國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

機器學習於合約橋牌叫牌上之應用
Contract Bridge Bidding by Learning

何君彥

Chun-Yen Ho

指導教授：林軒田博士

Advisor: Hsuan-Tien Lin, Ph.D.

中華民國 103 年 7 月

July, 2014

國立臺灣大學碩士學位論文
口試委員會審定書

機器學習於合約橋牌叫牌上之應用

Contract Bridge Bidding by Learning

本論文係何君彥君（學號 R01922014）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 103 年 7 月 4 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

林軒田
（指導教授）

[illegible]　　[illegible]

系 主 任　許永真

誌謝

感謝林軒田老師這四年來在研究，學業，與其他各方面的指導與幫助。從起初支持我嘗試這個特別的題目，並在將近兩年的失敗中提出各樣的指引與鼓勵，使我最終能在這個題目上做出一些有趣的結果並完成這份論文。更感謝老師在為人處事上的身教與言教，這樣的典範使我在研究之外亦受益良多。

感謝口試委員李育杰老師與林守德老師百忙之中撥冗參加口試，並提出了許多進一步研究的方向。

感謝 CLLab 的所有同學們，在討論與提問中提出了許多好的建議。特別感謝李俊良同學在論文寫作與口試上提供的各項建議與幫助。

感謝我的家人，在各方面上支持並關心我，使我能沒有後顧之憂的完成這篇論文。

感謝神，在研究的過程中賜下豐富的恩典，在我的軟弱中重新加我力量，並在我人生的旅途中一直與我同在。

何君彥，2014 年 7 月

中文摘要

合約橋牌是一種具有不完全資訊特性的遊戲，電腦在此遊戲中通常無法勝過人類的橋牌專家。其中，人類橋牌玩家的叫牌決定對於電腦程式而言特別難以模仿，這使得自動化叫牌仍然是一個具挑戰性的研究問題。另一方面，使用不模仿人類玩家的方法進行自動化叫牌的可能性目前尚未被充份研究，在這篇論文中，我們在無競叫叫牌問題上首先探討使用此種方法的可能性。我們提出一個獨創的機器學習架構以使電腦程式學習自己的叫牌決定。在這個架構下，我們將叫牌問題轉換為機器學習問題，並精心設計一個基於成本導向分類器和信心值上界演算法的模型以解決此問題。我們以實驗驗證所提出的模型，並發現此模型與模仿人類玩家叫牌決定且多次贏得冠軍的電腦橋牌程式相較具有相當的競爭力。

關鍵字: 機器學習, 合約橋牌, 情境式拉霸問題, 信心值上界, 成本導向分類器.

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.

Keywords: Machine Learning, Contract Bridge, Contextual Bandit Problem, Upper-Confidence Bound, Cost-Sensitive Classification.

Contents

List of Figures

List of Tables

Chapter 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. Currently, there are computer programs for some of these games that can outperform human experts. On the other hand, a more challenging class of incomplete information games such as poker and bridge continues to be of research interests [1]. A popular research direction is to exploit machine learning to analyze data in an effort to find better game-playing strategies [2–4].

Contract bridge, or simply *bridge*, is an example of 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 her/his 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 using a standard set of rules, commencing with the player next to the declarer playing the first card on the table. 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.

Previous research that attempted to implement a computer bridge player using AI suggested that the bidding stage is more difficult to implement than the playing stage. For example, in 1998, the GIB program [5] attained 12th place among 35 human experts in a par contest, which is a contest without bidding [6]. 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 strategies from human players directly during the bidding stage. The strategies of human players are commonly called a bidding system, 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 [7]. By first constructing a decision network from a bidding system, [8] proposes 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 [9], 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 completely 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 thesis, 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. Next, 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 that the performance of the model is competitive to the contemporary champion-winning computer bridge software that implements a human-designed bidding system.

The remainder of this thesis is organized as follows. In Chapter 2, we introduce the assumptions and formally define the bridge bidding problem. In Chapter 3, we illustrate the proposed learning model. Finally, we present the experimental results in Chapter 4 and conclude our study in Chapter 5.

Chapter 2

Problem Setup

As discussed in Chapter 1, the general bidding problem allows competition between two opposing teams. That is, each team can overcall the bid made by the opponents in order to obtain the contract it hopes to play, and/or obstruct the opponents' communication. Then, we can separate the general bidding problem into two sub-problems: 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 [8] and [9], and study the sub-problem of bidding without competition in this work.

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 $\mathbf{b}[k] \neq$ PASS for $k > 1$. Note that the $\mathbf{x}$ used in $g(\mathbf{x}, \mathbf{b}^k)$ for every pair of consecutive bids need to originate from different players. Then, by assuming without loss of generality that the first player is always the North player, we have $\mathbf{x}_k = \mathbf{x}_n$ for odd k and $\mathbf{x}_k = \mathbf{x}_s$ otherwise. Under the decomposition, the bidding process ends with exactly three consecutive PASSes if we consider the opponents.

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 relate 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. This would prohibit us from scaling the data, which is important for learning a better bidding strategy. To overcome this issue, we use the double dummy analysis [10] 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. Provided that the contract and the declarer are given, the solution becomes unique. Second, the analysis is fast. Whereas using computer agents to play every contract of a deal once might require more than several hours, 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 $\mathbf{b}_i = G(\mathbf{x}_{ni}, \mathbf{x}_{si})$.

Chapter 3

Proposed Model

3.1 Baseline and Optimistic Methods

First, we consider the bidding strategies G that only predict sequences $\mathbf{b}$ of length one. That is, we let $g(\mathbf{x}_s, \mathbf{b}^1) = \textsc{pass}$. 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 [11], with many existing algorithms available [12, 13]. Here, we consider two regression-based algorithms, cost-sensitive two-sided regression (CSTSR) and cost-sensitive one-sided regression (CSOSR) [13], 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 rather than the player. That is, we merge $\mathbf{x}_n$ and $\mathbf{x}_s$ as a feature, and use $D_{cheat} = \{([\mathbf{x}_{ni}, \mathbf{x}_{si}], \mathbf{c}_i)\}_{i=1}^{N}$ to learn a CSC classifier. Such an optimistic method solves a relaxed (unrealistic) version of the bidding problem. Therefore, the performance of this unrealistic classifier hints an upper bound that can be achieved by machine learning with the particular algorithms.

3.2 The Difficulty of Learning to Bid

Next, we use the bidding sequence for better information exchange and better performance. We start by analyzing the difficulty of learning to bid. To learn the bidding function $g(\mathbf{x}, \mathbf{b}^k) \mapsto \mathcal{B}$, we ideally need to have examples that consist of the tuple $((\mathbf{x}, \mathbf{b}^k), \mathbf{b}[k+1], c)$, where c is the cost of bidding $\mathbf{b}[k+1]$ given $\mathbf{x}$ and $\mathbf{b}^k$. However, we only have $\mathbf{x}$ in D, the "good" bidding sequences $\mathbf{b}$ are mostly missing, and the exact c hides in the cost vector $\mathbf{c}$.

For instance, let us consider extending the baseline method 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 $g(\mathbf{x}_n, \mathbf{b}^2) = \text{PASS}$. For learning $g(\mathbf{x}_s, \mathbf{b}^1)$, we readily have $\mathbf{x}_{si}$ in D, and we can enumerate all possible $\mathbf{b}_i[2]$ to get the associated c_i. Thus, except for the missing variables $\mathbf{b}_i^1$, the learning of $g(\mathbf{x}_s, \mathbf{b}^1)$ can be tackled by solving a CSC problem like the baseline method. So the "only" task is to generate the missing variables $\mathbf{b}_i^1$. However, the task is quite difficult. In particular, for learning $g(\mathbf{x}_n, \varnothing)$, both $\mathbf{b}_i[1]$ (the desired bids) and c_i (the cost of bidding $\mathbf{b}_i[1]$) are missing. That is, the task is an unusual unsupervised learning problem that requires clustering $\mathbf{x}_{ni}$ to the possible bids $\mathbf{b}_i[1]$ to indirectly help the teammate's classifier $g(\mathbf{x}_s, \mathbf{b}^1)$. Even if we manage to successfully cluster $\mathbf{x}_{ni}$, it would be difficult to assign a bidding label from $\mathcal{B}$ to each cluster.

We can have some insight on the challenging task above by considering what two human bridge players will do in this scenario. For simplicity, we will call $g(\mathbf{x}_n, \varnothing)$ as g_n, and $g(\mathbf{x}_s, \mathbf{b}^1)$ 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, the North player can improve his strategy g_n 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 the discussions above. We maintain two important aspects. 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 [14–16]. In this problem, we hope to earn the maximum total reward 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 [15] are some of the most popular contextual bandit algorithms. They 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.

3.3 The Multi-Layer Bandit Model

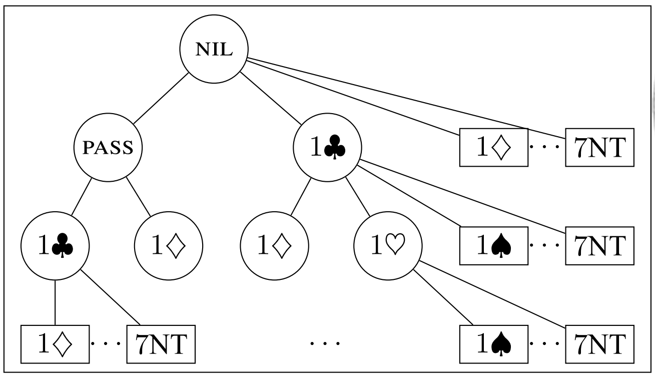
Whereas each player's decision making can be modelled by the contextual bandit problem, recall that our goal is to obtain a bidding strategy G that produces a sequence of decisions that satisfy bridge rules. We propose to represent each player's decision making with layers of "bidding nodes" V. With 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 $b = \textsc{nil}$ indicating the entering of the bidding stage. At each V, g is only allowed to predict $\textsc{pass}$ 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 $\textsc{pass}$. In addition, all nodes at layer $\ell + 1$ are terminal nodes.

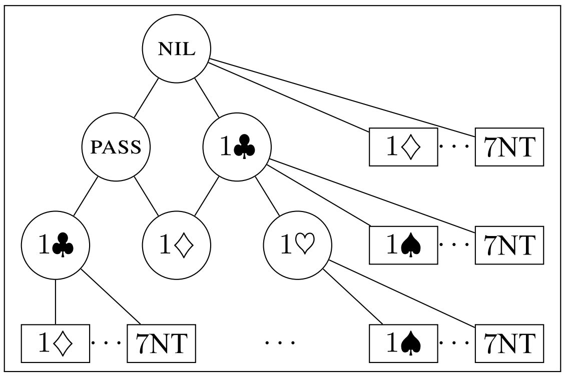
Since we form the nodes as a tree, each unique path from the root to V readily represents a bidding sequence $\mathbf{b}^k$. Thus, the classifier g of V only needs to consider the cards $\mathbf{x}$. We call such a structure the tree model, as illustrated in Figure 3.1(a) with $\ell = 3$ and $M = 2$. A variant of the tree model can be performed by 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 3.1(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 $\textsc{pass}$ 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



(a) Tree model with $\ell = 3$ and $M = 2$



(b) Layered model with $\ell = 3$ and $M = 2$

Figure 3.1: Tree model and layered model, the terminal nodes are not fully drawn

baseline methods that we have studied, 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 is used to compute a weight vector $\mathbf{w}_m$ for estimating the potential reward $\mathbf{w}_m^T\mathbf{x}$ of making the bid. During prediction, CSTSR predicts with the bid associated with the maximum potential reward. The computation of $\mathbf{w}_m$ takes

$$\mathbf{w}_m = (\mathbf{X}_m^T\mathbf{X}_m + \lambda I)^{-1}(\mathbf{X}_m^T\mathbf{r}_m),$$

where $\mathbf{r}_m$ contains all the rewards gathered when the m-th bid b_m is made by g and $\mathbf{X}_m$ contains all the $\mathbf{x}$ associated with those rewards. $\lambda > 0$ is the regularization parameter of ridge regression and I is the identity matrix.

Our final task is to describe the learning algorithm for the model structure with ridge regression. As discussed, we use the cost of the final contract (i.e., the last bid) to form

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 $\mathbf{w}_m$.
 1: repeat
 2:     Randomly select a data instance $(\mathbf{x}_{ni}, \mathbf{x}_{si}, \mathbf{c}_i)$
 3:     Set $V$ to the root node of the structure, and $\mathbf{x}$ to $\mathbf{x}_{ni}$
 4:     UPDATE($V$, $\mathbf{x}_{ni}$, i)
 5: until enough training iterations
 6: procedure UPDATE($V$, $\mathbf{x}$, $i$)
 7:     Compute the UCB reward $\mathbf{w}_m^T\mathbf{x} + \alpha \cdot$ uncertainty term for each possible $b_m$
 8:     Select the bid $b_m$ with the maximum UCB reward
 9:     if $b_m =$ PASS and $V$ is not root then
10:         Compute reward $r$ from $\mathbf{c}_i$ using $b_m$.
11:     else
12:         Set $\mathbf{x}$ to the feature of the other player, and call $r$ = UPDATE($V'$, $\mathbf{x}$, $i$).
13:     end if
14:     Update $\mathbf{w}_m$ with $(\mathbf{x}, r)$
15:     return $r$.
16: end procedure
```

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 [15] 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 [17], 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 $\mathbf{w}_m$.

The full algorithm is illustrated in Algorithm 1. We randomly select an instance $\mathbf{x}$ per iteration to satisfy the online nature of the UCB algorithms. Then, a bidding sequence is generated with either a series of LinUCB or UCB1 computations. Finally, all the nodes on the bidding sequence path are updated with the calculated rewards.

The uncertainty term is the key component for making the UCB algorithms work. First, we initialize all $\mathbf{w}_m$ with zeroes, and the uncertainty term is equally large for all possible bids. Therefore, the algorithm distributes instances to different bidding sequences somewhat randomly. Then, the uncertainty term decreases gradually after seeing more examples, which allows the reward term $\mathbf{w}_m^T\mathbf{x}$ to dominate the decision process. This allows the algorithm to focus on rewarding bidding sequences to fine-tune the bidding decisions.

3.4 Additional Techniques

In addition to the model and the core algorithms introduced in the previous section, we adopt several additional techniques to improve performance and computational efficiency. The first two techniques focus on improving the performance, and the last technique aims at improving computational efficiency.

Full Update. In the proposed model, whenever a bidding sequence $\mathbf{b}$ is sampled from the UCB algorithms for an instance $\mathbf{x}$, 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 can form additional examples by considering all the decisions of which reward can be calculated based on the above analysis for each bidding node on the bidding sequence $\mathbf{b}$, and include those examples in updating the associated bidding nodes. Such an update scheme is called

FULL UPDATE as opposed to the original SINGLE UPDATE scheme in the proposed model.

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. Thus, when UCB puts the top-layer classifiers in the exploitation stage, the bottom-layer classifiers may still be in the exploration stage. Even worse, 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 not on a terminal node (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) bidding sequences $\mathbf{b}$ to help update the lower layers of the model in this PENETRATIVE UPDATE scheme. The scheme is related to the famous epsilon-greedy algorithm for the contextual bandit problem [18].

Delayed Update. We adopt the contextual bandit algorithms in our model, which were designed for the online scenario where examples arrive one by one. Even with the Sherman-Morrison formula, updating the internal $\mathbf{w}_m$ right after an example arrives requires $O(d^2)$, where d is the dimension of $\mathbf{x}$. The updating step becomes the computational bottleneck of the algorithms. In view of the efficiency, we consider a DELAYED UPDATE scheme that does not update $\mathbf{w}_m$ immediately after each example is formed, but waits until gathering a pile of examples. Experimental results in Chapter 4 will show that such a scheme substantially decreases the amount of training time without loss of performance.

Chapter 4

Experiments

Next, we study the proposed model and compare it with the baseline and optimistic methods. In addition, we compare the model with a well-known computer bridge software, Wbridge5 [19], 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 study two different representations for $\mathbf{x}$: binary features and condensed features. The binary features are represented by a 52-dimensional binary vector, where each dimension representing the existence of the corresponding card. The condensed features contain two parts that are widely used in real-world bridge games and human-designed bidding systems, high card points (HCP) and number of cards in each suit. The HCP is a method for evaluating the round-winning power. It is calculated by summing up the values of cards, which is defined by Ace = 4, King = 3, Queen = 2, Jack = 1, and 0 otherwise. For both representations, a constant dimension is added to reflect the bias term.

We obtain the cost vectors $\mathbf{c}$ from International Match Points (IMP). The IMP is an integer between $\{0, 1, \cdots, 24\}$, widely used for comparing the relative performance of two teams in real-world bridge game [20]. We obtain $\mathbf{c}$ by comparing the best possible contract of the deal to each contract and calculate the IMP, where higher IMP indicates that the contract is far from the best one and should suffer from a higher cost. When transforming the costs to the rewards in the proposed model, we take 24 minus the cost as

Table 4.1: Results of baseline and optimistic methods

Method	Dimensions	Baseline	Optimistic
CSOSR - binary	53	3.9659	2.5657
CSOSR - condensed	6	3.8329	1.8985
CSTSR - binary	53	3.9399	2.7270
CSTSR - condensed	6	3.9428	2.7697
CSTSR - condensed + 2nd order expansion	21	3.8465	2.1106
CSTSR - condensed + 3rd order expansion	56	3.8272	1.9228
Wbridge5	N/A	2.9550	N/A

the reward to keep the rewards non-negative. [1]

4.1 Baseline and Optimistic Methods

First, we present the performance of the baseline and the optimistic methods in Table 4.1. In CSOSR, SVM with the Gaussian kernel implemented with LIBSVM [21] is used as the base learner. In CSTSR, ridge regression is used as the base learner. Because SVM training is time consuming, we only sub-sample $20,000$ instances for CSOSR. For CSTSR with condensed features, we also extend its capability by considering simple polynomial expansion of the features. 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 4.1.

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 have room for improvement. 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.

[1]One technical detail is that the cost vector $\mathbf{c}$ is generated by assuming that the player who can win more rounds for the contract is the declarer. We will discuss the effect in the end of this chapter.

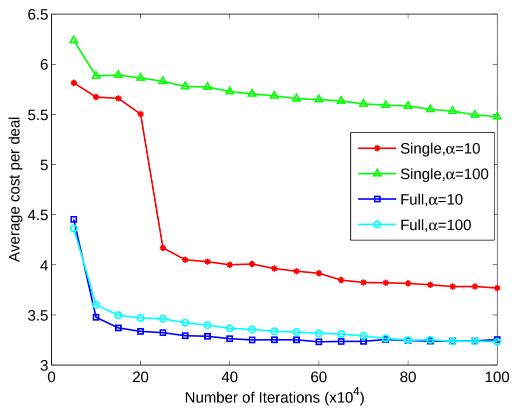
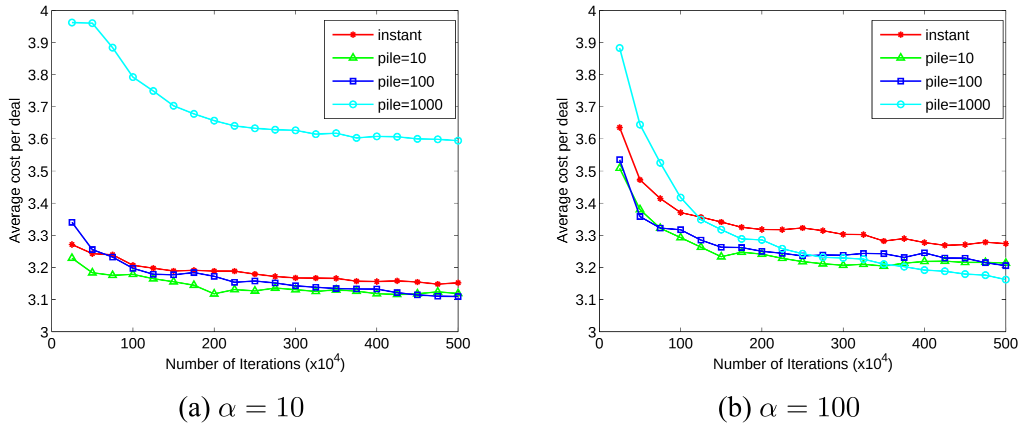


Figure 4.1: Full Update versus Single Update



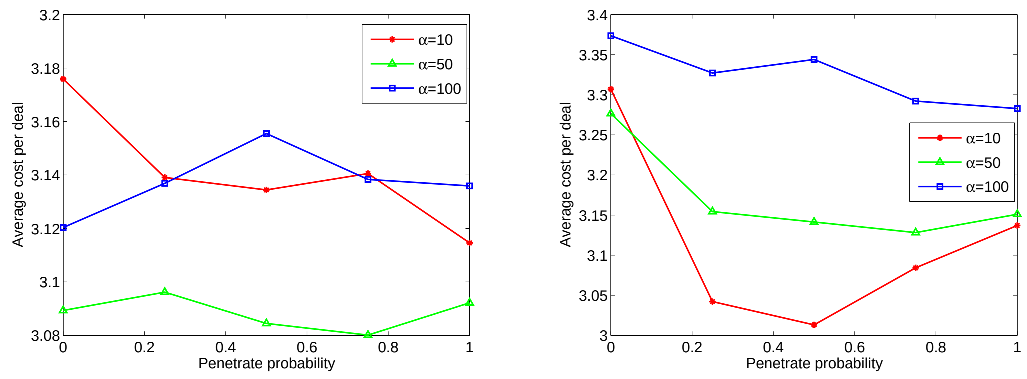
(a) $\alpha = 10$ (b) $\alpha = 100$

Figure 4.2: Delayed Update versus Instant Update

4.2 Effect of Applying Techniques

In Section 3.4, three techniques are proposed to improve the model. We first compare Full Update with Single Update. Figure 4.1 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 $\alpha \in \{10, 100\}$. We can easily observe that Full Update outperforms Single Update, which justifies that the additional examples used for Full Update capture valuable information to make the cost estimation more precise. Thus, we adopt Full Update in all the next experiments.

Then, we compare Delayed Update with Instant Update. Figure 4.2 shows how the average validation cost varies with the number of iterations on the same tree model used for Figure 4.1. For Delayed Update, we consider piles of size $\{10, 100, 1000\}$ instances



(a) $\ell = 4$ (b) $\ell = 6$

Figure 4.3: Penetrative Update with different p

per update. We find that when α is small, Instant Update or a small pile reaches the best performance, whereas larger α could use a larger pile. Overall, Instant Update not only fails to reach the best performance, but is also quite inefficient, as shown in the table of approximation training time below.

instant	pile = 10	pile = 100	pile = 1000
4hours	2hours	60mins	50mins

In view of the efficiency needed for extensive parameter selection, we decide to take Delayed Update with piles of size 100 in the next experiments.

Finally, Figure 4.3 shows the average validation cost when varying different penetration probability p on tree models with $M = 5$ coupled with ridge regression with $\lambda = 10^{-3}$ and UCB1 with $\alpha \in \{10, 50, 100\}$. We can see that a non-zero p (actual Penetrative Update) works well for small α and large ℓ. However, the benefit of Penetrative Update is less obvious when α is large. This is because when a larger α is used, the UCB term readily allows more instances to go to the next layer through exploring different bids, and therefore Penetrative Update does not have many early pass calls to avoid. Overall, we cannot find a fixed penetration probability p to work with different α and ℓ values. Thus, we decide to fine-tune p along with α and ℓ in the next experiments.

Table 4.2: Average Cost Using Different Model Structures

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	$\mathbf{3.0527}$	$\mathbf{2.9550}$

4.3 Comparison on 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 are equivalent when $\ell = 2$.

Table 4.2 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.

Several additional observations can also be found from Table 4.2. First, all the proposed models in Table 4.2 perform better than the baseline methods in Table 4.1. The results justify that the proposed models successfully make use of the bidding sequence for information exchanging between teammates. Second, the model structure can affect the choice of the UCB algorithm. The tree model generally works better with UCB1, while the layered model matches LinUCB better. This suggests a future research direction using other UCB algorithms to improve the performance. Third, the tree model, with its higher model complexity, generally performs better than the layered model. Nevertheless, similar to what is discussed during PENETRATIVE UPDATE, for models with higher complexity

Table 4.3: Comparison with Wbridge5 by the type of contract

(a) By contracts from the proposed model

Type	Difference	Number of Deals
PASS	1205	1907
PARTIAL	1506	5900
GAME	-1878	2131
SLAM	-164	61
GRAND SLAM	11	1

(b) By contracts from Wbridge5

Type	Difference	Number of Deals
PASS	-12	2116
PARTIAL	4205	4779
GAME	-1607	2670
SLAM	-1612	406
GRAND SLAM	-294	29

(such as a tree model or models with larger ℓ), it can be difficult for some nodes to obtain sufficient data for learning. Designing complex models while providing each node sufficient data is yet another future research direction.

4.4 Comparison with Wbridge5

Table 4.2 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 really high scores, human players (and hence human bidding systems) often prefer bidding towards those. Table 4.3 show the total cost of Wbridge5 minus the total cost of the proposed model in each category. 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 (the majority of the deals) 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.

Table 4.4: Effect of Using Real Declarer

	With assumed declarer	With real declarer
Best bidding model	2.8870	2.9435
Wbridge5	2.9550	3.0314

4.5 Effect of Using Real Declarer

In the previous experiments, we assume for simplicity that the player who can win more rounds for the contract is the declarer when we generate the cost vector **c**. This is different from the setting of a real bridge game, where the player from the bid-winning team who called the trump suit first become the declarer. Table 4.4 shows the average cost per deal of the best proposed model and the Wbridge5 software when we use the real declarer. We can observe that the performance of the proposed model and the Wbridge5 software decrease about the same amount. This shows that the effect of the declarer is minor under our problem setting.

We think that there are two reasons for making declarer less important. First, since the declarer only influence the player who play the first card, most deals in the data are declarer independent. That is, the North and the South players usually win the same number of rounds for most of the contracts in a deal. Second, whereas the information revealed to the opponent team varies for different declarers in a real bridge game, the double dummy analysis is based on perfect information and thus not influenced.

Chapter 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.

As an initiative of bidding by learning, the proposed model has reached promising performance. One possible direction on improving the model is to use more data to train a deeper model, which hopefully improve the performance of the 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 may call for a mixture of the proposed model (collaboration between teammates) and well-studied models for competition-based games such as Chess.

Appendix A

Table of Opening Bids

Table A.1 compare the opening bids of the best tree model with $\ell = 4$ and $\ell = 6$ with the SAYC bidding system [22], which is widely used by human players. The opening bids of the proposed model is generated by enumerating and predicting for all the combinations of features. As the prediction of the proposed model is made by CSC classifiers, there is no explicit rule for each opening bid. Instead, an approximate rule is provided in the table.

Several observations can be made from Table A.1. First, the opening rules of the proposed model is very different from the SAYC bidding system. This shows that the bidding methods learned by computer may be dissimilar to a human designed one. Second, whereas the terminal opening bids ($\{1\text{NT}, \cdots\}$) of the two tree models are similar, the non-terminal opening bids ($\{\text{PASS}, \cdots, 1\spadesuit\}$) are completely different. This shows a property of the proposed model. For terminal bids, a deterministic estimation of the reward can be generated from the cost vector $\mathbf{c}$, thus the corresponding CSC classifiers learned each time are similar. On the other hand, there is a randomness in the learning process of the non-terminal bids, thus the CSC classifiers learned each time could be very different. Third, the "Not used" bids in the proposed model show that the bidding process is not fully utilized in the proposed model. There is still a room for improvement if we can further enhance the information exchanging process.

Table A.1: Table of Opening Bids

Bid	Tree model, $\ell = 4$	Tree model, $\ell = 6$	SAYC
PASS	0-11 HCP	0-12 HCP	0-11 HCP
1♣	10-19 HCP, no many ♡	9-19 HCP, 4-6 ♡	12+ HCP, 3+♣
1♢	Not Used	8-18 HCP, short ♠ and 4-6 ♣	12+ HCP, 3+♢
1♡	9-19 HCP, 4-6 ♡	12-23 HCP, w/o long suit	12+ HCP, 5+♡
1♠	16-23 HCP, near balanced	10-19 HCP, 4-6 ♠	12+ HCP, 5+♠
1NT	Not used	Not used	15-17 HCP, Balanced
2♣	0-17 HCP, long ♣	0-17 HCP, long ♣	22+ HCP
2♢	0-17 HCP, long ♢	0-17 HCP, long ♢	5-11 HCP, 6+♢
2♡	0-13 HCP, long ♡	0-13 HCP, long ♡	5-11 HCP, 6+♡
2♠	0-13 HCP, long ♠	0-13 HCP, long ♠	5-11 HCP, 6+♠
2NT	Not used	Not used	20-21 HCP, balanced
3♣	14-19 HCP, long ♣	15-19 HCP, long ♣	5-11 HCP, 7+♣
3♢	14-19 HCP, long ♢	15-19 HCP, long ♢	5-11 HCP, 7+♢
3♡	Not used	Not used	5-11 HCP, 7+♡
3♠	Not used	Not used	5-11 HCP, 7+♠
3NT	19-29 HCP, w/o a long suit	19-29 HCP, w/o a long suit	25-27 HCP, balanced
4♣	Not used	Not used	5-11 HCP, 8+♣
4♢	Not used	Not used	5-11 HCP, 8+♢
4♡	10-29 HCP, long ♡	11-29 HCP, long ♡	8+♡
4♠	10-29 HCP, long ♠	11-29 HCP, long ♠	8+♠
4NT	27-29 HCP, near balanced	27-29 HCP, near balanced	Not used
5♣	16-27 HCP, long ♣	16-27 HCP, long ♣	very long ♣
5♢	17-25 HCP, long ♢	17-25 HCP, long ♢	very long ♢

Bibliography

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

[2] Michael Bowling, Johannes Fürnkranz, Thore Graepel, and Ron Musick. Machine learning and games. *Machine learning*, 63(3):211–215, 2006.

[3] Marc JV Ponsen, Jan Ramon, Tom Croonenborghs, Kurt Driessens, and Karl Tuyls. Bayes-relational learning of opponent models from incomplete information in no-limit poker. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1485–1486, 2008.

[4] Luís Filipe Teófilo, Nuno Passos, Luís Paulo Reis, and Henrique Lopes Cardoso. Adapting strategies to opponent models in incomplete information games: a reinforcement learning approach for poker. In *Autonomous and Intelligent Systems*, pages 220–227. 2012.

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

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

[7] Takahisa Ando and Takao Uehara. Reasoning by agents in computer bridge bidding. In *Computers and Games*, pages 346–364. 2001.

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

[9] Lori L DeLooze and James Downey. Bridge bidding with imperfect information. In *IEEE Symposium on Computational Intelligence and Games*, pages 368–373. IEEE, 2007.

[10] Ming-Sheng Chang. Building a fast double-dummy bridge solver. 1996.

[11] Alina Beygelzimer, Varsha Dani, Tom Hayes, John Langford, and Bianca Zadrozny. Error limiting reductions between classification tasks. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 49–56, 2005.

[12] Zhi-Hua Zhou and Xu-Ying Liu. On multi-class cost-sensitive learning. *Computational Intelligence*, 26(3):232–257, 2010.

[13] Han-Hsing Tu and Hsuan-Tien Lin. One-sided support vector regression for multiclass cost-sensitive classification. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1095–1102, 2010.

[14] Wei Li, Xuerui Wang, Ruofei Zhang, Ying Cui, Jianchang Mao, and Rong Jin. 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*, pages 27–36, 2010.

[15] Wei Chu, Lihong Li, Lev Reyzin, and Robert E Schapire. Contextual bandits with linear payoff functions. In *International Conference on Artificial Intelligence and Statistics*, pages 208–214, 2011.

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

[17] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[18] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in Neural Information Processing Systems 20*, pages 817–824, 2008.

[19] Yves Costel. Wbridge5 bridge software, 2014. URL: http://www.wbridge5.com/.

[20] Introduction to bridge scoring, 2005. URL: http://www.acbl.org/learn/scoreTeams.html.

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

[22] Standard american, 2014. URL: http://en.wikipedia.org/wiki/Standard_American.