Contract Bridge Bidding by Learning

Chun-Yen Ho and **Hsuan-Tien Lin**
Department of Computer Science & Information Engineering, National Taiwan University, Taipei, Taiwan
r01922014@ntu.edu.tw, htlin@csie.ntu.edu.tw

Abstract

Contract bridge is an example of an incomplete information game for which computers typically do not perform better than expert human bridge players. In particular, the typical bidding decisions of human bridge players are difficult to mimic with a computer program, and thus automatic bridge bidding remains to be a challenging research problem. Currently, the possibility of automatic bidding without mimicking human players has not been fully studied. In this work, we take an initiative to study such a possibility for the specific problem of bidding without competition. We propose a novel learning framework to let a computer program learn its own bidding decisions. The framework transforms the bidding problem into a learning problem, and then solves the problem with a carefully designed model that consists of cost-sensitive classifiers and upper-confidence-bound algorithms. We validate the proposed model and find that it performs competitively to the champion computer bridge program that mimics human bidding decisions.

1 Introduction

Game-playing is a rich field for artificial intelligence (AI) research. The vast majority of research has focused on full information games such as chess and Othello. On the other hand, a more challenging class of incomplete information games such as poker and bridge continues to be of research interests (Sandholm 2010). A popular research direction is to exploit machine learning to analyze data in an effort to find better game-playing strategies (Bowling et al. 2006; Ponsen et al. 2008; Teófilo et al. 2012).

Contract bridge, or simply *bridge*, is an incomplete information game that is played with a standard 52-card deck. The game requires four players, commonly referred to as North, East, West, and South. Players compete on two opposing teams, North-South and East-West, with the objective of earning the highest score in a zero-sum scenario.

A bridge game consists of several deals, each comprising two stages—the bidding stage and the playing stage. At the beginning of each deal, each player is dealt 13 random cards. In the bidding stage, the two teams engage in an auction in an attempt to find the most profitable contract for the playing stage. During the auction, each player can only see their own 13 cards and not those of the other players including their teammate. The auction proceeds around the table, with each player deciding to PASS or to increase the value of the bid from an ordered set of calls $\{1\clubsuit, 1\diamondsuit, 1\heartsuit, 1\spadesuit, 1\text{NT}, 2\clubsuit, \cdots, 7\text{NT}\}$ or by a more sophisticated call, such as doubling the current bid. The auction proceeds until it is terminated by three consecutive PASS calls, at which time the final bid becomes the contract of the deal. The contract consists of a number and a symbol. The symbol indicates the trump suit and the number indicates the number of rounds that the auction-winning team expects to win during the ensuing playing stage.

The player from the auction-winning team who first called the trump suit becomes the declarer. The playing stage comprises 13 rounds of a card-strength competition. During the playing stage, the auction-winning team attempts to make the contract, while the auction-losing team aims block the opponent team from making the contract. Ultimately, the scores of the teams are determined by comparing the contract with the actual number of winning rounds.

The bidding stage is often regarded as being more difficult to implement than the playing stage. For example, in 1998, the GIB program (Ginsberg 2001) attained 12^{th} place among 35 human experts in a contest without bidding (Ginsberg 1999). This demonstrates that computer bridge players can compete against expert human players in the playing stage. On the other hand, nearly all the computer bridge programs that are currently available borrow human bidding systems, which contain human-designed rules for communicating information between teammates. Traditional bridge AIs have attempted to convert these rules into computer programs. However, human-designed rules often contain ambiguities and even conflicts, which complicate the task of programming a bidding system. In addition, using only human-designed rules limits the capability of the machines.

There have been several successful efforts to improve the bidding AI. For example, a reasoning model for making decisions with a rule-based bidding system has been proposed in (Ando and Uehara 2001). By first constructing a decision network from a bidding system, Amit and Markovitch (2006) propose a Monte Carlo Sampling approach to decision making in the presence of conflicting bids. Further, the authors propose a decision tree based learning method for resolving conflicts. In the work of DeLooze and Downey

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

(2007), a learning method based on self-organizing maps is proposed for the problem of learning a human bidding system from examples, rather than from explicit statement of the rules. However, each of the previous work is based on a bidding system that is pre-designed by human experts.

In this work, we consider a different way for improving the bidding AI. We take an initiative to study the possibility that the machines can learn to bid without relying on a human-designed bidding system. In particular, we intend to lay out approaches for the machines to "learn their own bidding system" from raw data that contain only random deals. Our study not only opens a new route for improving the bidding AI, but also examines whether a machine-learned bidding system can be competitive to a human-designed one.

The incomplete information properties of the bidding stage make this a difficult task. Because each player can see only her/his 13 cards, it is difficult for the player to infer the best contract for the team directly. Thus, the bidding stage itself is usually considered a channel for players to exchange information. However, because the auction follows the rule of using monotonically increasing bids, players must avoid exceeding the optimal contract when exchanging information. Moreover, each team could interfere with the other's ability to exchange information. Together, these properties render the design of machine learning approaches for automatic bidding a difficult task.

In this paper, we propose a novel framework for applying machine learning approaches to the task of automatic bridge bidding without competition. We transform the bidding rules to a formal learning problem along with corresponding scalable data generation steps in Section 2. Next, in Section 3 we evaluate the potential of machine learning approaches by designing several baseline methods and analyzing the key challenge of solving the problem using machine learning. Then, we propose an innovative learning model with layers of cost-sensitive classifiers and upper-confidence-bound algorithms for solving the problem. We empirically demonstrate in Section 4 that the performance of the model is competitive to the contemporary champion-winning computer bridge software that implements a human-designed bidding system.

2 Problem Setup

We first separate the general bidding problem into two subproblems: bidding with competition, and bidding without competition. Both problems cover considerable amounts of deals in real-world bridge games. For the first initiative toward allowing the machine to learn its own bidding system automatically, we use settings similar to existing works (Amit and Markovitch 2006; DeLooze and Downey 2007), and study the sub-problem of bidding without competition.

The sub-problem can be formalized as follows. We use $\mathbf{x}$ to denote the cards of a player, and a length-ℓ sequence $\mathbf{b}$ to denote the bids of that player and the bids of her/his teammate. Each component of $\mathbf{b}$ is within an ordered set $\mathcal{B} = \{\text{PASS}, 1\clubsuit, 1\diamondsuit, \cdots, 7\text{NT}\}$, where $\mathbf{b}[1]$ is the first bid made by the team, $\mathbf{b}[2]$ is the second bid, etc.. For simplicity, we can further assume that the team that is bidding sits at the North-South positions, and by bidding without competition, the opponent team sitting at the East-West always calls PASS. The goal is to learn a bidding strategy $G(\mathbf{x}_n, \mathbf{x}_s)$ for predicting $\mathbf{b}$ given the cards $\mathbf{x}_n$ of the North player and $\mathbf{x}_s$ of the South player.

To meet the rules of bridge bidding, we further decompose G as follows. Let $g(\mathbf{x}, \mathbf{b}^k) \mapsto \mathcal{B}$ be the bidding function used for predicting bids in the bidding strategy G, where $\mathbf{b}^k$ denotes the bidding sequence until the k^{th} bid. Then for every $\mathbf{b} = G(\mathbf{x}_n, \mathbf{x}_s)$, we require $\mathbf{b}[1] = g(\mathbf{x}_1, \varnothing), \mathbf{b}[2] = g(\mathbf{x}_2, \mathbf{b}^1), \cdots, \mathbf{b}[\ell] = g(\mathbf{x}_\ell, \mathbf{b}^{\ell-1})$, such that $g(\mathbf{x}_{\ell+1}, \mathbf{b}) = \text{PASS}$, and $\mathbf{b}[1] < \mathbf{b}[2] < \cdots < \mathbf{b}[\ell]$. The monotonicity constraint makes $\mathbf{b}[k] \neq \text{PASS}$ for $k > 1$. Note that the $\mathbf{x}$ used in $g(\mathbf{x}, \mathbf{b}^k)$ for consecutive bids need to originate from different players. Without loss of generality, we assume that $\mathbf{x}_1 = \mathbf{x}_n$: the first player is always the North player.

To learn a good strategy G (or a good g within), we need to provide related data to the learning algorithm. The input (feature) part of the data is easy to generate, because all the relevant information can be obtained from the cards of the players in a deal. The $(\mathbf{x}_n, \mathbf{x}_s)$ can then carry some representation of the cards, and some feature expansion techniques can be applied also to achieve better performance. How can we generate data that relates to the desired output? Note that $\mathbf{b}$ is not directly available when allowing the machine to learn its own bidding system. However, we can indirectly know the goodness of some $\mathbf{b}$ by taking $\mathbf{b}[\ell]$ to the playing stage and obtaining the resulting score. Nevertheless, attempting to play each possible contract $\mathbf{b}[\ell]$ of a deal by either computer or human agents can be extremely time-consuming. To overcome this issue, we use the double dummy analysis (Chang 1996) to approximate the playing stage for evaluating the scores for each contract.

The double dummy analysis is a technique that computes the number of tricks taken by each team in the playing stage under perfect information and optimal playing strategy. Whereas the best game-playing strategy with only partial information might be different from that with perfect information, there are several advantages for using the latter to approximate the former. First, the result is deterministic and independent from the bidding stage. Second, the analysis is fast. Applying the double dummy analysis for a deal requires only several minutes. Finally, the approximation is usually good. In real bridge games, the result of a deal is usually close to that of the double dummy analysis when players are sufficiently strong.

After the double dummy analysis, we not only obtain the score of the best contract, but also have the scores of all possible contracts. We can store the differences between the best score and those scores as a cost vector $\mathbf{c}$. Also note that during data generation, we can drop the cards of the East-West team after the double dummy analysis. Then, we can formally define our learning problem as a cost-sensitive, sequence prediction problem with specialized constraints from bridge bidding rules. Given data $D = \{(\mathbf{x}_{ni}, \mathbf{x}_{si}, \mathbf{c}_i)\}_{i=1}^N$, where N is the number of instances in the dataset, we want to learn the bidding strategy $G(\mathbf{x}_n, \mathbf{x}_s)$ that minimizes the average cost of the predicted contracts (i.e., the final bid).

For this purpose, the objective function to be minimized in the training stage can be written as $\frac{1}{N}\sum_{i=1}^{N}\mathbf{c}_i[\mathbf{b}_i[\ell]]$, where $\mathbf{b}_i = G(\mathbf{x}_{ni}, \mathbf{x}_{si})$.

3 Proposed Model

Baseline and optimistic methods. First, we consider the bidding strategies G that only predict sequences $\mathbf{b}$ of length one. That is, we let $g(\mathbf{x}_s, \mathbf{b}^1) = \text{PASS}$. Thus, $\mathbf{x}_s$ is not needed, and all the constraints are trivially satisfied. Then, the objective function is reduced to minimize $\frac{1}{N}\sum_{i=1}^{N}\mathbf{c}_i[\mathbf{b}_i[\ell]]$ given $D_{base} = \{(\mathbf{x}_{ni}, \mathbf{c}_i)\}_{i=1}^{N}$, which is the standard formulation of the cost-sensitive classification (CSC) problem (Beygelzimer et al. 2005), with many existing algorithms available (Zhou and Liu 2010; Tu and Lin 2010). Here, we consider two regression-based algorithms, cost-sensitive two-sided regression (CSTSR) and cost-sensitive one-sided regression (CSOSR) (Tu and Lin 2010), as our baseline methods because of their close connections to the model that we shall propose next. The latter is one of the state-of-the-art algorithms for CSC. Both algorithms use regression models to estimate the cost for each bid, and predict the bid with minimum estimated cost. The difference is that CSTSR considers the plain-vanilla squared regression objective function, whereas CSOSR considers a more sophisticated function.

The baseline methods hint a lower bound that can be reached by machine learning. How about an upper bound? One possibility is to "cheatingly" reveal the full information to the learner, which simulates what is seen from the audience. That is, we use $D_{cheat} = \{([\mathbf{x}_{ni}, \mathbf{x}_{si}], \mathbf{c}_i)\}_{i=1}^{N}$ to learn a CSC classifier. The performance of this classifier hints an upper bound that can be achieved by machine learning.

The multi-layer bandit model. Next, we use the bidding sequence for better information exchange and better performance. As a simple start, let us consider extending the baseline methods to produce sequences $\mathbf{b}$ of length two. That is, $g(\mathbf{x}_n, \varnothing) = \mathbf{b}[1]$, $g(\mathbf{x}_s, \mathbf{b}^1) = \mathbf{b}[2]$ and $g(\mathbf{x}_n, \mathbf{b}^2) = \text{PASS}$. We can obtain some insight by considering what two human bridge players will do in this scenario. For simplicity, we refer to $g(\mathbf{x}_n, \varnothing)$ as g_n, and to $g(\mathbf{x}_s, \mathbf{b}^1)$ as g_s. Consider two human players who are unfamiliar with each other's bidding strategy and decide to practice together. After the North player uses his strategy g_n to make the first bid $\mathbf{b}[1] = g_n(\mathbf{x}_n, \varnothing)$, the South player has no choice but to use g_s on $\mathbf{x}_s$ and the given $\mathbf{b}[1]$ to make the final bid $\mathbf{b}[2]$. Then, after the score c of the contract is revealed, the South player can now improve her strategy g_s with $((\mathbf{x}_s, \mathbf{b}[1]), \mathbf{b}[2], c)$. The very same c indicates how good g_n is in helping g_s. Then, g_n can improve his strategy with $((\mathbf{x}_n, \varnothing), \mathbf{b}[1], c)$.

Nevertheless, if the North player is lousy and starts with a bad g_n (for instance, some g_n that always calls PASS), he might never know whether some alternative bids can help the South player better. This hints to the need for him to explore other possible bids, rather than sticking to his own strategy. On the other hand, the North player should not make exhaustive random predictions for exploration, because a uniformly random $\mathbf{b}[1]$ gives the South player almost no information to further improve her strategy g_s. That is, the North player should also use g_n to exploit some "good" bids that are known to work well with g_s. By balancing exploration (for improvement) and exploitation (for maintaining the team's chemistry), the two players can build a better bidding system together.

Our proposed model hinges on two aspects of the discussions above. First, the cost of the final contract (i.e., the last bid) can be re-used to hint the cost of intermediate bidding decisions. Second, players need to explore other bidding choices while exploiting the known good bids. The two aspects lead us to consider the Upper Confidence Bound (UCB) algorithms in the contextual bandit problem.

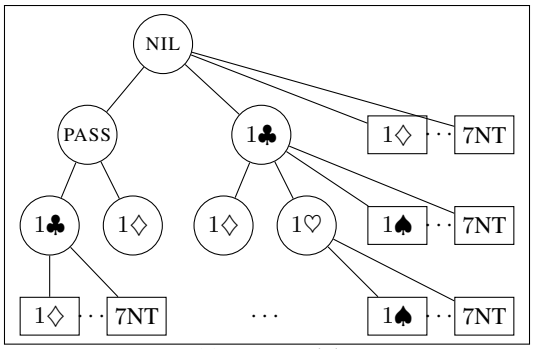
The contextual bandit problem has recently become a popular research topic in machine learning (Li et al. 2010; Chu et al. 2011; Langford and Zhang 2007). In this problem, we hope to earn the maximum total reward within some iteration by strategically pulling a bandit machine from M given ones subject to a dynamic environment context within some iterations. Since there is no additional information available about the M given bandit machines in the beginning, balancing exploration (of other bandit machines) and exploitation (of knowingly good machines) is important. The UCB algorithms (Chu et al. 2011) are some of the most popular contextual bandit algorithms. The algorithms cleverly use the uncertainty term to achieve balance.

We can now pose an analogy of a player's decision to the contextual bandit problem. The possible bids $\mathbf{b}[k+1]$ correspond to the bandit machines, and the context corresponds to the cards $\mathbf{x}$ on hand and the earlier bids $\mathbf{b}^k$. The reward can simply be considered as the maximum possible cost minus the cost calculated from the final contract.

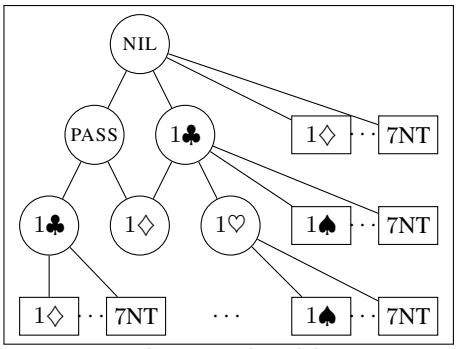
Whereas each player's decision making can be modeled by the contextual bandit problem, recall that our goal is to obtain a bidding strategy G that produces a sequence of decisions that satisfy bridge rules. We propose to represent each player's decision making with layers of "bidding nodes" V. With a careful design to structure these nodes, we can ensure that the bridge rules are all satisfied.

We define a bidding node V as a pair (b, g), where $b \in \mathcal{B}$ is called the bid label that V represents, and g is the bidding function subject to $\mathbf{x}$ and $\mathbf{b}^k$. We propose to structure the bidding nodes as a tree with $\ell + 1$ layers, where the first layer of the tree contains a single root node with the first bidding function $g(\mathbf{x}_n, \varnothing)$ and $b = \text{NIL}$ indicating the entering of the bidding stage. At each V, g is only allowed to predict PASS or something higher than b to satisfy the bridge rules. Every prediction of its g connects V to a child bidding node V' at the next layer such that the prediction equals the bid label of V'. We restrict only the lowest M predictions of g to connect to non-terminal nodes to control the model complexity. Other nodes are designated as terminal nodes, which contain a constant g that always predicts PASS. In addition, all nodes at layer $\ell + 1$ are terminal nodes.

Since we form the nodes as a tree, each unique path from the root to V readily represents a bidding sequence $\mathbf{b}^k$. Thus, the classifier g of V only needs to consider the cards $\mathbf{x}$. We call such a structure the tree model, as illustrated in Figure 1(a). A variant of the tree model can be performed by



(a) Tree model



(b) Layered model

Figure 1: Tree model and layered model with $\ell = 3$ and $M = 2$, the terminal nodes are not fully drawn

combining the non-terminal nodes that represent the same bid label in each layer. The combination allows the nodes to share their data to learn a better g. We call the variant the layered model, as illustrated in Figure 1(b).

Given the model above, a bidding strategy G can be formed by first inputting $\mathbf{x}_n$ to g at the root node, following the prediction of g to another node that represents $\mathbf{b}[1]$ in the next layer, then inputting $\mathbf{x}_s$ to the node, and so on. The process ends when a PASS call is predicted by some g of a non-root node.

After a particular model structure is decided, the remaining task becomes learning each g from data. We propose using CSTSR with ridge regression, which is among the baseline methods that we discussed earlier, as the learning algorithm, because it is a core part of the LinUCB algorithm that we adopt from the contextual bandit problem. Following the notations that are commonly used in the contextual bandit problem, we consider the reward r, which is defined as the maximum possible cost minus the cost, instead of the cost c. For each possible bid b_m, ridge regression with parameter $\lambda > 0$ is used to compute a weight vector $\mathbf{w}_m = (\mathbf{X}_m^T \mathbf{X}_m + \lambda I)^{-1}(\mathbf{X}_m^T \mathbf{r}_m)$ for estimating the potential reward $\mathbf{w}_m^T \mathbf{x}$ of making the bid, where $\mathbf{r}_m$ contains all the rewards gathered when the m-th bid b_m is made by g, $\mathbf{X}_m$ contains all the $\mathbf{x}$ associated with those rewards, and I is the identity matrix. During prediction, CSTSR predicts the bid associated with the maximum potential reward.

Our final task is to describe the learning algorithm for the

Algorithm 1 The Proposed Learning Algorithm

Input: Data, $D = \{(\mathbf{x}_{ni}, \mathbf{x}_{si}, \mathbf{c}_i)\}_{i=1}^N$; a pre-defined model structure with all weights $\mathbf{w}_m$ within all CSTSR ridge regression classifiers initialized to 0.
Output: A bidding strategy G based on the learned $\mathbf{w}_m$.

repeat
  Randomly select a data instance $(\mathbf{x}_{ni}, \mathbf{x}_{si}, \mathbf{c}_i)$
  Set V to the root node of the structure, and $\mathbf{x}$ to $\mathbf{x}_{ni}$
  UPDATE(V, $\mathbf{x}_{ni}$, i)
until enough training iterations
procedure UPDATE(V, $\mathbf{x}$, i)
  For each possible b_m, compute the UCB reward $\left(\mathbf{w}_m^T \mathbf{x} + \alpha \cdot \text{uncertainty term}\right)$
  Select the bid b_m with the maximum UCB reward
  if $b_m =$ PASS **then**
    Compute reward r from $\mathbf{c}_i$ using b_m
  else
    Set $\mathbf{x}$ to the feature of the other player
    Call r = UPDATE(V', $\mathbf{x}$, i)
  end if
  Update $\mathbf{w}_m$ with $(\mathbf{x}, r)$
  return r.
end procedure

model structure with ridge regression. As discussed, we use the cost of the final contract (i.e., the last bid) to form the rewards for intermediate bidding decisions. Then, we follow the UCB algorithms in the contextual bandit problem to update each node. The UCB algorithms assume an online learning scenario in which each $\mathbf{x}$ arrives one by one. First, we discuss the LinUCB algorithm (Chu et al. 2011) to balance between exploration and exploitation. During the training of each node, LinUCB selects the bid that maximizes $\mathbf{w}_m^T \mathbf{x} + \alpha \sqrt{\mathbf{x}^T (\mathbf{X}_m^T \mathbf{X}_m + \lambda I)^{-1} \mathbf{x}}$, where the first term is the potential reward on which CSTSR relies, and the second term represents the uncertainty of $\mathbf{x}$ with respect to the m-th bid. The $\alpha > 0$ is a parameter that balances between exploitation (of rewarding bids) and exploration (of uncertain bids). After LinUCB selects the bid for the root node, we follow the bid to the bidding node in the next layer, until a PASS call is predicted by LinUCB. Then, we know the cost of the bidding sequence, and all the nodes on the bidding sequence path can be updated with the calculated rewards using ridge regression. The full algorithm is illustrated in Algorithm 1.

Another choice for the UCB algorithms is called UCB1 (Auer, Cesa-Bianchi, and Fischer 2002), which replaces the uncertainty term $\sqrt{\cdots}$ in LinUCB with $\sqrt{\frac{2 \ln(T)}{T_m}}$, where T is the number of examples used to learn the entire g, and T_m is the number of examples used to update $\mathbf{w}_m$.

In addition to the model and the core algorithms introduced above, we adopt several additional techniques to improve the performance and computational efficiency.

Full update. In the proposed model, whenever a bidding sequence $\mathbf{b}$ is sampled from the UCB algorithms for an in-

stance $\mathbf{x}$, the reward r can be calculated from $\mathbf{c}[\mathbf{b}[\ell]]$, and the example $((\mathbf{x}, \mathbf{b}^k), \mathbf{b}[k+1], r)$ is formed to update the bidding nodes. A closer look shows that some additional examples can be calculated easily with $\mathbf{b}$. In particular, the cost for calling PASS immediately after k bids can be calculated by $\mathbf{c}[\mathbf{b}[k]]$, and the cost for selecting a terminal node with a bid label b can be calculated by $\mathbf{c}[b]$. Thus, we consider forming additional examples based on the above analysis, and include those examples in updating the associated bidding nodes. Such an update scheme is called FULL UPDATE.

Penetrative update. We consider the UCB algorithms to balance the need for exploration in the proposed model. In some ways, the UCB algorithms are not properly designed for the multi-layer model, and thus can lead to some caveats. For example, in the tree model, the number of instances that pass through a classifier in the top layer can be much more than those in the bottom layer. If the classifiers in the top layers often result in an early PASS, the ones in the bottom layer might not receive enough examples, which result in a worse learning performance. To solve this problem, we consider a probabilistic "penetrative" scheme to continue bidding during training. That is, whenever a classifier predicts a bid that results in an early PASS, we select another bid and call the corresponding UPDATE with some probability p. We require that the selected bid comes with a next bid (i.e., not resulting in an early PASS) and to be of the highest UCB term. In other words, with some probability, we hope to generate longer (but good) sequences $\mathbf{b}$ to help update the lower layers of the model in this PENETRATIVE UPDATE scheme.

Delayed update. We adopt the contextual bandit algorithms in our model, which were designed for the online scenario where examples arrive one by one. Somehow updating instantly per example becomes the computational bottleneck of the algorithms. In view of the efficiency, we consider a DELAYED UPDATE scheme that updates $\mathbf{w}_m$ until gathering a pile of examples.

4 Experiments

Next, we study the proposed model and compare it with the baseline and optimistic methods. We also compare the model with a well-known computer bridge software, Wbridge5 (Costel 2014), which has won the computer bridge championship for several years. A randomly-generated data set of $100,000$ instances (deals) is used in the experiment. We reserve $10,000$ instances for validation and another $10,000$ for testing, and leave the rest for training. We consider the condensed features for $\mathbf{x}$, which contain two parts that are widely used in real-world bridge games and human-designed bidding systems, high card points and number of cards in each suit. We have considered sparse binary features for representing the existence of each card, but find that they do not work better than condensed ones.

We obtain the cost vectors $\mathbf{c}$ from International Match Points (IMP), an integer between 0 and 24 that is widely used for comparing the relative performance of two teams in real-world bridge games (ACBL 2005). We obtain $\mathbf{c}$ by comparing the best possible contract of the deal to each contract and calculate the IMP. When transforming the costs to the rewards in the proposed model, we take 24 minus the cost as the reward to keep the rewards non-negative. [1]

Method	Dimensions	Baseline	Optimistic
CSOSR - condensed	6	3.8329	1.8985
CSTSR - condensed	6	3.9428	2.7697
CSTSR - 2nd POLY	21	3.8465	2.1106
CSTSR - 3rd POLY	56	3.8272	1.9228
Wbridge5	N/A	2.9550	N/A

Table 1: Results of baseline and optimistic methods

Baseline and optimistic methods. First, we present the performance of the baseline and the optimistic methods in Table 1. In CSOSR, SVM with the Gaussian kernel implemented with LIBSVM (Chang and Lin 2011) is used as the base learner. In CSTSR, ridge regression is used as the base learner. For balancing the time spent of the two algorithms, we only sub-sample $20,000$ instances for CSOSR. For CSTSR with condensed features, we also extend its capability by considering simple polynomial expansion (POLY) of the features (Abu-Mostafa, Magdon-Ismail, and Lin 2012). For parameters, we consider $C \in \{10^0, 10^1, 10^2, 10^3\}$ and $\gamma \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^0\}$ for CSOSR, and $\lambda \in \{10^{-6}, 10^{-5}, \cdots, 10^3\}$ for CSTSR. We choose the best parameters based on the validation set and report the average test cost in Table 1.

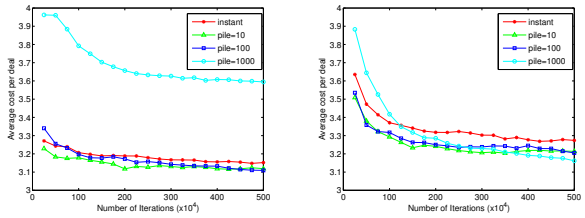
Unsurprisingly, we find that the performance of the optimistic methods to be much better than their baseline counterparts. This justifies that the information in both players are valuable, and it is important to properly exchange information via bidding. In addition, note that the optimistic methods can often achieve lower test cost than the Wbridge5 software. This suggests that the human-designed bidding system within the computer software may not be perfect yet. Comparing over all the baseline methods, we see that using the 2nd order expansion with the condensed features reach decent performance by the baseline CSTSR with only 21 expanded features. Thus, we will take those features within the proposed model in the next experiments.

Full and delayed updates. Next, we study two of the techniques proposed in Section 3 to improve the model. We first compare FULL UPDATE with SINGLE UPDATE.

Iterations (10^5)	2	4	6	8	10
Single Update	5.50	4.00	3.92	3.81	3.77
Full Update	3.33	3.26	3.23	3.24	3.25

The table above shows how the average validation cost varies with the number of iterations on a tree model with $\ell = 4$, $M = 5$ coupled with ridge regression with $\lambda = 10^{-3}$ and UCB1 with $\alpha = 10$. We can easily observe that FULL UPDATE outperforms SINGLE UPDATE, which justifies that

[1] One technical detail is that $\mathbf{c}$ is generated by assuming that the player who can win more rounds for the contract is the declarer.



(a) $\alpha = 10$ (b) $\alpha = 100$

Figure 2: DELAYED UPDATE versus INSTANT UPDATE

the additional examples used for FULL UPDATE captures valuable information to make the cost estimation more precise. Thus, we adopt FULL UPDATE in the next experiments.

Then, we study DELAYED UPDATE. Figure 2 shows how the average validation cost varies with the number of iterations on the same tree model used for the previous experiment. We consider piles of size $\{10, 100, 1000\}$ instances per update and $\alpha \in \{10, 100\}$. We find that when α is small, INSTANT UPDATE or a small pile reaches the best performance, while larger α could use a larger pile. Overall, DELAYED UPDATE does not lead to much loss of performance. Note that INSTANT UPDATE takes more than 4 hours to train a decent model, while DELAYED UPDATE with piles of size 100 only need less than 1 hour. In view of the efficiency needed for extensive parameter selection, we take DELAYED UPDATE with piles of size 100 in the next experiments.

Different model structures. Next, we compare the performance of different model structures to the Wbridge5 software. We consider the tree model and the layered model with $\ell \in \{2, 4, 6\}$, fix $M = 5$, and equip them with either UCB1 or LinUCB. For each model/algorithm combination, we take grid search on (p, α) with the validation set to choose the penetration probability parameter $p \in \{0, 0.25, 0.5, 0.75, 1\}$, and the UCB parameter $\alpha \in \{2^0, 2^2, 2^4, 2^6, 2^8\}$. Note that the tree model and the layered model is equivalent when $\ell = 2$.

Table 2 lists the average training/validation/test cost on all the model/algorithm combinations. The results suggest that the tree model with $\ell = 4$ or 6 coupled with UCB1 performs the best among all models. Furthermore, those best performance are competitive to the result reached by the Wbridge5 software. This marks a successful initiative toward learning to bid without relying on a human-designed bidding system. Also, all the proposed models in Table 2 perform better than the baseline methods in Table 1. The results justify that the proposed models successfully make use of the bidding sequence for information exchanging between teammates.

Comparison with Wbridge5. Table 2 readily lists the competitive performance of the proposed model to Wbridge5. Next, we make a more detailed comparison to understand the strengths and weaknesses of the proposed model. In real-world bridge games, a contract can roughly be divided into five categories based on its raw score from low to high, namely PASS, PARTIAL, GAME, SLAM, and GRAND SLAM. Because categories GAME and beyond result in higher scores, human players (and hence human bidding systems) often prefer bidding towards those. The table below shows the total cost of Wbridge5 minus the total cost of the proposed model in each category.

Best Contract Type	Cost Difference	# Deals
PASS	828	1816
PARTIAL	5014	5165
GAME	-2424	2339
SLAM	-2152	521
GRAND SLAM	-586	159

We see that the proposed model performs much better than Wbridge5 in PARTIAL contracts, which contribute to the majority of the deals. This shows that the proposed model is indeed guided by data rather than human design (that prefers GAME and beyond). On the other hand, a human-played bridge game often contains *competition* when the best possible contract is PARTIAL. Thus, the strength of the proposed model on PARTIAL contracts will need to be compensated with future studies on automatic bidding with competition. Lastly, the weakness of the proposed model on GAME and beyond may be due to the fact that there is insufficient data to warrant decent learning performance in those categories. Some sampling techniques can be applied in the future to focus on those categories of contracts.

5 Conclusions and Future Works

We formally defined the problem of bridge bidding without competition by learning, and proposed an innovative model for undertaking this problem. The model predicts a bidding sequence with layers of classifier (bidding) nodes, and trains each classifier with the aid of UCB algorithms for contextual bandit. The UCB algorithms allow the machines to learn their own bidding system by balancing the exploration for less-considered bids and the exploitation of well-learned bids. We show in experiments that the proposed model can achieve a performance similar to the champion-winning program in the computer bridge. Our initiative justifies the possibility that machine learning may be able to do better than human-designed bidding systems on bridge bidding problem.

One possible direction to improve the model is to use more data to train a deeper model towards valuable contracts such as the Grand Slam. The ultimate challenge is the other sub-problem: bidding with competition by learning. Such a challenge might require a mixture of the proposed model (collaboration between teammates) and well-studied models for competition-based games like Chess.

6 Acknowledgments

We thank Profs. Yuh-Jye Lee, Shou-De Lin, the anonymous reviewers and the members of the NTU Computational Learning Lab for valuable suggestions. This work is partially supported by the Ministry of Science and Technology of Taiwan via the grant MOST 103-2221-E-002-148-MY3.

model	UCB	train	validation	test
Tree/Layered, $\ell = 2$	UCB1	3.1197 ± 0.0177	3.1981 ± 0.0268	3.0755 ± 0.0173
	LinUCB	3.1242 ± 0.0089	3.2190 ± 0.0121	3.0933 ± 0.0112
Tree, $\ell = 4$	UCB1	$\mathbf{2.9013 \pm 0.0079}$	3.0769 ± 0.0118	$\mathbf{2.9672 \pm 0.0096}$
	LinUCB	3.0918 ± 0.0344	3.1804 ± 0.0298	3.0672 ± 0.0379
Tree, $\ell = 6$	UCB1	$\mathbf{2.9025 \pm 0.0210}$	$\mathbf{3.0484 \pm 0.0226}$	$\mathbf{2.9616 \pm 0.0234}$
	LinUCB	3.0124 ± 0.0249	3.1301 ± 0.0264	3.0477 ± 0.0243
Layered, $\ell = 4$	UCB1	3.0779 ± 0.0179	3.1656 ± 0.0198	3.0561 ± 0.0230
	LinUCB	3.0492 ± 0.0214	3.1325 ± 0.0218	3.0290 ± 0.0252
Layered, $\ell = 6$	UCB1	3.1366 ± 0.0176	3.2451 ± 0.0208	3.1214 ± 0.0168
	LinUCB	3.0825 ± 0.0209	3.1781 ± 0.0268	3.0660 ± 0.0224
Wbridge5	N/A	N/A	$\mathbf{3.0527}$	$\mathbf{2.9550}$

Table 2: Average Cost Using Different Model Structures

References

Abu-Mostafa, Y. S.; Magdon-Ismail, M.; and Lin, H.-T. 2012. *Learning from Data: A Short Course*. AMLBook.

ACBL. 2005. Introduction to bridge scoring. http://www.acbl.org/learn/scoreTeams.html.

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

Ando, T., and Uehara, T. 2001. Reasoning by agents in computer bridge bidding. In *Computers and Games*. 346–364.

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Beygelzimer, A.; Dani, V.; Hayes, T.; Langford, J.; and Zadrozny, B. 2005. Error limiting reductions between classification tasks. In *Proceedings of the 22nd International Conference on Machine Learning*, 49–56.

Bowling, M.; Fürnkranz, J.; Graepel, T.; and Musick, R. 2006. Machine learning and games. *Machine learning* 63(3):211–215.

Chang, C.-C., and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2:27:1–27:27. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Chang, M.-S. 1996. Building a fast double-dummy bridge solver. Technical report.

Chu, W.; Li, L.; Reyzin, L.; and Schapire, R. E. 2011. Contextual bandits with linear payoff functions. In *International Conference on Artificial Intelligence and Statistics*, 208–214.

Costel, Y. 2014. Wbridge5 bridge software.

DeLooze, L. L., and Downey, J. 2007. Bridge bidding with imperfect information. In *IEEE Symposium on Computational Intelligence and Games*, 368–373. IEEE.

Ginsberg, M. L. 1999. Gib: Steps toward an expert-level bridge-playing program. In *Proceedings of International Joint Conference on Artificial Intelligence*, 584–593.

Ginsberg, M. L. 2001. Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research* 14:303–358.

Langford, J., and Zhang, T. 2007. The epoch-greedy algorithm for contextual multi-armed bandits. *Advances in Neural Information Processing Systems* 20:1096–1103.

Li, W.; Wang, X.; Zhang, R.; Cui, Y.; Mao, J.; and Jin, R. 2010. Exploitation and exploration in a performance based contextual advertising system. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 27–36.

Ponsen, M. J.; Ramon, J.; Croonenborghs, T.; Driessens, K.; and Tuyls, K. 2008. Bayes-relational learning of opponent models from incomplete information in no-limit poker. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1485–1486.

Sandholm, T. 2010. The state of solving large incomplete-information games, and application to poker. *AI Magazine* 31(4):13–32.

Teófilo, L. F.; Passos, N.; Reis, L. P.; and Cardoso, H. L. 2012. Adapting strategies to opponent models in incomplete information games: a reinforcement learning approach for poker. In *Autonomous and Intelligent Systems*. 220–227.

Tu, H.-H., and Lin, H.-T. 2010. One-sided support vector regression for multiclass cost-sensitive classification. In *Proceedings of the 27th International Conference on Machine Learning*, 1095–1102.

Zhou, Z.-H., and Liu, X.-Y. 2010. On multi-class cost-sensitive learning. *Computational Intelligence* 26(3):232–257.