DTAC: A method for planning to claim in Bridge

Paul M. Bethe

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

Deparment of Computer Science

NEW YORK UNIVERSITY

May, 2010

Professor Ernest Davis

Professor Allan Gottlieb

Acknowledgements

This thesis would not have been written without the support of many people. Firstly my adviser, Professor Ernest Davis, who was patient with all the talk of bridge and a valuable resource to all things computer science. Many thanks to my parents and family, who always inspired me with their thirst for knowledge and provided valuable bridge conversations. Most importantly, thanks to my fiancée, Lindsay Nathan, for always being there for me.

Paul Bethe, New York City, April 2010

ABSTRACT

DTAC: A method for planning to claim in Bridge

Paul M. Bethe

The DTAC program uses depth-first search to find an unconditional claim in bridge – that is, a line of play that is guaranteed to succeed whatever the distribution of the outstanding cards among the defenders. It can also find claims that are guaranteed to succeed under specified assumptions about the distribution of the defenders' cards. Lastly, DTAC can find a claim which requires losing a trick at some point. Using transposition tables to detect repeated positions, DTAC can carry out a complete DFS to find an unconditional ordered claim in less than 0.001 seconds on average, and less than 1 second for claims which lose a trick. The source code for DTAC is available from: http://cs.nyu.edu/~pmb309/DTAC.html

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In planning a line of play in bridge, a declarer may have to consider adversarial plans: if the defenders do this, then I can do that. Often, however, a declarer can plan an unconditional line of play that is guaranteed to work regardless of the defenders' responses. In such a case, declarer will often save time in the play by *claiming*; that is, laying down his/her hand, and showing this sequence of plays.

In this paper, I present an algorithm for declarer to find a claim. This is the first published bridge algorithm to compute a line of play that is guaranteed to succeed. Previous bridge programs used Monte Carlo testing with random distributions of the unknown cards; thus the plans they returned are only probabilistically valid. Our algorithm can also be applied to find a claim that is valid under an assumption about the splits in given suits; e.g. assume that the 5 spades held by defenders split either 3-2 or 4-1 and that the 6 diamonds split either 3-3 or 4-2. The algorithm has been implemented in a program named DTAC (an acronym for the common phrase, "draw trumps and claim").

In bridge, a claim occurs during the play when a player faces their cards and in most cases provides an ordering of their cards that will take a certain number of tricks, despite any unstated distributions of the opponents' unknown cards. Although a defender may sometimes claim, it is based on having all winners in their own hand, so it is not as frequent as a claim by declarer and not part of this research. The ordering when stated by declarer will include cards from both his and dummy's hands. In bridge, claims are encouraged as they speed up the play. So, it is a natural advancement for bridge "bots" to be able to make and accept claims.

The ability to find claims should be useful for computer-bridge and the general class of declarer play as it will: i) allow for a faster and more enjoyable game against computers through early claiming; ii) improve the strength and speed of double dummy imperfect-information search, as well as bidding; and finally iii) by using claim(K), $K < N$, provide a baseline for the number of certain tricks when trying for more.

From the perspective of automated planning, claiming in bridge is an interesting application of *domain specific conformant planning:* i.e. finding an unconditional plan that is guaranteed to solve a problem despite incomplete information [Hoffmann and Brafman, 2006; Palacios and Geffner, 2009; Rintanen, 2004]. Abstractly speaking, the technique being used here is to map the actual partial-knowledge planning problem P into a complete knowledge problem P' (the combined defender, described in Section 3.2.1), which is a conservative, or pessimistic approximation in the sense that any solution to P' is also a solution to P. We suspect that the success of this technique in bridge is related to the rather simple causal structure of bridge play, in that successful play can be approximated to zeroth-order by planning playing the high cards. Also, there is a limited number of interactions and interferences, thus very good heuristics limiting search can be developed. As we shall discuss in Section 3.5.1, a substantial percentage of bridge hands end in a state where the declarer can claim the remaining tricks. Therefore, it is worthwhile searching for such a claim.

Chapter 2

Background

In order to fully appreciate the problem of claiming, the reader must have a background in the card-play stage of Bridge, general search, game tree search, and finally the current state of applying those Computer Science techniques to Bridge.

2.1 Blind Search, Adversarial Game Tree Search, minimax and A-B pruning

If a human were given a standard chess board and asked to place eight Queens on the board, such that no queen can take any other, how might it be done? Most likely, one Queen would be placed on the board, then each additional Queen would be placed so as not to attack any others. Eventually, either a solution would be found, or the human would find that the 7th or 8th Queen has no valid location on which to be placed.

Q	/	/	/	/	/	/	/
/	/	/	/	Q	/	/	/
/	Q	/	/	/	/	/	/
/	/	/	/	**2**	Q	/	/
/	/	Q	/	/	/	/	/
/	/	/	/	/	/	**1**	/
/	/	/	Q	/	/	/	/
/	/	/	/	/	/	/	/

Figure 2.1: Chess-board trying to place a 7th Queen

In Figure 2.1, the only location to safely place the seventh Queen is location **1** but that fails as there is no valid location in the 8th column. At this point in the search, the previous move, in the 6th column, would be discarded and one would look for the next valid location. However there is none, so the search backtracks to the 5th Queen, which can be placed at location **2**, and now the 6th Queen is placed, and so on. This method of backtracking is guaranteed to cover all possible ways to place queens on the board, and so if a solution exists, one will be found. Only one solution matters, so once a single valid placement is found, the human is done and need look no further. This is an example of *blind search* or *state-space search*[1], an operation which is easy to implement effectively on a computer, as it only requires a method of enumerating the states and a verification function to test if the result is still valid. However, the number of possible moves which may be searched through is exponential in size, such that the search can take a long time. A set of good heuristics can give a better guess as to which move to try and can provide significant speed up, as once a single solution is found, the search is over. Of course, there are better methods to solve the N-Queens problem, (see for example [Norvig and Russell, 2009] pp 151).

2.1.1 Minimax in Games

When playing a game with opponents, blind search runs into a problem. Let us examine the easiest example, using Tic-Tac-Toe as in Figure 2.2:

?		O
?	X	
X	O	

Figure 2.2: Tic-Tac-Toe with a winning position

At X's turn, a human can clearly see that playing in the top or middle left box will deliver victory, because O is unable to block in two places. But how can a computer discover these two plays are the only winning ones? At each turn when considering X's move, the computer must try each counter play for O, assuming that the opponent will play optimally. Consider if X tried to play in the bottom right corner, if O plays anywhere but top left, X wins. A blind DFS would have its solution, but not an *adversarial search.* The search must backtrack to the last play for O and try every possible play. In this case, top left would be found as blocking the win, so the search must continue.

This type of search can be represented by a minimax tree, where X's moves are *or nodes*, as only a single valid solution is needed at those, but O's moves are *and nodes*, as X's previous winning move must work for all of O's possible moves. X is considered the *maximizer* as he is trying to win, while O is the *minimizer*, as from X's perspective he is trying to force X to lose.

Looking at Figure 2.3, a lot depends on the move order. We know which two moves are winning, but the computer X does not. It may pick any of the others first and waste time

[1]Although I have moved slightly to an informed search by recognizing that each Queen must have its own column

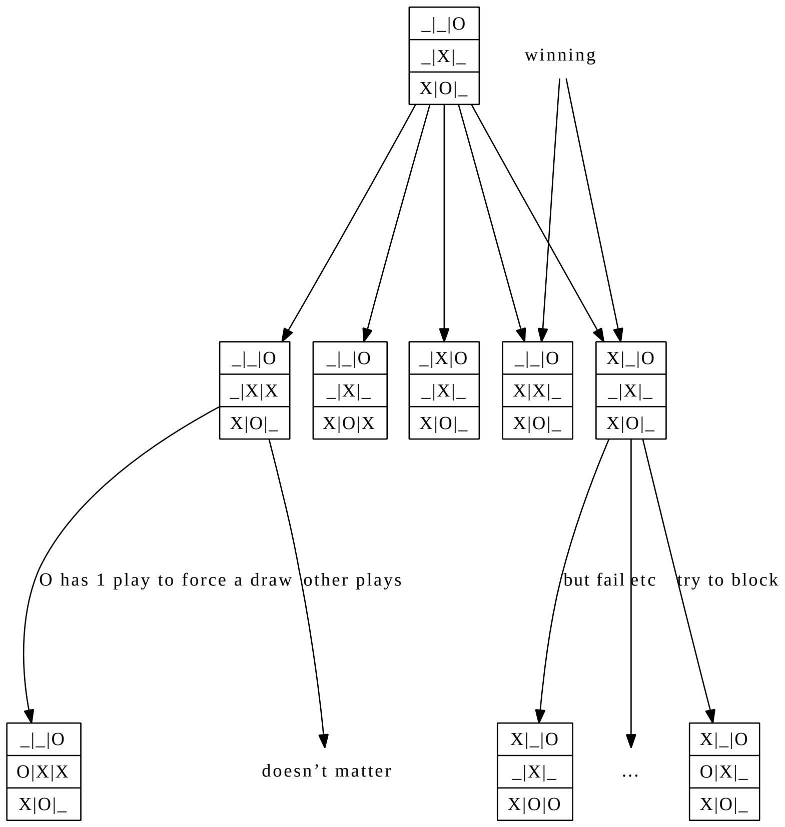

Figure 2.3: Tic-Tac-Toe minimax

discovering that they are unsuccessful. When it does pick one of the two winning moves, it must exhaust all of O's possible moves before it can declare a win.

This search space is much larger than a simple blind search. In general, verifying a successful plan requires presenting a tree with branches for every possible move for O; the problem is thus not in NP (the class of problems whose solution verification takes polynomial time). The unfortunate consequence is that all of the improvement on the N-Queen search discussed in [Norvig and Russell, 2009] Chapter 5 are not applicable to adversarial search.

2.1.2 Alpha-Beta Pruning

Alpha-Beta pruning is an enhancement to *minimax* search which prunes nodes early when it discovers that there is no reason to continue searching. Alpha corresponds to a global known value that max can already achieve, where Beta is the value that min can confirm. If at any point at a max-node (*or-node*) the *maximizer* can play to a better outcome than Beta, the search is abandoned, as min will make a better choice earlier in the search tree. The same is true when evaluating *and-nodes*, if the *minimizer* has a play which will lead to a lower option than Alpha, the search is abandoned, as max will make a better choice higher in the tree.

Figure 2.4 shows a a portion of a simplified game tree where the value of a combination of moves is assigned a score (rather than win/loss/tie as in Tic-Tac-Toe). The max-player is trying to maximize the outcome, and the min-player to minimize it.

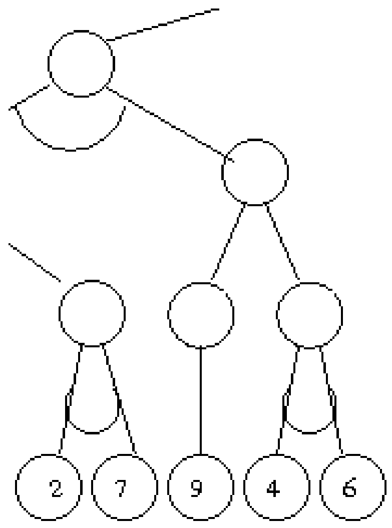

Figure 2.4: Subsection of a game-tree to demonstrate Alpha-Beta game pruning

The left two nodes go to a parent or-node (denoted by the arc connecting the branches), and a parent and-node (not shown), whose parent is the top or-node we can see. Let us assume that at the unseen *and-node*, the best available play yielded 1. At the given *or-node* both branches are visited, and the minimizer determines that 2 is the best outcome from here. The unseen parent *and-node* determines that it can do better than the previous 1, by taking the rightmost branch to the sub-tree we just analyzed. At the root *or-node*, the minimizer determines that by playing left, it can guarantee a score of 2, and so it sets Beta to be 2. Now, the search continues to the right of our root, then goes left at the *and-node* first (by convention). However, when the search reaches the leaf with a result of 9, this entire sub-tree can be pruned. The minimizer set a Beta of 2, so they will never play right from the root *or-node*, knowing that the maximizer could play left and score 9.

2.1.3 Conformant Planning and Kriegspiel

Conformant planning is the procedure of generating a plan under uncertainty which will succeed regardless of the true state of the world [Hoffmann and Brafman, 2006]. The initial problem is transformed from one with uncertainty to a state space search, where the current state is actually a set of *belief-states* about the possible worlds. This is somewhat similar to Ginsberg's Lattice approach, which will be covered later in section 2.3.3.

Russell and Wolfe explored a new method of solving imperfect information games via their *belief-state AND-OR search* in [Russell and Wolfe, 2005]. It is similar to DTAC only in that it explores a method of securing a guaranteed plan to an adversarial game. Unlike DTAC, it uses belief-state space to do an exact search up to resource limitations. This technique showed solid success on a limited database of checkmate positions for 3-ply and 5-ply searches.

2.2 Bridge

Bridge is a partnership card game played by four players, using a standard deck of 52 cards. Compass directions North, East, South, and West are used to identify the players, with the order of the bidding and the play in that order, clockwise around the table, and the partners seated across from each other: either N and S, or E and W. All of the cards are dealt out, and then the bidding proceeds clockwise from the dealer. The process of the bidding is irrelevant here (though in actual Bridge it can contain information useful for play). The result of the bidding is a *contract*, which specifies a particular partnership as holders, the number of tricks they are to win, and either a trump suit, or Notrump. The bidding process also determines which of the players in the team winning the contract is *declarer* and which is *dummy*; the opposing team is called the *defenders*. By convention, the number of tricks to be won is stated in terms of the number of tricks beyond a six-trick "book"; for example, a contract of 4♠ is a commitment to win 10 tricks with spades as trump (out of the 13 available on each deal).

The player to the left of the declarer leads to the first trick ('the opening lead'). Hereafter, the winner of the previous trick leads any card they wish to begin the next trick. After the opening lead, the partner of the declarer (*the dummy*) lays his cards on the table for all to see, and the declarer is charged with playing cards from both hands to maximize his side's ability to take the most tricks. Thus, the declarer knows the cards in his hand and in dummy's hand (where *hand* is a common description of the cards held by one player), but he does not know how the remaining 26 cards are divided between the defenders. Each of the defenders knows his own hand and after the opening lead dummy's. In a single trick, the players each play one card in sequence. The first player of a trick (the *leader*) may play any card in his hand. Each of the remaining players must play a card in the suit that has been led (by *following suit*) unless they are *void* (out of cards) in that suit. If they are void, they may *discard* any card in their hand and are said to *show-out* of the suit led. The winner is then determined as the highest card played in the suit led. Or, if there is a trump suit, any player void in the suit led may discard a trump, commonly known as *ruffing* or *trumping*, and in that case, the highest trump played determines the trick winner (as multiple players may ruff on the same trick).

Following the last trick, the number won by the declaring side is compared to the contract, and a score is obtained. As previously mentioned, at any point during the play, a player will often speed up the game by claiming all or some of the remaining tricks.

2.2.1 Diagrams and Nomenclature

A typical diagram involves 1, 2 or 4 hands. The single hand diagrams ♠AK ♡AKQ ♢AKQJ ♣AKQJ are most often used to explore bidding problems, as in the play, each player can see the dummy and therefore has knowledge of 2 hands.

A diagram showing two paired compass directions (East and West, or North and South) is used to tackle declarer play problems.

♠AK		♠432
♡AKQ	N	♡432
♢AKQJ	W E	♢5432
♣AKQJ	S	♣432

Figure 2.5: Sample two-hand diagram

All unseen cards are assumed to be held by the opponents, in any possible distribution. We will use these two-handed diagrams often to give examples of our claim engine. Sometimes these problems also include the bidding, as that makes distributional inference available.

There are also problems where a defender is given the dummy as information to decide what to do, these will not be used in this paper, so no examples are given. Finally, there are double-dummy diagrams where all 4 hands are given, either for compactness of a problem, or to examine exact search possibilities (see Figure 2.6).

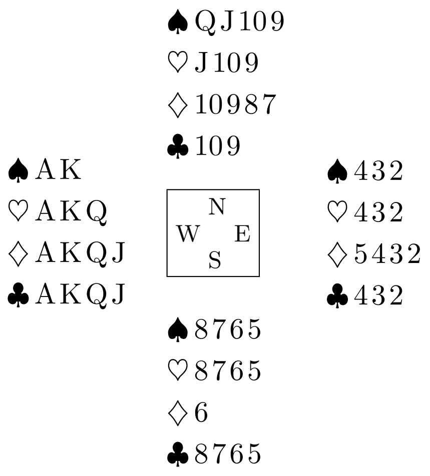

Figure 2.6: Example double-dummy layout

’-’ versus ’=’ It is often useful to speak of an unknown distribution of cards in a suit through shorthand. For example ’3-0 clubs’ describes either opponent holding all three missing clubs (and the other none), but with both options available. However in ’3=0 clubs with North’, by using ’=’ the shorthand implies the specific layout of three clubs with North and none with South (so 3-0 is the combination of 3=0 and 0=3).

A *finesse* is a technical play by any player that attempts to take an extra trick conditional on a certain card being held by a specific opponent. For example, in Figure 2.7, West is on lead at Notrump and leads any spade, with North then *playing small* (shorthand for playing any small card in the same suit).

Of course while the Ace will always win, playing the Queen will also win whenever the King is with North, but lose when South holds it (as South gets to play the King after seeing the Queen played). If the finesse is successful, the declarer will *cash* the Ace (playing a card which is guaranteed to take a trick).

2.2.2 Ending with a Claim

In actual play, almost every hand ends with a *claim*, where the declarer faces his cards and states a line of play to take a certain number of the remaining tricks. The opponents can then accept the claim and enter the score, or reject the claim. In the latter case, they do

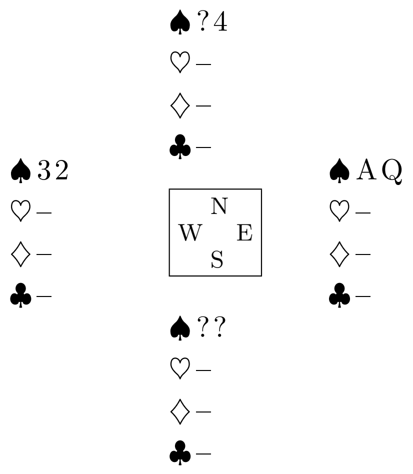

Figure 2.7: Simplest finesse

not continue play, but instead call the *director* (the 'referee' who adjudicates matters of law and irregularities), to figure out what would have happened. In such a case, the declarer is assumed to play reasonably given the actual distribution of the cards, but also to take a wrong turn whenever there was a normal choice of plays. It is important to recognize that while the opponents' unknown cards could be distributed in numerous ways, it is implied in a claim that there does not exist a distribution which can defeat the proposed line of play.

2.3 State of Bridge Play, GIB, and Monte Carlo

Since the first fully automated Bridge program, Bridge Baron, was published [Throop, 1983], there have been large strides in AI Bridge. Bridge has 3 distinct types of problems: bidding, single-dummy play (imperfect-information), and double-dummy play (perfect information). Each of these have different rules, objectives, and inferences that can be drawn, and research has tended to focus on only one aspect at a time. In this paper we focus on the work related to single-dummy play.

Where previous and parallel work had tried to encode Bridge player heuristics and planning techniques to solve the single-dummy problem [Smith *et al.*, 1998; Frank *et al.*, 1998], Ginsberg took advantage of modern computing power and explored a declarer-play algorithm that used a Monte Carlo simulation. Each path was represented by an exact distribution of the cards, consistent with current information, and then solved using a double-dummy solver [Ginsberg, 2002].

Double-dummy is a Bridge term referring to a situation with perfect information, and as such, an exact solution, can be solved for using the previously discussed technique of adversarial search with Alpha-Beta pruning. However a double-dummy solver with only Alpha-Beta was still not fast enough. Considerable work was done by Ginsberg to build on existing adversarial search pruning techniques, and by adding in his 'partition-search' algorithm [Ginsberg, 1999; Ginsberg, 1996], solutions to the double-dummy problem could be computed very efficiently. At each play the move returned is the one with the maximal expected value over the layouts that have been analyzed via the solver. The algorithm has a number of shortcomings. Obviously, the choice of move is based on a statistical estimate. More subtly, there is the problem of *strategy fusion* [Frank and Basin, 1998] which is the following: the algorithm estimates the value of a move in a state of partial knowledge as its expected value averaging over all complete knowledge elaborations of that state. But there are circumstances where that estimate is not valid, because it assumes that all future moves can be chosen correctly, whereas they may in fact depend on information that will not be known.

2.3.1 Strategy-fusion

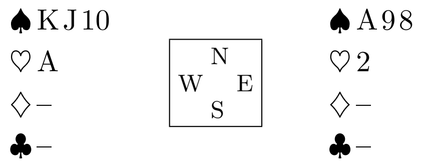

Figure 2.8: Example of strategy-fusion

In Figure 2.8, West is declarer, on lead to take the rest of the tricks. The opponents are known to hold 4 hearts, 1 diamond, and 3 spades including the Queen. Due to earlier play (for example, South discarding on diamonds), North is known to hold the remaining diamond. If spades are 2-1, the Queen will always fall, but if they are 3-0, which way should declarer guard for the *finesse*[2]? Playing the King of spades first will only succeed when North holds all three spades, but playing to the Ace fails in that case. Conversely playing to the Ace succeeds when South has all three spades, but fails when North turns out to have them.

A sound declarer would always make this contract without guess, by making a "discovery play". By taking the Ace of hearts first, declarer learns what he needs to know. If North discards the diamond, all four hearts are known to be in the South hand, and by inference the Queen and all of the spades are in the North hand. Three spade tricks can be taken via the "marked" finesse through North. If, on the other hand, North follows to the first heart, he can have at most 2 spades. This is known by inference, as North is also known to hold at least one diamond, and started with only four cards. Declarer no longer needs to guard against North holding all 3 spades, and so he plays the Ten of spades to dummy's Ace, able to take the "marked" finesse coming back if North has revealed the position by discarding.

[2]defined in Section 2.2.1

From the perspective of a Monte Carlo approach, leading the Ten of spades always succeeds. In 3=0 splits where North has the Queen the finesse is taken successfully, as the double-dummy solver “knows” it will work. In other cases, the solver plays the Ace and also takes all the tricks. When taking an expectation over possible leads, the King of spades fails when the Queen is with South, but the other two leads of the Ace of hearts and the Ten of spades work equally (taking the Ten and Jack as equals). So one of the two “winning” options is chosen at random, and 50% of the time, Monte Carlo has chosen the inferior line, which may fail in 11% of the actual distributions[3].

2.3.2 HTN

One attempt at trying a different approach involved building an algorithm using Hierarchical-Task-Network planning [Smith *et al.*, 1998]. HTN planning, as in the name, uses a hierarchy of “recipes” to solve a problem by applying high-level concepts, each of which contain smaller tasks, which may also contain even smaller ones until *atoms* are reached [Ghallab *et al.*, 2004]. However, it was slow, could only handle endings with 4 or fewer cards per player, and it did not provide better solutions often enough to be more useful than Ginsberg [Ginsberg, 2002].

2.3.3 Lattice Methods, and Squeaky Wheel Optimization

In an attempt to overcome *strategy-fusion*, Ginsberg employed a lattice method for solving *binary decision diagrams* [Nielsen, 2001]. This technique worked and was found to be useful for small end-positions, but as the number of cards remaining increased, the time increased as well (to over 1 minute for 8-card positions). However, Ginsberg modified this technique using *achievable sets* and a modification to the selection order called *squeaky wheel optimization* [Joslin and Clements, 1999], which created an approximation of the problem that was solvable in reasonable time. Profiling this technique against GIB’s Monte Carlo engine showed a small but significant improvement in the play. This is the method that is currently used by GIB, but it is still an approximation that does not allow for a claim.

2.3.4 Recent Research

Since Ginsberg’s paper in ’02 , computers have been considered experts at the play of the hand, and the small amount of published research on Bridge has looked at improving bidding [Ginsberg, 2002; Amit and Markovitch, 2006].

It was assumed that no more research was needed on the play, but this paper will show otherwise.

[3]taking the probability of any 3-0 to be 22%

A more in depth review of Bridge research can be found by reading my review paper from 2009 [Bethe, 2009].

2.3.5 Current Programs

GiB is available as an opponent on BridgeBase.com, but no longer competes in the WCBC (World Computer Bridge Championships). Over the last five years, two other programs have reigned as champions of the WCBC : Jackbridge and Wbridge5. Jackbridge, written by Kuijf and Heemskerk from the Netherlands, won in 2006 and 2009. Wbridge5, written by Yves Costel from France, won in 2005 and 2007-8. Both programs are rumoured to be written using a customization of the Monte Carlo and double-dummy techniques first proposed by Ginsberg [Norvig and Russell, 2009]. However exact details of these two program are proprietary.

In practice, none of these programs nor published works display the ability for a computer player to be able to claim with certainty.

Chapter 3

Planning for a Claim

3.1 Introduction

3.1.1 What is a Claim?

A *claim* in bridge is a statement by any player except the dummy, that indicates a number of tricks remaining that will be taken by the claiming side, from zero to all of the remaining tricks.

The complexity of a claim can vary from very easy to very complicated. There are very easy claims, for example if one of the declaring side's hands has all high winners. Slightly more complicated examples involve using the proper sequence of plays to unblock certain cards to end up in the right hand at the right moment. The most complicated of claims offered by declarer usually involves a trick to be lost at some point in the play, after which no matter what the opponents try, declarer will succeed.

The key part of a claim, which makes it different from an individual bid or play, is that it is a guaranteed line of play (possibly with conditional plan when losing a trick), which has no possible losing outcome no matter what the actual distribution of the opponents cards is.

3.1.2 Why are we interested in Claims?

Claims allow the game to be played faster, as the players do not need to waste time playing out a deal with a known conclusion. Nearly all bridge hands end in a claim or concession, and that is almost always made by declarer. In the data we have used from championship play, only 95 out of 55,343 deals were played out through the twelfth trick, where the rest were recorded as a claim. In fact, the average number of tricks claimed in our data was six. So it follows that having algorithms to verify a claim presented by an opponent, and more importantly to find a valid claim are important components of an expert computer player.

3.1.3 Why is a new algorithm needed?

Let us see what happens if Monte Carlo methods are used to solve for a claim of all the tricks on this hand, with West to lead at Notrump (Figure 3.1).

```
♠A                 ♠–
♡KQJ10    N        ♡A
        W   E
♢–        S        ♢–
♣–                 ♣AKQ2
```

Figure 3.1: An easy claim

For this example, assume that the opponents hold four clubs, and the Monte Carlo solver is attempting to find a winning line. The raw probability of the opponents' clubs dividing no worse than 3-1 (neither opponent having none) is 91%. When N random deals of the opponents cards are made, if the distribution follows the probability, on those deals with 2-2 or 3-1 clubs, one winning line which could be found by the double-dummy searcher is: any heart to the Ace, then the Ace, King, Queen and deuce of clubs. The deuce of clubs will win the last trick as the opponents are out.

A possible perfect layout during Monte Carlo is in Figure 3.2

```
          ♠10
          ♡8
          ♢109
          ♣10
♠A                 ♠–
♡KQJ10    N        ♡A
        W   E
♢–        S        ♢–
♣–                 ♣AKQ2
          ♠9
          ♡9
          ♣J98
```

Figure 3.2: A possible layout dealt by the Monte Carlo

So on this layout, the solver records that for all possible leads by declarer (which is just two: hearts or spades), all the tricks are taken. However, in the other 9% of cases, clubs break 4-0 and the only winning line starts with spades. As of the last published work, GIB was using 100 paths as a reasonable amount to solve[1] [Ginsberg, 2002]. The number of distributions of 10 cards into 2 hands is $\binom{10}{5} = 252$, so not all deals can be evaluated. The probability that the sampling of deals includes at least one where clubs are

[1]with advances in computing that is likely to be more today.

4-0 is $100 * (1 - [(0.91)^{100}]) = 99.992\%$, so there is a good chance that the right line will be found. However, in that very small chance that it can go wrong MC finds both lines to work equally well and will then choose randomly.

Even in the case where MC finds the right line, from the perspective of the solver, there is no way to assert a claim. The solutions may have exhibited *strategy-fusion* (see Section 2.3.1), or simply not have included some outlying case in the random draws.

3.2 Moving beyond Monte Carlo

To claim in bridge, you need 100% confidence that a proposed solution is valid, and it is clear that Monte Carlo cannot provide that.

3.2.1 Conformant Planning with the Combined Defender

In order to be able to search for a plan in reasonable time, we needed to be able to map the problem of finding a claim from a complete adversarial search to a blind search. In order to do so we conceived of a construct which could replace the pair of defenders by an imaginary *combined defender*. Conceptually, the combined defender plays two cards in each trick; he can always play the highest card held by either defender; and he can trump a trick if it is possible (relative to the declarer's knowledge) that either defender is void in the suit that has been led. The combined defender is thus an over-estimate of the power of the two separate defenders. If a pair of cards can actually be played by the actual separate defenders under any division of the outstanding cards between them, then it can be played by the combined defender; if the actual defenders can win a trick, then the combined defender can; if a line of play succeeds against the combined defender, it will succeed against any distribution in the actual defenders.

Moreover, since we are evaluating claims (unconditional play by declarer), we can further simplify the problem by conceptually requiring the declarer to state his entire line of play at the outset. The combined defender can choose the entire response on the basis of the declarer's entire plan.

Use of the combined defender simplifies the problem of finding a claim in two ways. First, it renders the uncertainty of the division of cards between the two defenders irrelevant. Second, it is easy to compute whether a claim of all tricks is valid. Let TC be the time at which the claim is announced (or considered) and TT be the time at some later trick that is part of the claim. Assume that declarer is planning to lead suit SL at TT and to win this trick with card C in suit SW. Assume that he has already played KL tricks in SL, and KT tricks in trump between TC and TT. Let ML be the minimum number of cards that either defender may hold in suit SL at TC. Then the combined defender can win the trick if any of the following three hold.

A. SL=SW at TC: combined defender had at least KL+1 cards in SL and his top card in SL was better than C.

B. SL=SW; ML $\leq$ KL: so one of the defenders may now be void in SL at TT; and combined defender had more than KT trump at TC. That is, defender can win a trick by trumping when declarer was planning to win in another suit.

C. SW=trump $\neq$ SL; ML $\leq$ KL: combined defender held more than KT trump at TC; and defender's top card in trump is greater than C. That is defender can over-trump declarer's intended ruff of this trick.

Since it is easy to compute whether the combined defender can defeat a claim, it is not necessary to do any search among his possible plays. The only search necessary is in space of plays for declarer. Thus, we have converted a problem of partial knowledge, adversarial search into a problem of complete knowledge, state space search.

3.3 Implementation

The requirements for implementing state space search are:

- A method for enumerating over the moves available to the player on lead, and the valid plays by their partner.
- A recursive function *findClaim* which searches through the pairs of moves available to declarer at the current trick returning False if all are defeatable by the combined defender. For all accepted moves, a recursive call to *findClaim* is made down to the final trick.
- An *accept* function which returns True if the combined defender cannot defeat the trick.

The *accept* function is simple: assume that the opponents can and will do the worst available to them. So if the defenders combined holding in a suit includes four cards, including the Jack, do not accept any play in the first four rounds of the suit that is not the Ace, King, or Queen. Inputs to *accept* are: the defensive constraints, the cards involved in this trick, and the rounds played for each suit in this search.

Definition 1. *Let the combined defensive assets be represented as a quad of triples (H, L, M), one for each suit. Where:*
H is the highest card the opponents have in that suit.
L is the max length that either opponent could have in a suit.
M is the min length that either opponent has in a suit.
Which can be represented as:
CD= [♠=(H,L,M); ♡=(H,L,M); ♢=(H,L,M); ♣=(H,L,M)]

This data structure represents the relevant aspects of declarer's knowledge of the state of the combined defender. It can also be used to encode further hypothetical constraints on the division of cards in a suit between the two defenders.

The function

accept(CD,Trumps,RoundsPlayed,Lead,Follow)

takes as arguments the state of the combined defender, the trump suit, the number of rounds played in each suit, the card led in the proposed trick, and the card played by their partner. (That is, if declarer is leading then Follow is the card played by dummy and vice versa.) It returns False if defenders can win the trick and True if declarer will necessarily win the trick.

We can now define the function

findClaim(CD,Trumps,K,State)

which returns a claim, if one can be found. The arguments CD, Trumps, and K are fixed for the search, representing the initial constraints ascribed to the combined defender, the trump suit, and the number of tricks K that declarer is trying to take. State is a data structure recording the cards held by declarer and dummy, the state of the CD, and the identification of which player holds the lead (which can by either of *declarer* or *dummy*). *FindClaim* does a simple DFS through the space of states. An operator corresponds to a choice of cards for the contract team. The accept function is used to prune search; if accept is false, then the current branch of the DFS fails. The base-case used to indicate success in the DFS is when the total rounds played is equal to K.

In practice, the depth-first search can be made much more efficient by rules that determine that two cards are equivalent, for example, if the declarer is about to try playing both the 9 and the 7 of the same suit, but knows that the 8 has already been played, only one of these must be tried as they have become logical equivalents. Further pruning can be done by adding rules that determine that some plays are pointless, as well as heuristics that guide the most promising lines of play to search first: like drawing trumps or leading honors from the short hand first, and, as described further in Section 3.3.3, hashing states to avoid repeated search.

3.3.1 Revisiting our Example

Consider the same Notrump hand from Figure 3.1 with West to lead:

Two claim lines exist, one of which is to cash the ♠A discarding the ♡A, making the west hand high, and the other which is to discard the club deuce on that first round, making the dummy high.

However our algorithm, unlike a Monte Carlo approach (when the specific distribution allows), will reject a line that tries to win with the deuce of clubs, as an attempt to win

♠A		♠–
♡KQJ10	N	♡A
♢–	W E	♢–
♣–	S	♣AKQ2

Figure 3.3: Easy claim again

the fourth club trick after taking the Ace, King, and Queen will fail *accept* as either of the opponents might have started with four clubs. DTAC finds the line of pitching the small club under the the Ace of spades, and the CD is unable to defeat any of the proposed tricks (Figure 3.4).

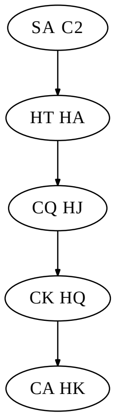

Figure 3.4: DTAC's Unconditional Claim

Consider the hand in Figure 3.5 at spades, where the opponents hold the spade Ten, two hearts, and some cards in the other two suits.

♠AKJ		♠Q9
♡2	N	♡–
♢–	W E	♢2
♣–	S	♣2

Figure 3.5: Simple ruffing claim

This easy claim for a human, is simply to ruff the heart high, with the Queen, then win with high trumps. Again, a Monte Carlo approach should find the winning line, but without the extra "proof" that all the tricks will be taken. The DTAC program find the easy claim, and the solution file is displayed in Figure 3.6.

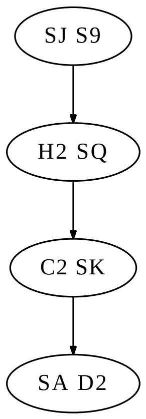

Figure 3.6: Unconditional Claim w/ a trump suit

3.3.2 Constrained Search

In order to find a claim, we assumed the worst of the opponents' card distribution, then looked for a solution. However, the presented algorithm does not have to be run for just the worst-case distribution.

Constraints on splits such as "spades are not split 5-0" can be easily incorporated by limiting the plays open to the combined defender and adjusting the *findClaim* algorithm to use different values of L and M. This could be used to mimic the way a human often analyzes a hand, by restricting the opponents to have at least one or two cards in every suit, and look for a winning line of play conditioned on those constraints.

♠A86532		♠KQJ
♡A6	N	♡K10732
♢KQ	W E	♢A87
♣853	S	♣AJ

Figure 3.7: Hand with an optimal line via constraints

In Figure 3.7, against West's contract of 7♠, North leads the King of clubs, and declarer wins dummy's Ace. An unconditional claim is impossible (DTAC confirms no solution), as with trumps possibly 4-0, after playing the King, Queen, Jack, declarer must cross to a winner via a different suit. However, this trick is not accepted, as the CD construct could trump in.

However, the highest percentage line[2] involves assuming that: trumps do not break 4-0, hearts break 3-3 or 4-2, and diamonds break 4-4 or 5-3. If a new search is made, with these assumptions in the combined defender, in about 0.001s (after optimizations discussed in

[2]expert consensus

3.3.3), the correct line is returned by DTAC (See figure 3.8). This is the highest percentage line available, and the search for it does not require a Monte Carlo approach.

This winning line (under these assumptions) is to the play the King and Queen of diamonds and then Ace of hearts (playing low each time from the dummy is implied when nothing else is written). Cross to dummy via a low spade to the King, play the Ace of diamonds to discard the low heart, and then lead a low heart, ruffing with a low spade[3]. Next, a low spade to dummy's Queen and another low heart, this time ruffed with the Ace (protecting against North having started with only two hearts). A third round of spades by playing low to the Jack draws the outstanding trumps (under the 3-1 assumption), and now King and Ten of hearts provide discards for the losing 8 and 5 of clubs (the Ten of hearts is the fifth round, and wins if neither opponent started with more than four). This sequence is quite complicated, and would not be found by the average bridge player, as the play of discarding a heart on the Ace of diamonds is very advanced play.

3.3.3 Optimization via Transposition Table

There were several hands in our data set which could take up to 30 or more seconds to search for a simple claim. Only small improvements were achieved by adding better heuristics for which move to pick, and eliminating touching cards: e.g. with the choice of the 9 or 7 of a suit, but with the 8 already played, consider only one move. More was needed to get the search to a usable speed, and the simplicity of information required to uniquely describe a search position lead to the definition and use of a transposition table.

Definition 2. *The cards left to be played, the rounds played in each suit, and the player next to lead, uniquely define a searchable position.*

Given that, a position can be defined as a 69-bit string: 52 bits to hold which cards are left to be played, 4-bits for each suit, to allow rounds played up to 12, and a single bit to indicate which player is next to lead. A transposition table now becomes a hashset on this key.

We modified the search algorithm to check for an already visited position at the beginning of *findClaim*, and then added a position's key if a move was accepted as a final step before making the recursive call. Once the search algorithm was so modified, the resulting speed-up was impressive and will be demonstrated later in the results section.

3.4 Losing a Trick

Consider this simple hand at Notrump with just declarer's cards (as dummy's are irrelevant):

[3]here *cross* is used to indicate that the intent is win the trick in the opposite hand from the last, to alter the next to lead

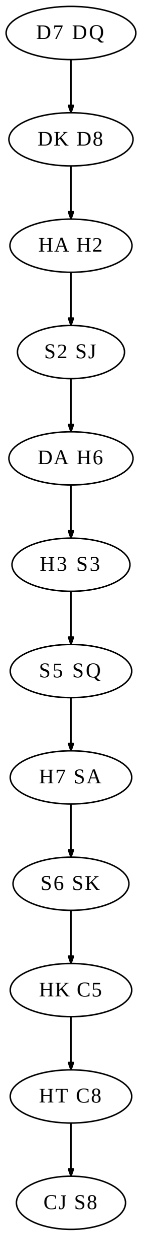

Figure 3.8: Claim when trumps not 4-0, hearts not 5-1, and diamonds not 6-2

♠ A
♡ A
♢ A
♣ K Q J

Figure 3.9: Simplest 1-loser hand

It is easy to see that five tricks are automatic. The declarer plays clubs, trying to lose to the Ace. If the Ace is taken, there is a claim, as this player is prepared and able to win any suit returned by the opponents.

However, until now, our algorithm did not handle the notion of losing the lead. To find such a solution we make use of a single adversarial and-node, which requires that winning lines are available for all suits that the opponents have to lead.

When the King of clubs is lead, if the opponents hold 2 or more clubs, the player holding the Ace may choose to duck by playing low, or they may win their Ace. If they duck, the regular search continues with declarer to determine the next play. However if they win, declarer must be able to find a claim in each suit that the defenders may lead. Notice that in the case when they duck, at the next round, declarer constructs another and-node which requires valid claims for the opponents ducking, as well as all suits.

After DTAC was modified to incorporate this improvement, a solution was found for this example (see Figure 3.10).

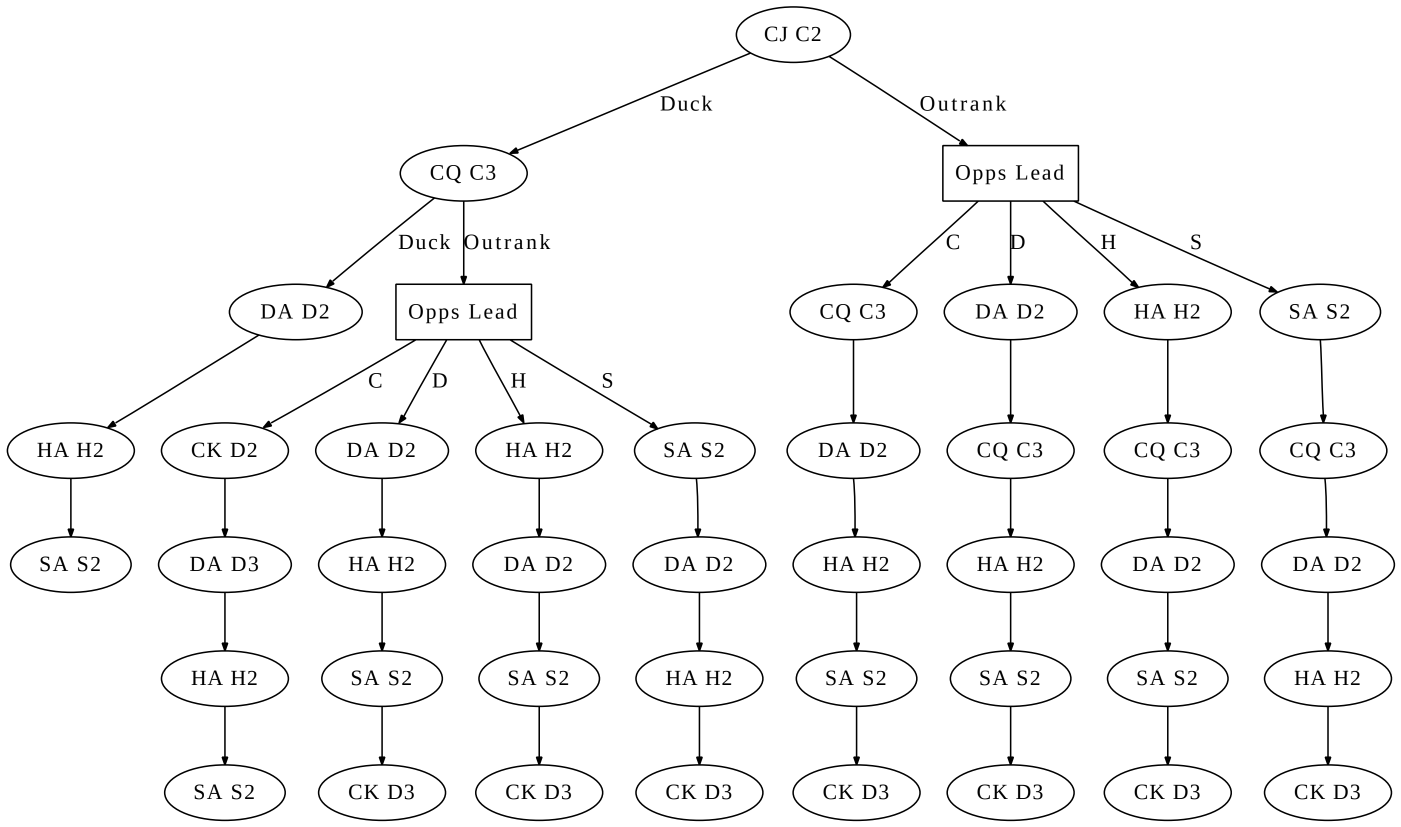

Figure 3.10: DTAC's classic plan

In our method, once a trick was lost, the search proceeds as before and no more losers are allowed. While solutions are no longer verifiable in linear time, there is at most one and-node per depth of the tree, which is limited by the number of cards. Thus verification is quadratic, and an improvement on classic adversarial search.

In fact, we can use this technique to solve much more complicated hands where the declarer would not claim at the table, but DTAC is able to provide a best line for Figure 3.11.

♠–		♠7
♡–	N	♡–
♢AK	W E	♢–
♣AJ8	S	♣K743

Figure 3.11: Small example of a 1-loser claim in trumps

With clubs as trumps and declarer having ruffed once already (leaving the opponents with 5 total trumps), he wants to take all the tricks but one (which we assume fulfills his contract). If trumps split 3-2, the problem is easy to solve. (DTAC's solution is illustrated in Figure 3.12)

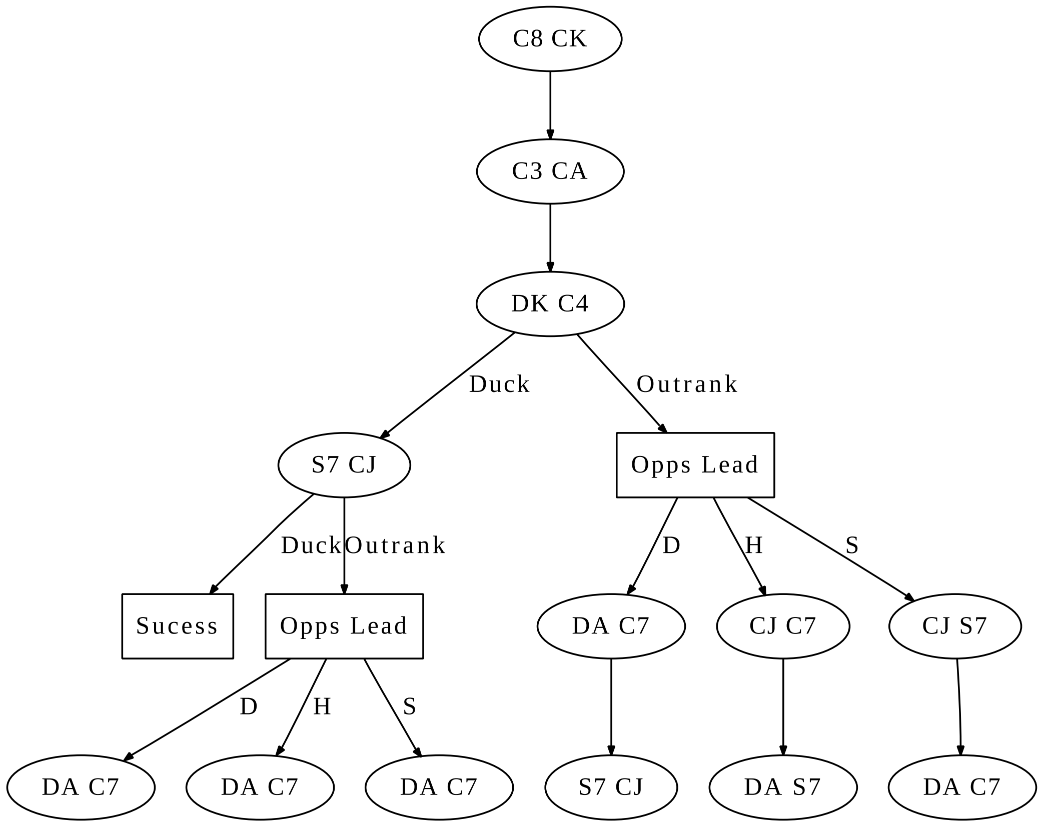

Figure 3.12: Claim when trumps 3-2

With the knowledge that trumps are 3-2, DTAC quickly finds the claim of drawing two trumps with the King and Ace, and then cashing[4] side winners and waiting for the opponent with the *master trump* (the highest outstanding trump, most likely here, the Queen), to take it. At that point DTAC is prepared to win the required number of tricks after any return that is made. However the same search fails with trumps allowed to break 4-1.[5]

In a more complicated example of requiring trumps to split, we look at Figure 3.13:

♠10		♠76543
♡AKQ	N	♡2
♢KJ1043	W E	♢AQ5
♣AJ85	S	♣K743

Figure 3.13: A more complicated hand for trumps 3-2

Playing in a contract of 5♣ the opponents lead and win the Ace of spades, and continue with another high spade. Declarer ruffs in (with the Five of clubs), and needs to take 10 of the remaining 11 tricks. At this point, assuming trumps 3-2, DTAC provides a conditional plan that involves drawing two trumps and then playing winners. DTAC's solution is shown in Figure 3.14, and is just a more complex version of the solution from Figure 3.11.

Perhaps the bridge "bot" will continue to think or look for a higher probability solution, but it also can compute the percentage of the 3-2 split and consider that plan amongst its options.

My last example is a hand from example 28 of one of the great Reese and Trézel books [Reese and Trézel, 1978]. Declarer plays at 6♢ on the lead of a club (a spade would have been tougher). A Monte Carlo solver should find the solution, but it will discover that in 75% of the distributions, North holds one of the spade honors, and when declarer always guesses correctly in that solver (knowing perfect information), it will expect to make the contract on all of those hands by simply drawing trump. So again, depending on the sampling, there is a very small percentage chance that all paths will include a spade honor onside, and MC will be forced to pick randomly between the winning line and the inferior one (which suffers from strategy-fusion).

DTAC can find a guaranteed solution if told only that each opponent has at least 1 club, 1 spade, and 2 hearts (meaning no ruffs are coming when a trick is lost). A low trump is lead to dummy, which allows an opponent to win the Jack (which would not have been a loser with the opponents holding only 3 trumps). Now any return can be won, and the dummy entered with a low diamond to the 8, which is now high. The Ace and King of

[4]playing known winners

[5]Note that deciding when to settle for a solution under certain distributions is not addressed in this paper.

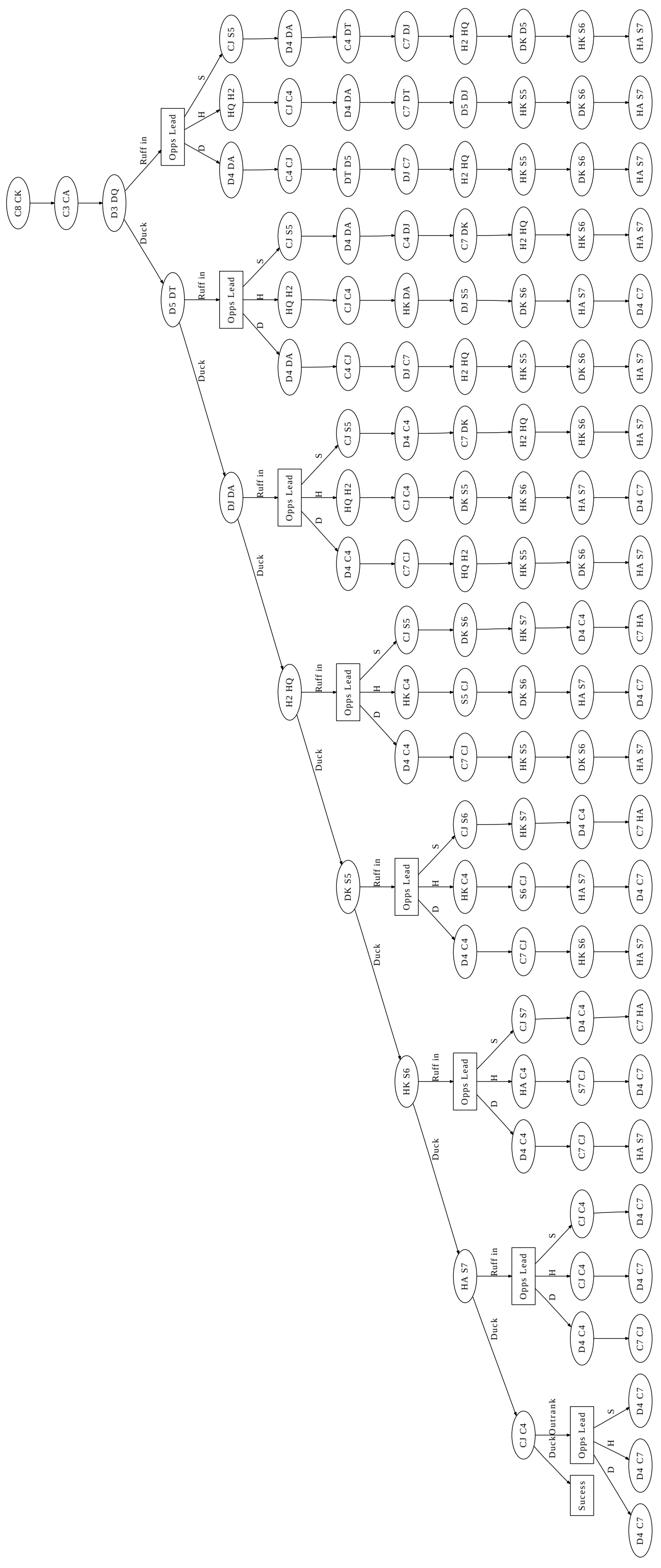

Figure 3.14: Claim when trumps 3-2

♠A65		♠Q1072
♡–	N	♡AK32
♢AKQ109732	W E S	♢86
♣AK		♣432

Figure 3.15: Forcing an entry in 6♢

hearts provide discards for the small spades, and low heart is ruffed with a high trump to return to hand, and trumps are drawn (if they broke 3-0). This line of play rates at about 95%, and is found by DTAC. (DTAC's solution diagram is in figure 3.16)

3.4.1 Addendum to Optimization with a Loser

A position with an allowed loser is not the same as one without. So in order to share the transposition table that showed such improvement in simple claims, one extra bit is required to indicate if a loser was allowed. That way if a position is reached which had no solution when no losers were allowed, it is still a viable position to search when a loser is an option. Note that in this last case, if no solution was found with a loser, then there is clearly also no solution without an allowable loser.

Because of the *and-node* that is now needed for an opponent on lead, it is possible that a solution may be found to a particular position and lead that is not accepted due to failure under other suits lead. Therefore it is important to note that when adding the solution to the hashtable, in case the player reaches this position again but gets to choose the lead (which is different). Note we have moved from a hashset indicating visited positions to a hastable with possible NILL entries for visited positions with no solution.

3.5 Experimental Results

These results were obtained on a Quad-core Intel Xeon machine, with a 64-bit distribution of RedHat Linux (kernel 2.6.18), using gcc-4.1.2 to compile the C++ code for DTAC.

3.5.1 Compiling Data Sets

Two data sets were created for analysis. The first was obtained from the USBF (U.S. Bridge Federation: usbf.org), containing 21,797 deals played in team-trials to represent the U.S. in International competitions, and also from International competition[6]. The second set of

[6]http://usbf.org/docs/vugraphs/*/pbns, http://usbf.org/vugraphs/*/pbns/

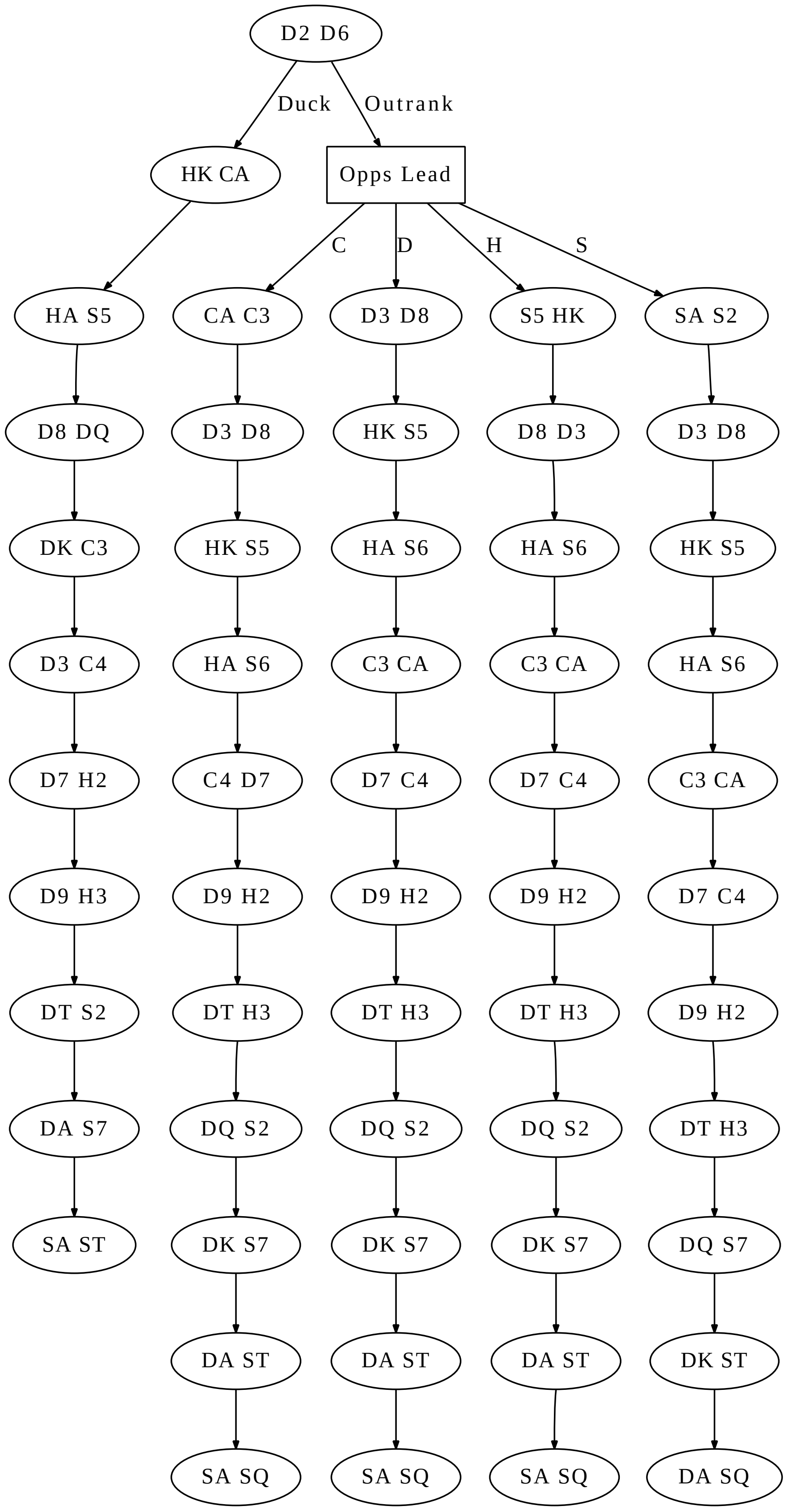

Figure 3.16: Claim via forced entry

33,546 deals was acquired from the PBN (Portable Bridge Notation[7]) archive[8], and contains a somewhat disparate sampling of National and European championship tournaments.

The data includes a record of the hand the final score entered, and several rounds of card play. Thus it is possible to infer the number of tricks claimed from the last round and the score. However, as these files are human generated, there is the possibility of error in the data if some of the last rounds were not recorded, although we believe that to be only a small number.

The first test for our dataset was relevance. How often did claims appear to happen, and would this work be useful? A quick scan of the data revealed that 33% of the deals in the USBF data set ended with an early claim of the remaining tricks, and 99% of the rest ended in a claim (but of fewer than all the remaining tricks). Less than 1% of deals were completely played till the end. The average number of tricks claimed was 6.

3.5.2 Without Losing a Trick

Encouragingly 70% of deals in our dataset with a claim of all the tricks was also found by the DTAC program. This means that our program was relevant in at least 23% of the deals. We also looked at deals which ended prematurely, but with declarer claiming K tricks, $K <$ the number remaining. 67% of the USBF deals appeared to end that way, and DTAC (when given K) found solutions in 49% of those cases, increasing the relevant number of deals to roughly 50% of the dataset. However there is more work to be done by a program in order to decide to claim only K tricks, rather than trying for more.

Data Set	deals	% found	DTAC %
USBF, Claim All	21797	33%	23%
ARCH, Claim All	33546	18%	10%
USBF, Claim K	21797	67%	33%
ARCH, Claim K	33645	81%	31%

Table 3.1: Percentage of hands with claims and then found by DTAC

Initially, while DTAC was satisfactorily solving these datasets, and passing several regression tests for example hands where claims should and should not be found, the time required was discouraging.

Table 3.2 shows timing results before adding transposition tables, which included two particularly long deals requiring 402sec and 6003sec.

The data in Table 3.3 shows the results after transposition tables were added as discussed in Section 3.3.3. The branching factor was reduced significantly, but more importantly, the

[7]http://www.tistis.nl/pbn/

[8]http://www.angelfire.com/games2/pbnarchive/pbn/index.htm

Data set	Average time	Std. dev.	b-factor
USBF data	0.4294s	2.0676s	2.6
PBN arch	0.19s	6.8s	2.6
USBF claim-K	0.59s	51s	2.6
ARCH claim-K	0.36s	39s	2.6

Table 3.2: Unconditional search timing data without transposition tables

searches experienced a hit rate of 70%, and an average 1000 to 1 speed-up. The longest search now takes less than 0.01s. In these tables the “b-factor” or *branching factor* is the average number of moves tried at each node, and is an indicator to the size of the search space.

Data set	Average time	Std. dev.	b-factor
USBF data	0.00032s	0.0033s	2.3
PBN arch	0.00032s	0.0033s	2.2
USBF claim-K	0.00014s	0.0019s	2.3
ARCH claim-K	0.00015s	0.0026s	2.2

Table 3.3: Unconditional search with optimization

With the speed and relevance of this algorithm established, it now becomes reasonable for computer players to look for a simple claim at each round they are to play, as well as use DTAC to enhance play and bidding code to look for guaranteed lines under some constraints.

3.5.3 When Losing a Trick

Any of the data which did not have a solution by winning the first K tricks to match the human result was run through the lose-a-trick module of DTAC.

Data Set	% of original	# of deals	DTAC finds %
USBF, Claim with Loser	38%	8030	26%
ARCH, Claim with Loser	52%	17015	12%

Table 3.4: Percent of remaining hands solvable for 1-loser with transposition tables

So of those remaining deals, DTAC can find a solution with loser 26% of the time for the USBF data, and only 11% of the time for the PBN archive (as indicated in Table 3.4). Again, the number of deals that this covers is large enough that it appears useful, but the time required (even with transposition tables) was not as promising.

Data set	Average time	Std. dev.	Max	b-factor
USBF claim-K	0.75s	3.3s	89s	2.4
ARCH claim-K	0.49s	3.2s	140s	2.4

Table 3.5: Time spent solving claim-K for 1-loser

Unfortunately, several of the searches which bore no fruit also took a long time to exhaust the search space. The addition of the losing trick, even with the transposition tables can be slow. While still useful to bridge programs, the search will have to be done with time-out. It seems possible that better pruning heuristics about which tricks to try after the loser could speed up the search. If at the *and-node*, we search three suits and find a solution, but the 4th is a failure, we could have saved time had we known to try that 4th suit first (and failed).

3.6 Limitations

There are certain types of claims that an expert bridge player would make that DTAC is unable to find. During the play, one of the defenders may *show out* of a suit; that is, discard another suit when they were out of the led suit. It is now possible to make a claim that can be guaranteed valid given this information (for example taking a finesse), but our algorithm is not able to do so. For example declarer and dummy hold:

A54 □ KJ10

if declarer wins a trick with the A of spades in his hand, dummy playing the Ten, and the player after dummy discards (another suit). The Queen is now known to be held by the player in front of dummy. When declarer leads a low spade and the player in front of dummy plays low, declarer will play the Jack from dummy. If instead the opponent produces the Queen, it will be beaten with the King, and the Jack is high. This *finesse* is considered *marked* as the previous round indicated who held the Queen (*marked* describes a situation where the holder of a card is known). It is routine for a claim here to use the phrase "finessing for the the Queen of spades".

Consider the last five cards of play to be in a single suit which has yet to be played by any player, where declarer and dummy hold:

AK982 □ Q1054

(dummy's other card being a known loser). An expert player would claim the remaining five tricks, by stating a conditional plan: play the Ace, and if both defenders follow, the Jack must fall two rounds under the King and Queen. But if one defender *shows out* by discarding another suit, declarer follows with a conditional plan to finesse the defender known to hold the Jack, as described in Section 2.2.1.

As with losing a trick, both of these types of advancement seem possible with the same type of partial adversarial search.

It has even been reported that to speed up the play, top experts will make a claim based on a *squeeze* position that is expected due to knowledge gained from the bidding and play, but we have no expectation of being able to duplicate this. For those interested, an example *squeeze* position is in Figure 3.17.

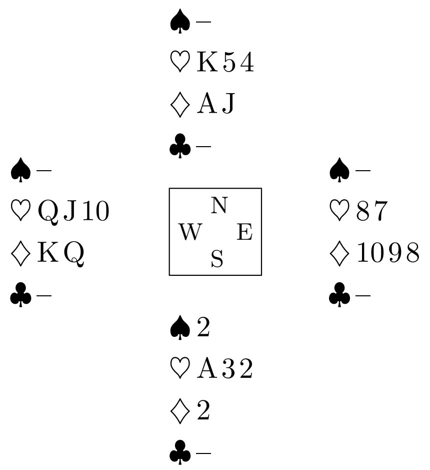

Figure 3.17: Example of a claimable squeeze

In the example of Figure 3.17 with South playing at Notrump and needing all the tricks, West had opened with a bid of 1♡ and was therefore known to have started with most of the missing high cards (the Jack, Queen, King, and Ace in any suit) and at least 5 hearts. By this end position, West required both the King and Queen of diamonds for enough *points* (a common method of evaluating hands in the bidding) to have opened the bidding. Thus South is playing almost double-dummy, in knowing that East has at most 2 hearts, and the location of the diamond *honors* (the Jack, Queen, King and Ace). When the last spade is cashed, West is *squeezed*: a diamond discard allows the Jack to score, while a spade discard allows declarer to score the third round. While fully conditional play is beyond the current scope of DTAC, Monte Carlo methods have proven quite good at solving these end-positions.

Chapter 4

Conclusions and Further Work

4.1 Conclusions

The large percentage of deals for which DTAC finds a solution, as well as the speed in which it is done, has proven that computer bridge programs will soon be claiming. With constrained search and losing trick search, computers will be able to find baseline solutions as alternatives to their results, as well as providing search bounds.

4.2 Further Work

While DTAC adds value to computer bridge, there are still lots of hands for which the existing methods may not find the best solution. *Best* in this case meaning the one which works on the highest probability of hands.

The cover of the famous *Test Your Play* book [Kantar, 1981], includes the problem in Figure 4.1. On a trump lead to South's contract of 7♡, the reader is asked whether it is better to finesse in spades or diamonds, but this is a diversion. The correct play is first to try for clubs 4-3, then fall back on the diamond finesse. Picking one finesse gives a 50/50 chance, but clubs break 4-3 62% of the time, and if they don't then you can still try the diamond finesse, which combines the odds of success to 81%.

As expected, DTAC finds a solution if told that clubs are 4-3. (See Figure 4.2) But notice that from the algorithm's perspective, leading to the diamond Ace at the 5th trick is fine as it expects clubs to break. The optimal line enters dummy the second time by ruffing a spade to ruff the 3rd round of clubs. At this point you discover whether clubs are breaking or not. DTAC's line does not allow the fall-back option of the diamond finesse after this discovery.

♠ 4
♡ K 7 6 3
♢ A Q J
♣ 6 5 4 3 2

N
W E
S

♠ A K J 9
♡ A Q 9 8 5 4
♢ 5 3
♣ A

Figure 4.1: Hand w/ best line assuming 4-3 clubs

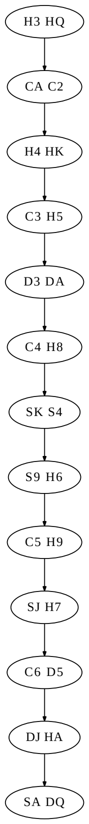

Figure 4.2: Claim for clubs 4-3

♠AKJ1098		♠Q42
♡K	N	♡AJ10xxxxx
♢QJx	W E S	♢x
♣AJx		♣x

Figure 4.3: Optimal line available via discovery play

4.2.1 Discovery Play

Consider the hand in Figure 4.3. In a contract of 6♠ after North's taking the first trick with the Ace of diamonds, they then play a club to their partner's King which is taken by declarer's Ace. Declarer must take all the remaining tricks to make the contract, so any distribution where that is not possible is not worth worrying about.

The play of this hand is tricky. An expert (or double-dummy search) would see that when hearts are 3-1 and spades 2-2, a winning line is to play the King of hearts, play a spade to dummy's Queen, ruff a low heart; draw the last trump, and ruff a club or diamond to the good hearts (when started from the Ace). A double-dummy solver would also see that if reversed (spades 3-1 and hearts 2-2), a winning line is to cash the King of hearts, draw three rounds of spades ending with dummy's Queen, and run hearts from the Ace.

However, while both of these lines of successful play start with the cashing of the King of hearts, at the second trick the play diverges, and the declarer will not have the information needed to know which line to take. This is another example of *strategy fusion* (as discussed in 2.3.1), and a known shortcoming of approximating imperfect game search via an expectation of perfect solutions [Frank and Basin, 1998].

When given either set of constraints, DTAC finds a solution (Figures 4.4 and 4.5), but as described above, they diverge too soon to be usable.

A human expert would see this, and realize that there is another winning line which works in both cases by making a *discovery play* (A bridge term describing a choice of plays which provide the partial knowledge needed to pick correctly from diverging plans). The King of hearts is played, overtaking with the Ace. A low heart is lead and ruffed with any spade, and one then discovers if the hearts are breaking 2-2 or 3-1 by observing the opponents' plays. At this point the lines diverge, but the human has learned the necessary information to decide whether to draw trumps in three rounds ending in the dummy with good hearts, or draw trumps in two rounds to dummy, in order to ruff another round of hearts to then re-enter dummy via a ruff.

So, each distribution where a successful line exists has two options, each of which overlap with a companion, one of which "discovers" the winning line, but the other one does not. Will current programs that utilize double-dummy search find the winning line? The solver will see that overtaking the King of clubs with the Ace or just allowing the King to hold, make on the same probability of distributions (as to the MC solver, it will guess which distribution to play for). So it is purely random whether MC finds the discovery play.

How would DTAC aid in solving this? Given a set of constraints, each solution can be obtained using a modified version of *findClaim* that returns multiple solutions. Rather than searching through every possible layout of the missing cards, DTAC would search through only the 225 different distributions (enumerating 3 for 4-0, 3-1, 2-2 hearts, etc.). Then, and this is where further research must be done, some method of merging plans must be engaged, with the additional requirement that at the time of plan divergence, enough information must be available to select the correct one. Another promising approach is to build upon Ginsberg's Lattice approach, by using the *combined defender* approach to solve deals.

We reported that DTAC found a solution to hands losing a trick under certain constraints. However, the method a computer bridge player would use to know which set of constraints to search under is still an open problem of interest.

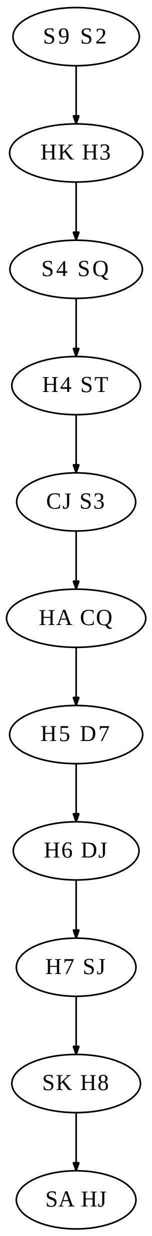

Figure 4.4: Claim for only spades 2-2, hearts 3-1

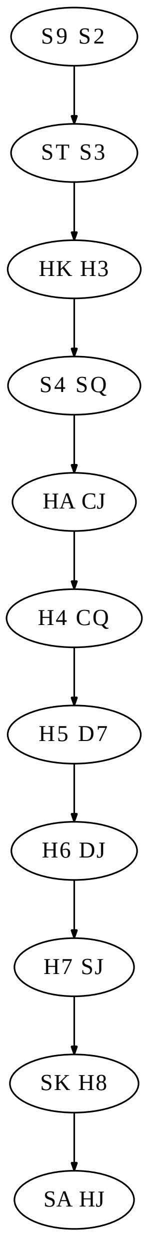

Figure 4.5: Claim for only spades 3-1, hearts 2-2

Bibliography

[Amit and Markovitch, 2006] A. Amit and S. Markovitch. Learning to bid in bridge. *Machine Learning*, 63(3):287–327, 2006.

[Bethe, 2009] P. Bethe. The State of Automated Bridge Play, August 2009. `http://ephman.org/bridgeReview200908.pdf`.

[Frank and Basin, 1998] 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.

[Frank *et al.*, 1998] I. Frank, D. Basin, and H. Matsubara. Finding optimal strategies for imperfect information games. In *Proceedings Of The National Conference On Artificial Intelligence*, pages 500–508. John Wiley & Sons Ltd, 1998.

[Ghallab *et al.*, 2004] M. Ghallab, D.S. Nau, and P. Traverso. *Automated Planning: theory and practice*. Morgan Kaufmann Publishers, 2004.

[Ginsberg, 1996] M.L. Ginsberg. Partition search. In *Proceedings Of The National Conference On Artificial Intelligence*, pages 228–233, 1996.

[Ginsberg, 1999] M.L. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 584–593. Citeseer, 1999.

[Ginsberg, 2002] M.L. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:313–368, 2002.

[Hoffmann and Brafman, 2006] J. Hoffmann and R.I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[Joslin and Clements, 1999] D.E. Joslin and D.P. Clements. Squeaky Wheel. *Optimization. Journal of Artificial Intelligence Research*, 10(5):353–373, 1999.

[Kantar, 1981] E. Kantar. *Test Your Play, Volume 2*. Wilshire Book Co., 1981.

[Nielsen, 2001] J.L. Nielsen. BuDDy-A Binary Decision Diagram Package. *IT Univ. Copenhagen (ITU), Copenhagen, Denmark: Tech. Rep*, 2001.

[Norvig and Russell, 2009] P. Norvig and SJ Russell. *Artificial intelligence: a modern approach*. Prentice Hall, 3rd edition, 2009.

[Palacios and Geffner, 2009] H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[Reese and Trézel, 1978] T. Reese and R. Trézel. *Those Extra Chances in Bridge*. F. Fell Publishers, 1978.

[Rintanen, 2004] J. Rintanen. Complexity of planning with partial observability. In *ICAPS*, pages 345–354, 2004.

[Russell and Wolfe, 2005] S. Russell and J. Wolfe. Efficient belief-state AND-OR search, with application to Kriegspiel. In *International Joint Conference on Artificial Intelligence*, volume 19, page 278. Citeseer, 2005.

[Smith *et al.*, 1998] S.J.J. Smith, D. Nau, and T. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–106, 1998.

[Throop, 1983] T. Throop. *Computer bridge*. Hayden, 1983.