Learning to Bid in Bridge

Asaf Amit ASAF@CS.TECHNION.AC.IL
Shaul Markovitch SHAULM@CS.TECHNION.AC.IL
Department of Computer Science
Technion—Israel Institute of Technology
Technion City, Haifa 32000, Israel

Abstract

Bridge bidding is considered to be one of the most difficult problems for game-playing programs. It involves four agents rather than two, including a cooperative agent. In addition, the partial observability of the game makes it impossible to predict the outcome of each action. In this paper we present a new decision-making algorithm that is capable of overcoming these problems. The algorithm allows models to be used for both opponent agents and partners, while utilizing a novel model-based Monte Carlo sampling method to overcome the problem of hidden information. The paper also presents a learning framework that uses the above decision-making algorithm for co-training of partners. The agents refine their selection strategies during training and continuously exchange their refined strategies. The refinement is based on inductive learning applied to examples accumulated for classes of states with conflicting actions. The algorithm was empirically evaluated on a set of bridge deals. The pair of agents that co-trained significantly improved their bidding performance to a level surpassing that of the current state-of-the-art bidding algorithm.

1. Introduction

Game-playing has long been used by artificial intelligence researchers as a domain for studying decision-making in competitive multi-agent environments. The vast majority of researchers have dealt with full information two-player games such as chess or checkers. Such games present two challenges: that of decision making in very large search spaces when resources are limited, and that of dealing with an opponent agent. Other games such as poker present the additional challenge of decision making with partial information.

One of the most interesting games for multi-agent reasearchers is bridge. It presents all the above problems with the addition of both competitive and cooperative agents.[1] Furthermore, the communication between the cooperative agents is restricted.

The game of bridge consists of two parts, bidding and playing. The playing stage proved to be relatively easy for computers, and many programs have shown good playing performance. Examples include Bridge Baron (Smith, Nau, & Throop, 1998a), GIB (Ginsberg, 2001), and Finesse (Frank, 1998). In 1998, GIB attained 12^{th} place among 35 human experts in a par contest[2](Ekkes, 1998).

The problem of bidding, however, proved to be much harder. Ginsberg (2001) admits that "...the weakest part of GIB's game is bidding." Bidding is probably the weakest part of most existing bridge programs.

1. Henceforth, we will call a cooperative agent a *co-agent* and an opponent agent an *opp-agent*.
2. A *par contest* is a competition where the players are given an auction and compete only in playing.

The goal of the research presented in this paper is to develop a decision-making algorithm that is suitable for bridge bidding. The most common decision-making algorithm used for resource-bounded two-player full-information games is limited-depth minimax with various extensions such as pruning, selective deepening, and transposition tables. There are, however, several properties of bridge bidding that make minimax unsuitable for this problem:

1. It involves four agents rather than two.
2. It involves a cooperative agent.
   (a) If we don't have any knowledge about the co-agent, we don't know what type of decision to make in the search nodes associated with it. Neither maximum nor minimum seems appropriate.
   (b) In the likely event that the co-agent's strategy is familiar to us, it is not clear how to incorporate it into the search.
3. Because of the partial information available to the agent, there is uncertainty about the utility of search tree leaves.
4. It is extremely difficult to evaluate intermediate states in the bidding stage. Therefore the agent needs to search down to the leaves of the game tree.
5. Bidding is based on a bidding system – a set of bidding rules that the two partners agree on. Each bid serves two roles: it should lead to a contract that is makable and it should be used for information exchange between partners (based on the bidding system). This dual role and the dependency on the bidding system makes bidding extremely difficult.

We begin with a brief background on bridge bidding. We follow with a discussion of the above problems and present the PIDM (Partial Information Decision Making) algorithm which solves them. The algorithm allows models to be used for both opp-agents and co-agents and uses a novel model-based Monte Carlo sampling method to overcome the problem of hidden information. We then present a learning algorithm that reduces the branching factor of the search tree, thus allowing deeper and faster search. One of the problems with offline learning of cooperative agents is the circularity in the adaptation process. Each agent needs to know the current strategy of its co-agent in order to adapt to it, but the strategies of both agents are constantly changing. We present a co-training algorithm where the co-agents periodically exchange their current strategy and continue to train. Finally, we evaluate our algorithms in the bridge bidding domain and show that a pair of agents significantly improves its bidding performance after co-training.

The contributions of the papers are:

1. We introduce a method for partial modeling of opponents and partners. Instead of trying to learn to predict what action the other agent will take, we predict the "reasonable" actions it might consider.
2. We present a new algorithm for model-based decision making in partially observable environments.

3. We present a new methodology for co-training of cooperative agents.

4. We present a state-of-the-art bridge bidding algorithm that is able to achieve, after learning, a bidding ability close to that of a human expert.

2. Background: Bridge Bidding

In this section we first give a brief introduction to the rules of bridge, and then discuss the difficulty in building an algorithm for bridge bidding.

2.1 The Rules of Bridge

The game of bridge is played by four players commonly referred to as North, South, East, and West, playing in two opposing partnerships, with North and South playing against East as West. The game is played using a standard deck of 52 cards, divided into 4 suits, each containing 13 cards from the A down to 2.

Every round (called *a deal*) in bridge is divided into two phases: the auction and the play. Each player is given 13 cards and the player designated as the dealer opens the auction. Each player can bid in its turn and suggest a trump suit and the level of the contract (the number of tricks it promises to take). Some of the bids are called conventions and used for interaction between the partners. If a player does not wish to make a higher bid, it can double an opponent bid, redouble an opponent double, or pass. The auction ends when a call is followed by three pass bids.

The final bid becomes the contract and its trump suit becomes the trump suit of the deal. The player from the highest bid team who called this trump suit first becomes the declarer, its partner the dummy, and their opponents the defenders.

After the auction the play starts. The play consists of 13 rounds (tricks). During each round, each of the players puts one card on the table. The play starts with the player that sits next to the declarer playing its first card and the dummy exposing its cards to everyone. During the play, the declarer plays both his cards and the dummys. Each of the 13 tricks is won by one partnership. The player who wins the trick has the right to lead for the next trick. A player must follow the led suit if possible or play another suit. The highest card in the led suit (or in the trump suit if played) wins the trick.

The scoring depends on the number of tricks taken by the declarer and the final contract. If the declarer takes the number of tricks he committed to, his side gets a score and the defender gets minus that score, otherwise the positive score is given to the defenders. The players are encouraged to commit to a greater number of tricks through bonuses that are awarded on a "game" (9-11 tricks) and a "slam" (12-13 tricks). However, they might face a negative score if they fail, even if they took most of the tricks. In most bridge tournaments, the score is compared to other scores achieved with the same cards at the other tables. There are two main methods for computing the final result of a game. The IMP method computes the arithmetic difference between the scores and converts it using a conversion table. The MP method gives 2 points for each score worse than the pair's score, 1 point for each equal score, and 0 points for each better score.

2.2 Problems in Bridge Bidding

The auction is considered to be the hardest and most important part of the game. During human world championships, there is little variation in the level of the players during card playing, making the quality of the bidding the decisive factor in the game. The final game of the 2000 world championship is a good example. During the 128-deal match between Italy and Poland, 66 deals led to a more-than-one-point swing[3]. 54 of the swings can be attributed to superior bidding performance, while 12 were due to better playing.

During the auction stage the players, in turn, make a call. Players can see only their own 13 cards, but not the additional 39 cards dealt to the others. Players decide which call (out of the 20-30 possible calls) to make on the basis of their own cards, and on the basis of the history of calls by the other players. The process of making calls continues until a call is followed by three "pass" calls.

During the bidding stage, it is particularly difficult for a bidding algorithm to decide which calls to make. First, it is impossible to evaluate the leaves of the game tree because only partial information about states is available. There are $6.35 * 10^{11}$ possible hands (of 13 cards). Each of these possible partial states can be completed to a full state in $8.45 * 10^{16}$ ways. The only information that can be used to infer the complete state is the history of calls by other agents.

It is very difficult to infer a full state from a partial state. First, each call can carry one or more of the following meanings:

1. Suggesting an optional contract.
2. Exchanging information between partners.
3. Interfering with opponents.

A different intended meaning would lead to a different inference for the same call. To be able to understand each other, the partners maintain a list of agreements about the meaning of calls called a *bidding system*. Because it is impossible to store a table of all 10^{47} possible call sequences, a bidding system is expressed by a set of rules where the condition specifies a pattern over the current auction and a pattern over the current hand, and the action is a call[4].

The main problem with using bidding systems is their ambiguity. When a player looks for a rule that matches the current state in order to choose a call, it is very likely that more than one rule with conflicting actions will be found. When a player tries to infer her partner's hand based on her calls, the set of possible hands can be too large to be useful.

To reduce this ambiguity and allow better understanding between partners, human bridge players usually devote much time to practicing together. Books written by multiple world champions Belladonna and Garozzo (1975), and by Goldman (1978), are dedicated to getting partners to understand each other better during the bidding stage. These books almost entirely ignore the playing stage. In their introductions, the writers mention the substantial time they spend practicing with their partners in order to better understand their actions.

3. This means that one of the teams won a significant number of points for the deal.
4. For an example of a common bidding system see `http://www.annam.co.uk/2-1.html`.

Even when the actual meanings of a sequence of calls are known, it is very difficult to infer three hands that are consistent with it. Checking only the constraints inferred from the auction is not enough. Often the possible calls do not describe disjoint sets of hands. In this case, we must subtract from the set the possible hands that could be described by higher priority calls. In addition, we should consider the fact that a player can take an action that conflicts with the inferred constraints.

Another problem in bidding is the lack of indication of progress in the middle stages. The expected utility of an auction is determined only after the auction's last bid, and there are no obvious features of a partial auction that can serve as indicators of its quality (as material advantage does in chess).

Most bridge bidding programs make decisions using a rule-based approach (for example, (Carley, 1962; Wasserman, 1970; Lindelöf, 1983). The conditions of a rule consist of auction patterns and constraints on the possible hands of the other players, and the action is the call that should be made.

Size limitations, as well as the large number of states that a rule base should cover, prevent it from working perfectly in any situation. Sometimes two or more rules with conflicting actions may match the current state. There may be other states that match no rules. To solve such problems, a combined rule-based system and lookahead search was suggested. This approach was used by Gambäck, Rayner, and Pell (1993) and by Ginsberg (2001). The lookahead approach requires a function for evaluating the leaves of the lookahead tree. Ginsberg implemented such an evaluator in his GIB program.

The lookahead approach can help to rectify the problem of incomplete rule bases. However, when an agent acts under time limits, as is the case in bridge playing, the possibility of performing lookahead is quite constrained. In the following sections we present a learning methodology that can help us to overcome these problems.

3. The PIDM Algorithm

In Section 1 we listed several problems that make minimax inappropriate for bridge bidding. Here, we discuss each of these difficulties and consider alternatives for overcoming them. The result of this discussion is our novel PIDM algorithm which generalizes previous approaches to allow decision making in partial-information games with both cooperative and competitive agents such as bridge.

3.1 Dealing with Incomplete Information

Bridge is a game with incomplete information, meaning that each agent has access to only part of the current state. More formally, let $V_1, \ldots, V_n$ be a set of *state variables*. The set of *legal states* is

$$S = \{\langle v_1, \ldots, v_n \rangle | v_i \in D_i, C(v_1, \ldots, v_n)\},$$

where D_i is the domain of variable V_i and C is a set of constraints. A *full state* is a member of S. A *partial state* is a state in S with some of its values replaced by ?. Let $A_1, \ldots, A_n$ be a set of agents. We denote the partial state accessible to agent A_i out of a full state s as $PS(s, i)$.

The full state of a bridge game during the auction stage contains (1) the distribution of cards between the four players (the deal),(2) the current sequence of calls for the auction, the identity of the dealer, and (3) the *vulnerability* of each of the pairs and the bidding system of both pairs. During the auction stage, only the sequence of calls changes while the deal and the bidding systems remain unchanged. The set of constraints, C, requires that each hand contain 13 cards and that the union of the 4 hands be a legal deck.

During the auction, each player is exposed to only part of the information – the hands of the other three players remain unknown at least until the end of the auction stage. Using the above notation, a partial state accessible to agent *North* may look like:

$$PS(s, N) = \langle(\spadesuit AQJ93\ \heartsuit AKJ6\ \diamondsuit J65\ \clubsuit 2), ?, ?, ?, 1\spadesuit - Pass - 2\spadesuit - Pass, N, None, (2/1, Acol)\rangle.$$

Given a partial state p, and an agent A_i, we define the set of all full states consistent with p for agent A_i as

$$CFS(p, i) = \{s \in S | PS(s, i) = p\}.$$

The size of this set determines the agent's uncertainty regarding the current full state.

There are several methodologies for dealing with such uncertainty. One possibility is to use a Bayesian network where the input variables describe the current state and the output variables stand for possible bids. We discovered, however, that writing such a network for the bridge-bidding domain is extremely complicated, involves many variables, and requires extensive expert knowledge. In addition, bridge bidding requires partner cooperation through a specific bidding system. Therefore, such a network would need to be rebuilt for each partner.

Another possible alternative is to enumerate all the members in $CFS(p, i)$, perform regular lookahead for each of them, and select the action with the highest expected value. Such an approach was taken by Poki (Billings, Davidson, Schaeffer, & Szafron, 2002). Enumeration should indeed work for $CFS(p, i)$ of small size. In bridge bidding, however, where the size of $CFS(p, i)$ is always $\frac{39!}{(13!)^3} = 8.45 \cdot 10^{16}$, this approach is infeasible.

One way to reduce the size of the set of consistent states is through abstraction operations such as bucketing – partitioning the set into equivalence classes. Such an approach was applied by Shi and Littman (2001), and by Billings, Burch, Davidson, Holte, Schaeffer, Schauenberg, and Szafron (2003), to the domain of poker. This method requires finding abstraction operators that are aggressive enough to reduce $CFS(p, i)$ to a size that allows enumeration, while retaining the subtleties of the state. One abstraction operator that is commonly used by bridge players treats all minor cards (below 10) as equivalent. This abstraction reduces the size of $CFS(p, i)$ to $6 \cdot 10^{12}$, which is still too large to allow enumeration. However, this abstraction is considered by bridge experts as too rough to be used for bidding decisions.

The approach we eventually decided to adopt is *Monte Carlo sampling*, which is used by GIB (Ginsberg, 2001) for both bridge bidding and playing. Under this approach, we generate a sample of $CFS(p, i)$, perform lookahead for each of the states in the sample, and select the action leading to the highest expected value. Ginsberg suggested drawing the sample not from $CFS(p, i)$ but rather from a subset of it "... consistent with the bidding thus far". In Section 3.4 we show an algorithm that uses acquired models of the other agents to find such a subset.

3.2 Decision Nodes for Other Agents

We assume some heuristic function that can evaluate the leaves of the tree. In this case it is clear that when agent A_i performs lookahead search, we should select, at nodes associated with this agent, an action with maximal value. It is less clear what decision should be assumed at nodes associated with other agents. In traditional minimax search with limited-depth lookahead, we select a minimal value action at an opponent's decision node. This is a conservative approach that assumes the worst case[5].

In the case of a rational opponent in full-information zero-sum games, it is likely to select a move with a value similar to the worst-case value assumed by minimax. Similarly, a rational *co-agent* is likely to select a move with a high utility (from the agent's point of view), and therefore an extension of the minimax approach would maximize co-agent nodes.

In imperfect information games, however, the nodes associated with other agents are more problematic. The information available to the other agents is different than that available to the current agent. Minimax assumptions under such circumstances are not feasible – assuming that an opp-agent always takes the worst action from your point of view carries an implicit assumption that it always knows your current hand (as it does the current hand of the co-agent).

In poker, for example, such an assumption would lead the agent to the infeasible assumption that the opponent always knows when to raise and when to fold.

An alternative strategy would try to *simulate* the other agents in order to predict their actions. This approach falls under the general framework of *opponent modeling search* (Carmel & Markovitch, 1996b; Iida, Uiterwijk, van den Herik, & Herschberg, 1993, 1994; Donkers, 2003). There, the action of the opponent is predicted by simulating its decision process using an opponent model (typically a minimax algorithm using a given or learned evaluation function).

The M* algorithm (Carmel & Markovitch, 1996b) was designed for two agents that play in turn. We generalize the algorithm to allow any number of players. As in M* (and minimax), we require that only one player be active at a time, but do not require them to take turns. Instead we assume that it is possible to determine for each state which agent should act. The M* approach defines a *player* as a pair consisting of a strategy and a model of the other agent, which is (recursively) also a *player*. Here we extend the definition to allow n agents: A *player* P_i is a tuple

$$[M_1, \ldots, M_{i-1}, U_i, M_{i+1}, \ldots, M_n],$$

where U_i is the agent's strategy represented by its utility function (over states) and M_j is also a *player* representing the model of agent j. We assume that the depth of the recursion is finite.

For imperfect-information games, we would need to simulate the process where the other agents work with incomplete information. The extended version of M*, which is capable of handling multiple agents and partial information, is illustrated in Figure 1. The process begins with Monte Carlo sampling, which generates a set of full states consistent with the partial state observable by the agent. For each member in the generated sample, a lookahead

5. This is true for limited-depth trees. In the case of full game trees, it can be justified because it is the *optimal solution*.

search is performed. Gray round nodes represent states where the agent who called the algorithm should act, while white square nodes represent states where other agents are simulated. On the right, we can see an illustration of one of the simulations. The full state is passed to the simulator through a masking filter which hides the information not observable by the simulated agent. In bridge, for example, this masking is easily implemented by hiding the three hands belonging to the other players.

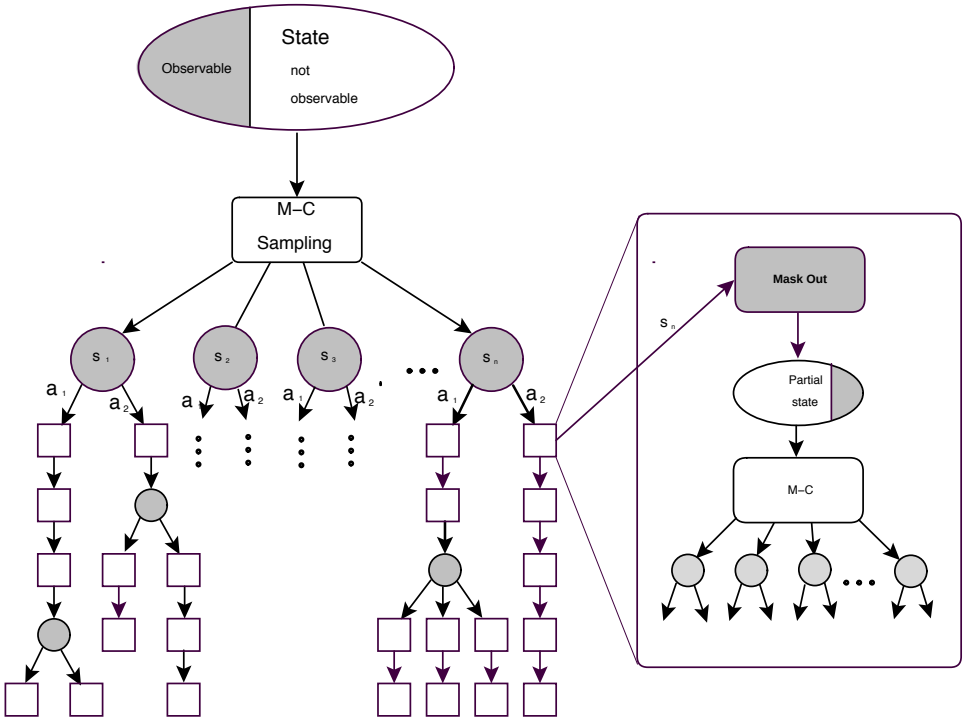


Figure 1: Incorporating Monte Carlo sampling into the lookahead search in partially observable environments. On the right, an illustration of each of the simulations of the other agents.

3.3 Resource-Bounded Reasoning

Bridge bidding, like many other MAS interactions, limits the time that agents can spend on decision making. Assume that we are allocated T time for the decision making process[6]. First we need to decide the size N of the sample used for the Monte Carlo sampling process. Then we need to decide how much time (denoted by G) should be allocated to generating the samples.

This process needs to be constrained because significant resources are consumed in looking for a sample that is consistent with the other agents' actions. If we make a simplifying

6. In bridge, as in other games, there is a time limit on the entire game. The question of how to distribute the total resources between the individual states is challenging and was studied by Markovitch and Sella(1996). In the current work we assume that each state is allocated the same decision-making time.

assumption that each instance in the generated sample is allocated the same time R, then we can conclude that $R = \frac{T-G}{N}$.

Given allocation of R time for a lookahead search, we need to decide how to distribute it within the tree. For simplicity, we assume that at nodes corresponding to the current agent the remaining resources are distributed equally between the subtrees corresponding to the different possible actions. Simulation nodes present a more difficult problem: How to distribute the remaining resources between the simulation process and the search applied to the resulting action. Again, we use a simple scheme where a fixed portion of the remaining resources is devoted to simulation.

One problem with the simulation approach is the relatively few resources available for each simulation. This leads to very shallow simulation trees, increasing the likelihood of simulation error. Furthermore, in bridge bidding it is very difficult to evaluate non-leaf nodes. One way of searching deeper using the same resources is by reducing the branching factor of the search tree. This approach is known in the game-playing community as *plausible move generation* or *selective search*.

Finkelstein and Markovitch (Finkelstein & Markovitch, 1998) describe an approach for learning a selection strategy by acquiring move patterns. Their approach, which depends on a graph-based representation of states, is appropriate for games similar to chess but not for bridge. In the next section we describe an alternative approach for representing a selection strategy. The proposed approach depends only on the availability of state features, making it more general than the former work. For the purpose of this section we assume that we possess a selection strategy $\varphi : S \rightarrow 2^A$ where S is the set of possible states and A is the set of possible actions. We now extend the strategy of a *player* to include its selection strategy as well:

$$P_i = [M_1, \ldots, M_{i-1}, \langle U_i, \varphi_i \rangle, M_{i+1}, \ldots, M_n].$$

3.4 Resource-Bounded Model-Based Monte Carlo Sampling

Assume we are given models of the selection strategies of the other agents, $\varphi_1, \ldots, \varphi_n$, and the history of the interaction $\langle a_1, A_{i_1} \rangle \ldots, \langle a_m, A_{i_m} \rangle$, where a_j is an action and A_{i_j} is the agent taking it. We can now design an algorithm for generating a sample consistent with the past actions. The first phase of the sampling process is to generate an instance, s, of the state distribution consistent with the current partial observation of the agent. In bridge, for example, this stage is easily implemented by randomly dealing the 39 cards which are not in the player's hand to the other three players. Then, the new instance s is tested for consistency with the history actions. First, we find the previous full state, s_{m-1}, by applying the inverse version of the last action, a_m^{-1}, to the generated state s. We then check if a_m is consistent with the model of agent A_{i_m} by testing for membership in the set of actions returned by $\varphi_{i_m}(PS(s_{m-1}, i_m)$. The process then continues through the entire history.

Ideally, we would continue the generation process until we find a consistent sample of the requested size K. In practice, however, we also get a limit G on the resources we can expend on the sampling process. Therefore, we use a contract algorithm that returns the K most consistent states found within the allocated resources. The inconsistency of a state is measured by the number of actions in the history that disagree with it. The process of

```
function RBMBMC(PartialState, History, Agent, ResourceLimit, SampleSize) → sample
    Let Agent be [M_1, ..., M_{i-1}, ⟨U_i, φ_i⟩, M_{i+1}, ..., M_n]
    Let History be ⟨a_1, A_{i_1}⟩ ..., ⟨a_m, A_{i_m}⟩
    repeat SampleSize times
        FullState ← random element of CFS(PartialState,i)
        InconsistCount ← Inconsistent(FullState, History, Agent, m)
        Sample ← Sample ∪ {⟨FullState, InconsistCount⟩}
    MaxInconsist ← Maximal InconsistCount in Sample
    while Spent Resources < ResourceLimit and MaxInconsist > 0
        FullState ← random element of CFS(PartialState, i)
        InconsistCount ← Inconsistent(FullState, History, Agent, MaxInconsist)
        if InconsistCount < MaxInconsist then
            NewElement ← ⟨FullState, InconsistCount⟩
            WorstElement ← An Element in Sample with InconsistCount = MaxInconsist
            Sample ← Sample ∪ {NewElement} − {WorstElement}
            MaxInconsist ← Maximal InconsistCount in Sample
    return Sample

function Inconsistent(FullState, History, Agent, CutOff) → [0,m]
    State ← FullState
    CurrentInconsist ← 0
    for i from m downto 1
        State ← a_i^{-1}(State)
        if a_i ∉ φ_{i_m}(PS(state, i_m) then
            increase CurrentInconsist by 1
        if CurrentInconsist = CutOff then
            return CutOff
    return CurrentInconsist
```

Figure 2: Resource-Bounded Model-Based Monte Carlo Sampling (RBMBMC sampling)

counting the number of inconsistencies can be cut off if the current number is already equal to the maximal one in the current K best states.

The resource-bounded model-based Monte Carlo Sampling algorithm (RBMBMC) is listed in Figure 2.

3.5 The PIDM Algorithm

Having described the way we solve the problems listed in the Introduction, we are now ready to present our main algorithm. We call this algorithm PIDM for Partial Information Decision Making. The algorithm receives a partial state, a *player* (as defined in Section 3.2), and a resource limit.

An agent's selection strategy is called to find the subset of actions that need to be considered for the given state. A sample of full states consistent with the history is then generated as described in Section 3.4. Each of the actions is then evaluated on each of the sample members, and the action with the maximal expected value is returned.

function PIDM(PartialState, Agent, ResourceLimit) $\rightarrow$ *Action*
  Let Agent be $[M_1, \ldots, M_{i-1}, \langle U_i, \varphi_i \rangle, M_{i+1}, \ldots, M_n]$
  $Actions \leftarrow \varphi_i(PartialState)$
  if $|Actions| > 1$ **then**
    $G \leftarrow g \cdot ResourceLimit$; $0 < g < 1$
    $S \leftarrow RBMBMC(PartialState, History(PartialState), Agent, G, SampleSize)$
    for $A \in Actions$
      $V[A] = \frac{1}{|S|} \sum_{s \in S} LookAhead(A(s), Agent, \frac{ResourceLimit - G}{|Actions||S|})$
    return action A with highest $V[A]$
  else
    return the *action*

function LookAheadSearch(State, Agent, ResourceLimit) $\rightarrow$ *Utility*
  if $ResourceLimit \leq \epsilon$ **then** ; ϵ = enough time for computing U
    return $U_i(State)$
  $NextAgentIndex \leftarrow ActiveAgent(State)$
  if $NextAgentIndex = i$ **then** ; The agent that called LookAheadSearch
    for $A \in \varphi_i(State)$
      $V[A] = LookAheadSearch(A(State), Agent, \frac{ResourceLimit}{|\varphi_i(State)|})$
    return maximal $V[A]$
  else
    $R_{sim} \leftarrow r \cdot ResourceLimit$; $0 < r < 1$
    $A \leftarrow PIDM(PS(State, NextAgentIndex), M_{nextAgentIndex}, R_{sim})$
    return $LookAheadSearch(A(State), Agent, ResourceLimit - R_{sim})$

function ActiveAgent(State)$\rightarrow$ Agent index
  Return the index of the agent that should act at the given state.

Figure 3: Resource-Bounded Partial Information Decision Making (PIDM algorithm)

The utility of an action with respect to a full state (a member of the generated sample) is evaluated by calling a lookahead procedure on the next state — the one that resulted when the action was applied. If there are insufficient resources for further lookahead, the procedure returns the current state's utility, which is computed using the strategy of the current player. Otherwise, it determines the identity of the agent that should act next. If the agent is the player that called the procedure, the lookahead continues with appropriate adjustment of the resource limit. If that agent is another player, the procedure simulates its decision making: it computes the partial state observable to that agent and calls the PIDM algorithm on that state with the other agent's model. This call to PIDM returns the action assumed to have been taken by the other agent. We apply this action on the current state and continue the lookahead process from it.

We avoid the potential complication of empty player models by assuming a default model that can be used by the algorithm. The formal PIDM algorithm is given in Figure 3.

3.6 The PIDM Algorithm and Bridge Bidding

In Section 1 we specify one of the aspects of bridge that makes bidding particularly hard – the use of bids for exchanging information between partners. How does the PIDM algorithm manage to resolve this problem?

PIDM solves this problem implicitly through the simulation process. Each bid is evaluated using a deep lookahead that simulates the other agents. A bid that conveys information to the partner results in a small set of possible states considered by the partner to be consistent with the player bids. The small set will lead to a more accurate Model-based Monte-Carlo simulation which will, in turn, yield a more efficient and accurate search.

The simulation process also covers the opposite case, where the player asks the partner for information. An "information-request" bid will be recognized down the lookahead tree by the simulated partner, who will return the required information through its own bid. This will reduce the space of possible partner hands and will yield more accurate Monte Carlo simulation that will result in a better bid.

The success of this process of information exchange depends on the quality of the "language" used – the bidding system. The problem with using bidding systems as a language is their high level of ambiguity. In the next section we show a co-training algorithm which reduces that ambiguity, making the PIDM algorithm much more efficient.

4. Learning to Cooperate

In the previous section we defined an agent as having both a selection strategy and a utility function. The selection strategy is used to reduce the branching factor of the lookahead search. Knowing the selection strategy of other agents helps to reduce the branching factor during simulation. In addition, the PIDM and RBMBMC algorithms use the other agents' selection strategies to find the subset of states consistent with the auction history, thus improving the sampling process.

The selection strategy can be manually built by the agent designer or learned from experience. Even if manually built, there is still a strong motivation to refine it by learning. First, the selection strategy returns a set of actions for each partial state. Refining the network will reduce the size of the returned action sets, thus reducing the branching factor of the tree expanded by the PIDM algorithm. Second, learning together with a co-agent can yield a refined selection strategy that is adapted to that of the co-agent.

The bidding system in bridge can be viewed as a manually built selection strategy. Bridge is particularly suitable for modeling approaches because of the rule that requires the pairs to expose their bidding systems to each other. This spares us the need to acquire a model of the opp-agents' strategy via learning.

Bridge partners face two main problems when adopting a bidding system: lack of coverage, i.e., the existence of many states for which no rule applies; and ambiguity, i.e., two or more rules applicable in a given state. Bridge partners therefore spend significant time refining their bidding system by co-training.

In this section we provide a framework and an algorithm for such co-training. We start by describing the data structure we use for representing that strategy, continue with an algorithm for experience-based refinement of a selection strategy, and conclude with an algorithm for co-training of partners.

4.1 Representing the Selection Strategy Using Decision Networks

Our goal is to come up with a representation scheme that is easy to learn and maintain as well as easy to use. Given that bridge bidding systems are represented by a set of rules, a rule-based system comes to mind as the obvious candidate for representing a selection strategy. This representation scheme is indeed modular – it is very easy to add or remove rules. However, rule-based systems are difficult to maintain because of their potential inconsistency – a problem that is easy to detect but hard to fix.

In this subsection we describe the *decision networks* mechanism for representing the decision strategy. The main advantage of decision nets is their hierarchical structure, which allows refinements and conflict elimination by means of specialization-based learning algorithms such as the one described in the next subsection.

A selection strategy is a function $\varphi : S \rightarrow 2^A$ where S is the set of possible states and A is the set of possible actions. We represent this function by a structure called a *decision net*. Each node n of the net represents a subset of states $S(n)$. The nodes are organized in an ISA hierarchy. A link from node n_1 to node n_2 means that $S(n_2) \subseteq S(n_1)$. There are two types of links – *regular* links and *else* links. The two types are used to impose a priority when accessing the links, as we will explain shortly.

The membership condition of a node is determined by its set of *state constraints (SC)*. Let $SC(n) = \{C_1, \ldots, C_n\}$ be the set of constraints of node n. The states belonging to node n are $S(n) = \{s \in S | C_1(s) \wedge \ldots \wedge C_n(s)\}$. Each node n also has a set of associated actions $A(n)$. The network has one root node, n_0, with $SC(n_0) = \emptyset$ representing the whole set of states S.

In bridge, for example, the set of possible actions includes 38 legal calls (Pass, Double, Redouble, and the bids from 1♣ up to 7NT). A state constraint in our implementation is of the form $L \leq f(s) \leq H$, where L and H are lower and upper bounds and $f(s)$ is a state *feature*. We use a set of about 250 bridge-specific features which are listed in Appendix A. Examples for common attributes are HCP, which sums the points associated with high cards,[7] and SUITLEN$_\heartsuit$, SUITLEN$_\spadesuit$, SUITLEN$_\diamondsuit$ and SUITLEN$_\clubsuit$, which count the number of cards in each suit.

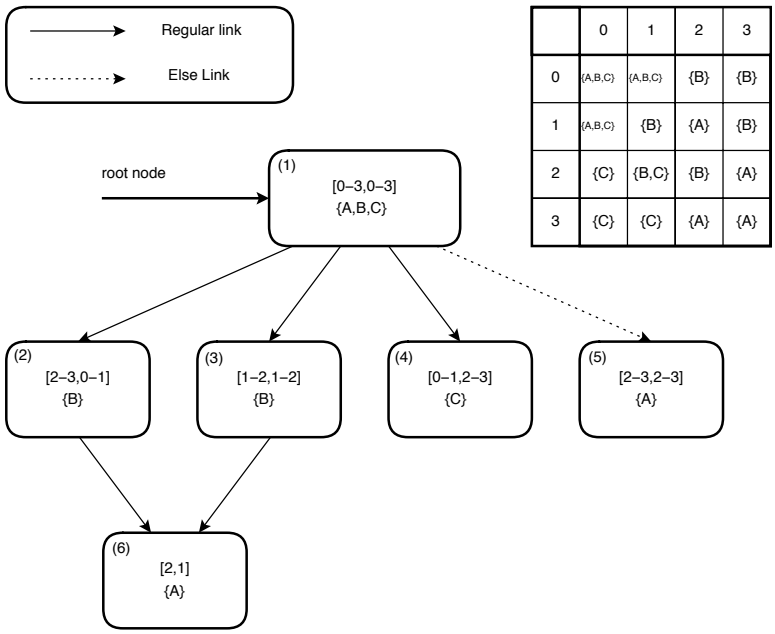
Consider for example a state s with the following hand: ♠ T5 ♡ AK954 ♢ QJ52 ♣ K3. For this state, $HCP(s) = 13$, SUITLEN$_\spadesuit(s) = 2$, SUITLEN$_\heartsuit(s) = 5$, SUITLEN$_\diamondsuit(s) = 4$ and SUITLEN$_\clubsuit(s) = 2$. One possible node that this state matches is

$$\langle \text{Actions:}\{1\heartsuit\}, SC : \{12 \leq HCP(s) \leq 23, 5 \leq \text{SUITLEN}_\heartsuit \leq 13\}\rangle,$$

meaning that one plausible opening call for this hand is 1♡.

To determine the set of actions associated with a state s, we take the union of the actions associated with the *most specific nodes* the state belongs to. These nodes are found by a recursive algorithm that starts from the net root and propagates the state down the hierarchy. For each node n, for each *regular* link (n, n'), the state is propagated if $s \in S(n')$. If s was not propagated down to any regular link, then a similar process is applied to the *else* links. If this also fails, then n is considered a most specific node for s and its associated actions $A(n)$ are accumulated. If the algorithm is given a partial state (that represents a subset of S), then the test $s \in S(n')$ is replaced by the test $s \subseteq S(n')$.

7. HCP = High Card Points - points for the picture cards in the hands. A=4; K=3; Q=2 & J=1.



	0	1	2	3
0	{A,B,C}	{A,B,C}	{B}	{B}
1	{A,B,C}	{B}	{A}	{B}
2	{C}	{B,C}	{B}	{A}
3	{C}	{C}	{A}	{A}

Figure 4: An example of a simple decision net. Each state is represented by two attributes. Each node contains a pair of range constraints on these attributes and a set of associated actions. The table shows the set of actions that the decision net associates with each state.

The algorithm can be illustrated with a simple example shown in Figure 4. Assume that world states are characterized by two attributes, f_1 and f_2, each with a domain of $[0-3]$. The selection strategy of our agent is represented by the given decision net of 6 nodes. The nodes are labeled (1) to (6) (for the sole purpose of facilitating the discussion). Consider, for example, the state $[0,2]$. The state is a member of node (1). It is also a member of one of its regular children, node (4). This is the only most specific node and therefore its associated set of actions $\{C\}$ will be returned. The state $[1,2]$ has two most specific nodes — (3) and (4). Therefore, the algorithm will return $\{B,C\}$. State $[2,1]$ is a member of (2) and (3). However, they are not the most specific nodes. This state will be propagated down two of the regular links to node (6) and the resulting set of actions will be $\{A\}$. Note that the actions associated with node (6) are not a subset of the actions associated with its parents. State $[2,2]$ is member of (3) and (5). But since (5) is an *else* link, it will not be considered and only the actions of (3), $\{B\}$, will be returned. The state $[3,3]$, however, does not belong to any of the regular children and will therefore be propagated through the *else* link to node (5), and its associated actions, $\{A\}$, will be returned.

Finally, because the state $[0,0]$ is not a member of any of the root's children, the propagation will stop there. The actions returned will be $\{A,B,C\}$.

The formal selection strategy algorithm is described in Figure 5.

The φ selection strategy, which is used both for action filtering and for state filtering, is implemented by calling the function *StateActions* from the root node.

```
function StateActions(Node, PartialState) → Set of action
    if PartialState ⊆ S(Node) then
        Actions ← ∅
        for Child ∈ Regular-Links(Node)                    ; Checking the regular nodes
            Actions ← Actions ∪ StateActions(Child,PartialState)
        if Actions ≠ ∅
            return Actions
        else                                               ; Checking the else nodes
            for Child ∈ Else-Links(Node)
                Actions ← Actions ∪ StateActions(Child,PartialState)
            if Actions ≠ ∅
                return Actions
            else
                return Node-Actions(Node)
    return ∅
```

Figure 5: The StateActions algorithm for finding the set of actions associated with a given partial state

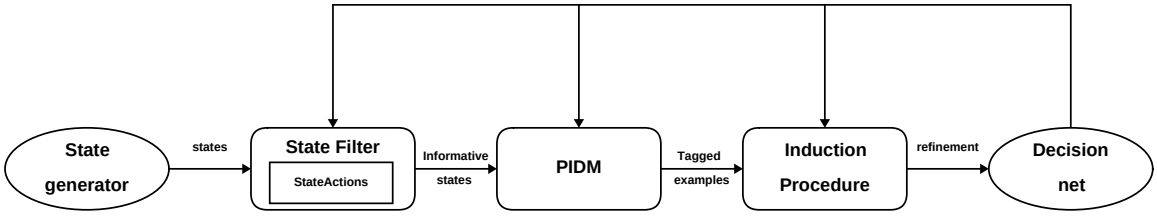


Figure 6: The information flow during the learning process

4.2 Learning Algorithms

The main goal of the learning process is to enhance the agent's performance. This is done by refining the selection strategy represented by the decision net and using the refined net to make the PIDM process more efficient. Our general learning framework is based on inductive learning from examples. The learning process has three main stages:

1. Generating informative training problems. These are states for which the StateActions function returns two or more actions.
2. Using the PIDM algorithm to tag each example with an action.
3. Generalizing over the tagged states and building a refined decision net (RDN).

The information flow of this process is shown in Figure 6.

4.2.1 Generating Informative Training Examples

The first phase of the process is to produce training instances. Since learning resources are limited, we are interested in producing instances that are expected to be useful for the

learner. Obviously, a state s for which $|\varphi(s)| = 1$ is not useful for the learner, since the selection strategy already knows what to do in this state. We can use our current decision network to generate states that are members of nodes with $|Actions(n)| \geq 2$. This is similar to the uncertainty-based experience generation employed by Scott and Markovitch (1993). There are, however, four problems with this approach:

1. Some states are more likely to occur than others. The samples that result from the aforementioned sampling process deviate significantly from the state distribution.

2. In general, we do not always know how to generate a state that satisfies a given set of constraints.

3. Sometimes $|\varphi(s)| \geq 2$ while no one node contains two actions. This situation can occur if s belongs to two different nodes with different actions.

We use here a modified procedure that takes the above problems into account. We assume the availability of a generator that generates random states according to their distribution. We pass each generated state through the net and collect only those for which $|\varphi(s)| \geq 2$. Since we are able to collect many such states, we prioritize them by the resources required for processing them. We prefer investing learning resources in states that require more search resources. Learning such states would reduce the branching factor associated with their decision nets, therefore leading to a greater reduction in search efforts.

4.2.2 Tagging the Examples

To build an example for the induction algorithm we must tag each state with its "correct" action. One way to obtain such tags is to call the PIDM algorithm with a resource allocation that is much larger than the one available during a real interaction. Such a large resource allocation either allows the lookahead procedure to search the whole state tree down to its leaves, or at least search it deep enough to make a reliable decision.

This is a classical form of *speedup learning*, where the learner exploits the large resources available during *offline* learning to perform deep lookahead. The results of this deep search can be used as the "real" values. The induction process then infers a fast procedure that attempts to approximate the "real" one. Samuel's checkers player (Samuel, 1959), for example, tried to infer a linear predictor that approximates the deep minimax value.

Since we are dealing with a multi-agent setup, even searching the whole tree will not necessarily yield the best possible action. This is because the PIDM algorithm relies on the models of the other agents. In the co-training framework described in Section 4.3, we show how the co-agents can exchange selection strategies during the learning process to tailor the tagging to these specific co-agents.

4.2.3 Learning Refined Decision Nets

The tagged examples of states and actions are used to refine the agent's decision net. The examples are inserted into the decision net and the net is refined in two main phases. The first phase is to insert the examples into the net.

Each example $\langle s, a \rangle$ is propagated down the decision net according to s and stored at the most specific nodes that s satisfies. The decision net is not structured to prevent conflicts.

It is quite possible that a state will belong to more than one specific node. Our approach is to resolve such conflicts only when they are realized during learning. Let $N = n_1, \ldots, n_k$ be the set of most specific nodes that s satisfies. If $k > 1$, we create a new node, $\hat{n}$, that is the intersection of $n_1, \ldots, n_k$, i.e., $SC(\hat{n}) = \bigcap_{n \in N} SC(n)$.

The second phase of the algorithm is to generalize over the tagged examples and resolve conflicts. After all the examples have been inserted into their proper place in the net, each node is analyzed according to the examples that were propagated to it. The easiest case occurs when the same action, a, is suggested by all the examples stored in node n. If $A(n)$, the set of actions associated with the node, is different from $\{a\}$, then we simply set $A(n)$ to be $\{a\}$. If the set of examples stored in n has a non-uniform set of actions, we use an induction process to build a classifier. The attributes for the induction process can be the same ones used also for specifying constraints over states. The input of the produced classifier is a feature vector representation of a state. Its output is an action (or a set of actions).

If the learning process were stopped at this point, we could use any induction algorithm to generalize over the examples. However, since we would like to continue the learning process and perhaps refine the generated hypothesis when more examples are available, we prefer an induction mechanism whose output can be merged into the decision net. One such algorithm is ID3, which generates decision trees that can be easily linked to be included in the decision net.

A problem may arise if the node where the learning takes place has children. While the examples used for learning do not belong to this node's children (since otherwise they would have been propagated to them), the generalization process may very well yield nodes that conflict with these children.

To make the learning process monotonic (in the sense of not producing new conflicts), we connect the resulting decision tree through an *else* link. Such a link will give priority to the existing children of the current node.

4.2.4 Illustrative Example

In this subsection we present a simple bridge example to illustrate the learning algorithm described above. Assume that our initial bidding system contains the following two rules for the opening bid:

1. If the hand is balanced and the high card points are between 15 and 17, then bid 1NT.

2. If the high card points are between 11 and 23 and the hand contains at least 5 hearts, then bid 1♡.

These two rules are included in most common bidding systems. The rules are represented by the decision net shown in Figure 7(a). The root node is common to all decision nets and contains the fallback decision: If no match was found, consider all possible actions. The two children correspond to the two rules.

Assume now that during learning our algorithm encounters the state $PS(s, N) = \langle(\spadesuit AQ2\ \heartsuit QJ854\ \diamondsuit K4$ Propagating the state through the decision net will end up in two nodes with two conflicting

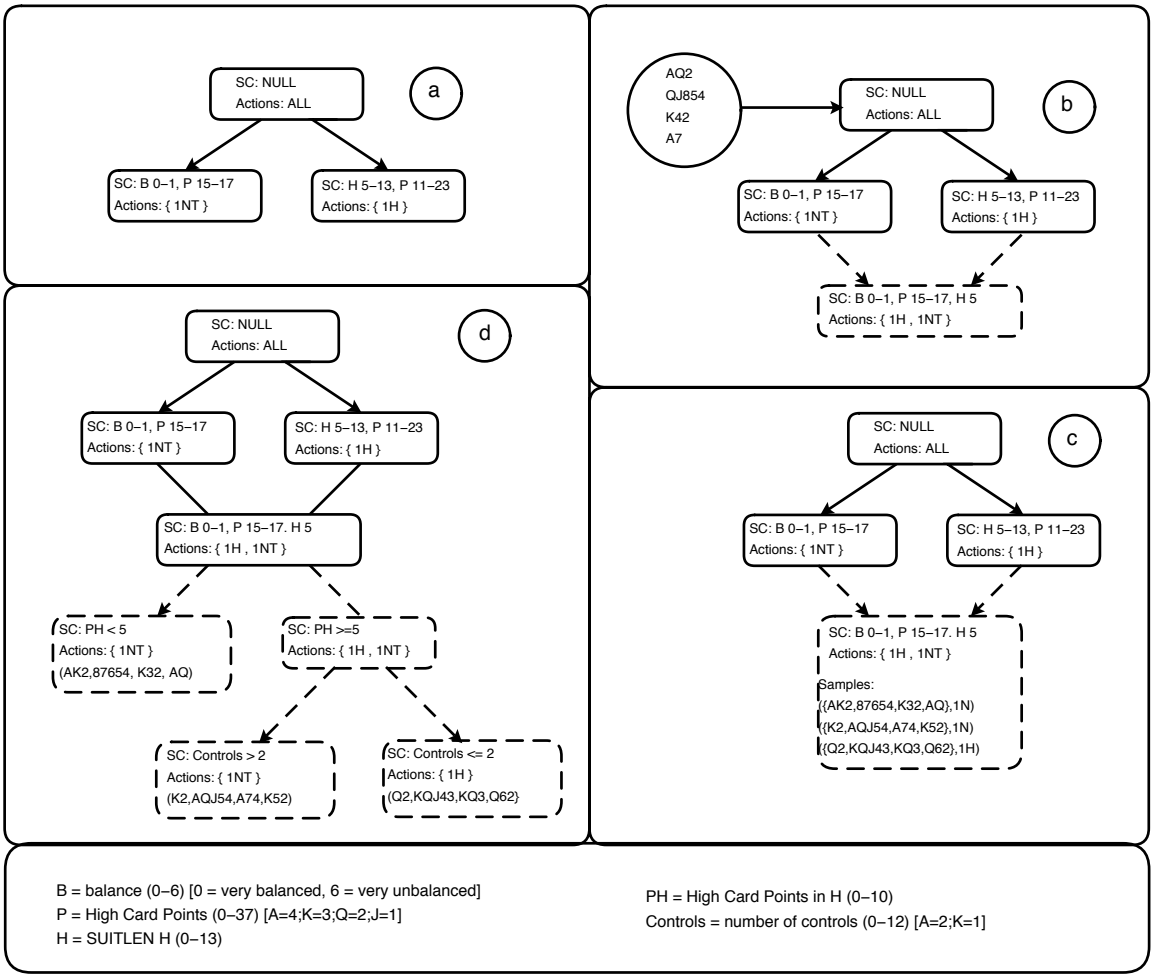


Figure 7: The learning algorithm refines the decision net by (a) finding an example that causes conflict; (b) creating an intersection node; (c) generating tagged examples for the intersection node; (d) connecting a newly induced decision net using an *else* link.

action sets. The algorithm therefore generates an intersection node with the conjunction of the conditions of its two parents. This process is illustrated in Figure 7(b).

After more training examples are processed, some of them will be stored in the new node as illustrated in Figure 7(c). The algorithm then calls the induction algorithm on the accumulated set of tagged examples and generates the tree illustrated in Figure 7(d).

The resulting tree represents a refined bidding system containing an additional new set of three rules:

1. If (a) the hand is balanced, (b) the high card points are between 15 and 17, (c) the hand contains 5 hearts, and (d) the high card points in hearts is less than 5, then bid 1NT.

2. If (a) the hand is balanced, (b) the high card points are between 15 and 17, (c) the hand contains 5 hearts, (d) the high card points in hearts is at least 5, and (e) the hand contains at least 3 controls, then bid 1NT.

3. If (a) the hand is balanced, (b) the high card points are between 15 and 17, (c) the hand contains 5 hearts, (d) the high card points in hearts is at least 5, and (e) the hand contains less than 3 controls, then bid 1♡.

4.3 Learning with Co-agents

The learning process as described in the previous section can be performed separately by each agent. There are, however, advantages to cooperative learning.

First, we use the PIDM algorithm during learning to evaluate each action. At each of the nodes associated with other agents, the algorithm attempts to predict the action they would take. For opponent agents, the algorithm uses modeling to make this prediction. During a real interaction, communication between co-agents may be restricted and therefore the agent must model the co-agents as well. During learning, however, it is quite possible that such restrictions are not imposed. In this case, the agent can ask its co-agents specifically for the actions they would have taken in the associated states. This "simulation service" provided by the co-agents is likely to yield more accurate tagging.

Then, during RBMBMC sampling, the agent uses its model of the other agents' selection strategy to reduce the population of states to sample from. During co-learning, the co-agents can exchange their selection strategies to make the RBMBMC process more accurate.

A very important issue is the scheduling of such a co-learning process. If we schedule the co-agents to learn sequentially, the decisions made by each during the co-learning process will be very different from the actual decisions reached at the end of the process. This is because these decisions will change significantly during each learning turn. Therefore we need to schedule the learning processes of the co-agents to be executed in parallel. In this way, all the agents will progress simultaneously, and the simulation results, especially toward the end of the learning process, are likely to be accurate. Also, when performing parallel co-learning, the co-agents need to exchange their selection strategies often, for the same reasons. This process is illustrated in Figure 8. Consider the bridge example given in Section 4.2.4. After co-training, the partner uses the refined decision net of Figure 7(d) received from the player to constrain the space of Monte Carlo sampling. There, if the player bids 1NT at the beginning, the space of possible states will be smaller than the space before training, leading to a more accurate search. In addition, after receiving the net, the partner is able to get an exact simulation of the player at the relevant states without using search for the simulation.

5. Empirical Evaluation

To test the utility of the framework presented in this paper we conducted a set of experiments in the domain of bridge bidding. The algorithms described in the previous sections were implemented in BIDI - a bridge bidding system. We start by describing the research methodology, including a specification of the dependent and independent variables, and continue by describing and analyzing the results.

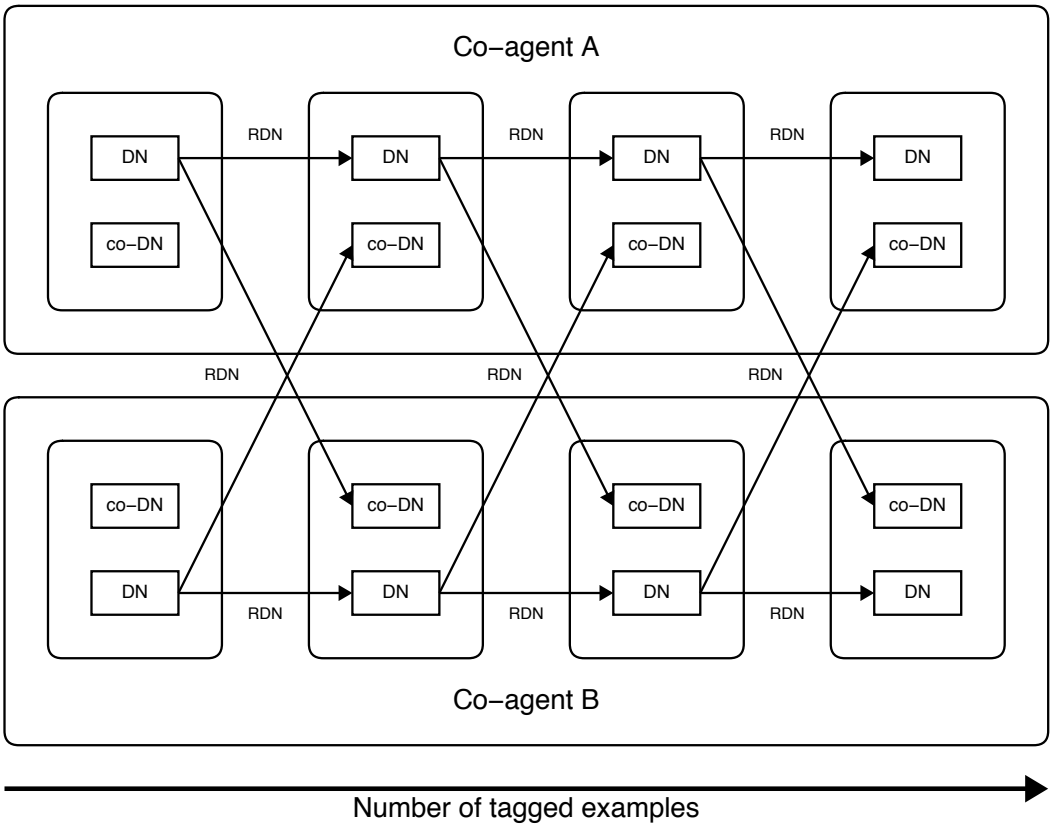


Figure 8: Exchanging decision nets during co-training. After each exchange, the agents continue the process with the refined decision nets.

5.1 Experimental Methodology

Our experimentation started with a co-training session. A set of 2000 random deals was generated and used as input to the active learning process as described in Section 4. Then, on the basis of the examples collected during the learning process, the system began to refine the decision network using ID3. We stopped the learning session after 100 decision nets were refined. The learning process had a time limit of 5 seconds for each tagging process and a sample size of 10. The bidding system used during the test was a standard two-over-one system with some common conventions.

To evaluate the performance of our bidding algorithm we tested it against GIB, which is considered to be one of the strongest bridge programs. We conducted the test with a pair of our bidders competing against a pair of GIB bidders. Since our focus was on the bidding stage of the game, we used GIB to perform the playing stage for both pairs. The details of this tournament are described in Section 5.2.

In addition to evaluating the system's performance, we are also interested in performing a parametric empirical evaluation to get a better understanding of the algorithms involved. Since the above method involves manual manipulation by an operator who has to feed

the GIB program with BIDI's bids, it is not suitable for an extensive parametric study. Therefore we added *bidding challenges:* an additional method that allows an automatic evaluation of our bidder.

A bidding challenge is a very common method for evaluating the performance of human bridge pairs. The method is used by pairs to evaluate their progress while practicing. It is also used in various competitions to compare the bidding performance of pairs of experts. The method was also used to test computerized bidders such as COBRA (Lindelöf, 1983), Wassermann's bidder (1970), and BRIBIP (Stanier, 1975).

Bidding challenge problems test the bidding ability of a pair of players *without the need for an actual opponent pair*. Each problem consists of two hands dealt to the pair together with a predetermined call sequence for the opponent that is considered to be "reasonable". We used two sets of bidding challenge problems. The first set consists of 100 randomly generated deals. To avoid trivial problems, a filter is used to ensure that at least one hand is suitable for an opening bid. Such filtering usually yields deals where the tested pair has a stronger hand. Therefore we can make the common assumption that the opponent pair does not interfere and its bids can be assumed to consist of a sequence of "pass" calls. The other set consists of problems taken from the book "Bidding Challenge" (Cohen, 2002).

A bidding challenge problem is solved by a sequence of calls that results in a contract. The quality of the solution is then measured by evaluating this contract.

The method we use in this work to evaluate contracts is the *IMP scoring* used in many top international events. As explained in Section 2.1, the score of a bridge deal is determined by comparing the number of tricks committed to in the contract to the number of actual tricks taken. A reference score (known in bridge as a *datum score*) is then used to normalize this score, and the resulting score is the IMP score. In bridge competitions the reference score is the average score obtained for the same deal by the other pairs participating in the event.

There are two issues that need to be resolved when evaluating BIDI's performance. First, we need an objective reference score that can be automatically computed. For the random problems, we evaluate the expected score for each of the 35 possible contracts, and take their maximum as the reference score. The expected score of a contract is computed from the two hands of the partnership using the LTC principle. This principle will be explained in Section 5.1.2. The reference scores for Cohen's problems are given in the book and are taken from a world class field of experts.

Second, because the playing stage is omitted, we need to estimate the score of the actual contract reached. We do so using the same LTC evaluation applied to the contract. Thus, the difference between the maximal LTC value and the actual LTC value is the IMP value of the deal.

5.1.1 Dependent Variables

We measure several aspects of the bidder's performance during the testing phase: the average scores when solving the problems, the resources used during the process, and the number of times that learned decision nets were actually used during testing.

1. *IMPs/problem:* The average IMP gain/loss per problem. This is the most important aspect of the performance.

2. *Generated instances:* The number of instances generated by the Monte Carlo sampling algorithm during the execution of PIDM. This number includes the samples that were checked and rejected during the RBMBMC process. We found that this measurement is the platform-independent variable that best correlates with execution time.

3. *Lookahead invocations:* When PIDM is called, it first calls the selection strategy to propose possible actions. Only if two or more actions are returned is a lookahead process initiated. This variable measures the number of times the PIDM process initiated a lookahead due to a conflict with the decision net associated with the state.

4. *Lookahead invocation ratio:* The ratio between the number of times a lookahead was initiated and the number of times the PIDM process was used. As learning progresses, we expect this ratio to decrease.

5. *Learned positions utilization:* The number of refined decision nets that were used during the test. As learning progresses, we expect this measurement to increase.

6. *Sampling failures:* The number of times the RBMBMC process could not find a set of instances that is 100% consistent with the auction because of insufficient resource allocation.

7. *Lookahead stage:* The average number of calls in the states where a lookahead was initiated.

5.1.2 Independent Variables

Several variables affect the performance of the bidder. We tested the effect of the following parameters:

1. *Sample size:* The number of instances returned by the RBMBMC process.

2. *Resource allocation:* There are two main parameters that dictate how the resources are distributed during decision making:

   (a) *Sampling resources:* The maximum portion of the total resources dedicated to the RBMBMC sampling process.

   (b) *Simulation resources:* The maximum resources allocated to each recursive PIDM call used for agent simulation.

3. *Utility function:* The ultimate value of an auction to a player is determined directly by the number of tricks taken. The number of tricks can be estimated by performing full lookahead in the game tree that describes the *playing* stage. This approach, however, requires high computational resources. The approach used by BIDI is based on the well-known *Losing Trick Count* (LTC) principle. The number of tricks is estimated from the hand of the player and the hand of the partner. We estimate the number of tricks for each suit and then sum them. For each suit, A, K and Q are considered winners. Every card over the first 3 is also a winner (because we can assume the opponent will run out of cards in the suit at the fourth or fifth round). The cards that are not winners are considered losers. The losers may be covered by high cards of the

partner [8]. The contract and the number of expected tricks determine the expected number of points[9]. More detailed explanation of LTC can be found in many bridge text books (for example (Klinger, 2001)).

4. *Bidding system:* BIDI uses a variant of the common *two-over-one* system. We prefer a "human-type" bidding system over computer oriented bidding systems like COBRA (Lindelöf, 1983) since the later have trouble handling different opponent systems. Bidding systems are traditionally represented by a set of bidding rules. Each rule contains an auction pattern, a call, and information revealed by that call. BIDI accepts such input and converts instantiated rules to decision nets on the fly whenever necessary. The full structure of the bidding rules and the algorithm for on-the-fly construction of the decision nets are explained in Appendix B. Our initial bidding system contains about 2000 rules.

5. *Learning resources:* The resources devoted to learning are controlled by three parameters.

   (a) *Training sequences:* The basic learning procedure involves the refinement of a decision net associated with a bidding sequence.

   (b) *Training examples:* The number of tagged examples produced for each decision net being refined.

   (c) *Resources/Example:* The resources (in seconds) dedicated for each example.

   These three parameters are multiplied to obtain the total resources used for learning.

5.2 The performance of BIDI

We tested the performance of BIDI after learning with the performance of GIB by conducting a match of 48 boards. The match was conducted in the same way as an official bridge match. We simulated a team of two pairs played by GIB and a team of two pairs played by BIDI. The match consisted of two tables – one with BIDI playing North/South and one with BIDI playing East/West. Each of the boards was played by the two tables.

The bidding process was conducted with the help of a human operator. Each call by GIB was manually fed into BIDI, and the call returned by BIDI was manually fed into GIB. At the end of the bidding process we used the "auto play" feature of GIB, thus allowing its playing engine to play both sides.

For each board we computed the score of the two teams by comparing their points on both tables and converted the difference into IMPs.

The results of the match were 103 IMPs to BIDI versus 84 IMPs to GIB, which is equivalent to 0.4 IMP/board.

8. For example, assume the player has Axxx for a certain suit and its partner has Kxxx. Axxx contains 2 losers. One of them is covered by the K in the partner's suit, and therefore 3 tricks are expected.
9. For example, if the pair is able to take 11 tricks with hearts as the trump suit, the function will return +200 for a 2♡ contract (commiting to take 8 tricks), +450 for a 4♡ contract (commiting to take 10 tricks and getting a game bonus), and -50 for a 6♡ contract (commiting to take 12 tricks and failing).

5.3 Learning curves

To analyze the behavior of the learning system we measured various dependent variables periodically throughout the training process.

5.3.1 Performance Curve

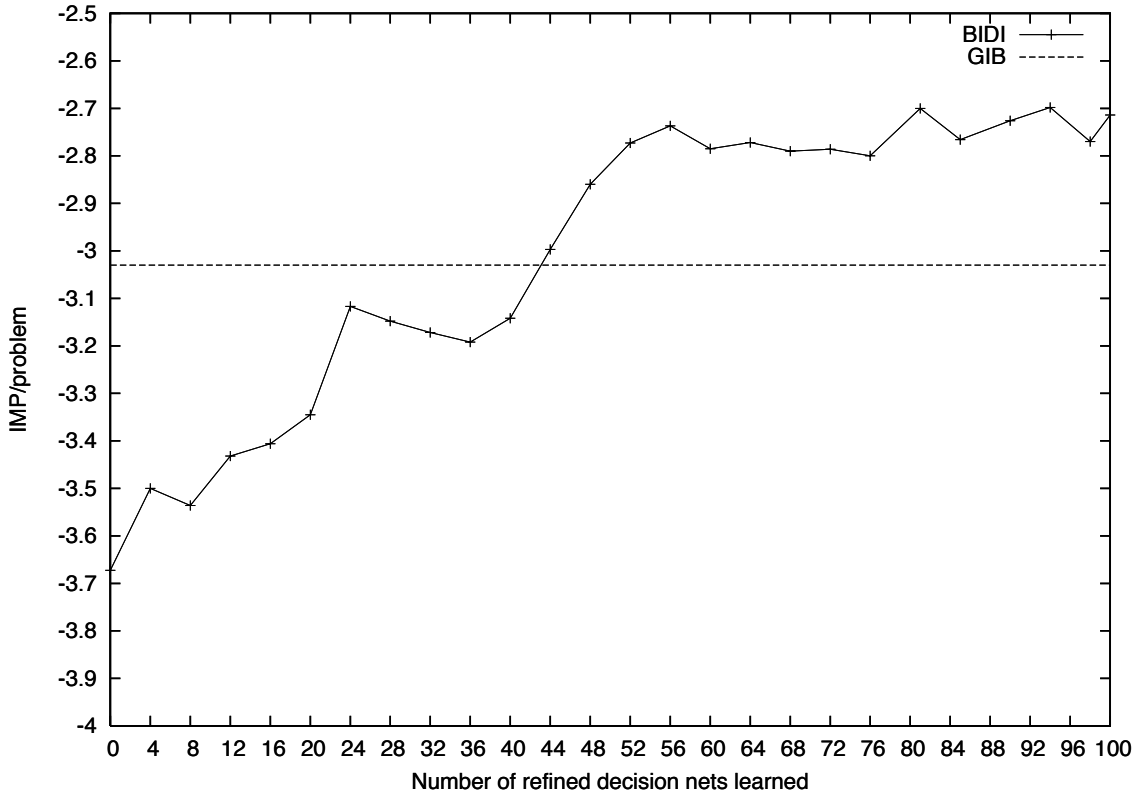


Figure 9: Average score on 100 random problems

The most interesting dependent variable to measure is the performance of the system as a function of the amount of acquired knowledge. Since the evaluation methodology described in the previous subsection involves human operator, we decided to use instead a set of 100 bidding challenges as explained in Section 5.1. After refining each 4 nets, we turned off learning, performed a test, and then resumed the learning process. During the test, a time limit of 0.5 seconds was given to each call and a sample size of 3 was used. Each test was repeated 10 times and the resulting scores were averaged.

Figure 9 shows the learning curve of BIDI. The x-axis measures the learning resources, expressed by the number of decision nets refined. The y-axis shows the performance of BIDI on the test set, measured by the average IMP per problem. We can see that BIDI improved its performance by about 1 IMP during learning. This improvement is substantial – it is customary to estimate the performance difference between beginners and experts at about 1.5 IMPs.

One potential critique of the presented graph is that the system started with very weak bidding ability and therefore was easy to improve. To test this, we compared the performance of the system *before* learning to the performance of world experts on the set of bidding challenges in the aforementioned book by Cohen(2002). BIDI achieved an IMP score of 0.3, meaning that its bidding ability on these problems is above average relative to the reference group of world experts.

BIDI's performance on the test set after learning is about -2.7 IMPs per deal. We can put this number in perspective if we take note of the fact that, during the 2003 World Bridge Championship Final, each of the teams scored about -2.5 IMPs per deal. Note however that BIDI achieved the score in a bridge challenge where the opponent does not interfere.

Another point of reference is the performance of the GIB system (Ginsberg, 2001) on our test. We applied GIB (version 6.1.0) on our test set and measured its bidding performance. The horizontal line in the graph shows its performance (an average score of -3.03 IMPs/problem). We can see that before learning GIB has an advantage over BIDI of about 0.6 IMPs/problem. After learning, however, BIDI's performance exceeds GIB's by about 0.3 IMPs/problem.

5.3.2 The Use of Refined Decision Nets During the Experiment

Two variables serve to evaluate the use of refined decision nets during the experiment: the number of lookahead invocations and the number of refined nets actually used. The learning process should reduce the number of lookahead invocations because the refined nets should solve the conflicts that invoke them. Avoiding lookahead search also eliminates the recursive calls to the PIDM algorithm. Since the performance of the system improves with learning, we expect that the number of refined nets used during testing will increase as learning progresses.

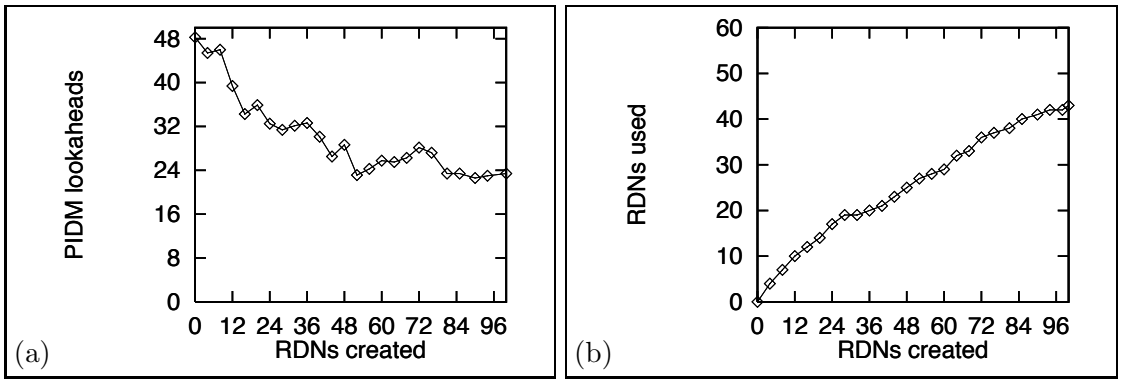


Figure 10: The use of refined decision nets during the experiment. (a) The number of times a lookahead was initiated during the PIDM process; (b) The number of learned auction sequences used during the test.

Figure 10(a) shows the number of invoked lookahead processes as a function of learning resources. As expected, we observe a reduction in the number of lookahead calls invoked

during the test. Figure 10(b) shows the number of refined nets used during the test as a function of learning resources. This number increases as expected.

5.3.3 The Effect of Learning on the Sampling Process

We expect that learning will have two main effects on the sampling process. Since learning reduces the number of lookahead invocations, we expect the number of instances that need to be generated to decrease. Furthermore, since the bidder spends less time on lookahead, it can allocate more time to model-based sampling and generate samples that are more consistent with the agents' history.

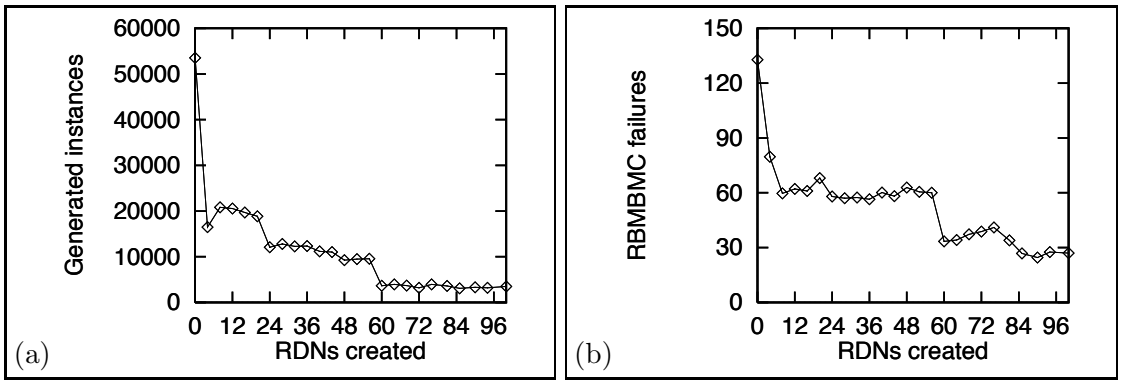


Figure 11: The effect of learning on the sampling process. (a) The total number of samples that were created during the test by the RBMBMC sampling process; (b) The number of times that the RBMBMC process failed to create a perfect sample of hands suitable to the auction.

Graph 11(a) shows the number of generated instances during the test as a function of learning resources. Indeed, the graph shows significant reduction in the number of generated instances. Figure 11(b) shows the number of inconsistent samples as a function of learning resources. This number declines as expected.

5.3.4 The Effect of Learning on the Lookhead Stage

It is more advantageous to perform lookahead in the later stages of an auction than in the earlier stages. There are two main reasons. First, the leaves of the game tree are closer, and therefore the searched tree is shallower. Second, the history is longer, and therefore the RBMBMC process has more data and is able to generate more accurate samples.

Our active learning algorithm prefers to refine nets that require more resources to process. Since states associated with earlier stages of the auction require deeper lookahead, we expect our learner to process them first.

Figure 12 shows the average length of the auction when lookahead is invoked. Indeed, the graph shows that the remaining lookahead calls are those invoked in later stages of the auction. The graph shows that the average stage is delayed by 4 calls, thus reducing the depth of the search trees by 4.

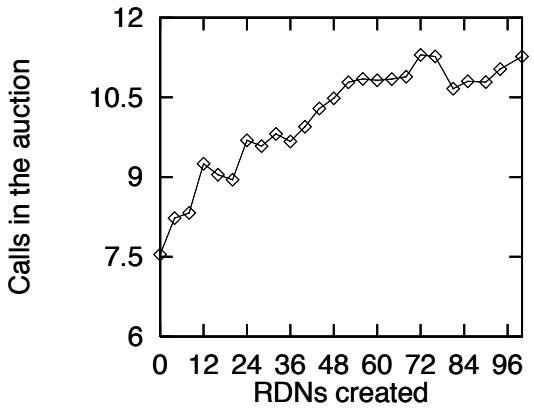


Figure 12: The average number of calls when invoking lookaheads

5.4 Parametric Study of the Learning Algorithm

To better understand the nature of the algorithms presented in Section 4, we performed a parametric study where for each experiment we kept all the parameters but one at their default values and tested how this one parameter affected the system's performance.

Since parametric study is a time-consuming process, we decided to accelerate the learning by working on a restricted set of states.[10] Such a state space was used both during training and testing. The smaller state set allows efficient learning using fewer resources than the unrestricted set would have required. The experiments in this subsection mea-

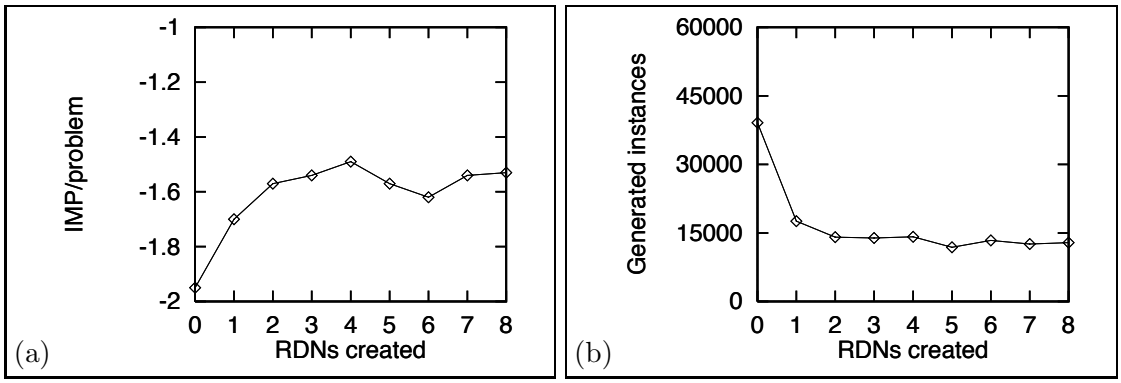


Figure 13: Result of the parametric study. (a) Average IMP/problem results for BIDI; (b) The number of instances generated by the RBMBMC sampling process during the test.

sure the two most important dependent variables: the IMP/problem score and the number of generated instances for sampling. For comparison, Figure 13 shows the basic learning curves for the restricted state space. The results are similar to those presented in Figures 9 and 11(a) except that, as expected, the learning rate is much faster for the restricted set.

10. Specifically, we concentrated on auctions that start with a single major raise, i.e., 1♡-2♡ and 1♠-2♠.

5.4.1 The Effect of Sample Size on Performance

The size of the sample used for the Monte Carlo simulation affects system performance in two ways. The larger the sample is, the better the estimation is, and therefore better decisions can be expected. However, larger samples mean that more resources are required by the PIDM algorithm, which performs lookahead search for each instance. In addition, the larger sample size also increases the resources needed for the RBMBMC process, which tests each instance for consistency with the other agents' models. Since we assume a contract framework for our decision maker, a larger resource requirement would reduce decision quality.

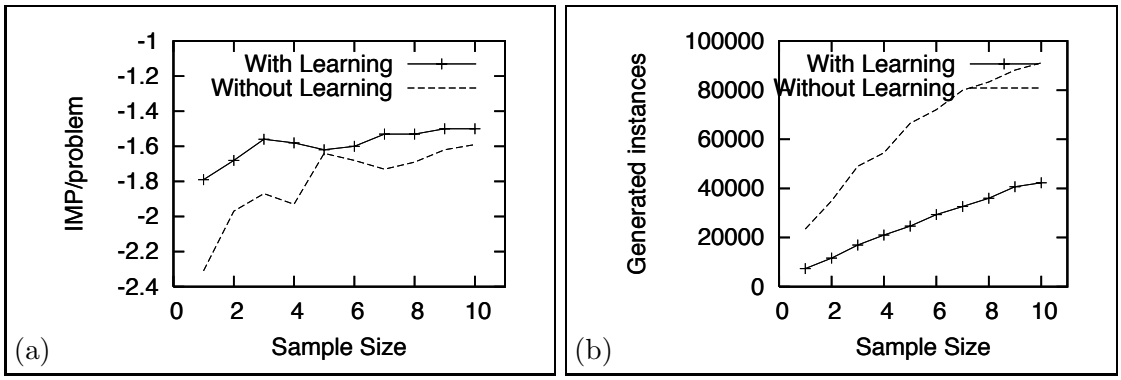


Figure 14: The effect of sample size on performance. (a) Average results (IMPs/problem) when changing the sample size parameter; (b) Average number of generated instances created when changing the sample size.

Figure 14 indeed shows that performance improves with increased sample size and the resource requirement decreases. We can also see that the main reason for the improved performance of the learning process is the speedup. The agent could have improved its performance by denser sampling instead of by learning, but the required resources would have increased significantly. Indeed, combining the results from the previous two graphs shows that a better score was achieved soon after learning (sample size = 3), using fewer resources (15,000 instances) than the best score without learning (sample size = 10, instances = 90,000).

5.4.2 How the Time Limit Benefits Learning

BIDI is basically a speedup learner. It replaces time-consuming lookahead processes with efficient decision net lookups. Therefore, as the total allocated time for the decision process decreases, we expect the benefits of the learning process to increase.

Figure 15 shows that the benefit of learning as a function of the allocated time does indeed increase significantly.

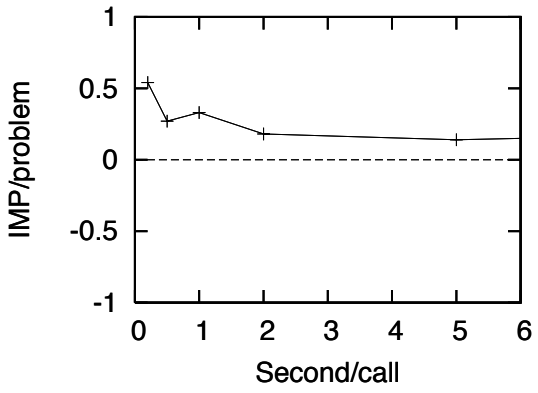


Figure 15: Δ IMP/problem (Learning - No learning) when changing the time limit for each call

5.4.3 The Benefit of Continuous Exchange of Models

In Section 4.3 we described a process where the agent and the co-agent continuously exchange their models (strategies) so that training is performed with the most updated model. We performed an experiment to test whether such an algorithm is beneficial. Since the most influential positions during learning were the first two, we repeated the initial stage of learning with the continuous exchange mechanism turned off. This means that the agents exchange their strategies only at the end of learning positions. The following table shows the algorithm is indeed beneficial.

RDN Created	Without Exchange	With Exchange
1	-1.76 IMP	-1.69 IMP
2	-1.80 IMP	-1.57 IMP

5.4.4 The Effect of Using Different Evaluation Functions

All the experiments so far were carried out under the assumption that the agent and the co-agent use the same utility function (described in Section 5.1.2) for evaluating search tree leaves. This is quite a reasonable assumption for cooperative agents. We decided, however, to test how system performance will be affected if the experiments are carried out without this assumption. We repeated the experiment, having the agent use the same utility function as in Section 5.1.2 and the co-agent a different utility function.[11]

Obviously, we expect the less accurate simulation to reduce performance (during testing only). During co-training, the agent can ask its co-agent for its expected action in each position.

Figure 16 shows the results obtained. We can see very little reduction in performance when using different functions, indicating that our method is insensitive to the particular functions used.

11. The function implements the simplified total tricks principle where 3 HCPs equal one trick and every trump equals a trick.

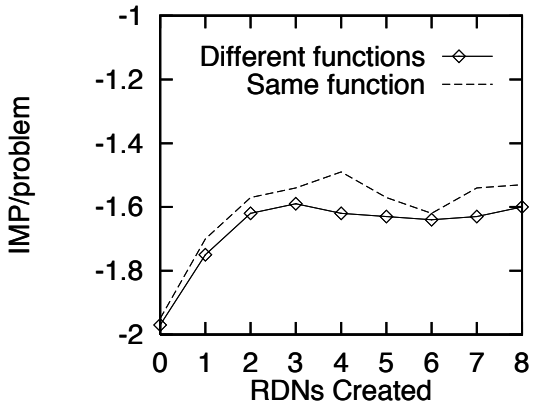


Figure 16: IMP results for two partners using the same and different utility functions

6. Related Work

The work described in this paper is related to several research areas. We divide our discussion of related work into three subsections. The first concerns reasoning in imperfect information games, the second concerns bridge bidding algorithms, and the third concerns opponent modeling.

6.1 Reasoning in Imperfect Information Games

Most of the research in the game playing community has been concentrated in perfect information games. Imperfect information games have been always considered to be more difficult due to their partial observability. One of the games that attracted significant attention is poker, mainly due to the extended research done by the University of Alberta games group. The group developed two main poker systems, Poki (Billings, Papp, Schaeffer, & Szafron, 1998; Billings, Pena, Schaeffer, & Szafron, 1999; Billings et al., 2002) and PsOpti (Billings et al., 2003). Poki evaluates hands by computing their expected return. The expected return is computed relative to a distribution over the possible opponent hands. The distribution is maintained and updated during the game according to the opponent actions. This method is somewhat analogous to our model-based Monte-Carlo sampling. In both cases the goal is to bias the sampling towards states that match the history of the game.

BIDI, however, does not maintain an explicit representation of the hands of the other agents. Instead it filters the random sample such that only states consistent with the other agents' actions are generated. The reason why we could not use Poki's method is the difference in the size of the space of hidden information. In bridge each player has a hand of 13 cards; therefore the size of each table would have been $\binom{39}{13} = 8,122,425,444$. PsOpti performs a full expansion of the game tree but reduces the size of the set of possible states by partitioning the set into equivalence classes. These abstraction operations are called bucketing. They also reduce the depth of the tree by eliminating betting rounds.

Bridge is not only an imperfect information game; it is also a multi-player game. Multi-player algorithms in imperfect information games have not received much attention. Sturte-

vant (2004) suggested performing Monte-Carlo sampling and then using the Max^n algorithm (Luckhardt & Irani, 1986). We decided not to use this approach because, during the search, the opponent agents act as if they have access to the full information — which results in very unlikely moves.

3-player Tarok is a multiplayer imperfect information game that bears some similarities to bridge, although the bidding stage in Tarok has a much lower complexity than bridge. Lustrek, Gams, and Bratko (2003) developed a program for bidding and playing in Tarok. They used Monte-Carloi sampling in combination with search. One novel aspect of their program is their use of hierarchical clustering to make the sampling more efficient. In their future work discussion, Lustrek et al. suggest that using the players actual moves to bias the sampling process may make the sampling more efficient. Indeed, in Section 3.5 we describe a method for such biased sampling.

Our method of representing and learning a selection strategy by means of a decision net is related to early work on discrimination nets (Feigenbaum & Simon, 1984). There, the net is used to organize memory and to access stored objects on the basis of external object features (stimuli). We store *sets* of objects (actions) and access them on the basis of external state features. When a conflict is found, EPAM extends the net to resolve it. We take a different approach – we accumulate a set of examples for the node containing the conflict and apply a powerful batch induction algorithm (ID3) to resolve the conflict. Our representation is similar to that of DIDO (Scott & Markovitch, 1991). There the network was also used for storing actions, and examples were also accumulated. But the conflict resolution process was much simpler than ours, extending the net by one level. The most important difference between our approach and previous approaches is *how* the net is used. While previous approaches use the net as a decision maker, we use it as a *component* of a search-based decision maker. The network is used to reduce the branching factor of the search tree, thus allowing deeper lookahead.

Refining a given selection strategy via search bears some similarity to refining book moves. For example, Hyatt (1999) uses the results of the search just after leaving a book line to modify the value associated with that line, thus affecting future selection of that book line.

6.2 Bridge Bidding

Bridge algorithms have been developed since the advent of computers. One of the first bridge bidding programs was written as early as 1953 by Ronnie Michaelson for the HEC-2 computer (Bird, 1999). There are not many published details about this program – only that it was based on the Acol bidding convention.

In 1962, Carley (1962) wrote a M.Sc. thesis[12] on a computer program that plays bridge. His program used a list of rules of thumb to make its decision during play and simple rules for the opening bid and the responses to bids. Due to memory limitation (the program was written in assembler), the performance of the program was rather limited. Carley outlined a proposal for a more sophisticated bidding algorithm that uses information about the bidding system, the bidding history, the known hand, and a priori probabilities to predict features of the concealed hands. This estimation was used to evaluate each of the plausible

12. Under the supervision of Claude Shannon.

bids along four dimensions associated with the four roles associated with bidding (asking for information, revealing information, deceiving the opponent and setting the level of bidding).

Wasserman (1970) extended Carley's idea of using bidding patterns. In addition to the opening and responding patterns, Wasserman's program could recognize several common patterns of bidding, such as take out double and ace asking.

One of the strongest rule-based systems is COBRA (Lindelöf, 1983). COBRA makes its decisions on the basis of a combination of static attributes from the player's hand and dynamic attributes revealed during the auction.

World bridge champion Mahmood Zia offered 1,000,000 pounds sterling[13] to anyone who could write a bridge program beating him (Mahmood, 1994). Levy (1989) wrote a controversial paper claiming that writing such a program is feasible using existing tools. His suggestion was to use the COBRA bidding system and a fast double dummy solver (like GIB) for the play phase. However, this system as a major drawback: its COBRA bidding component is not flexible enough to cope with different opponent systems.

Some programs, like BRIBIP (Stanier, 1975), MICROBRIDGE (Macleod, 1991), and the one developed by Ando, Kobayashi, and Uehara (2003), tried to infer the cards of the other players on the basis of their calls. The programs applied the rules backwards to generate constraints over the hidden hands of the other players and used them both for the bidding and play stages.

Jamroga (1999a) analyzed bridge bidding conventions and considered them as a form of communication. He used natural language processing methods to represent bidding rules. Gambäck et al. (1993) also discussed the role of bidding conventions as a language for communication and pointed out several examples of speech acts expressed by bidding rules. Their method for generating samples is different than ours. While we continue to generate full states and return a sample consisting of the best matched states, they perform local search on each unmatched state until a match is found. Such a method bears the risk of producing biased samples. Gambäck et al. used a trained neural network for evaluating contracts.

Ginsberg (2001) used Monte Carlo sampling for bridge and restricted the sampled set to be the states compatible with the agents' calls according to the bidding system. Our RBMBMC algorithm generalizes this approach, formalizes it, and extends it to a contract algorithm that finds the most consistent sample it can within the allocated resources.

The structure of the bidding rules we use is similar to that used by other bridge programs (Frank, 1998; Jamroga, 1999b; Schaeffer, 2000; Ginsberg, 2001). There is, however, a major advantage to our learning approach. Using a nonadaptive system to add a new state attribute would have required a major revision to the rule base. Our system, on the other hand, requires only the definition of the attribute. The decision net inducer will automatically use the new attribute whenever necessary.

Because our focus is on bidding, we limit our discussion of previous works to this aspect of the game. We would like, however, to give a brief summary of works concerned with the playing stage. As mentioned above, the first programs used expert rules for the playing stage. Smith et al. (1998a, 1998b, 1996) used the HTN model of hierarchical planing to decide on the play actions. They made abstractions of sequences of actions and used

13. Unfortunately, this bet was canceled due to the progress in computer play.

these abstractions to significantly reduce the size of the search tree. Frank (1998) developed algorithms for playing a particular suit against an opponent with optimal defense. GIB (Ginsberg, 2001) used Monte-Carlo sampling in combination with partition search to reduce the size of the game tree.

6.3 Opponent Modeling

PIDM, the main algorithm in this paper, builds and extends previous work on agent modeling in general and opponent modeling in particular.

There are several main approaches for using agent models. One approach associates a probability distribution with the set of possible actions for each possible state (Jansen, 1992; Billings et al., 1999). Such an approach is possible when the state space is sufficiently small. For larger spaces, several works extend the above approach by generalizing over states (Billings et al., 2003; Sen & Arora, 1997; Donkers, 2003).

PIDM is based on the simulation approach, where models of other agents' strategies are used to simulate their decision making in order to predict their actions (Iida et al., 1993, 1994; Donkers, Uiterwijk, & van den Herik, 2001; Donkers, 2003; Carmel & Markovitch, 1998, 1996b). PIDM extends the previous works by allowing any combination of opp-agents and co-agents and by combining search with model-based Monte Carlo sampling for partially observable states.

Jamroga (2001) extends the idea of opponent modeling to imperfect information games. He proposes the *reasonably good defense* model: "If nothing suggests the contrary, the opponent should be assumed capabilities similar to the player. Thus, MAXs knowledge and skills, and his model of MIN should be symmetrical."

One major issue with this model that has received little attention is that of the resources available for simulation. Assume that an agent performing the simulations has computational resources similar to those of the other simulated agents. Then, there will be a smaller amount of resources available to that agent at each node than would have been used by the simulated agent in a real interaction. Thus, performance-reducing prediction errors are likely to occur (Gao, Iida, Uiterwijk, & van den Herik, 1999; Carmel & Markovitch, 1996a). One way to overcome this problem is to model only certain aspects of the agents' strategy. Markovitch and Reger (2005) learn the opponent's weaknesses and bias the search towards states where the opponent's strategy is expected to be weak.

In this work, we address the resource problem in two ways. First, the PIDM algorithm includes a resource allocation component for each of the simulations, and for the sampling process. Here we use a rather simplistic resource allocation strategy. In Section 7 we describe our planned enhancements for this component.

Our more significant contribution in this area is in learning and using the selection strategy. This learning mechanism can be viewed as a speedup learner where a rough approximation of the other agents' decision procedure is learned and expressed by induced decision nets.

Some of the above works include algorithms for learning opponent agent models on the basis of their past behavior (Billings et al., 2003; Davidson, Billings, Schaeffer, , & Szafron, 2000; Sen & Arora, 1997; Bruce, Bowling, Browning, & Veloso, 2002; Markovitch & Reger, 2005). The learning framework presented here focuses on adapting to co-agents.

This allows us to incorporate mechanisms that would not have been possible with opponent agents. These mechanisms include a co-training process where the agents continuously exchange their adapted selection strategies and consult one another about their expected behaviors during training.

Iida (2000) presented the pair playing cooperative search (PPC search) for complete information environments. Like Iida, we also concentrate on modeling the partner for increased accuracy during our lookahead searches. However, the PIDM search is different in several aspects. First, the search is done in a partially observable environment. Second, there is no restriction on the number of co/op agents or on the order of the turns. Finally, we use simulation to predict the other agents' moves instead of building a minimax tree.

7. Discussion

This paper presents new algorithms for decision making and cooperation in bridge bidding. While the motivation, application and empirical evaluation of the algorithms were in the context of bridge, the framework can be used for model-based decision making and learning in multi-agent systems in partially observable environments. The main contributions of this work are twofold.

First, we introduce a general algorithm, PIDM, for model-based decision making. The algorithm employs *model-based Monte Carlo sampling* to instantiate partial states and model-based lookahead to evaluate alternative actions. This method of sampling has not been used before in general, and in bridge bidding in particular.

Second, we present a learning framework for co-training of cooperative agents. The agents refine their selection strategies during training and exchange their refined strategies. The refinement is based on inductive learning applied to examples accumulated for classes of states with conflicting actions. The continuous exchange of strategies allows the agents to adapt their strategies to those of their co-agents. This algorithm differs from existing bidding algorithms in that it allows an agent to adapt to its partner.

The learning process speeds up decision making by generating more refined selection strategies. These lead, in turn, to lookahead trees with smaller branching factors. When the selection strategy manages to select exactly one action, the agent avoids lookahead altogether. Learning results in lower resource requirements, and the freed resources can then be used for better decision making.

One important aspect of this research is the great emphasis placed on resource allocation. We assume the framework of bounded rational agents with limited resources available for decision making. Therefore, resource allocation must be carefully planned. Our general PIDM algorithm allows the resources to be distributed flexibly between the sampling process and the simulations. Our *Resource-Bounded Model-Based Monte Carlo* sampling process is a contract algorithm that finds the best sample it can within the allocated resources. This sample may not be consistent with *all* the past actions of the other agents.

Currently, we use a simple resource allocation scheme to decide how to distribute the resources between sampling and lookahead within the lookahead tree. In future research we intend to develop more sophisticated resource allocation schemes. For example, we could employ a learning framework such as the one described by Markovitch and Sella (1996), where inductive learning is used to identify states that can benefit from larger resource

allocation. Such a framework can be employed both for allocating resources for each PIDM simulation node and for allocating them between the PIDM processes within a sequence of interactions. Likewise, we can improve the resource allocation for the different actions inside the PIDM function. Instead of uniform resource allocation for all the actions, we can plan to allocate more resources to the more complicated actions.

In this work we concentrated on adapting to co-agents. The co-agents continuously refine their selection strategy during learning so that co-agents with different selection strategies can adapt to and better cooperate with each other. Co-agents can also cooperate within the PIDM process during learning. For modeling opp-agents, we cannot assume cooperation during training, and other techniques, such as those developed by Carmel and Markovitch(1998), and by Sen et al.(1998), should be used.

The framework presented in this paper was implemented and empirically tested in the domain of bridge bidding. The experiments show two significant improvements in the achievements of the bidder: one in the actual score achieved and the other in the resources invested during the process. We also conducted a parametric study of different variables of the algorithm.

This framework is general and not limited to the bridge domain. It can be applied to other partially observable MAS domains with any number of opp-agents and co-agents by defining the following components:

1. A set of informative features over states. These features will be used by the learning process to refine the selection strategy.

2. An optional initial selection strategy. There is nothing in our algorithm that requires such an initial strategy, but supplying it can significantly improve the learning process.

3. A utility function for evaluating states.

We believe that the algorithms presented in this paper enhance our ability to build agents capable of adapting to their co-agents in order to cooperate against other agents with conflicting goals.

Acknowledgements

We would like thank Irad Yavneh for his helpful remarks during this study.

This study was supported by the Marco and Louise Mitrani Memorial Fellowship, funded by Mr. and Mrs. Belmonte, and generous financial assistance from the Technion.

Appendix A. Hand Attributes Used in BIDI

The decision net nodes, either given or learned, use state constraints to determine the membership of states in nodes. The constraints are based on state features which are usually domain-dependent and require expert knowledge. The version of BIDI used for the experiment described in this paper uses a set of 35 bridge-specific feature classes. Each of the feature classes has instances for various combinations of suits. Some of the features apply some function to a single suit of the hand, some to a pair of suits, and some to the whole hand.

In the following tables we list the 35 feature classes. For each feature class, we mark the variations this class takes. The possible variations are:

1. $S = \{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$. One feature instance for each of the four suits.
2. $A = \{\spadesuit, \heartsuit, \diamondsuit, \clubsuit, \spadesuit \cup \heartsuit \cup \diamondsuit \cup \clubsuit\}$. Five instances – one for each suit and one for the whole hand.
3. $TS = \{\langle s_1, s_2\rangle | s_1, s_2 \in S\}$. One feature instance for each pair of suits, the first of which is the trump suit, and the second, which can be any suit (including the trump). 16 feature instances in all.
4. $TA = \{\langle s_1, s_2\rangle | s_1 \in S, s_2 \in A\}$. One feature instance for each pair of suits, the first of which is the trump suit, and the second, which can be any suit (including the trump) or the whole hand. 20 feature instances in all.
5. T^*A. Same as above but s_1 can also be NT.
6. $2S = \{\langle s_1, s_2\rangle | s_1, s_2 \in S, s_1 \neq s_2\}$. One feature instance for each combination of two different suits (ordered). 12 feature instances in all.

A.1 Attribute Classes for Evaluating High Cards

Attribute	Description
HCP_A	The weighted count of High Card Points where A=4; K=3; Q=2 & J=1. Range is [0,10] for each suit and [0,37] for the whole hand
CONTROLS_A	The weighted count of aces and kings: A=2 & K=1. Range is [0,3] for each suit and [0,12] for the hand
HC_A	The count of aces, kings and queens. Range is [0,3] for a single suit and [0,12] for the hand.
HONOR_A	The count of cards higher than 9. Range is [0,5] for a single suit and [0,13] for the hand.
ACE_A	The count of aces. Ranges are [0,1] and [0,4].
KING_A	The count of kings. Ranges are [0,1] and [0,4].
QUEEN_A	The count of queens. Ranges are [0,1] and [0,4].
JACK_A	The count of jacks. Ranges are [0,1] and [0,4].
TEN_A	The count of tens. Ranges are [0,1] and [0,4].
RKCB_S	The count of aces in the whole hand + the count of kings in the suit. Range is [0,5].

A.2 Attributes for Evaluating the Distribution of the Hand

Attribute	Description
BAL	How balanced is the hand? (0=balanced-6=freak)
DP_{TA}	Points for short non-trump suits. 1 point for each doubleton, 2 points for each singleton and 3 points for void. Ranges are [0,3] and [0,9].
DIF_{2S}	The difference in length between 2 suits. Range is $[-13,13]$.
SUITS	The number of suits that are longer than 3. Range is [1,3].
$SUITLEN_S$	The length of the suit. [0,13].
$SUITLEN_{2S}$	The total length of the two suits. [0,13].
LMAJ	The length of the longer major suit. [0,13].
LMIN	The length of the longer minor suit. [0,13].
$LONGEST_S$	Is the suit the longest in the hand? [0,1].
$LONG_S$	Is the suit not shortest in the hand? [0,1].
$FIRSTBID_S$	Is the suit appropriate for opening in 5M system? [0,1].
$UPLINE_{2S}$	Is it appropriate to bid s_1 when the last bid suit is s_2? [0,1].

A.3 Attributes for Evaluating the Quality of the Suits

Attribute	Description
$BIDDBLE_S$	The number of times we can bid the suit throughout the auction. [0,8].
$INTER_A$	The weighted count of the intermediate cards (T = 2, 9 = 1, 8 = 0.5). Ranges are [0,3.5] for a suit and [0,14] for the hand.
$LOSERS_{TA}$	Expected number of losers in s_2 (a suit or the hand) when s_1 is played as trump. Ranges: [0,3] and [0,12].
$STOP\text{-}NT_S$	Stopper quality of the suit when playing in NT. [0=no stopper - 4=double stopper].
$STOP\text{-}TRUMP_{TS}$	Stopper quality of the s_2 when playing with trump s_1. [0=no stopper - 4=double stopper].
$QLTY_S$	A general quality assessment of the suit [0=bad-9=excellent].

A.4 Attributes for Evaluating the Quality of the Hand

Attribute	Description
NTLP	Lindelöf points (1983) for the purpose of NT playing.
RATIO	The ratio between the sum of HCP of the two longest suits and the total HCP of the whole hand. [0,1].
$POINTS_{TA}$	HCP_{TA} + DP_{TA}.
$UNLOS_S$	Estimated number of losers in the hand without partner support for the longest suit.

A.5 Dynamic Attributes

Attribute	Description
COVER_{TA}	Estimated number of cover cards in s_2 when s_1 is played as trump. [0-13].
FAST_{T*A}	Estimated number of fast losers (that can be cashed by the opponent) in s_2 when s_1 is played as trump.
LP	Lindelöf points.

Appendix B. On-the-Fly Decision Net Construction

As mentioned in Section 5.1.2, BIDI is capable of converting rule-based bidding systems to decision nets.

B.1 Structure of the Bidding Rules

Each bidding rule has three parts:

1. A set of conditions about the auction history.
2. The call (action) associated with the bidding rule.
3. A set of constraints about the bidder's hand.

During bidding, if the auction conditions and the hand constraints match the current auction and hand, then the associated call is added to the list of calls suggested for the current state.

During the sampling process, if the auction conditions are matched, and the associated call is the one made by another player, then the set of constraints about that player's hand is inferred.

The auction conditions are represented by rules that contain three parts:

1. The pattern of the previous calls of the auction.
2. The conventions used by the players during the auction.
3. The previously inferred information about the players' hands.

The third part of the rule contains a list of pairs. Each pair consists of a constraint list and a number representing a priority. The priority helps to resolve conflicts. When used for bidding, the priority helps decide between a set of recommended calls. When used for inference, the priority helps to resolve conflicting constraints that arise from different rules. In addition, the third part of the rule contains the name of the convention used.

Figure 17 shows an example for a bidding rule recommending an opening call of MULTI-2♢.[14]

Negative bidding rules tell the players about actions they *should not* take. Those actions will later be removed from the list of actions suggested by the positive rules. For example,

14. The 2♢ opening used by BIDI describes weak two in major, strong in any suit, or 4441 distribution with a strong hand.

Name	$2\diamondsuit$ opening bid	
Auction Conditions	Call Pattern	$(\text{Pass})^*$
	Convention Used	Null
	Revealed Information	Null
Action	$2\diamondsuit$	
Revealed Information	Constraints	$\langle 100, (5 \leq HCP \leq 10, \text{SUITLEN}(\spadesuit) = 6, \text{SUITLEN}(\heartsuit) < 4, \text{LOSERS}(\spadesuit, \spadesuit) \geq 2)\rangle$
		$\langle 100, (5 \leq HCP \leq 10, \text{SUITLEN}(\heartsuit) = 6, \text{SUITLEN}(\spadesuit) < 4, \text{LOSERS}(\heartsuit, \heartsuit) \geq 2)\rangle$
		$\langle 110, (HCP \geq 18, \text{SUITLEN}(\spadesuit) \geq 6 \text{ Suits} = 1, 4 \leq \text{LOSERS}(\spadesuit, ALL) \leq 5)\rangle$
	Sets	$\langle 110, (HCP \geq 18, \text{SUITLEN}(\heartsuit) \geq 6 \text{ Suits} = 1, 4 \leq \text{LOSERS}(\heartsuit, ALL) \leq 5)\rangle$
		$\langle 110, (HCP \geq 18, \text{SUITLEN}(\diamondsuit) \geq 6 \text{ Suits} = 1, 4 \leq \text{LOSERS}(\diamondsuit, ALL) \leq 5)\rangle$
		$\langle 110, (HCP \geq 18, \text{SUITLEN}(\clubsuit) \geq 6 \text{ SUITS} = 1, 4 \leq \text{LOSERS}(\clubsuit, ALL) \leq 5)\rangle$
		$\langle 130, (HCP \geq 20, \text{SUITS} = 3, \text{LMAJ} = 4, \text{LMIN} = 4)\rangle$
	Conventions	MULTI-2D

Figure 17: An example of a bidding rule

Name	No passing in GF situations	
Auction Conditions	Call Pattern	$(\text{any})^* - XY - Pass$ $XY < 4\heartsuit \cup XY \neq 3N$
	Convention Used	We - GAME-FORCE
	Revealed Information	Null
Action	Pass	
Revealed Information	Constraint Sets	$\langle -1, \emptyset\rangle$
	Conventions	NULL

Figure 18: An example of a negative bidding rule

Figure 18 shows a simple rule that prevents a pass before the game level is reached, in game-forcing positions.

B.2 Converting Rules to Decision Nets

As discussed in Section 5.1.2, BIDI converts rules to decision nets on the fly whenever necessary. Given a current auction history h, BIDI first collects all the rules matching h. The collected rules are then converted to a set of triplets of the form $\langle Call, SC, Priority\rangle$ according to the second and third parts of the rules.

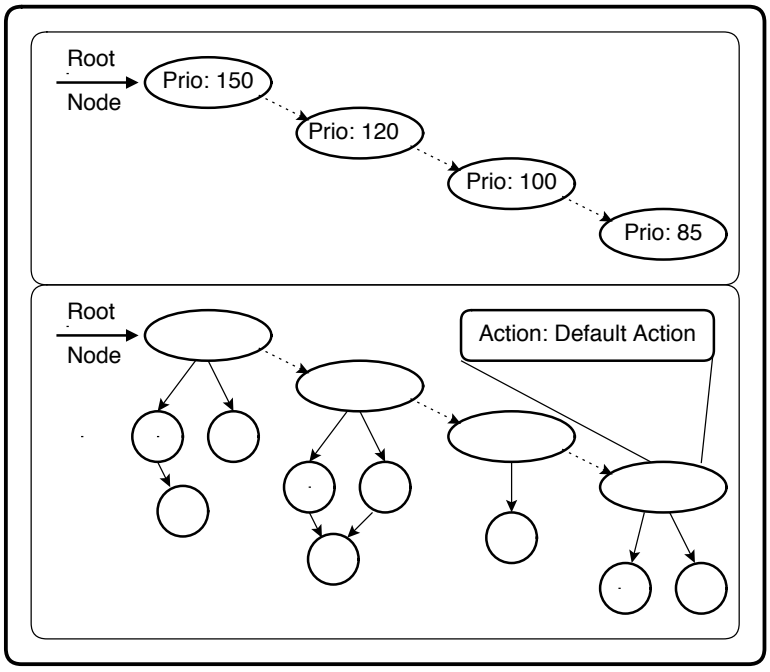


Figure 19: The construction stages of the on-the-fly decision net

The net is created in three phases:

1. A skeleton net is built according to the triplets' priorities. The net is built as a chain, using the *else* links, from the highest priority to the lowest. The node with the highest priority becomes the root node of the net. The upper part of Figure 19 illustrates this stage.

2. The triplets are converted into nodes. The state constraints part of the node (SC) is taken from the constraint list of the triplet. The action part of the node is taken from the triplet's call. Each of the nodes is propagated down the net and placed according to its SC and its priority.

3. The set of default actions is the set of all actions minus the set of actions prohibited by negative rules. This set is inserted into the last node in the skeleton net (the one with lowest priority). The lower part of Figure 19 illustrates the state of the net at the end of this stage.

Figure 20 illustrates the possible options in the recursive propagating algorithm. The input for the function is new nodes, created according to the triplets. The new nodes are propagated recursively down the tree, one after the other.

First, the nodes are propagated on the skeleton net, using the *else* links, according to the priority of the nodes (case 1). Afterward, each new node's SC is compared to the proper net node's SC. If those SCs are equal (case 2), the nodes are merged by joining their sets of actions.

If the new node's SC is a sub-SC of the net's node, we have to check the relations between the SC and the node's children. There are two possibilities in this case:

1. The new node's SC is not a sub-SC of any children. The new node then becomes a new child of the node. If one or more current children of the net's node are sub-SCs of the new node, those children move from the net node to the new nodes (case 4); otherwise the process stops (case 3).

2. The new node is a sub-SC of one or more children. In this case, the new node is recursively propagated down to all of those children (case 5).

When the new node is not a sub-SC of the net's node, the process stops without having done anything (case 6). Because the root node's SC is $\emptyset$ and the skeleton net contains every possible priority, we can be sure that every new node will fit in at least one place. The formal algorithm is given in Figure 21.

References

Ando, T., Kobayashi, N., & Uehara, T. (2003). Cooperation and competition of agents in the auction of computer bridge. *Electronics and Communications in Japan*, *86*(12).

Belladonna, G., & Garozzo, B. (1975). *Precision & Super Precision Bidding*. Cassell, London.

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., & Szafron, D. (2003). Approximating game-theoretic optimal strategies for full-scale poker. *International Joint Conference on Artificial Intelligence (IJCAI)*, 661–668.

Billings, D., Davidson, A., Schaeffer, J., & Szafron, D. (2002). The challenge of poker. *Artificial Intelligence*, *134*(1-2), 201–240.

Billings, D., Papp, D., Schaeffer, J., & Szafron, D. (1998). Opponent modeling in poker. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 493–499.

Billings, D., Pena, L., Schaeffer, J., & Szafron, D. (1999). Using probabilistic knowledge and simulation to play poker. In *AAAI/IAAI*, pp. 697–703.

Bird, R. (1999). Btm's first steps into computing. *Computer Resurrection: The Bulletin of the Computer Conservation Society*.

Bruce, J., Bowling, M., Browning, B., & Veloso, M. (2002). Multi-robot team response to a multi-robot opponent team. In *Proceedings of IROS-2002 Workshop on Collaborative Robots*.

Carley, G. (1962). A program to play contract bridge. Master's thesis, Dept. of Electrical Engineering, Massachussets Institute of Technology, Cambridge, Massachussets.

Carmel, D., & Markovitch, S. (1996a). Learning and using opponent models in adversary search. Tech. rep. CIS9609, Technion.

Carmel, D., & Markovitch, S. (1996b). Incorporating opponent models into adversary search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 120–125, Portland, Oregon.

Carmel, D., & Markovitch, S. (1998). Model-based learning of interaction strategies in multi-agent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, *10*(3), 309–332.

Cohen, L. (2002). *Bidding Challenge*. Master Point Print, Toronto.

Davidson, A., Billings, D., Schaeffer, J., , & Szafron, D. (2000). Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence*, pp. 1467–1473.

Donkers, H. (2003). *Nosce Hostem - Searching with Opponent Models*. Ph.D. thesis, Universiteit Maastricht.

Donkers, H., Uiterwijk, J., & van den Herik, H. (2001). Probabilistic opponent-model search. *Information Sciences*, *135*(3-4), 123–149.

Ekkes, O. (1998). The gib zone (7): The par contest. *Imp Magazine*, *9 Oct/Nov*(7).

Feigenbaum, E., & Simon, H. (1984). EPAM-like models of recognition and learning. *Cognitive Science*, *8*(43), 305–336.

Finkelstein, L., & Markovitch, S. (1998). Learning to play chess selectively by acquiring move patterns. *ICCA Journal*, *21*(2), 100–119.

Frank, I. (1998). *Search and Planning Under Incomplete Information*. Springer.

Gambäck, B., Rayner, M., & Pell, B. (1993). Pragmatic reasoning in bridge. Tech. rep. 030, Sri International Cambridge Computer Science Reasearch Center.

Gao, X., Iida, H., Uiterwijk, J., & van den Herik, H. J. (1999). A speculative strategy. *Lecture Notes in Computer Science*, *1558*, 74–92.

Ginsberg, M. L. (2001). Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*.

Goldman, B. (1978). *Aces Scientific*. Max Hardy.

Hyatt, R. M. (1999). Book learning – a methodology to tune an opening book automatically. *International Computer Chess Association Journal*, *22*(1), 3–12.

Iida, H., Uiterwijk, J. W., & van den Herik, H. (2000). *Cooperative Strategies for Pair Playing*, pp. 189–201.

Iida, H., Uiterwijk, J., van den Herik, H., & Herschberg, I. (1993). Potential applications of opponent-model search. *ICCA journal*, 201–208.

Iida, H., Uiterwijk, J. W., van den Herik, H., & Herschberg, I. (1994). Potential applications of opponent-model search. *ICCA journal*, 10–14.

Jamroga, W. (1999a). Modelling artificial intelligence on a case of bridge card play bidding. In *Proceedings of the 8th International Workshop on Intelligent Information Systems*, pp. 267–277, Ustron, Poland.

Jamroga, W. (1999b). Modelling artificial intelligence on a case of bridge card play bidding. In *Proceedings of the 8th International Workshop on Intelligent Information Systems (IIS'99)*, pp. 267–277, Warsaw, Poland.

Jamroga, W. (2001). A defense model for games with incomplete information. *Lecture Notes in Computer Science*, *2174*, 260–274.

Jansen, P. J. (1992). *Using Knowledge about the Opponent in Game-tree Search*. Ph.D. thesis, Carnegie Mellon University.

Klinger, R. (2001). *Modern Losing Trick Count*. Orion.

Levy, D. (1989). The million pound bridge program. *Heuristic Programming in Artificial Intelligence - First Computer Olympiad*, 93–105.

Lindelöf, E. (1983). *COBRA - The Computer-Designed Bidding System*. England: Victor Gollancz.

Luckhardt, C. A., & Irani, K. B. (1986). An algorithmic solution of n-person games. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-86)*, pp. 158–162, Philadelfia, PA.

Lustrek, M., Gams, M., & Bratko, I. (2003). A program for playing tarok. *ICGA Journal*, *26*(3).

Macleod, J. (1991). Microbridge - a computer developed approach to bidding. *Heuristic Programming in Artificial Intelligence - the First Computer Olympiad*, 81–87.

Mahmood, Z. (1994). *Bridge My Way*. Natco Press.

Markovitch, S., & Reger, R. (2005). Learning and exploiting relative weaknesses of opponent agents. *Autonomous Agents and Multi-agent Systems*, *10*(2), 103–130.

Markovitch, S., & Sella, Y. (1996). Learning of resource allocation strategies for game playing. *Computational Intelligence*, *12*(1), 88–105.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 211–229.

Schaeffer, J. (2000). The games computers (and people) play. In *AAAI/IAAI*, p. 1179.

Scott, P. D., & Markovitch, S. (1991). Representation generation in an exploratory learning system. In Fisher, D., & Pazzani, M. (Eds.), *Concept Formation: Knowledge and Experience in Unsupervised Learning*. Morgan Kaufmann.

Scott, P. D., & Markovitch, S. (1993). Experience selection and problem choice in an exploratory learning system. *Machine Learning*, *12*, 49–67.

Sen, S., & Sekaran, M. (1998). Individual learning of coordination knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, *10*(3), 333–356.

Sen, S., & Arora, N. (1997). Learning to take risks. In *AAAI-97 Workshop on Multiagent Learning*, pp. 59–64.

Shi, J., & Littman, M. (2001). Abstraction models for game theoretic poker.. *Computers and Games*, 333–345.

Smith, S. J. J., Nau, D., & Throop, T. (1996). A planning approach to declarer play in contract bridge. *Computational Intelligence*, *12*(1), 106–130.

Smith, S. J. J., Nau, D., & Throop, T. (1998a). Computer bridge: A big win for AI planning. *AI Magazine*, *2*(19), 93–105.

Smith, S. J. J., Nau, D., & Throop, T. (1998b). Success in spades: Using ai planning techniques to win the world championship of computer bridge. In *Proceedings of AAAI*, pp. 1079–1086.

Stanier, A. M. (1975). Bribip: A bridge bidding program. In *The Proceedings of the 1975 International Joint Conference on AI*, pp. 374–378.

Sturtevant, N. (2004). Current challenges in multi-player game search. In *Proceedings of Computers and Games*, Israel.

Wasserman, A. (1970). Realization of a skillful bridge bidding program. *Fall Joint Computer Conference, Houston, Texas.*

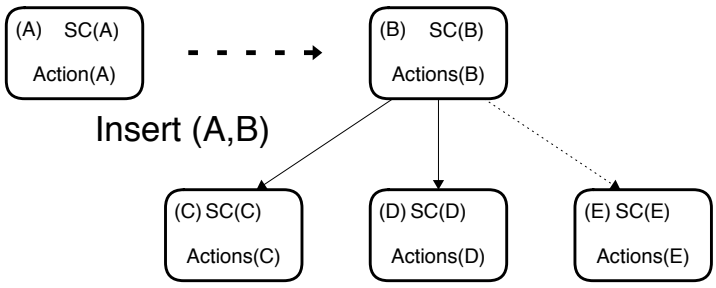
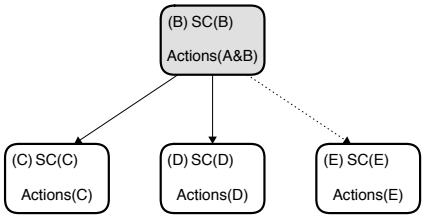
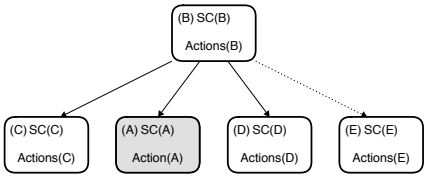
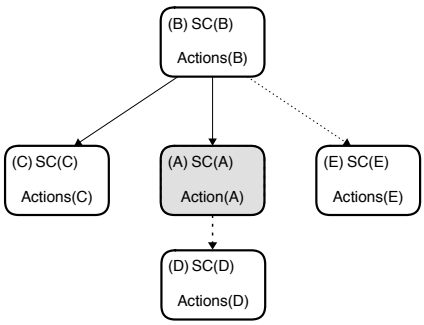
<table>
<tr><td colspan="2"> 
</td></tr>
<tr><td>(1) $Priority(A) < Priority(B)$</td><td>$Insert(A, E)$</td></tr>
<tr><td>(2) $SC(A) \equiv SC(B)$</td><td> 
</td></tr>
<tr><td>(3) $SC(A) \subset SC(A) \subset SC(B)$
$SC(A) \not\subseteq SC(C) \cup SC(A) \not\subseteq SC(D)$</td><td> 
</td></tr>
<tr><td>(4) $SC(D) \subset SC(A) \subset SC(B)$
$SC(A) \not\subseteq SC(C)$</td><td> 
</td></tr>
<tr><td>(5A) $SC(A) \subseteq SC(C) \cup SC(A) \subseteq SC(D)$
(5B) $SC(A) \subseteq SC(C) \cup SC(A) \not\subseteq SC(D)$</td><td>$Insert(A, C)$ & $Insert(A, D)$
$Insert(A, C)$</td></tr>
<tr><td>(6) $SC(A) \not\subseteq SC(B)$</td><td>$Null$</td></tr>
</table>

Figure 20: Possible way to propagate new nodes into a decision net during the construction process

```
Build-Decision-Net (triplets) → net
Let triplet be ⟨Call, SC, Priority⟩
*** building the skeleton net. ***
    Priorities ← find all the possible priorities of the options.
    For each Priority in Priorities
        Node ← create-node (priority, null-view, no-actions, no children, no attributes)
    push node into nodes.
    sort nodes according to priority.
    for each node in nodes
        node-attribute ← next node
*** Inserting the triplets ***
    Net ← First node.
    for each triplet in triplets
        node ← CREATE-NODE(triplet-priority, triplet-SC, triplet-action,
                no children, no attributes)
        INSERT-NODE (net, node)
*** Inserting the default actions. ***
    node-actions(last nodes) ← default-actions.
    return net.

INSERT-NODE(net, node) → Boolean
    if net-priority > node-priority then (case 1)
        INSERT-NODE(net-attribute, node)
    elseif net-SC = node-SC then (case 2)
        push node-action into net-actions.
        return TRUE.
    elseif net-view > node-view then (case 3-5)
        for each child in net-children
            Insert-Node (child, node)
        if exists child such child-SC ≥ node-SC then (case 5)
            return TRUE
        else (case 3-4)
            common-child ← FIND-COMMON-CHILDREN(net, node)
            move common-child from net-children to node-children
            push node into net-children
            return TRUE
    else (case 6)
        return NIL
```

Figure 21: The process of On-the-fly construction of decision nets.