Give Me a Hand: A Bridge Agent

Mike Liao (SUNET: mikeliao)

Stanford University

1 Introduction

The goal of the project is to build an intelligent agent that can play the card game **Bridge**. Bridge is a complex and highly strategic game. The game has proven to be quite enduring and some players have played the games for ages and never grew tired of it[1]. Although the game is a lot of fun, it is a totally different story when it comes to developing an agent to play the game. The game involves many moving pieces that require a lot of efforts. In the project, we apply a states-based model and a variable-based model, each with a different purpose to solve parts of the problem. Also, we will only tackle the actual card playing and skip the auctioning part of the game. This is to remain focused on the topic that is reasonably scoped for a 10-week course project. Even so, we will soon learn the challenges and the complexity of the project in the following sections.

Brief Introduction to Bridge. The game involves four players who form 2 teams, the EAST-WEST team and the NORTH-SOUTH team. Each of the players will be dealt 13 cards in the start of the game. The players will then take turns to bid in the auction that results in a contract (the player who wins the auction is referred to as the declarer). The players then take turns to play one card at a time. At each round, or trick, after all four players have their chance to play a card, the cards are compared based on the rule of the game and the contract so that the player whose card dominates those of the other 3 players wins the trick. The winner of the trick then proceeds to lead a new card in the next trick and this continues until all 52 cards are played in the 13 tricks. There are greater detailed in each part of the game, a more detailed introduction can be found here[2].

The Infrastructure. The simulation of the game is an important aspect of the project. The one we use is based on a project developed by Lorand Dali [1]. The game engine has an existing agent, called BEN, which the author of the project developed using neural networks. We have refactored parts of the original code to install our agents to participate in the game.

[1] Warren Buffet and Bill Gates are two famous enthusiasts of the game
[2] https://en.wikipedia.org/wiki/Contract_bridge

2 Literature Review

1. The State of Automated Bridge Play [2] - the paper reviews some of the works done by the AI researchers for different aspects of the game, such as card playing, bidding, and double dummy analysis. This provides a good starting point to research other related works.
2. Search Algorithms for a Bridge Double Dummy Solver [3] - this paper discusses the implementation details and provides pseudo code for the popular Double Dummy Solver (DDS) tool. The source code of the DDS project is hosted in Github [4]. DDS is implemented using zero-window search, an optimized version of the alpha-beta pruning.
3. GIB: Imperfect Information in a Computationally Challenging Game [5] - the Bridge program, GIB, discussed in this paper is based on Monte Carlo simulations. The paper also discusses optimization techniques such as partition search to limit the saearch space. GIB was at the time the most advanced computer bridge program in the world.
4. The $\alpha\mu$ Search Algorithm for the Game of Bridge [7] - the paper discusses some of the shortcomings resulting from the use of Monte Carlo sampling in developing bridge agents.

It appears the advanced bridge programs nowadays are mostly based on Monte Carlo techniques, including the current world robot champion, which are commercial based and closed sourced [6].

3 Dataset

Our data is compiled bridge games downloaded from the **PBN Homepage**[3]. The dataset is contained in PBN files. Since the BEN game engine already parses most of the data needed for simulation, we only needed to make some changes to parse the details of the auction and contract from the PBN files.

Sample Game

```
[Deal "N:52.J76.KQJ874.T9 AK3.T9854.T.K832 Q976.Q.9632.QJ74 JT84.AK32.A5.A65"]
[Declarer "E"]
[Contract "4H"]
```

4 Baseline and Oracle

Our baseline is a rule-based agent that tries to win a trick whenever possible or plays defensively when it appears there is no chance in winning the current trick. Since the agent cannot see all of the cards, for those unseen cards, we just distribute them randomly between the two hidden hands.

[3] https://www.tistis.nl/pbn/

If the agent is leading the trick, it will either play a card such as an honor card[4] that dominates the other three players or a plain card[5] in the suit that helps its partner to win the trick. For example, the agent will play a **S5** if it determines its partner holds a **SA** to win the trick. On the other hand, if there is no such card that is guaranteed to win the trick, the agent will just play defensively with a plain card without sacrificing an honor card such as **SQ** which still has the chance to win a trick down the road.

For the oracle, we adopt the Double Dummy Solver (DDS) developed by Bo Haglund [3]. In a double dummy analysis, all cards are visible. As a result, the DDS algorithm can compute the maximum number of tricks that either team can possibly take by assuming each team plays optimally. This is not how the normal games proceed since each player only sees her own hand and the dummy hand. So technically, by having the full visibility, the oracle is cheating.

5 Main Approach

5.1 Minimax

The core of our agent involves searching through the game tree to find a best card to play in each trick. We adapt the minimax agent from the course for a 4 player turn-based game where the *NORTH* and *SOUTH* are the max players and *EAST* and *WEST* are the min players. The value to maximize or minimize is the number of tricks taken by the *NORTH-SOUTH* team. Specifically,

$$V_{\text{minimax}}(s,d) = \begin{cases} \text{Utility}(s) & \text{if IsEnd(s) is True} \\ \text{Eval}(s) & \text{if } d = 0 \\ \max_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a), d') & \text{if Player(s)} \in \{\textit{NORTH}, \textit{SOUTH}\} \\ & d' = d-1 \text{ if LastCardInTrick(s)} \\ \min_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a), d') & \text{if Player(s)} \in \{\textit{EAST}, \textit{WEST}\} \\ & d' = d-1 \text{ if LastCardInTrick(s)} \end{cases}$$

We use the unplayed cards (52 of them in the start of the game) as the states and the played card will be popped off from the states based on the action taken by the players in the game tree. In addition, we will keep track of the number of tricks won by the *NORTH-SOUTH* team in our states so that we could decide how a particular leaf in the tree is performing.

Due to the partially observable nature of the game (half of the cards are hidden at the start of the card playing), there are some uncertainties involved. However, we think expectimax model is not a suitable model since there is no reason to believe our opponents will play randomly. The uncertainty comes from the imperfect information rather than random plays by the opponents. We will

[4] Honor cards are the cards in Aces, Kings, Queens, and Jacks.

[5] Plain cards are the rest of the cards that are not honor cards.

deal with the uncertainty in a different fashion later on. On a lesser note, based on the states of the game, the playing sequence could be altered from trick to trick since the winning hand can lead a card in the next trick.

Lastly, to better match the flow of the game, we will define a level (depth) in the game tree in terms of a trick. Since the search space is huge, we apply depth-limited search and alpha-beta pruning. The depth level is decremented in the recursive call when a player plays the last of the four cards to complete a trick. As EVAL(S) is called, we compute the maximum tricks we can possibly take from the remaining cards using the DDS algorithm [3].

5.2 Card Distribution and its Impact on Trick Taking

Like the baseline, our initial minimax agent also assigns unseen cards randomly in the hidden hands. For human players, reasoning and calculating the likelihood of card holding is an essential element in forming a playing strategy. The placement of the cards and the sequences of card playing could become a decisive factor.

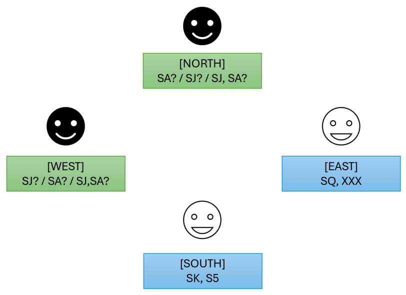


Fig. 1. Delimma

Consider a case (Figure 1) where *EAST* is the dummy so the hand of *EAST* is known and *SOUTH* needs to lead a card in a new trick. SOUTH has two choices in **Spades**, **SK** or **S5**. Furthermore, suppose **SA** and **ST** have not been played yet and could be held by *NORTH* or *WEST*. If *NORTH* holds **SA**, *SOUTH* can play **S5** so that NORTH can win the trick or plays **SK** first to win a trick and leads with **S5** again for *NORTH* to play **SA**. However, if *WEST* holds **SA**, *SOUTH* should not play **SK** since **SK** will lose to **SA** played by *WEST*. *SOUTH* will waste a trick by playing **SK** since the card still has the chance to win a trick down the road. As our agent only assigns the unseen cards randomly, the mistake of playing **SK** too early as in the above case could be made with 50% chance.

5.3 Modeling the Placement of Unseen Cards

To be fair, the random assignment is based on the probability that **SA** has equal chance of being held by either *NORTH* or *WEST*. Without any other information, this might be the best assumption we can make. However, during the progress of the game, players will make decisions based on the cards they have and the different behaviors from the card playing should be taken into account to infer with greater confidence which player holds what cards.

One important aspect of the game that we have so far neglected is the auction. In general, players bid the suits in which they hold a strong hand. For example, if player bids **Spades**, it generally means she has either many cards in **Spades** or has some of the honor cards in **Spades**. The art of biding goes far beyond this simple observation, biding could be made to convey certain information to the partner or could be based on strategic moves to outbid the opponents. However, since we do not focus on this part of the game, we will make some simplified assumption to model the distribution of the unseen cards.

Let's consider a Bayesian Network where C represents the number of cards a player holds in a suit (**Spades**, for example). H represents the number of honor cards she holds in **Spades** given C. H is conditioned on C since the number of honor cards in **Spades** is conditioned on the number of **Spades** cards a player holds. B is an indicator variable whether the player bids **Spades** given C and H. The relationship can be represented in Figure 2.

Suppose X number of cards in **Spades** are in the hidden hands and there are 13 cards hidden for each of the two hidden hands (note that the number of hidden hands is always 2 after the auction concludes). Also, among X hidden cards in **Spades**, there are Y honor cards. We see that the distributions for $\mathbb{P}(C = c)$ and $\mathbb{P}(H = h \mid C = c)$ are known and can be written as

$$\mathbb{P}(C = c) = \frac{\binom{X}{c}\binom{26-X}{13-c}}{\binom{26}{13}} \tag{1}$$

$$\mathbb{P}(H = h \mid C = c) = \frac{\binom{Y}{h}\binom{X-Y}{c-h}}{\binom{X}{c}} \tag{2}$$

Since we do not know the distribution for $\mathbb{P}(B = b \mid C = c, H = h)$, we will use the maximum likelihood estimator (i.e. counting the occurrences of bidding in **Spades** given $C = c, H = h$) to learn the distribution table. The distribution is hard-coded in our project so that we could lookup a value when needed.

Then, we will ask the question: when we see a player bids **Spades**, what are the likely number of cards (plain and honor) she holds in **Spades**. This can be computed from inference as

$$p(C = c, H = h \mid B = 1) \propto p(C = c)p(H = h \mid C = c)p(B = 1 \mid C = c, H = h) \tag{3}$$

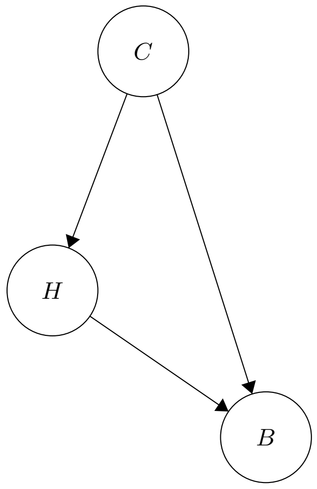


Fig. 2. Bayesian Network for Bidding

5.4 Solving for the Bayesian Network

From lecture, we know that we could reduce Bayesian Network to Markov Network and map out the corresponding factor graph. Ultimately, we will need to assign the unseen cards between the hidden hands, so our goal is to find the assignment of C and H. We will frame the Markov Network as a CSP problem and solve for the assignment of maximum weights. By only considering the assignment of max weights, we understand we risk disregarding some significant amount of probability associated with all other assignments. For example, if our maximum assignment is $\{C = 5, H = 1\}$ and $p(C = 5, H = 1 \mid B = 1) = 40\%$, we essentially commit to an assignment which only occurs 40% of the time.

Alternatively, we could sample from the inferred distribution of $\mathbb{P}(C = c, H = h \mid B = 1)$ defined in Equation 3. However, in this method, we might need to run simulation based on different sampled results and compute expectation of some sort. This is an approach better fitted for Monte Carlo simulation. Since it is not our approach, we will not go this route.

5.5 Inferring Distribution of Unseen Cards during Trick Playing

So far, our probabilistic model focuses on the distribution of 26 unseen cards at the start of trick playing. However, as more hidden cards get revealed as the game progresses, ideally we should adapt our model to account for the new observation such as what card gets played or even what does not get played and update our belief in the possible distribution of the remaining unseen cards. This requires deeper knowledge in the game playing card-by-card or learning from data to derive the relationship between card holding and card playing. We did not attempt to solve this problem in this project. However, we will include one rather obvious observation. That is, when a player is **void** in a suit (**Spades** for example)[6], the probability that she holds any card in **Spades** should drop to 0. In our actual implementation, we just need to add an additional constraint in the factor graph and solve the updated CSP problem.

6 Evaluation Metric

For the project, our agents will always play the *SOUTH* seat against the existing robots BEN. This helps us to establish a common standard to evaluate our various agents, **the baseline**, **the oracle**, and **the main approach**. If *SOUTH* is the declarer, our agent will play both the *SOUTH* and the *NORTH* since *NORTH* is now the dummy. At the end of each simulated game, we will log the number of tricks taken by the *NORTH-SOUTH* team for analysis. One exception is that when *NORTH* is the declarer, we will not record the result of that game since our agent does not participate in the game as a dummy hand.

We will use the oracle as the benchmark and for each of the simulated games, compute the differences between the number of tricks taken by the oracle and that by the other agents. For instance, if the oracle won 10 tricks in a particular game while a minimax agent won 8 tricks, we will log the differences as -2. The smaller the difference is for an agent, the more performant the agent is.

7 Results and Analysis

We use 4 PBN files from our datasets to run the simulation and track performances. Excluding the games where *NORTH* is the declarer, we record a total of 113 games where *EAST* plays the declarer 34 times, *SOUTH* 42 times and *WEST* 37 times. Our baseline only has one implementation. On the other hand, we have three flavors of the minimax agents. Specifically,

1. **minimax_d1**: minimax with depth one ($d = 1$) search
2. **minimax_d1_bayes**: 1st agent with inferred card distribution
3. **minimax_d1_opt**: 2nd agent with configuration when *SOUTH* plays the declarer.

The total differences in the number of tricks taken are summarized in Figure 3. All of our minimax agents performed better than the baseline. The baseline took 158 fewer tricks than the oracle while the **minimax_d1** agent took 124 tricks fewer. On average (per game), the baseline took 1.4 fewer tricks than the oracle while **minimax_d1** took 1.1 tricks fewer (as in Figure 4).

The comparisons between the different minimax agents show that **minimax_d1_bayes** did marginally better than **minimax_d1**. After a closer look,

[6] A **void** in **Spades** means the player has no more cards in **Spades**. Based on the rules of the game, a player must follow suit of the lead card unless impossible. As a result, we can observe whether a player still holds any card in a suit.

we found that although **minimax_d1_bayes** performed better than **minimax_d1** at the games where *EAST* and *WEST* were the declarer (Figure 5 and Figure 6), it did worse in the games when *SOUTH* played the declarer (Figure7). In the **minimax_d1_opt** agent, we attempted to use probabilistic inference only in the cases when *SOUTH* was not the declarer and **minimax_d1_opt**'s performance did improve by a wider margin to −112 from −124 in **minimax_d1** (Figure 3).

8 Error Analysis

In Figure 8, we saw that increasing the search depth in our minimax agents did not improve our results. In fact, our **minimax_d2** did worse than **minimax_d1**. This might seem surprising at the first glance. However, this behavior has to do with the error in the assumption the agent made about card distribution in the hidden hands. Increasing the search depth requires the agent to build up a larger game tree with more actions from the hidden hands. If the agent's belief in card distribution in the hidden hands deviates from the truth in the first place, searching through a deeper tree with more errors in the tree nodes would just make the matter worse. This is an issue we described earlier with an example in Section 5.2.

On the other hand, if we compare the minimax agents with Bayesian Network using different search depths, such as **minimax_d1_bayes** versus **minimax_d2_bayes** in Figure 8, increasing depth does seem to improve the result slightly. However, since the difference is not really significant, it might have been caused by the randomness inherited in the procedures of card assignment. The behavior we observed illustrates the limitation of the minimax algorithm in dealing with a game with hidden information such as bridge.

9 Future Work

As we discussed in Section 8, there are some fundamental deficiencies in the minimax algorithm when it comes to games with imperfect information. Our adoption of a probabilistic inference model for card distribution seems to move in the right direction but it has to be more sophisticated. For example, we could try to model card playing from the hidden hands using probabilistic models with transitional states. As the game progresses and we observe more cards played by the hidden hands, we should adapt our belief in the card distribution and make the necessary adjustment. So far, we only deal with the case where a player is void in a particular suit. This is bare minimum that has to be done. In a real bridge game, all card playing comes with many calculations and players keep updating their knowledge about the game. It is interesting to see if there is a way to model these dynamics using the techniques we know.

Alternatively, we could take a totally different approach. As far as we know, the most state-of-the-art bridge programs are based on Monte Carlo simulations. It might be interesting to explore the application of the Monte Carlo techniques

Fig. 3. All Games

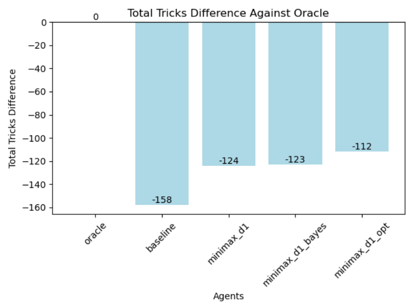


Fig. 4. Average In All Games

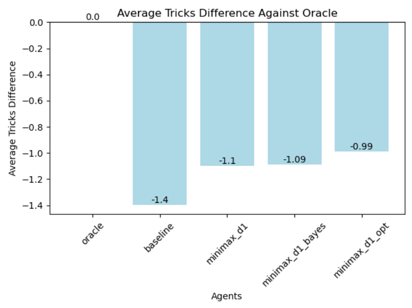


Fig. 5. *EAST* Declarer Game

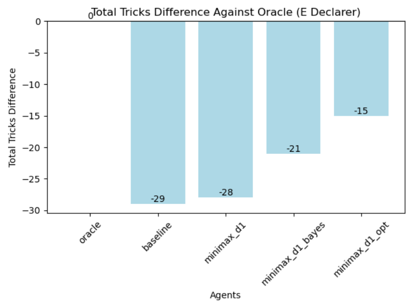


Fig. 6. *WEST* Declarer Games

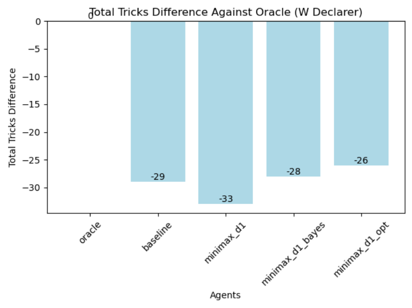


Fig. 7. *SOUTH* Declarer Games

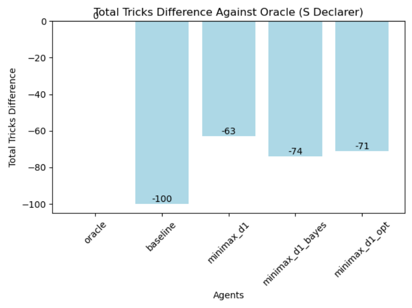
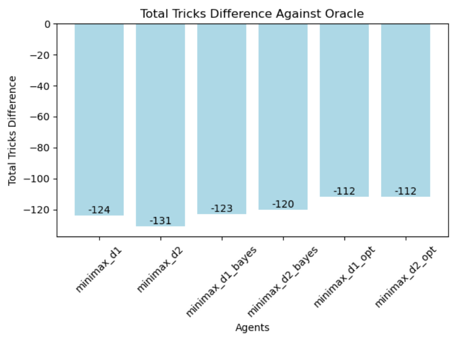


Fig. 8. D1 Versus D2: All Games



and see if we did end up with better results. Fortunately, all the infrastructure we built for this project will become useful even if we are to change our approaches.

Lastly, we could start looking into the auction part of the game. As discussed in Russell and Norvig's book, the authors mentioned that bridge programs have become fairly competitive in playing the cards but still lag behind when it comes to the bidding phase because the agents do not completely understand the conventions used in the communications with the partners [6]. It will be interesting to learn what the challenges are.

10 Code

The project is hosted on Github https://github.com/mikeliaohm/cs221-final-project. Most of our works are inside the SRC/AGENT directory, including the different agents and card assigners for the hidden hands. All of the dataset can be found in SRC/AGENT/BOARDS/DATASET and results can be found in the SRC/AGENT/RESULTS directory.

References

1. Dali, L.: Bridge Engine, BEN, https://github.com/lorserker/ben
2. Bethe, P.: The State of Automated Bridge Play (2010)
3. Haglund, B., Hein, S.: Search Algorithms for a Bridge Double Dummy Solver (2014)
4. Haglund B: Double Dummy Solver, https://github.com/dds-bridge/dds
5. Ginsberg, M.,: GIB: Imperfect Information in a Computationally Challenging Game (2001)
6. Russell, S., Norvig, P.: Artificial Intelligence A Modern Approach Fourth Edition p224-225
7. Cazenave, T., Ventos, V.: The $\alpha\mu$ Search Algorithm for the Game of Bridge (2019)