest possible opponents provides a lower bound on the payoff that can be expected when the opponents are less well informed.

Note that under this model, both opponents have the same information. Hence, we can actually model single-suit problems using two players, MAX (the declarer) and MIN (the opposing side). MAX plays the North and South cards and MIN plays those of East and West.

3.3 Our Bridge Tactics

Our tactics represent a new formalisation of the domain of single-suit Bridge play, distilling this knowledge into seven distinct and simple manœuvres. To produce our tactics, we generalised from examples of card play we found in Bridge books. As we mentioned in the Introduction, the tactic set we produced in this way is sufficient to generate optimal solutions for all the problems in the Encyclopedia of Bridge [1].

We were surprised to find that only seven tactics are required to generate all useful strategies for playing single-suit Bridge. Although the tactics represent knowledge that human players usually learn through experience, we are not aware of any Bridge text that explicitly presents the required knowledge in such a compact and manageable way.

FINESSE's tactics are easy to explain to Bridge players. The two simplest tactics are 'cash' and 'duck'. We saw an example of a cash in the suit combination of Figure 1. Cashing corresponds to playing a *master* (i.e., a card higher than any of East or West's cards), which will always win the trick. Ducking, on the other hand, is the act of deliberately losing a trick. It is carried out by simply playing the lowest possible cards from both the North and South hands.

Another simple tactic is one we call 'sequence'. This tactic handles situations where there is no element of manœuvre. Typically, this will be the result of holding enough high cards to play one on each trick. Two examples of this are shown in Figure 2 (where we follow the standard Bridge convention of using an 'x' to stand for cards whose actual value has no effect on the analysis). In both these situations, one card from the sequence K-Q-J will lose to the opponents' Ace, while the other two will win a trick. The sequence tactic corresponds to either North or South playing a card from the highest sequence held by the combined hands.

K Q J	Q x x
N	N
W E	W E
S	S
x x x	K J x

Fig. 2. Suit combinations where there is no element of manœuvre

The most interesting tactics are the remaining four, which are for finessing. In describing these tactics, we will call a 'finesse card' a card with which declarer tries to win a trick, even though it is not a master. Also, the 'lead card' is the card that declarer leads in order to initiate the finesse. And a 'cover card' is a card in the third hand capable of beating at least one of the outstanding cards above the finesse card. The four types of finesse are then:

Type 1. The finesse card and cover card are in the same hand (in Bridge this is called a *tenace*), opposite the lead card. This was the type of finesse we encountered in §3.1, where the A-Q formed a tenace, allowing the Queen to be used as a finesse card against the King. Other examples are shown in Figure 3. In the first, the A-Q again form a tenace, but the Q-10 is a further tenace that can be used to finesse against the Jack. In the second, the A-J-10 form a tenace that can be used to finesse against the missing King and Queen. This finesse can be carried out twice, producing three tricks when the King and Queen are both held by West.

A Q 10	A J 10 x
N	N
W E	W E
S	S
x x x	x x x

Fig. 3. Two suit combinations allowing Type 1 finesses

Type 2. The finesse card is also the lead card, and the opposite hand contains the cover card, and also a 'duck card'. Further, the finesse card must belong to a sequence of two or more cards with no outstanding cards between them held by the declarer's combined hands. Figure 4 illustrates this type of finesse. In both examples, declarer can finesse against the missing King by leading the Jack from the South hand.

A Q x x	A x x
N W E S	N W E S
J x x	Q J

Fig. 4. Two suit combinations allowing Type 2 finesses

Type 3. The finesse card and a 'duck card' are opposite the lead card. No cover card is necessary. For instance, in the first example of Figure 5 declarer can finesse against the opponents' Ace by leading low from the South hand. In the second example, declarer finesses against the opponents' King by leading low to the Queen.

K Q x	A x x
N W E S	N W E S
x x x	Q x x

Fig. 5. Two suit combinations allowing Type 3 finesses

Type 4. The finesse card is the lead card and is run to the third hand, which holds a cover card. The card directly below the finesse card is held by East or West, and the card directly below that is held by the declarer's side. Either the finesse card is a singleton, or there is a continuation for the next trick that uses the card two ranks below it as a finesse card for a Type 1 or Type 2 finesse. Figure 6 gives examples of this finessing situation. In both examples, declarer can finesse against the opponents' Queen by leading the Jack. If West plays the Queen, declarer covers with the master and follows up with a finesse against the opponents' 10.

K x x	A 9 8
N W E S	N W E S
A J 9	K J x

Fig. 6. Two suit combinations allowing Type 4 finesses

To enable the seven tactics to be used to heuristically constrain search, we specify their preconditions declaratively. As part of this specification, we represent suit combinations as a data type: a 6-tuple that describes the game state from the declarer's perspective. A *suit-combination tuple* records (1) the cards held by North and South, and (2) the outstanding cards. An opponent may also be known to be *void* (have no cards) in the suit (e.g., because of failing to follow the lead on a previous round). Hence we also record (3) a flag indicating whether East is void, and (4) a similar flag for West. Finally, a small amount of pre-processing is carried out to identify (5) the number of possible *losers* in the suit (the number of rounds lost assuming the worst possible split of cards among the opponents) and (6) the *critical cards*, which are those cards whose location could make a difference in winning a trick.

In the Appendix we give examples of two tactics: the cash and the Type 1 finesse. There, we have given the original Prolog code used in the FINESSE system. Each Prolog clause has a suit-combination tuple (represented as `State`) for its first argument. For example, the preconditions for the Type 1 finesse are that (1) there must be at least one loser in the suit, and (2) there must be an outstanding critical card with other outstanding cards below it. Further, (3) the declarer must have a sequence of one or more cards beneath the critical card, and (4) one of declarer's hands must contain both a card from this sequence and a card that can beat the critical card (this forms a tenace). Finally, (5) the hand opposite the tenace must have a duck card to lead towards it, and (6) the first opponent is not known to be void.

If the preconditions in the body of the clause succeed, the result is a tactic, represented as a Prolog term. FINESSE's representation of tactics as Prolog predicates will sometimes appear in the figures of this paper. These should be fairly self-explanatory. For example, `cash(a,h)` denotes the cashing of the Ace of hearts, `duck(d)` denotes ducking a round in the diamond suit, and `finesse(1, west, q, s)` denotes a Type 1 finesse of the Queen of spades against West (the optimal play for the AQ-2 example of Figure 1).

4 Search: Vector Propagation

We have described FINESSE's tactics, which encode its knowledge about Bridge. We now introduce FINESSE's search algorithm for finding optimal strategies in imperfect information games. We describe the algorithm as we apply it in FINESSE, where it searches a tree of tactics resembling a minimax tree.

FINESSE's tactics specify possible ways to play a single trick. The search space for any given problem is then the tree of possible tactic applications. Each node in the tree is labelled by a suit-combination tuple, which characterises the game state at that point in play. First, a root (MAX) node is generated, representing the initial game state. Successor (MIN) nodes are then produced by iterating through each of the applicable tactics. From each of these nodes, a new level of MAX nodes is generated by iterating through the possible plays by East and West. MAX nodes are recursively expanded until no more tactics apply. In general, there is more than one tactic applicable to a particular state and East/West may also make a number of different responses. Hence trees are of the form illustrated in Figure 7.

This example shows the tree generated by FINESSE for our AQ-2 example. The North and South cards are shown at each non-terminal MAX node, and the MIN nodes are represented by circles. The labels on the MIN branches denote the cards played by West and East on the first trick (*e.g.*, 'K/v' denotes West playing the King with East unable to follow in the suit). After the first trick, MAX has only one remaining card. If this can be cashed a further branch is generated.

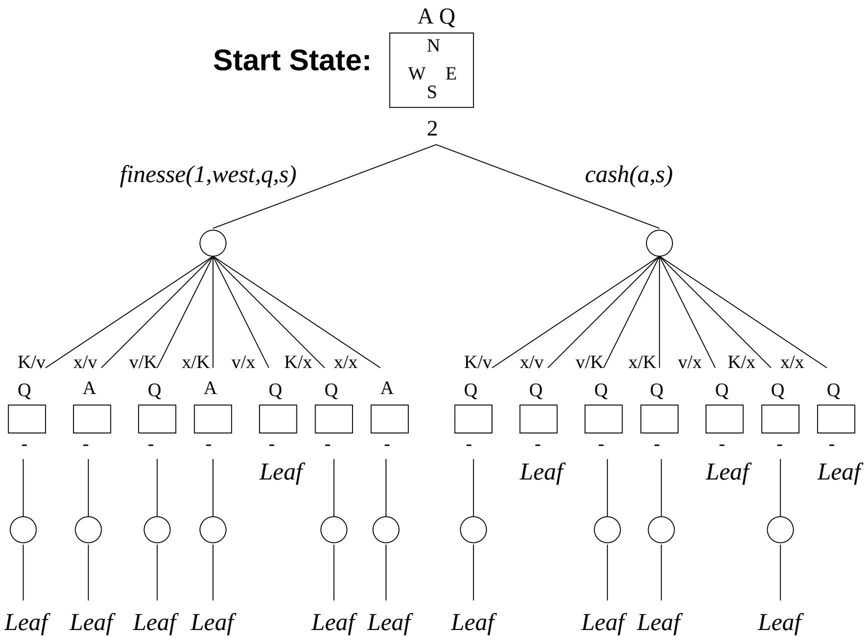

Fig. 7. An example game tree for the suit combination of Figure 1

The MIN branches of Figure 7 can only be followed by East/West if they were dealt the appropriate cards (*e.g.*, West can only play the King if he was dealt it). If we do not distinguish between the nine low cards, there are 20 distinct outcomes of the initial deal (for either East or West, they may hold between 0 and 9 low cards, and either hold the King or not). We call each of these possibilities a *world* and label the leaf nodes of a tree with the payoff in each world. To formalise this, for any leaf-node ν of a tree with n possible worlds, let the function *payoff-vector*(ν) return the n-element vector $\vec{K}$ in which $\vec{K}[j]$ (the jth element of $\vec{K}$) takes the value of the payoff at ν in world j ($1 \leq j \leq n$). For any world j in which ν cannot be reached, the value of $\vec{K}[j]$ is $\perp$, *i.e.* undefined. For example, the vector for the leftmost branch of Figure 7 would consist of one payoff of 2 (the world where East has no cards in the suit) and nineteen $\perp$s (all the remaining worlds).

Say we have an imperfect information game tree with n worlds. A specification of exactly one branch at each MAX node in the tree defines a MAX *strategy*. For any such strategy, we can determine its chance of success with respect to the best defence model by a bottom-up pass through the tree, as follows: (1) at leaf nodes, back up the n-tuple *payoff-vector*(ν); (2) at MAX nodes, back up the vector from the daughter node on the branch specified by the strategy; and (3) at MIN nodes that have m daughters with the vectors $\vec{K}_1$, $\vec{K}_2$, $\cdots$, $\vec{K}_m$, back up:

$$\min_i \vec{K}_i = (\min_{i=1\cdots m} \vec{K}_i[1], \min_{i=1\cdots m} \vec{K}_i[2], \cdots, \min_{i=1\cdots m} \vec{K}_i[n])\,, \tag{1}$$

where $\min(x,\perp)=\min(\perp,x)=x$, for all x. This minimisation correctly models the best defence assumptions that MIN has perfect knowledge of the world state by restricting MAX's payoff in any individual world to the smallest value possible.

Given a game tree and a strategy, the process just described produces a single vector at the root of the tree that gives the strategy's payoff in each world. By iterating this procedure over each of the (finitely many) strategies, we could find an *optimal* strategy. However, for general game trees this straightforward approach will be too inefficient, since the number of strategies in a tree is exponential in the tree's size. Unfortunately, it is not easy to do better without sacrificing optimality as the problem of finding an optimal strategy (against best defence) is NP-complete in the size of the game tree [18]. We have previously described a number of tractable heuristics for this problem [21,18], but to *solve* single-suit Bridge we require optimal play.

In this paper, we propose an optimal algorithm whose average running time is much better than that of the straightforward enumerative approach described above, although it too has exponential running time in the worst case. Our *vector propagation* algorithm is based on two ideas: examining the results of all strategies by propagating *sets* of payoff-vectors *simultaneously*, and using a notion of *subsumption* to dispose of many of the inferior strategies during propagation. In the resulting algorithm, each one of the vectors in the set produced at a node represents the outcome of one of the possible strategies in the node's subtree. We will see that for our trees of Bridge tactics, the prunings made by the subsumption check are sufficient to allow us to produce optimal results in real-time.

The algorithm for vector propagation (without the subsumption check) is given in Figure 8, where $sub(t)$ is a function that computes the set of immediate subtrees of a tree t. At MAX nodes, no single branch is selected; instead, the results of all possible MAX branch selections are retained by collecting the vectors from all the daughter MIN nodes. At MIN nodes with m branches, the function Π first forms a set in which each element is itself a set containing exactly one vector from each of the initial m sets. It then applies Equation 1 to each element $\{\vec{K}_1, \vec{K}_2, \cdots, \vec{K}_m\}$ of this set.

Algorithm *vec-prop*(t):
Take the following actions, depending on t.

Condition	Result
t is leaf node	$\{payoff\text{-}vector(t)\}$
root of t is a MAX node	$\cup_{t_i \in sub(t)} vec\text{-}prop(t_i)$
root of t is a MIN node	$\Pi_{t_i \in sub(t)} vec\text{-}prop(t_i)$

Fig. 8. The vector propagation algorithm

In general, the number of vectors produced by *vec-prop* at any node will be the same as the number of strategies in the subtree rooted on that node, which is exponential in the subtree's size. However, in practice, including a subsumption check can lead to substantial reductions. We implement subsumption as follows: if the collection of vectors at a node contains members that are pointwise less than or equal to any other member at that node, these vectors are removed, as they inevitably give rise to inferior strategies. Note that we previously suggested this approach in [15], and used it in [20] to provide a benchmark for other algorithms on binary trees. However, this is the first time it has been implemented for a real game. For our trees of Bridge tactics, the subsumption check allows all the problems in our test databases to be solved, in an average time of just over 0.6sec.

To illustrate *vec-prop*, we give an example using the simple three-world tree of Figure 9. This figure shows the sets of vectors produced by *vec-prop* at each node. At the leaf nodes, the sets of vectors have just one element. The set of vectors at node d also has just one element, since the node has just one branch. However, the set of vectors at node e is formed by taking the union of two sets, and therefore has two elements. At node b, *vec-prop* initially forms the set $\{\{(3,0,0),(2,0,0)\},\{(3,0,0),(0,1,1)\}\}$ to represent the payoffs that can be produced by the two MAX strategies possible in its subtrees (MAX can either pick the left-hand branch at node e or the right-hand branch). It then applies Equation 1 to each element of this set to produce $\{(2,0,0),(0,0,0)\}$. Finally, the vector (0,0,0) is removed by the subsumption check. At the root of the tree, there are two possible vectors, one corresponding to the MAX strategy of picking the left-hand branches at nodes a and e and the other corresponding to choosing the right-hand branch at node a.

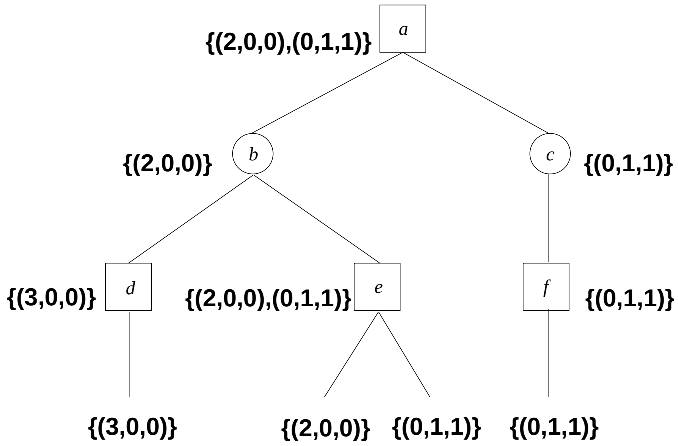

Fig. 9. Applying *vec-prop* to a tree with three worlds

Selecting the 'best' vector from the set produced by *vec-prop* at the root of a tree depends on the particular goal. We have implemented the three alternatives shown in Figure 10. We use a version of *vec-prop* that backs up information on branch choices together with the vectors, so that identifying the best vector for a given goal also identifies a strategy. For our simple tree with three worlds, the best strategy for the goal *max* is to choose the left-hand branch at the root. However, assuming that each of the three worlds is equally likely, the best strategy for guaranteeing a payoff of at least 1 is to pick the right-hand branch at the root. For the goal *expected* both options are equal. For our Bridge example of the AQ-2, the best strategy for all three goals is the same: finesse the Queen.

Goal	Required strategy optimises chances of
max	taking the maximum number of tricks
Ntricks	taking at least N tricks, $N <$ maximum number
expected	highest expected number of tricks

Fig. 10. Three possible goals for a suit combination

4.3 Payoff-reduction and Beta-reduction

The results we present in §6 show that *vec-prop* allows FINESSE to solve single-suit Bridge. Hence, in this domain, the subsumption heuristic is effective in reducing the size of the sets of strategies produced at any node. However, for general trees, this is not always the case, and even on some Bridge problems the sets can become large. We therefore describe here two additional heuristics that we also employ in FINESSE, which can substantially reduce the size of these sets.

The first heuristic we explain is payoff-reduction [21]. This is a pre-processing step that recognises that under the best defence model, the play will never actually reach some nodes in the tree. Consider a leaf node ν that gives MAX a payoff of p in world k. Play will never reach ν in world k if the ancestors of ν include a MIN node with a branch on which MAX's payoff in world k is lower than p. The pre-processing carried out by payoff-reduction for a tree t with n worlds is therefore as follows:

(1) Apply standard minimax to t backing up just the payoffs in world k, $1 \leq k \leq n$, recording for every MIN node its minimax value, m_k, in each world.
(2) Examine the payoff vectors of each leaf node of the tree. Reduce the payoffs p_k in each world k to the minimum of p_k and all the m_k of the node's MIN ancestors.

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. However, since the payoffs are all lowered there is a greater chance that the subsumption check of vector propagation will be able to prune vectors.

A similar pruning of leaf nodes is carried out by beta-reduction [18]. However, this time the reductions are not based on a pre-processing step, but are incorporated into the search itself. Whereas payoff-reduction uses the minimax values of the game tree in each world as upper bounds for leaf node payoffs, beta-reduction uses the vectors already produced at MAX nodes. Essentially, it is an adaptation of the well-known procedure of alpha-beta pruning. Successfully analysing one daughter of a MIN node provides an upper bound on the result that can be produced at the node's siblings. Such upper bounds, associated with MIN nodes, are called *beta* values.

We motivate beta-reduction on our game trees by returning to the simple three-world tree we introduced in Figure 9. Consider the effect, at node b, of producing the vector (3,0,0) at node d (see Figure 11). Since MIN chooses the smallest possible payoff in each world (by applying Equation 1), the vector

(3,0,0) imposes an upper bound on the payoffs of any of the vectors that can now be produced at b. Thus, the payoffs beneath node e can be reduced if they are above these values. In particular, in the second and third worlds, MAX cannot achieve a payoff of 1 at node e (since zero is an upper bound; MIN can always choose the left-hand branch at node b in these worlds). In the example of Figure 11, reducing the leaf node payoffs in this way results in a decrease in the size of the set of vectors at node e. We show in §6 that for larger trees the impact of such reductions can be significant.

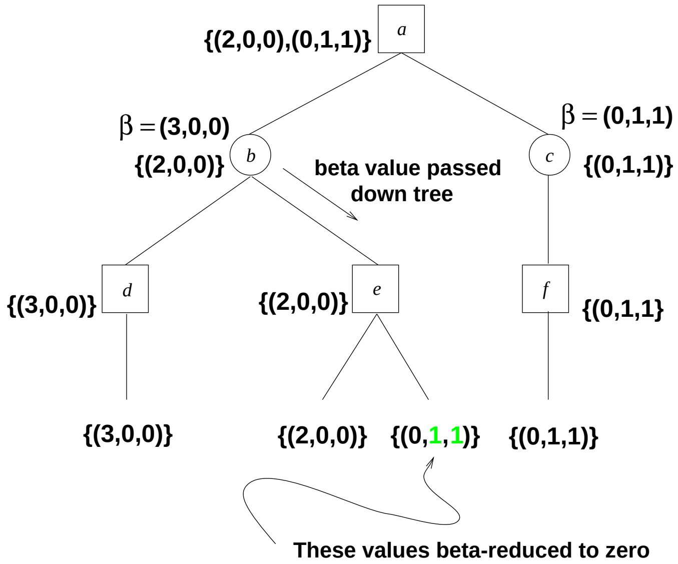

Fig. 11. Applying *vec-prop* with beta-reduction to a tree with three worlds

For a general MIN node ν, analysing one MAX daughter produces a *set* of vectors. The upper bound $\vec{U}$ on the payoff that MAX can achieve by choosing a single element of this set is a vector formed by taking the pointwise maximum of all the set's members. If m branches have been analysed at ν, producing the vectors $\vec{U}_1, \vec{U}_2, \cdots, \vec{U}_m$, the beta value representing the bound for the remaining branches at ν is $\min_i \vec{U}_i$, where $\min_i$ is as defined in Equation 1. The *pruning* of branches (for instance, with a full vector-based version of alpha-beta, as described in [18]) is difficult in our Bridge trees, as MIN branches are associated with card plays. Each MAX node in a tree is therefore reachable in at least one world for which its siblings are not. Thus, the vectors produced at a MAX node always have at least one numerical payoff where the vectors of its siblings have a $\perp$, and vice-versa.

5 Generating Explanations

As defined in the Introduction, a computer solves a game when its play is both optimal and *explainable*. We now show how the optimal trees of tactics produced by FINESSE's combination of knowledge and search can be automatically converted into explanations comparable to those given by human experts. FINESSE's strategy explanation procedure has three steps. The first identifies from a strategy the paths that need to be explained. The second removes from consideration move sequences or game situations that would be considered too simple to explain to human players. Finally, the third produces plain text with the aid of patterns and idioms. These steps are largely game-general. We describe each one below.

Step 1: Path Extraction

The strategy to be explained defines a subset of the paths in the tactic search space by virtue of selecting specific MAX branches (e.g., a finesse of a Queen). Hence, our first step is to extract those paths that form part of the optimal strategy. In doing so we build a new tree.

To decide which paths to explain, we take into account the goal (from Figure 10) that a strategy optimises. When explaining a strategy for the goal *max*, some paths in the strategy may fail to produce the maximum possible number of tricks. For example, when finessing in our example of AQ-2, the MIN branch x/K (where the Queen loses to East's King) leads to a leaf node where declarer has won just one trick (with the Ace). Such paths that fail to produce the maximum number of tricks are ignored. Similarly, for the goal *Ntricks*, we extract only paths that achieve or exceed N tricks at the leaf node. And, for *expected*, we extract only paths for which the payoff at the leaf nodes is higher than the *minimum* possible payoff for the problem.

The paths extracted from a search space in this way are recombined into a new tree where paths that differ in just their MIN plays are merged. For example, Figure 12 shows the new tree produced by FINESSE for the optimal strategy when playing AQ-2. The two branches represent four paths through

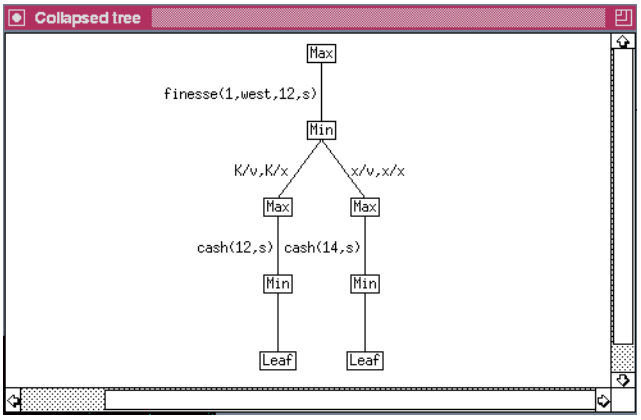

Fig. 12. The paths extracted by FINESSE from the game tree for the AQ-2 problem

the original tree. The left-hand branch represents the two paths where West plays the King, so that declarer wins with the Ace and cashes the Queen on the second round (MIN branches K/v and K/x). The right-hand branch represents the two paths where West plays low and the Queen wins, so that the Ace can then be cashed (x/v and x/x). It should already be clear that this reduced tree is more amenable to explanation than the original of Figure 7. In general, the MAX branches in these extracted trees are labelled with a single tactic (each path in the tree corresponds to one possible sequence of tactics) whereas the MIN branches have multiple labels.

Step 2: Pruning

Sometimes parts of a strategy describe situations that are so straightforward that only a beginner would expect an explanation. As described below, we implement two functions that identify and prune (1) bad moves, and (2) situations that are particularly easy to play. This leaves the parts of the game tree containing only the 'interesting' parts of the original strategy.

The first pruning function identifies clearly bad moves. In our trees, each MIN branch can only be followed under certain distributions of the outstanding cards. We therefore find bad moves by looking for branches that should *never* be followed by the East and West. Such branches can be identified by checking that for every possible world in which they can be followed, there is a different path through the tree that gives MAX a lower payoff.

As an example, consider again the finesse branch in the tree of Figure 7. We did not draw attention to this before, but in this tree there are actually not four paths that lead to two tricks, but five. The extra path is one where MAX leads the two and West discards from another suit with East playing the King (branch v/K). If the first opponent discards in this way when a type 1 finesse is attempted, the tactic is doomed to fail (since all the remaining cards in the suit are held by the other opponent). FINESSE therefore aborts the finesse attempt after seeing West's play and plays the Ace from the North hand on the first trick. For East to play the King under North's Ace is clearly giving away a free trick to the Queen, so branch v/K should never be followed by the defence. In practice, FINESSE identifies such branches by applying payoff-reduction, as described in §4.3, to the game tree determined by a strategy.

The second pruning function identifies easy playing situations. We first prune MIN nodes if each branch of the node leads to a linear tree. The justification for this is that linearity implies that MAX's best moves are not affected by the responses made by MIN: whatever moves MIN makes, MAX's best strategy is straightforward. Second, we prune a particularly trivial kind of Bridge situation: any MIN node with a subtree containing only cash tactics. This heuristic is justified by noting that cashing a top card is the simplest possible play. Without this pruning rule, FINESSE would generate lengthy explanations with strings such as 'then cash the ten and the nine and the eight'. The reader of FINESSE's explanations is expected to bear in mind that if no explicit direction is given, the default action is to cash a top card. A simple example of pruning is provided by Figure 12. Both the linearity and the cashing heuristic identify the top-most MIN node as an easy situation, leaving just a single MAX branch to be explained: finesse the Queen.

Step 3: Pattern Matching

The final step in automatically explaining strategies is performed by pattern matching, which maps the branches of an extracted and pruned tree into plain text. Most of the basic operations carried out in the pattern matching phase are game-general. We describe three representative examples here.

First, for a tree such as that of Figure 13, the natural explanation is not 'Do tactic T1 then whatever MIN does, do tactic T2,...'. Much better is 'Do tactics T1 and T2,...'. We incorporate a sequentialisation step to compile down such linear sequences in trees. Second, MIN branches can have multiple labels

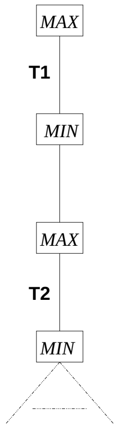

Fig. 13. A linear sequence in a tree

(*e.g.*, as in Figure 12). When this is the case, the branches are ordered so that the branch with fewest labels comes first. This allows the final (longest) branch label to be explained as 'otherwise...'. Finally, as a special case of sequentialisation, a tree may contain a linear sequence where T1 and T2 are the same tactic. This is rendered as 'Do tactic T1 and repeat if necessary'.

Explanations are produced by a simple recursive function that incorporates the above operations. The basic action is simple: for each branch at the root of a tree, output some text describing the branch and then explain the subtree. If the root is a MAX node, the output text describes the tactic, and if the root node is a MIN node, the text describes the MIN moves. For formatting, line-breaks are added after each MAX tactic, and the ith MIN level is indented i spaces.

Only limited Bridge knowledge is required to explain the tactics, since they already correspond directly to Bridge plays that humans understand. It is also relatively simple to describe the MIN plays in terms of the cards they represent. Just one further piece of Bridge-specific knowledge is required. If the extracted tree is linear, the tree can only contain cash tactics. We explain such trees as simply 'cash top honors'. Finally, note that there is nothing in our account that requires explanations to be in the *English* language. The generation rules are simple enough for us to also construct another set that works in, say, Japanese or German.

6 Experience

We tested our algorithms on the suit combinations from the Official Encyclopedia of Bridge [1]. This book contains a 55-page section presenting optimal strategies for a selection of 665 single-suit problems, chosen for their coverage of the possible play situations that arise in practice. There were several reasons for choosing this test set:

- to demonstrate that *vec-prop* always terminates quickly on a large and representative sample of problems that occur in single-suit play.
- to show empirically that the tactics significantly reduce the size of the search space without sacrificing optimality (i.e., any strategy optimal in the space of legal moves is also present in the tactic space, for all the possible goals in Figure 10).
- To assess the time savings due to tactics.
- To compare the performance of human experts with that of FINESSE, both in finding optimal solutions, and in explaining them.
- To provide a basis for comparison with other algorithms for Bridge, such as Monte-carlo sampling.

To carry out our tests, we created three separate test databases from the problems in the Encyclopedia by collecting together its strategies for the three goals we gave in Figure 10 (*max*, *Ntricks*, and *expected*). For any given problem, the Encyclopedia gives a strategy for at least one of these goals (usually *max*), and, in some cases, accompanies this with solutions for one or more of the other goals. We omitted only ten of the Encyclopedia strategies, because they did not fit the best defence assumptions: four involving East/West strategies with probabilistic choices, five relying on assumptions about sub-optimal defence, and one involving constraints on the cards that could be played. In total, our databases contain 1567 problems, as summarised in Figure 14.

Strategy optimises chance of	Size (problems)	Database number
Maximum number of tricks	650	I
At least N tricks, $N <$ maximum	469	II
Highest expected number of tricks	448	III

Fig. 14. Databases created from the Official Encyclopedia of Bridge

6.1 Test Results

For each of the problems in our databases, we applied vector propagation to the space of possible card plays as well as the space of tactics. The optimal strategies in both cases were the same. This validates both that tactics can be used without sacrificing optimality, and that vector propagation is sufficient to solve single-suit Bridge.

On average, the trees of tactics have 1,330 leaf nodes and take 0.39sec to build (on an Ultra SPARC-II at 450MHz). In contrast, the trees of possible card plays (even if we generate just one branch for plays of cards from the same sequence) have an average of 47,000 leaf nodes and take 1.09sec to build. The basic implementation of *vec-prop* (with subsumption check) requires an average of 0.25sec to search the tactic trees and 2.79sec for the trees of possible card plays. Thus, the overall average time to construct and find a solution in the tactic space is 0.64sec, compared to 3.88sec in the space of possible plays. This is a modest saving, but the difference is more dramatic when comparing the *worst case* time to solve problems in the data set (see Figure 15).

Algorithm	With tactics		Without tactics	
	Avge (sec)	worst (sec)	Avge (sec)	worst (sec)
vec-prop	0.65	52.6	3.88	83.3
with beta-reduction	0.63	7.1	7.01	152.2
with payoff-reduction	0.67	7.8	—	—

Fig. 15. Performance of *vec-prop* when heuristics are added

For applications such as a tutoring system, or a program for playing complete games of Bridge, a worst-case time requirement of more than 80 seconds for one suit combination is at the outer limits of practical use. However, Figure 15 illustrates that tactics lead to a significant saving when using either of the heuristics for reducing payoffs. These heuristics both involve the overhead of maintaining the upper bounds, and then reducing the payoffs at the leaf nodes. However, for the trees of tactics, this overhead is compensated for by the reduction in the number of strategies produced at internal nodes by *vec-*

prop (the number of strategies produced at the root nodes is unaffected, the average number being 6.4). For the trees of possible card plays, on the other hand, the large number of leaf nodes results in an overhead that outweighs the saving to the point that speed is almost halved. Indeed, for the payoff-reduction heuristic, the memory requirements of storing payoffs prevented us from running a complete test on the trees of possible plays.

To provide a comparison of how other algorithms fare on the Encyclopedia databases, we also implemented the technique of Monte Carlo sampling. As discussed in §2.2, this is the approach used by the strongest Bridge playing programs. To give Monte Carlo sampling the best possible chance of success, we let it examine *all* possible worlds. Nevertheless, it made a total of 600 errors on the databases (when exploring the search space as defined by FINESSE's tactics). These errors are all small deviations from the optimal strategies that would result in missing the desired goal in an average of 7% of possible worlds. Thus, if the entire set of database problems were to be played out against randomly distributed cards, the expected number of cases where the goal would be missed is (7/100) * 600 = 42 (about 2.7%). The reasons for this sub-optimality are explained in [16]. We now go on to consider how the Encyclopedia's solutions compare to the (optimal) ones produced by FINESSE.

6.2 Discovery of Errors

FINESSE discovered a significant number of errors in the Encyclopedia solutions. This is somewhat surprising, as the Encyclopedia has gone through many editions, and has been a well-known reference for serious Bridge players for many years. The overall error rate uncovered by FINESSE was over 3%, as summarised in Figure 16.

Database	Encyclopedia Errors	Error Rate
I	16	2.5%
II	11	2.4%
III	31	6.9%
Overall	58	3.7%

Fig. 16. Errors discovered by FINESSE in the Encyclopedia

To illustrate the problems involved, the 16 errors discovered by FINESSE in database I (we ignore the simple typo of Problem 609) are summarised in Figure 17. In general, when the Encyclopedia gives a lower chance of success than FINESSE, the reason is that the optimal strategy has been overlooked. When the Encyclopedia's chance of success is higher, the cause is the Encyclo-

Best strategy overlooked					
Num	Cards (N-S)	Tricks	Ency	Finesse	Comment
(289)	KQ9x-Jxx	Max: 3	62%	63.7%	Finesse is better by 1.7%
(477)	AJ8-Txx	Max: 2	39%	39.7%	Finesse is better by 0.7%
(543)	KT8xx-Jxx	Max: 4	25%	26.6%	Finesse is better by 1.6%
(568)	ATxx-9xx	Max: 2	51%	57.5%	Finesse is better by 6.5%
(601)	Qxxxx-J9x	Max: 3	48%	50.3%	Finesse is better by 2.3%
(622)	KTxxx-9xx	Max: 3	63%	66.7%	Finesse is better by 3.7%

Errors in probability calculation					
Num	Cards (N-S)	Tricks	Ency	Finesse	Comment
(31)	A9xx-KQx	Max: 4	48%	47%	Faulty analogy with a previous problem
(430)	KQT9x-x	Max: 4	14%	13.3%	Probable rounding error

Best defence not modelled correctly					
Num	Cards (N-S)	Tricks	Ency	Finesse	Comment
(162)	AQ8x-Jxx	Max: 4	59%	56.8%	Against best defence, Ency solution has 54.4% chance
(437)	QT8xx-x	Max: 5	21%	19.4%	Against best defence, Ency solution has 1.6% chance
(459)	K987-Qxxx	Max: 3	23%	15.8%	Actual strategies same
(541)	Kxxx-J9x	Max: 2	47%	39.7%	Actual strategies same
(548)	KT8x-Jxx	Max: 3	27%	19.6%	
(590)	QJ98-xxx	Max: 2	63%	58.2%	Against best defence, Ency solution has 53.3% chance
(591)	QJ9x-xxx	Max: 2	55%	51.8%	
(595)	Qxxx-J9x	Max: 2	33%	31.8%	Against best defence, Ency solution has 30.2% chance

Fig. 17. Detail of Encyclopedia errors found by Finesse in database I. The problem numbers are the numbers in the Encyclopedia, and the percentages are those given by the Encyclopedia and Finesse for the maximum number of tricks.

pedia's calculation of probabilities. Overall, there are six problems where the Encyclopedia misses the best strategy, two where it makes errors in calculating probabilities, and eight where there are unstated or mis-stated assumption of sub-optimal defence. In this latter category, problem (437) is particularly striking. The strategy given by the Encyclopedia is slightly better than FINESSE's if we assume that East/West do not play optimally. However, against best defence, it has a chance of success of just 1.6%.

The discovery of these errors is evidence of the difficulty of the problem, and also of the effectiveness of our approach. It is also of interest to Bridge players. FINESSE's identification of errors in the Encyclopedia, and in particular its discovery of two novel strategies, was recently the subject of an article in the leading Bridge magazine Bridge World [17].

As for the other references on suit combinations, we have not used FINESSE to systematically analyse them (entering the data would be time consuming, and we know, in any case, that we can find optimal strategies by applying *vec-prop* to the space of possible card plays). However, we have made use of Roudinesco's Dictionary [30] to provide a third opinion when the Encyclopedia and FINESSE differ. The number of errors discovered by doing this stands at thirteen. For example, the Dictionary misses the best strategy for the Encyclopedia problems (568) and (601) (these were the two problems for which FINESSE's novel strategies were described in Bridge World).

6.3 Comparing Explanations

We used FINESSE to generate explanations for all the problems in our databases (generating explanations takes negligible time). Here we present a sample of FINESSE's explanations and compare them to those of the Encyclopedia. These show that FINESSE not only produces strategies that are *explainable* in human terms (and thus solves single-suit play in the technical sense described in the Introduction), but that FINESSE itself is capable of producing explanations comparable to those of human experts.

We present problems numbered as they are appear in the Encyclopedia, giving the suit combination and the maximum number of tricks that can be taken. For each problem, we give the Encyclopedia's and FINESSE's explanations of how to take the maximum number of tricks, with FINESSE's text in `boldface` font. We select a representative sample, to give an accurate picture of the strength of the algorithms in §5, and also of the differences between FINESSE's explanations and the Encyclopedia's.

Num	Cards (N-S)	Encyclopedia & Finesse explanations
(1)	AKQJ9-x (Max: 5)	Cash top honors in the hope of dropping the ten `cash top honors`
(2)	AKQJ9x-x (Max: 6)	Cash top honors `cash top honors`

(3)	AKQJ9-xx
	(Max: 5)
		Cash top honors in the hope of dropping the ten
		`cash the Ace`
		` if East shows out finesse the nine`

(4)	AKQ9x-Jx
	(Max: 5)
		Cash top honors in the hope of dropping the ten
		`cash the Jack`
		` if East shows out finesse the nine`

The phrase 'in the hope of dropping the ten' used in the Encyclopedia's explanations is not describing part of the strategy itself. Instead, it is a statement of the conditions under which the strategy succeeds. In this paper, we have given just our algorithm for the textual explanation of strategies, but extra information such as a textual description of the world states represented by a strategy's payoff vector is easily added. In (3) and (4), Finesse gives explicit detail on what to do when East discards from another suit on the first round. Finesse does not state the continuation of cashing Ace, King and Queen when East is *not* void, since the default action to take when Finesse's explanations give no explicit direction is to cash a top card.

Num	Cards (N-S)	Encyclopedia & Finesse explanations
(5)	AKQJ8-xx (Max: 5)	Cash top honors. (But against defenders who would not false-card from 109x or 109xx, cash the jack and finesse the eight if the nine or ten appears from East) `cash top honors`
(6)	AKQ8x-Jx (Max: 5)	See (5) above `cash top honors`

(7)	AKQ8x-Jxx
	(Max: 5)
		Cash the jack first in case East is void
		`cash the Jack`
		` if East shows out finesse the eight`
		`  and repeat if necessary`

(8)	AKQ10-x (Max: 4)	Finesse the ten `finesse the ten`
(9)	AKQ109-x (Max: 5)	Play off the top honors. This is fractionally better than the immediate finesse `cash top honors`

In §3.2 we discussed the implicit model of best defence that the Encyclopedia uses to assess Bridge strategies. However, as in (5), the Encyclopedia sometimes gives alternative strategies that are designed to exploit weak opponents. Usually (as in this case), if the assumption of weak defence is incorrect, the alternative strategy's chance of success will be lower than that of the 'best defence' strategy. Finding such strategies in Finesse would require relaxing the assumptions of the best defence model, for which we currently have no algorithm. However, in (9), the Encyclopedia contrasts the chance of success of the best strategy against the chance of success of the nearest competitor. Such comparisons are easily made in FINESSE, since the competing strategies will typically all be present in the set of vectors produced by *vec-prop*. In this example, for instance, playing the top honors wins in some worlds for which the finesse doesn't, and vice-versa. So, neither strategy subsumes the other and *vec-prop* returns them both in a set of size two for this problem. Note also that (7) provides an example of the recognition of a repeated tactic.

Num	Cards (N-S)	Encyclopedia & FINESSE explanations

(11)	AKQ10-xx (Max: 4)	Cash the queen, and then finesse the ten
		`cash the Ace`
		`if both play low or East shows out finesse the 10`

(12)	AKQ9-xx (Max: 4)	Finesse the nine: hope that West has both the jack and ten
		`finesse the nine`
		`and repeat if necessary`

(13)	AK10x-Qx (Max: 4)	Cash the queen, and then finesse the ten
		`cash the Queen`
		`if both play low or East shows out finesse the 10`

(14)	AK9x-Qx (Max: 4)	Play off the queen, king, and ace, hoping that the jack and ten fall in three rounds. (But against defenders who would not falsecard from J10x, cash the queen and finesse the nine if East drops an honor) `cash top honors`

(11)–(13) show FINESSE explaining the same strategies as the Encyclopedia in slightly different terms. The differences are simply a matter of style; any Bridge player will understand either explanation equally well. (14) shows that the Encyclopedia introduces variation in its explanations (*i.e.*, explaining how to cash the top honors, even though this is trivial).

Num	Cards (N-S)	Encyclopedia & FINESSE explanations
(22)	AK9xx-Q10 (Max: 5)	Cash the queen, king and ace `cash the Queen`
(23)	AKxxx-Q10 (Max: 5)	Finesse the ten `finesse the ten`
(24)	AK9xx-Qx (Max: 5)	Play off the top honors `cash top honors`

(25)	AKQ10-xxx (Max: 4)	Cash the king and queen; if both follow, play the ace. This is 2% better than a third-round finesse
		`cash the Ace and King`
		`if East shows out finesse the ten`

In (22) it is important to begin with the cashing of the Queen, since this allows 5 tricks to be won even when the Jack is singleton. Both the Encyclopedia and FINESSE explicitly state that the Queen should be cashed first, but neither state why. (The Encyclopedia also says to play the Queen first in (11), but there the order is *not* significant.) (25) shows an example of sequentialisation. Note again that FINESSE concentrates on giving explanations of when to finesse, rather than when to cash, since cash branches are specifically removed by the pruning heuristics.

Num	Cards (N-S)	Encyclopedia & FINESSE explanations

(26)	AKQ9-xxx
	(Max: 4)
		Cash the queen and king; if an honor drops from East, finesse the nine next. This is 6% better than cashing the three top honors regardless
		`cash the Ace`
		`if East shows out finesse the nine`
		`and repeat if necessary`
		`otherwise, cash the King`
		`if East shows out or plays the Jack or ten`
		`finesse the nine`

(27)	AK10x-Qxx
	(Max: 4)
		See (25) above
		`cash the Ace`
		`if both follow low cash the Queen`
		`if East shows out finesse the ten`
		`if East shows out finesse the ten`

(28)	AK9x-Qxx
	(Max: 4)
		See (26) above
		`cash the Queen`
		`if East shows out finesse the nine`
		`and repeat if necessary`
		`otherwise cash the Ace`
		`if East shows out or plays the Jack or ten`
		`finesse the nine`

Referring to a previous example, as in (27) and (28), is another way of varying the presentation style. There was a simple case of this in (6), but here the optimal strategies are not so trivially recognised as being equal.

Num	Cards (N-S)	Encyclopedia & FINESSE explanations

(31)	A9xx-KQx
	(Max: 4)
		See (26) above
		`cash the King and Queen`
		`if both follow low or East shows out or East`
		`plays the Jack or ten, finesse the nine`

(39)	AK9x-Q8xx
	(Max: 4)
		Cash the ace. If an honor appears, cash the next top honor from the hand on the left of the Jack or ten.
		`cash the Ace`
		`if East plays the Jack or ten, cash the Queen`
		`if East shows out finesse the nine`
		`otherwise, cash the King`
		`if West shows out finesse the eight`

(62)	AK98x-Jx (Max: 5)	Run the jack *or* lead small to the nine `run the jack and finesse the eight`

Although the Encyclopedia relates Problem (31) to (26), this is mistaken. In (31), it is not possible to win 4 tricks when East is void. FINESSE produces a correct strategy and explanation. For (39), FINESSE is not capable of replicating the complexity of the Encyclopedia's explanation, yet produces an equivalent description that is easy to understand. We finish with problem (62) since it illustrates another simple way to improve FINESSE's explanations: whenever there are multiple strategies that achieve the desired goal in the set produced by *vec-prop*, explain them all.

The above examples show that FINESSE's algorithm for converting strategies to plain text produces explanations comparable to those of the Encyclopedia. They also highlight the extra possibilities — over and above simply explaining an optimal strategy — that can be used to improve explanations in a reference text. We are using all these techniques in producing a book with FINESSE, except for the last, which remains future work:

- give descriptions of card distributions under which strategies succeed,
- vary the presentation and terminology when describing a number of similar problems, *e.g.*, 'play off the top honors' versus 'cash top honors',
- compare the optimal strategy with slightly inferior strategies for a problem,
- give pointers to problems with similar solutions,
- explain more than one strategy, when the optimal strategy is not unique, and
- mention alternative strategies that can take advantage of weak defence.

Even without including this kind of information in its explanations, FINESSE satisfies the requirements for solving single-suit play, automatically explaining optimal strategies in plain English text. In addition to producing an error-free reference text, this also makes FINESSE an ideal basis for a tutoring system capable of teaching users the optimal way to play suit combinations.

7 Conclusions and Further Work

We have shown that the combination of a set of seven tactics and a new search algorithm can be used to find and explain all optimal strategies for Bridge suit combinations. Using tactics to constrain search increased efficiency, and resulted in compact strategies constructed from human-understandable units. On a test set of Bridge problems from a definitive expert text, FINESSE identified 58 errors in the expert solutions and also discovered strategies that were new to the Bridge community. Moreover, FINESSE's explanations of its strategies were comparable to those produced by the human experts.

One promising direction for further work is investigating how *vec-prop* performs in other games with imperfect information. The single-suit results reported here for Bridge could also be extended towards a solution for the full game of Bridge, either by investigating how *vec-prop* performs on larger trees (such as endgame positions with cards from more than one suit) or by combining plans for suit combinations into global ones. The FINESSE system as it stands is also an ideal basis for a tutoring program for single-suit play.

Appendix: Example Tactic Preconditions

Below we give two examples of the preconditions of FINESSE's seven tactics: the cash and the Type 1 finesse. These preconditions use several metalanguage predicates. With the help of the comments, the meaning of these predicates should be evident. For instance, `state_data` accesses parts of the suit-combination 6-tuple that characterises the current game state.

Note that in the preconditions for the Type 1 finesse (which we also discussed in §3.4), the number of losers is used to identify situations where no manœuvre is necessary. In general, with no losers, the only necessary tactic is a cash. With just one loser, the duck tactic can still be excluded, since executing the duck itself gives a trick to the opponents, and this brings no advantage (in a single suit) if there is only one loser.

```
/***********************************************************************
* cash                                                                 *
*   Cashing top tricks. Examples:                                      *
*                                                                      *
*    (cash A)   (A or K)     (A or K)      (Jack)      (Ace)    (Ace)  *
*       A          A x          A K      A K Q x x x     A        A    *
*                                                                      *
*      K x        K x          Q J           J           K        -    *
*                                                                      *
*  Preconditions:                                                      *
*  (1) Declarer must have at least one card that is higher than any of *
*      the defence's                                                   *
*  (2) The opposite hand to one of these high cards must be able to    *
*      duck (or be void).                                              *
*  (3) This duck card must not be from the sequence of winners (unless *
*      declarer has more high cards than length in the suit and can    *
*      afford to throw winners on winners).                            *
*  (4) As with other tactics, we want to generate a maximum of two per *
*      state (one winner from North, one from South).                  *
***********************************************************************/
manoeuvre_applicable(State, cash(Card,Suit) ) :-
  state_data(north, State, North),            % get data
  state_data(south, State, South),            % get data
  state_data(out, State, Out),                % get data
  ns_sequence(North, South, Out, 15, Tops),   % find top declarer cards
  Tops \== [],                                % condition (1)
  slength(North, NorthLength),                % need to calculate lengths
  slength(South, SouthLength),                % of the longest of the
  max(NorthLength, SouthLength, Longest),     % North and South hands, and
  length(Tops, SeqLength),                    % the sequence of top cards
  cash_direction(North, South, First, Second), % suggest a hand to play first
                                              % [condition (4)]
  card_from_sequence(First, Tops, Card),      % first hand has cash card
  duck_card_or_void(Card, Second, DuckCard, _),% condition (2)
  if_then(  SeqLength =< Longest,             % condition (3)
            \+ member(DuckCard, Tops) ),
  North = suit(_,Suit,_).                     % instantiate Suit
```

```
/***********************************************************************
* finesse - Type 1                                                     *
*  A low card is led towards a tenace.                                 *
*                                                                      *
*    A Q 10        A J 10                                              *
*                                                                      *
*    9 4 2         8 6 5 3                                             *
*                                                                      *
*   This has the preconditions:                                        *
*   (1)  There must be at least one loser in the suit.                 *
*   (2)  There must be an outstanding critical card...                 *
*   (2a)  ...with other outstanding cards below it.                    *
*   (3)  Declarer has a sequence of any length beneath the Critical card.*
*   (4)  Declarer has a card that can beat the Critical card, in the   *
*        same hand as a card from the above sequence.                  *
*   (5)  The opposite hand has a low card to lead toward the tenace.   *
*   (6)  The player being finessed is not known to be void.            *
***********************************************************************/
manoeuvre_applicable(State, finesse(1, Finessed, FinCard, Suit) ) :-
  state_data(losers, State, Losers),           % get data on losers
  Losers > 0,                                  % condition (1)
  state_data(north, State, North),             % get other data
  state_data(south, State, South),             % ...
  state_data(out, State, Out),                 % ...
  state_data(crits, State, Critical),          % ...
  state_data(voide, State, VoidE),             % ...
  state_data(voidw, State, VoidW),             % ...
  critical_card(Critical, CritCard),           % condition (2)
  duck_card(CritCard, Out, _, _),              % condition (2a)
  ns_sequence(North, South, Out, CritCard, Sequence), % condition (3)
  finesse_direction(North, South, Sequence, VoidE, VoidW, % Given these...
    FirstHand, ThirdHand, VoidFirst, Finessed), % Suggest a finesse direction
  VoidFirst == no,                             % condition (6)
  lowest_card(ThirdHand, Sequence, FinCard).   % condition (4)
  cover_card(CritCard, ThirdHand, _, _),       % condition (4)
  duck_card(FinCard, FirstHand, _, _),         % condition (5)
  North = suit(_,Suit,_).                      % instantiate Suit
```

References

[1] ACBL. *The Official Encyclopedia of Bridge.* American Contract Bridge League, Inc., 2990 Airways Boulevard, Memphis, Tennessee 38116-3875, fifth edition, 1994.

[2] L.V. Allis, H.J. van den Herik, and I.S. Herschberg. Which games will survive? In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*, pages 232–243. Ellis Horwood, 1991.

[3] Andrew Robson Bridge Club. Man vs machine: Results of contest between Zia Mahmood and seven computers. `http://www.arobson.co.uk/pages/manvmach.htm`, September 1999.

[4] S. Brock. *Suit Combinations in Bridge.* B T Batsford Ltd, 1998. ISBN 0713481641.

[5] Alan Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.

[6] Alan Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993. Also available from Edinburgh as DAI Research Paper No. 567.

[7] Alan Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991. Earlier version available from Edinburgh as DAI Research Paper No 413.

[8] M. Buro. The Othello match of the year, Takeshi Murakami vs Logistello. `http://www.neci.nj.nec.com/homepages/mic/event.html`, August 1997.

[9] M. Buro. From simple features to sophisticated evaluation functions. In H. J.van den Herik and H. Iida, editors, *Computers and Games: Proceedings CG'98. LNCS 1558*, pages 126–145, Tsukuba, Japan, 1999.

[10] Gay Carley. A program to play contract bridge. Master's thesis, Dept. of Electrical Engineering, Massachussets Institute of Technology, Cambridge, Massachussets, June 1962.

[11] N. Charness. Components of skill in bridge. *Canadian Journal of Psychology*, 33(1):1–16, 1979.

[12] N. Charness. Expertise in chess and bridge. In D. Klahr and K. Kotovsky, editors, *Complex Information Processing: The Impact of Herbert A. Simon*, pages 183–208. Erlbaum, Hillsdale, NJ, 1989.

[13] Anthony Davey. *Discourse production : a computer model of some aspects of a speaker.* Edinburgh University Press, 1978.

[14] R. W. Engle and L. Bukstel. Memory processes among bridge players of differing expertise. *American Journal of Psychology*, 91(4):673–689, December 1978.

[15] I. Frank. *Search and Planning under Incomplete Information: A Study using Bridge Card Play.* PhD thesis, Department of Artificial Intelligence, Edinburgh, 1996. Also published by Springer-Verlag in the Distinguished Dissertations Series, ISBN 3-540-76257-4, 1998.

[16] 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.

[17] I. Frank and D. Basin. Make no mistake: Computers vs suit combinations. *Bridge World*, 2000. To appear.

[18] I. Frank and D. Basin. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, 2000. To appear.

[19] I. Frank, D. Basin, and A. Bundy. An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, pages 72–76, Vienna, Austria, 1992. Longer Version available from Edinburgh as DAI Research Paper No. 575.

[20] I. Frank, D. Basin, and H. Matsubara. Monte-carlo sampling in games with incomplete information: empirical investigation and analysis. In *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.

[21] I. Frank, D. Basin, and H. Matsubara. Finding optimal strategies for imperfect information games. In *Proceedings of AAAI-98*, pages 500–507, 1998.

[22] M. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of IJCAI-99*, pages 584–589, Stockholm, Sweden, 1999.

[23] M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.

[24] Deepak Khemani. Planning with thematic actions. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 287–292, Menlo Park, CA, 1994. AAAI Press.

[25] M. Kosmulski. *Rozgrywka pojedynczego koloru.* PZBS (Polski Zwiazek Brydza Sportowego), 1990. in Polish (English title: "Suit Treatment").

[26] 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.

[27] C.N. Napjus. *Declarer: A Learning Bridge-Playing Program which Generalizes and Infers.* PhD thesis, University of Washington, 1969.

[28] Y. Nygate and L. Sterling. Aspen — designing complex knowledge based systems. In *10th Israeli Symposium on Artificial Intelligence*, pages 51–60, 1993.

[29] A. Robinson. Informal rigor and mathematical understanding. In G. Gottlob, A. Leitsch, and D. Mundici, editors, *Computational Logic and Proof Theory*, pages 54–64. Springer-Verlag, Lecture Notes in Computer Science, vol. 1289, 1997.

[30] J.M. Roudinesco. *The Dictionary of Suit Combinations.* Guy Trédaniel, 1996. ISBN 2-85707-825-0.

[31] 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.

[32] S.J.J. Smith and D.S. Nau. An analysis of forward pruning. In *Proceedings of AAAI-94*, pages 1386–1391, 1994.

[33] A. Stanier. Planning to make tricks at bridge. In *Proceedings of the AISB Summer Conference*, pages 256–265, 1976. Also in D. Levy, editor, *Computer Games II*, Springer Verlag, New York, 1988.

[34] J.W.H.M. Uiterwijk, H.J. van den Herik, and L.V. Allis. A knowledge based approach to connect-four. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence – The First Computer Olympiad*, pages 113–133. Ellis Horwood, 1989.

[35] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behaviour.* Princeton University Press, 1944.