Combining Knowledge and Search to Solve Single-suit Bridge

Ian Frank
Complex Games Lab
Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305
ianf@etl.go.jp

David Basin
Institut für Informatik
Universität Freiburg
Am Flughafen 17
Freiburg, Germany
basin@informatik.uni-freiburg.de

Alan Bundy
Division of Informatics
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
A.Bundy@ed.ac.uk

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 Bridge tactics constrain search to the point where optimal strategies can quickly be identified, and also provide the key to explaining these strategies in human-understandable terms. We demonstrate this using a canonical set of single-suit Bridge problems from a definitive expert text. FINESSE 'solves' these problems in the technical sense that, in addition to always finding optimal solutions (and revealing a 3% error rate in the expert answers), it also explains each solution in simple, clear English text.

Introduction

There is a big difference between finding a solution to a problem and being able to communicate that solution to others. In computer games, this distinction has been referred to as the difference between *cracking* and *solving* a game (Allis, van den Herik, & Herschberg 1991). To crack a game, a program needs to play optimally. But to solve it, the program's play must also be explainable in human-understandable terms.

In practice, solving a game is much harder than cracking it. Considerable advances in search algorithms and specialised 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 contrast to this trend, we show in this paper 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. We consider a sub-problem of the game 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. There are a number of Bridge books that cover the problem in depth (ACBL 1994; Kosmulski 1990; Brock 1998; Roudinesco 1996)[1], and the author of one of these (a world life master), notes:

> '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. (Roudinesco 1996, p.9)

Cognitive studies (Engle & Bukstel 1978; Charness 1979; 1989) have shown that human players routinely plan their Bridge play by first considering individual suit combinations, and that human performance in Bridge can be attributed to the acquisition of high-level patterns and chunks of knowledge gained through experience. Our contribution here is to show that a computer can also use a set of patterns to find and explain optimal strategies for single-suit play.

To do this, we formalise expert knowledge about Bridge as a set of seven *tactics*, each of which represents a simple, distinct manœuvre, such as playing a winner or finessing (manœuvering against) missing cards. Any given suit combination is then solved by searching the space of possible tactic applications. This search is carried out by 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 the search, and it results in strategies that are trees of tactics rather than trees of individual moves. We show that a natural consequence of producing high-level trees is that strategies can then be explained in plain text.

Our ideas are implemented in a system called FINESSE. Although FINESSE's use of tactics significantly prunes the search space, all optimal strategies remain. We demonstrate this empirically by testing the system on a database of over 1500 problems from a canonical Bridge reference (ACBL 1994). FINESSE solves all these problems in an average of

[1] All these books implicitly use the *best defence* model (Frank & Basin 1998) to analyse Bridge problems. In this model, (1) the opponents have perfect information; (2) they choose their strategy second; and (3) the optimal strategy is a pure strategy (making no use of probabilistic choices such as '80% of the time do this, 20% of the time do that'). This is also the model assumed throughout the current paper.

just over 0.6sec, and also finds 58 errors in the test database (a rate of over 3%). Sometimes, FINESSE even discovers strategies that are new to the Bridge community; two of these new strategies were the subject of a recent article in a leading Bridge magazine (Frank & Basin 2000a).

Previous research on high-level approaches for playing computer Bridge includes the work of (Smith & Nau 1996), who won the 1997 computer Bridge championships with a program incorporating AI planning techniques. We have also previously shown the feasibility of using tactics for single-suit play (Frank, Basin, & Bundy 1992). Especially in the light of the cognitive research cited above, there are many worthwhile questions to be answered on how such knowledge-based approaches can be used to replicate the human ability of piecing together single-suit plans into global strategies. The more recent success of brute-force techniques such as GIB (Ginsberg 1999) suggests that the small but frequent errors made by a Monte Carlo sampling approach are no barrier to competing with strong human Bridge players, but the research reported in this paper represents the first time that a part of the game has been cracked or solved. FINESSE is fast enough for real-time use — for example as part of a program for playing complete hands — and its further ability to explain strategies in plain text suggests the viability of our approach as the basis of a real-time Bridge tutor.

A Bridge Example

For a detailed description of Bridge, readers are referred to one of the excellent books on the subject, such as (Goren 1986). Here, we give some idea of single-suit play with the following example:

♠ A Q

N
W E
S

♠ 2

In presenting single-suit problems such as this, it is conventionally assumed that South (S) is the *declarer* and his partner, North (N), places his cards on the table for all to see. The division of the remaining cards in the suit (held by East (E) and West (W)) is unknown, and the task is to specify the optimal way for South to play the cards from both his own hand and from North's. Typically, the goal is to win a certain number of *tricks*, or rounds of play.

In this example, declarer can win one certain trick by playing the Ace. However, if this card is played immediately, the only chance of making two tricks is if East or West holds the singleton King, so that it falls when the Ace is played. So, this play of *cashing* the Ace succeeds in winning two tricks for only two of the 2^{10} possible ways that the remaining 10 cards in the suit can be split between East and West.

A better solution is to play the two from the South hand. By covering whatever card West plays, declarer can then win two tricks whenever West has the King — a 50% chance. This play follows the elementary principle that the best results are obtained by forcing an opponent to play ahead of you. It is a typical example of a standard manœuvre called a *finesse*.

When planning a hand, human players will make use of their knowledge of commonly occurring patterns like the finesse to avoid having to consider all the possible combinations of plays of single cards. FINESSE attempts to replicate this capability by restricting declarer's options at each stage of the play to a similar set of such manœuvres, as defined by its tactics.

Our Bridge Tactics

FINESSE's tactics extend and improve the tactic set we introduced in (Frank, Basin, & Bundy 1992). They distill all possible attacking plays into seven distinct and simple manœuvres easily understood by Bridge players. Four of these tactics represent different types of finesse, which we simply number from 1 to 4. For example, the lead of the two we discussed in the previous section was a Type 1 finesse. The other three tactics are 'cash', 'duck', and 'sequence'. Cashing corresponds to playing a top card that is guaranteed to win (we also saw an example of a cash in the previous section). Ducking, on the other hand, is the act of deliberately losing a trick by playing the lowest cards from both the North and South hands. This is sometimes useful for increasing the chances of winning tricks with the remaining cards. Finally, sequencing handles situations that have no element of manœuvre; North or South simply plays a card from the highest sequence held by the combined hands.

To enable these seven tactics to be used to guide search, we formally specified their applicability preconditions as Prolog clauses. Each such clause takes a representation of the game state as its first argument. If the preconditions in the body of the clause succeed, the result is a tactic, represented as a Prolog term. The Prolog predicates used by FINESSE to represent tactics 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, 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).

Building the Space of Tactics

FINESSE's tactics specify the possible ways to play a single trick. The search space for a given problem is then the tree of possible tactic applications. To build this tree, a root (MAX) node is first 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 produced by iterating through the possible plays by East and West. The process of expanding MAX nodes by checking for applicable tactics is then repeated recursively, until no more tactics apply. In general, both the declarer and East/West will have a number of options at each node, so that trees will be of the form illustrated in Figure 1.

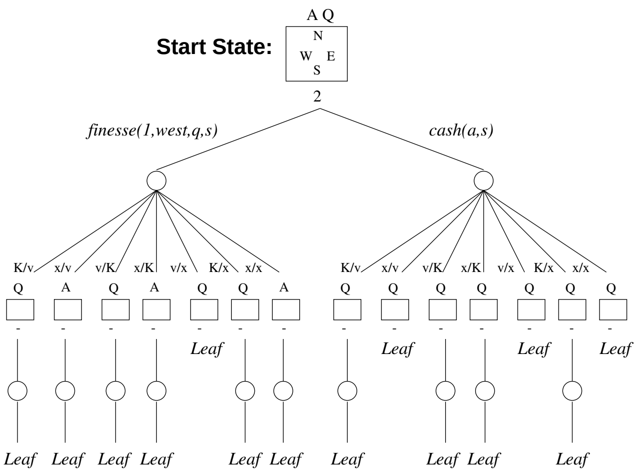

Figure 1: A game tree built with tactics

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), and an 'x' indicates an arbitrary low card. After the first trick, MAX has only one remaining card. If this cannot be cashed, then the node is a leaf node. Otherwise, the tree includes a further branch, representing the cash.

Searching the Space of Tactics

The MIN branches of Figure 1 can only be followed by East/West if they hold the appropriate cards (*e.g.*, West can only play the King if he was dealt it). If we don't distinguish between the nine low cards in this problem, 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 1 would consist of one payoff of 2 (the world where East has no spades) and nineteen $\perp$s (all the remaining worlds).

Given a game tree, 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 (see footnote on first page) 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 the n-tuple:

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

where $\min(x,\perp)=\min(\perp,x)=x$, for all x.

This simple procedure produces a single vector at the root of a game tree, giving the strategy's payoff in each world. To find an *optimal* strategy we could iterate this procedure over each strategy. In practice, though, it is more efficient to propagate payoff-vectors in parallel. To this end, we define a *vector propagation* algorithm that backs up *sets* of vectors so that the outcome of each possible strategy in a node's subtree is represented by one of the vectors in the set produced at that node. This algorithm is defined in Figure 2, 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.

In general, the number of vectors produced by *vec-prop* at any MAX or MIN node will be the same as the number of strategies in the subtree rooted on that node. This is exponential in the size of the tree. However, it is known that finding optimal strategies against best defence for this type of game tree is NP-complete in the size of the tree (Frank & Basin 2000b). Thus, unless P=NP, no efficient algorithm exists. To improve the (exponential) efficiency of *vec-prop*, we include a simple pruning step: 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 ignored as inevitably giving rise to inferior strategies. For our trees of Bridge tactics, this pruning is sufficient to allow all the problems in our databases to be solved, in an average time of just over 0.6sec.

Selecting the 'best' vector from the set at the root of a tree depends on the particular goal. We have implemented the three alternatives shown in Figure 3. 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 the ♠AQ-2 example, the best strategy for all three goals is the same: finesse the Queen.

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

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

Figure 2: The vector propagation algorithm

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

Figure 3: Three possible goals for a suit combination

Generating Explanations

FINESSE produces explanations of its strategies in three steps. The first identifies from a strategy the possible paths that need to be explained. The second removes from consideration move sequences or game situations that are too simple to warrant explanation. Finally, the third uses pattern-matching and knowledge of Bridge idioms to produce English text. We describe these three steps below. Note that each one is largely game-general, and that there is nothing in our account that requires explanations to be in *English*. The generation rules are simple enough for us to also construct another set that works in, say, Japanese or German.

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 branches at MAX nodes, such as a finesse of a Queen). To decide which of these paths to explain, we take into account the goal (from Figure 3) that the 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

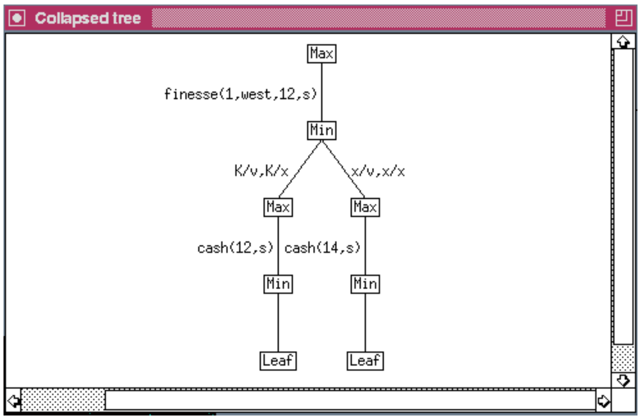

Figure 4: Screen capture of the tree extracted by FINESSE from the game tree for the ♠AQ-2 problem

recombined into a new tree where paths that differ in just their MIN plays are merged. For example, Figure 4 shows the new tree produced by FINESSE for the optimal strategy when playing ♠AQ-2. The two branches represent four paths through the original tree. The left 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 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).

Step 2: Pruning Game Trees. 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' branches 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 East/West. Such branches can be identified by checking that, for every possible distribution, there is a different path through the tree that restricts MAX to fewer tricks.

The second pruning function identifies easy playing situations. First, it prunes 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, it prunes 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. 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. In practice, this is always obvious: if the declarer still holds top cards, it is clearly advantageous to keep playing. A simple example of pruning is provided by Figure 4. Both the above pruning rules 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 English text. Most of the basic operations carried out in this step are game-general. We describe three representative examples here.

First, when there is a linear sequence such as MAX $\stackrel{T1}{\rightarrow}$ MIN $\rightarrow$ MAX $\stackrel{T2}{\rightarrow}$ $\cdots$ in a tree (*i.e.*, the MAX and MIN nodes each have a single daughter node), 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 (*e.g.*, as in Figure 4). 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 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'.

Performance: Discovery of Errors

We tested FINESSE on the suit combinations from the Official Encyclopedia of Bridge (ACBL 1994). This book contains a 55-page section presenting optimal strategies for 665 single-suit problems, chosen for their coverage of possible play situations. We created three separate test databases from these problems, corresponding to the three goals given in Figure 3. In total, these databases contain 1561 problems (less than 665×3 since the Encyclopedia often gives no strategy for one or two of the goals).

FINESSE solves all these problems correctly. On an Ultra SPARC-II at 450MHz, building the tree of tactics requires an average of 0.39sec per problem and applying *vec-prop* requires a further 0.25sec (in fact, we use a slightly modified version of *vec-prop* that incorporates the *beta-reduction* heuristic (Frank & Basin 2000b) — this leaves the average time per problem unaffected, but reduces the worst case solution time from 52sec to 7sec). The average number of leaf nodes in FINESSE's tactic trees is 1,330. 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. These larger trees can still be solved by *vec-prop* in an average of around 4sec, but the worst case solution time is over 80sec. For both types of tree, the average number of vectors produced by *vec-prop* at the root of a tree is 6.4.

FINESSE also discovers 58 errors in the Encyclopedia solutions (16 in the max database, 11 in the $Ntricks$ database and 31 in the $expected$ database). This overall error rate of 3.7% is somewhat surprising, as the Encyclopedia has gone through many editions and is a well-known reference for serious players. The discovery of these errors is evidence of the difficulty of the problem, and of the effectiveness of our approach. It is also of interest to Bridge players: FINESSE's identification of these errors, and in particular its discovery of two novel strategies, was recently the subject of an article in the magazine Bridge World (Frank & Basin 2000a).

Performance: Example Explanations

We used FINESSE to generate explanations for all the problems in our databases (generating explanations takes negligible time). We lack space here for a detailed analysis, but Figure 5 gives some typical examples. For instance, (12) illustrates the recognition of a repeated tactic, (25) provides an example of sequentialisation, and (26) and (28) show the effect of using 'otherwise...'

These examples show that FINESSE not only produces optimal strategies that are *explainable* in human terms (and thus solves single-suit play in the technical sense), but that FINESSE itself is capable of generating natural language explanations comparable to those produced by human experts. 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:

- give descriptions of card distributions under which strategies succeed, *e.g.*, (3), (12),
- vary the presentation and terminology when describing a number of similar problems, *e.g.*, 'play off the top honors' and 'cash top honors' in (5) and (24),
- compare the optimal strategy with slightly inferior strategies for a problem, *e.g.*, (25),
- give pointers to problems with similar solutions, *e.g.*, (28), (31) (although note that the Encyclopedia is mistaken to relate (31) to (26), as it is now not possible to win 4 tricks when East is void; FINESSE produces a correct strategy and explanation),
- explain more than one strategy, when the optimal strategy is not unique, *e.g.*, (62), and
- mention alternative strategies that can take advantage of weak defence, *e.g.*, (5). However, such strategies break the *best defence* assumptions that the opponents will play optimally; if the assumption of weak defence turns out to be incorrect, the chance of success of the alternative strategy will be lower than that of the 'best defence' strategy.

The last of these techniques is a topic for further research. We are using all the rest, however, in producing a computer-authored Bridge reference with FINESSE.

Conclusions

We have shown that a set of seven tactics can be used to represent all attacking plays for Bridge card combinations. The search space reduction due to searching at the tactic-level enabled us to apply a new optimal search algorithm, and resulted in compact strategies constructed from human-understandable units.

Not only are FINESSE's strategies *explainable*, but FINESSE explains its strategies *automatically*. We gave examples of these explanations and showed that they are comparable to those found in human texts. We also identified some of the further qualities beyond simply 'explaining an optimal strategy' that are found in texts written by human experts. We are currently adding these qualities to FINESSE, so that it too can author its own expert text.

Num: (3) AKQJ9-xx Max Tricks: 5

Cash top honors in the hope of dropping the ten

cash the Ace
if East shows out finesse the nine

Num: (5) AKQJ8-xx Max Tricks: 5

Cash top honors. (But against defenders who would not falsecard from 109x or 109xx, cash the jack and finesse the eight if the nine or ten appears from East)

cash top honors

Num: (11) AKQ10-xx Max Tricks: 4

Cash the queen, and then finesse the ten

cash the Ace
if both play low or East shows out finesse the 10

Num: (12) AKQ9-xx Max Tricks: 4

Finesse the nine: hope that West has both the jack and ten

finesse the nine
and repeat if necessary

Num: (23) AKxxx-Q10 Max Tricks: 5

Finesse the ten

finesse the 10

Num: (24) AK9xx-Qx Max Tricks: 5

Play off the top honors

cash top honors

Num: (25) AKQ10-xxx Max Tricks: 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 10

Num: (26) AKQ9-xxx Max Tricks: 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

Num: (28) AK9x-Qxx Max Tricks: 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

Num: (31) A9xx-KQx Max Tricks: 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

Num: (62) AK98x-Jx Max Tricks: 5

Run the jack *or* lead small to the nine

run the jack and finesse the eight

Figure 5: Comparison of Encyclopedia's (normal font) and FINESSE's (sans serif font) explanations for taking the maximum number of tricks. Problem numbers are those in the Encyclopedia, and 'x's denote arbitrary low cards.

References

ACBL. 1994. *The Official Encyclopedia of Bridge*. 2990 Airways Boulevard, Memphis, TN 38116-3875: American Contract Bridge League, Inc., fifth edition.

Allis, L.; van den Herik, H.; and Herschberg, I. 1991. Which games will survive? In Levy, D., and Beal, D., eds., *Heuristic Programming in Artifical Intelligence 2*. Ellis Horwood. 232–243.

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

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

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

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

Frank, I., and Basin, D. 1998. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence* 100(1–2):87–123.

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

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

Frank, I.; Basin, D.; and Bundy, A. 1992. An adaptation of proof-planning to declarer play in bridge. In *Proceedings of ECAI-92*, 72–76.

Ginsberg, M. 1999. GIB: Steps toward an expert-level bridge-playing program. In *Proc. IJCAI-99*, 584–589.

Goren, C. H. 1986. *Goren's New Bridge Complete*. Century Hutchinson Limited.

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

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

Smith, S., and Nau, D. 1996. A planning approach to declarer play in contract bridge. *Computational Intelligence* 12(1):106–130.