Machine Learning of Bridge Bidding

Dan Emmons
Computer Systems Laboratory
2008-2009

Abstract

This goal of this project is to create an effective machine bidder in the card game of bridge. Factors like partial information and the multiplicity of the meanings of bids make this task difficult. My plan to overcome these problems is to use a Monte Carlo sampling method for overcoming the limitation of partial information and to use a tree structure of constraints to store the bidding system used by a partnership. Then a machine partnership can train by continually swapping their new bidding inclinations in order to learn new decision networks. The results of this project will not only demonstrate the feasibility of having a machine learn to bid in this manner but also the machine partnerships may develop new bidding conventions useful to human bridge players.

Introduction

Games such as bridge are frequently used as test of and a medium for developing intelligent decision-making algorithms. So far the bidding phase of a bridge hand as proved to be very difficult for machines to perform well for several reasons. Unlike many other games, bridge involves interactions with both cooperative and opponent agents. Also, only part of the total information is available to each agent while bidding, making it difficult to evaluate the effectiveness of given bid until the entire bidding sequence is complete. Perhaps the greatest challenge of all is that bids made by human players usually have more than one meaning. Bids often times suggest contracts to play, but just as often are made to provide the bidder's partner with more information about the bidder's hand. Sometimes they even direct the bidder's partner about what to do while defending. Because of the limited amount of information available to each bidder it can be difficult to determine which of the meanings was intended, or if multiple where intended. The limited information even makes it hard to decide what information would be most useful to give to one's partner about their hand. All these issues would seem to indicate that machine bridge bidding requires a very different category of solution from solutions to problems in other games.

The goal of this project is to create a machine bidding agent as skilled as possible. This will be done using a flexible bidding system framework that can evolve through co-training over the course of many hands. The important developmental points in this project will be designing this framework or choosing from among those already designed, choosing an effective method to overcome partial information, and finding an effective way for partnerships to continually exchange information about their bidding tendencies so that they might improve through co-training. Progress will be measured using the average change in IMPs scored per hand in simulated team matches. This project will not deal with the playing phase of a hand beyond what is required to evaluate the performance of bidding agents.

Background

Prior research in this area has indicated that the Monte Carlo sampling method is an effective way to overcome the limitation in bridge of partial information (Amit and Markovitch). Using this method a bidder would generate a random sample of many hands that satisfy the constraints already placed upon each hand by the bidding and the known thirteen cards that must lie within the bidder's hand. Actions are evaluated based on these sample hands. Probabilistically, the bidder is evaluating based on what are close to the actual percentage chances of the scenario lying that way, so this method effectively allows the machine bidder to "imagine" the results of their actions based on the expected card distribution.

Double-dummy solving is generally considered to be a solved problem at the current state of the art in machine bridge bidding. Many techniques are known to be very effective in solving within large search trees within little time. The hash table, quick trick checking, and single suit analysis methods optimizations are at this point known to be helpful in solving quickly (Chang). These and other improvements are being adopted for the requisite double-dummy solver in this project in order to be able to measure the improvements in machine bidding agents at a more reasonable pace.

Development

The development of a fast double-dummy solver was the first step taken in this project. This tool is necessary for evaluating the performance of bidding agents.

The algorithm used to count the number of tricks available in a contract is based on the standard minimax algorithm. The algorithm is given the remaining cards of the hand that must play next, the cards that have been played so far in the current trick, a goal trick count to reach, and the number of tricks taken so far. It will recursively call itself with many different cards from its hand having been played each time. The algorithm is initialized first with the goal number of tricks being 7, and subsequent calls are made that shift the goal to the midpoint of the range that is known to contain the goal.

If the algorithm is run on a hand position in which the opponents have taken too many tricks to allow the declarer to reach the goal, the algorithm returns to its caller that the goal number of tricks cannot be reached from the hand position. If the algorithm is run on a hand position in which the declarer has already reached his goal number of tricks the algorithm returns to its caller that the goal number of tricks can be reached from the hand position. If a defender sees just one position in which the declarer cannot make the goal number of tricks, he prematurely returns that the goal cannot be reached. If the declarer or his partners see a position in which they can reach the goal number of tricks, they prematurely return this result.

This algorithm runs far too slowly to be able to evaluate a thirteen-trick hand, so many improvements and optimizations were made in order to facilitate this. The first optimization introduced was the use of a hash table. This hash table stores hand positions as keys and maps them to the known upper and lower bounds of tricks that the declarer can make from that position. This prevents positions that can be reached by many lines of play from being re-

evaluated, which saves a very significant amount of time. It also in some cases prevents re-evaluation of positions when the algorithm is initiated with a different goal value.

The hash table improves speed significantly, but quickly depletes memory when evaluating a full thirteen-trick hand. The most memory-consuming part of an entry was the hand position, so finding a way to generate a unique numerical value for each hand to store in the hash table vastly reduces the memory cost per entry. An even better method would be to take advantage of this value to cause hands that will have the same trick result to have a hash collision, making the second one adopt the trick score of the first without even evaluating it. The solution implemented uses a character array. The first four bytes hold the locations of the top four cards from each suit. Each pair of two bits holds the number of the player that holds the corresponding card. The next eight bytes hold the lengths of the suits of each player, using each set of four consecutive bits to hold the suit length for the corresponding player (although in almost every case only three bits are needed). The last byte contains the player who must play the next card. This character array is then converted into an integer by using the hash code function of strings. This function does permit some collisions but is erratic enough in its distribution of values across the hash table that it is impossible for two hands with the same associated value to be transformed into each other by any legal line of play, so unwanted collisions never occur anyway.

Another useful optimization reduces the average branching factor of the enormous search tree. A card play is not tried if another equal card has already been checked. An example of this would be if a player has both the jack and queen of diamonds; only one needs to be checked because the cards are for all intents and purposes equal-valued. Another example would be if a player had both the ten and the queen of diamonds and the jack had already been played. A similar optimization is only checking cards that can beat the current winning card for the trick and the lowest play option available. This has the effect of preventing a player from losing to a trick unnecessarily high. For instance, if the seven of hearts is lead, the next player need only check the cards above the seven and his lowest card.

In very simple but effective searching optimization is to search in a breadth-first manner when the declarer is only one trick away from either reaching or failing to reach is contract. This breadth-first searching is done with each player only checking the highest card they can play in each suit. Searching in this manner reduces the average search time by a factor of two, because it prevents the computer searching for an elaborate line of play to defeat a contract by more tricks in the future when there is a simple line of play to defeat it immediately.

The final optimization used to speed up the process of double-dummy solving is ordering card choices according to the likelihood that they will benefit the player. This allows me to give the computer simple bridge rules that I have learned are usually correct in my own experience with the game to try first. These help search times by more frequently forcing cutoffs of the search at nodes prematurely by finding a line of play that either reaches the goal number of tricks or prevents reaching it, depending on the player that must play next. It is not necessary that these rules be correct in every situation because the computer will continue on with the search at these nodes if such a line is not found. Examples of these principles include covering the opposing

team's honors with your own, playing low to a trick if you can't beat your partner's card, and if playing last to a trick take it with the minimum possible card.

Performance Analysis

My program has been tested by running it many times to obtain an average time for analysis after each adjustment to the code. Testing for correctness has been done by having skilled bridge players examine the hand until they have figured out the answers. This is very time consuming but is the only way to know for sure whether or not the program is correct. A program that was more frequently correct was always judged to be superior to a program that was faster.

Here is output data from a sample run of the double-dummy solving program. These values are correct and the solving process took approximately 2 seconds overall:

North:
Clubs: T 7 5 3 2
Diamonds: J
Hearts: A Q J T
Spades: T 9 7

West:
Clubs: 6
Diamonds: A K T 7 5
Hearts: 9 8 4
Spades: Q J 6 2

East:
Clubs: A J 8
Diamonds: Q 9 8
Hearts: 5 3
Spades: A K 8 5 4

South:
Clubs: K Q 9 4
Diamonds: 6 4 3 2
Hearts: K 7 6 2
Spades: 3

Trick Counts for Each Declarer (North, South, East, West):
Clubs: 9 9 3 3
Diamonds: 2 2 11 11
Hearts: 7 7 3 3
Spades: 0 0 11 11
No Trump: 2 2 8 8

Results

Currently no bidding agents have been implemented and so there are no results to report yet. However, once machine partnerships are able to co-train the double-dummy solving tool that has been completed will be able to compute the progress of the partnership.

Cited Literature

Chang, M. (1996, August 1). Building *a Fast Double-Dummy Bridge Solver*. Retrieved September 29, 2008 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.423

Amit, A., & Markovitch, S. (n.d.). *Learning to Bid in Bridge*. Retrieved October 21, 2008 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.1837

Appendix A: Rules of Bridge

Like chess, the card game bridge is exceptionally difficult to learn, although the rules are simple. The game uses a standard deck of playing cards, with an equal number of cards dealt randomly to each player. Bridge must be played with exactly four players; neither more nor fewer players are permissible. Each player is partnered with the player across from them and opponents with the other two players. A hand of bridge is played in two phases: the bidding and the playing. When playing the game bidding comes first, but it is necessary to have an understanding of the play before attempting to understand bidding.

The play consists of thirteen tricks, where each trick is a grouping of one card from each player. A trick commences with one player leading any card in their hand. Each subsequent player going around the table clockwise must play a card of the same suit from their hand. If they have no cards of the suit that was lead, they may play any card they wish. The player that wins the trick is the one who plays the highest card of the suit that was lead for the trick. There is also may or may not be a trump suit. If there is, the highest trump card played in the trick wins irrespective of whether trump was the lead suit or not. The player who leads for the next trick is the winner of the previous trick.

In the bidding each partnership must decide which trump suit it wishes to play in and how many tricks they think they can win between the two of them. The first person to bid is the dealer, and the bid moves clockwise around the table. On your turn you may bid a contract, pass, double, or redouble. A contract consists of a number and either a suit or no-trump. If a contract is bid it must be higher than all previous contracts. A contract is defined to be higher than another contract if either its number is higher or the numbers are equal and its suit is higher. The order of suits from lowest to highest is clubs, diamonds, hearts, spades, and lastly, no-trump. The suit represents what you wish to be the trump suit, and the number is the number of tricks the player thinks that him and his partner can take together minus six. The minimum bid is one, corresponding to seven tricks, and the maximum bid is seven, corresponding to all thirteen tricks. A can only be bid if the previous contract was bid by the opponents. The double increases the number of points the opponents score if they make their contract but increases the number that the doubling partnership gets if they fail to make their contract. A redouble increases these two values even more. A redouble may only be bid if the previous contract was bid by the redoubling partnership and the contract has already been doubled. The bidding sequence ends after three consecutive passes, unless the first three bids are passes, in which case the fourth player must pass to end the bidding.

If everybody passes both sides receive score zero for the hand. Otherwise the partnership that made the last contract must play the hand and take that many tricks. The first person of that partnership to have bid the suit that ended up as trump is called the declarer. Their partner is called the dummy. The person to the left of the declarer leads for the first trick. Immediately following this lead the dummy lays down all their cards so that everybody can see, and the declarer plays both hands attempting to take as many tricks as he bid.