A Bayesian Net Inference Tool for Hidden State in Texas Hold'em Poker

Michael A Terry
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar Street

Brian E Mihok
The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, MA 02139-3563

Abstract

We present the design, implementation, and evaluation of a tool for performing common inference tasks in the game of Texas Hold'em Poker. The design was based upon the generation of inferences from a Bayesian Net, which was generated from a combination of expert knowledge and Machine Learning of a large archive of previously played hands. Combining this data and expert knowledge of the problem domain, we generate a large database training set in terms of our variables of interest. Using the freely available Matlab Bayesian Net toolkit, we apply domain knowledge to generate topology for the Bayesian Nets, and use Matlab Bayes Toolkit's Maximum Likelihood parameter estimation to learn the parameters for this structure. Finally, we evaluate our sytem's ability to predict an opponent's hole cards in Texas Hold'em given our representation.

Introduction

One of the key goals in Artificial Intelligence is to create cognitive systems that can perform human-competitive reasoning tasks. When a particular problem domain requires that inferences be made with limited information, an additional element of difficulty is introduced. In real-world problems, one typically does not have all of the information necessary to make a completely informed decision, and must either perform an action to receive that information, or make an educated guess regarding the best plan of action. A number of challenges arise when attempting to develop reasoning systems which emulate this type of intelligence.

One of the main challenges is formulating a well-defined problem and appropriate computational representation within which to perform experiments. Games have traditionally provided many problems in Computer Science with an excellent test-bench for new ideas, because they can be clearly described in terms of set rules and well-defined sets of actions. Games of imperfect information (hidden state) provide models for an even broader range of problems, because they address the issues of uncertainty typically associated with most difficult inference tasks.

In addition, most games have expert strategists, which can provide additional insight into the design and evaluation of intelligent systems. As we will show in this paper, this incorporation of domain knowledge is an important theme amongst many techniques in Artificial Intelligence. Harnessing the knowledge accumulated by humans into the design process is often critical, and provides a great deal of benefit to the end-product.

A recent worldwide explosion in the popularity of Poker has stimulated interest in the game in many areas of Computer Science research. The most common form of poker, Texas Hold'em, is a game of simple rules and complex strategies. As in most cards games, much of the state information is hidden to the player. In addition, because there are complex interactions between multiple adversaries, it provides a great application for reasoning methods associated with multi-agent environments. Furthermore, because it is a zero-sum game of chance (odds) and clear rewards (money), one could conceivably extend many of the concepts for reasoning in Hold'em to disciplines such as Economics.

This paper presents our efforts to design, implement and evaluate a system for Bayesian inference in the domain of Limit Hold'em Poker. To address these tasks of reasoning under uncertainty, we investigate the possibility of using probabilistic inference methods that fall under the umbrella of Statistical Learning. These techniques have been used in a number of problem domains, and demonstrate a great deal of success and recent advances in Computer Science research. The next section of this paper describes the background information related to poker strategy and the statistical inference methods we have incorporated into our design. This is followed up by the requirements and motivation for our systems. Next, the design tradeoffs associated with our particular implementation are presented. Finally, we describe the results from our inference system on a number of validation tests.

Background

Games

Throughout this history of Games and Artificial Intelligence, a number of significant benchmarks have been achieved by computers that can produce human-competitive strategies. In the textbook example, Chess has been shown to be 'solvable' through exhaustive search methods. The degree of success of this approach is highly determined by the overall computational power of the hardware platform. However, in comparing these search algorithms (i.e. minimax) with the nature of human strategic reasoning, the parallels between the two are minimal. Humans reason according to relationships between different types of moves and positions, and determine a strategy accordingly. This is entirely different from machine strategy, which usually involves some calculations/evaluations and the use of heuristics. Essentially, humans are able reduce the search space of their exploration, based upon domain knowledge with seemingly little effort. More recent efforts have shown the possibility of capturing domain knowledge into search of a solution space (Koza *et al.* 2003). Although these and similar techniques show promise for performing a seemingly more intelligent search for solutions, the problem of representation of this domain knowledge remains a significant challenge. In addition, when a particular problem domain introduces elements of uncertainty, the level of complexity of this representation further increases. Poker is a game of uncertain information and complex strategy. Although the rules and action possibilities are very simple, good human strategy incorporates factors such as deception(i.e. bluffing), odds, and psychology. Thus, brute-force tactics would not be sufficient for determining an effective strategy. Unlike chess, algorithms for poker strategy must incorporate a concise representation of a great deal of domain knowledge, much of which is related to the uncertainty of the game. However, information about rules, strategy information, and other nuances of a particular problem domain are difficult to represent in machines using the programming techniques and data structures of modern digital computers. One intuitive way this information is conveyed in machine learning is through the description of game factors as variables. In games, a variable can represent a wide range of concepts including those related to the physical, conceptual, temporal realms. Rules of a game, a player's actions, and relevant state information can all be captured through the declaration of domains and variables. In machine learning, examples of actions in a game are presented in terms of these variables, and algorithms are used to 'learn' the patterns or relationships between these variables. When uncertainty is involved, Bayesian Nets provide a way to model probabilistic information, given some evidence or observations about the situation. These technique apply very well to the domain of poker, since the variables of interest are closely tied to the uncertainty of information.

Poker Strategy

In poker the goal is to maximize the expected value over all of your actions, so that a player sees the maximum long-term winning over a number of hands. Given a wide variety of information available at every stage in the game,session, or tournament, one must make their decisions to maximize this expectation. In live action poker, not only does a player have information about the game actions of an opponent, he also has a wide variety of physical cues, or ’tells’ regarding an opponent’s actions from which to infer information about the state of the game. These cues are critical for live action play, but nearly absent in Internet poker play. Fortunately for a computer inference strategy, this means a reduction in the number of variables it can take into account.

Good poker strategy incorporates numerous levels of thinking regarding the multitude of variables available to a player. Every action in the game releases more information about the hidden state of the game(opponents’ hole cards), and the player must use this information to construct a model about the state of the game. Our goal in this project is to generate a representation of the hidden state in Hold’em, and construct an inference tool to extract this information from betting patterns in a database of statistics from previously played hands.

To illustrate the complexity of inference tasks an intermediate player may consider in Hold’em, consider this simplified two-player scenario between the Hero and the Villain, a habitual bluffer.

Pre-flop actions:

Hero holds the Queen of Clubs and King of Clubs.

-reasoning that *his starting cards above average*,

Hero raises.

Villiain calls the raise.

They both proceed to the next round.

Flop: Two of hearts, Queen of hearts, Jack of spades

-reasoning *his hand is strong*(pair of queens), and *his hand can get stronger*(straight),

Hero bets.

Villain calls.

Turn Ace of hearts. -Hero reasons that the Ace hurts his overall ranking

-despite this belief Hero continues to show strength. A check here would reveal too much weakness.

Hero bets.

Villain raises.

At this point, the Hero is put in a tough situation and must think on multiple *levels* about how to act. Hero looks to answer questions of the following type:

- How strong is my pair of queens compared to other possible hands? Is the size of the pot big enough to justify proceeding with a weaker holding?(Level 0)
- What is the likelihood that my opponent holds an ace? (Level 1)
- What does he think I have? Does he know that my hand becomes much weaker when an Ace hits the turn? (Level 2)
- Does he know that I know he is a habitual bluffer? If he knows I know this information, would he continue bluffing with nothing here? (Level 3)

We place a great deal of emphasis on these *levels* of reasoning because they provide a framework from which to generate a winning strategy, and more easily classify inference tasks. Expert players are known to routinely perform up to level 4 or 5 reasoning.

The emphasis of existing poker research has been on opponent modeling via neural nets (Davidson 1999), and simplifying minimax search for game theoretic optimum for Hold’em games of two players(Billings *et al.* 2003) . Although the neural net strategy may be successful in detecting patterns in one’s opponents bets, it is not clear how to extract relevant information from these learned structures to generate an appropriate strategy. For our system, we are looking to build upon lower-level inferences to guide higher-level strategic reasoning.

Also, one would not anticipate that a strategy optimized for two-player(heads-up) play could be extensible to multi-way action. Our learning strategy makes no assumptions about the expected number of opponents.

Bayesian Nets

Bayesian Nets, also known as belief networks, provide a powerful tool for making inferences under situation of uncertainty. They are a graphical method for representing quantities of information in the form of connected graphs and conditional probability distribution. Critical to the idea is the use of Bayes' Rule to update our beliefs about the state of the world:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

The Bayesian Net essentially decouples the complex relationships between different variables(nodes) in the game state by explicitly specifying the direct influence of variables to other variables in the form of a parent-child connections. Any observations on variables goes into the model as evidence, and the probability distributions of the graph are updated accordingly. These observed nodes become 'instantiated', and often times simplify the inference process by jisjointing the graph into subnets of the full net.

One drawback to this approach is that a large training database is needed to automatically learn the structure and/or parameters for a Bayesian Net. For a particular problem domain, this approach may not be possible, or the generation or retrieval of this information may not be feasible. Therefore, either of these steps can be substituted by human design by experts. The benefit of using Bayesian Nets for inference is that very limited information is needed to perform inference. Also, since this is a acyclic graph representation, it is easy to represent the relationships between various quantities as connections in this graph.

Another area of difficulty for Bayesian Nets lies in active learning of the structure and parameters of a Bayesian Nets containing large numbers of continuous nodes. Traditionally, this problem has been addressed through the strategic discretization of continuous nodes into appropriate ranges of values. We have taken this approach, generating a very simple discretizaiton policy for our variables of interest.

For more recent advances in the treatment of continuous nodes for Bayesian Nets, see (Davies & Moore 2002) (Davies 2002).

Requirements and Motivation

Training Database

One of the crucial elements of developing the inference tool is to be able to provide the Bayesian Net with training data so that it could learn the parameters associated with making the inference of the villain's hole cards. Without this data, the Bayesian Net would have no way of knowing how to act and the project would simply not be possible. The best source of information for Texas Hold'em comes in the form of hand history databases, which we will simply refer to as hand histories. Hand histories contain all of the information about a game, including the players, the actions each player took, the community cards, and ultimately what each player had at showdown and how much they won. Stated simply, hand histories are a log of a game. Hand histories are available or can be generated through a variety of different sources, including several sources online. In addition, the University of Alberta has a freely available hand history database that can be used to train artificially intelligent poker players or inference agents.

However, each of the different sources comes with a different format on how the data is stored. As a result, a design tradeoff exists between taking the time to develop the most general information extraction method possible versus the complexity of implementation. On the one hand, creating a general information extraction method allows us to use multiple different hand history formats, thereby increasing the corpus of data available to train the Bayesian Net. However, unless the implementors have experience and are skilled in the area of general information extraction, this could quickly develop into a project all by itself.

Given these basic criteria, two options were considered for implementation. The first method considered was to implement a system similar to (Freitag & McCallum 2000) This paper was half of the subject matter we presented in our advanced lecture given March 30 in class. In this paper, Freitag and McCallum discuss the use of modified hidden Markov model structures to extract key information from human readable text. The HMM structure in their paper was modified to fundamentally divide the nodes into target states and non-target states. The target states are the nodes from which the key information is extracted. The non-target states are then subdivided into background states, prefix states, and suffix states. With this modified version of the HMM, the text must start in a background state, then transition into a chain of prefixes. From the prefix chain, the text must go to a target state, after which it must transition to a suffix and then back to the background. In doing so, the independence of the transitions for the HMM is no longer valid, but instead, the HMM must follow certain predefined paths. Training sets are selected and all the targets in the sets are labelled. The best HMM structure is then learned using the Viterbi and Baum-Welch algorithms.

The other type of information extraction method evaluated was to write a parser in a to-be-determined programming language for a specific type of hand history. Instead of attempting to learn a general structure that extracts the information from a generic format, the parser would be written to be format specific. If the format changed or a different type of hand history was desired, then the parser would either have to be modified or a new parser would have to be written.

Given the above information and the constraints of the project, writing a parser for a specific type of hand history format was selected. Several key factors caused this decision. One such factor was that the method described by Freitag and McCallum requires the creation of a separate HMM for each variable to be extracted. As a result, each variable contained in the hand history, such as action or the players position, would require the learning of a separate HMM. This would substantially increase the amount of time necessary to extract the data. Additionally, not all of the data is specifically stated in the hand history, the pot odds a player faces (the amount of money in the pot compared to the amount of money required for the player to call) can be calculated based upon the information contained in the hand history, but is never actually directly stated. This development meant that we would have to post-process the data anyway, regardless of whether a parser was written or not. It then seemed logical that if the information needed to be processed as it became available anyway, that the parsing could be incorporated into the same file. The fact that we would be limited to only one format type would not be detrimental if we obtained enough hands of a certain format to allow for the proper training. It turns out that this was indeed the case. Furthermore, we have had a variety of experience writing code to parse out files before while we had never before attempted the system described by Freitag and McCallum. As a result, we knew that the implementation of this system would end up being much harder than we anticipated through the application of Murphy's Law. Finally, since our ultimate goal was to train a Bayesian Net, we understood that it would behoove us to get the information required by the Bayesian Net as soon as possible to allow for time to tweak the Net. We felt that the parser would result in the production of the data in the quickest manner.

Game State Representation

Another critical aspect of producing a high fidelity inference tool for Hold'em is the representation used for the state of the game. This concept was alluded to in the previous section. It is simply not enough to know a player's actions as produced by a hand history. Instead, all of a villain's actions must be considered in the context of the current state of a game. Indeed, a bet when the villain is the first player to act in a given round and he/she has a large amount of players behind them is quite different from a bet in late position when everyone checked around to the villain. In the first scenario, the villain bets knowing that he/she has the potential to be called and or raised by numerous other players, putting them at risk of investing much more money into the pot than just their bet. The fact that they are still willing to bet demonstrates a much stronger hand than in the second scenario. Here, the villain has the advantage of knowing how all of the players in the game acted before he/she has to act. When all the previous players acted passively, the villain has more insurance that they have the best hand or that they will at least not have to invest much more money. Given the importance of the state of the game in interpreting how a player acts, a design implementation be carefully designed to efficiently, but completely represent the state of the game. The subsequent sections discuss some of the major issues associated with this design.

Two drastically different classes of designs were evaluated for the representation of the game. The first was a graphical method while the second was a code-based method. Included as options in the graphical method are HMMs and Stateflow Charts in Matlab while the code-based method requires the state to be tracked in variables of code. The graphical based method has the advantage of being much more intuitive and easier for users to follow along with the progress of a game. However, such methods tend to be much slower in runtime. By contrast, the code-based method is much more abstract, but has the advantage of being quite fast for efficiently written code. This balance between speed versus abstraction and complexity is discussed related to the described methods below.

The first potential representation method for the state of the game is to use HMMs to model the progression of a hand. The opponent's hole cards are used to define the initial state of the model. The probability of receiving any two hole cards is a simple application of entry level probability, yielding a flat probability mass function (PMF) for all possible hands.

However, the model can be simplified by bucketing the possible hands. At the preflop stage of the game, it does not matter if the opponent has an ace of hearts and an ace of spades as opposed to an ace of diamonds and an ace of clubs. Instead, what matters is that the opponent has a pair of aces. Thus, in the preflop model, the suits of any pair are irrelevant. Similarly, the particular suit can be ignored for the remaining combinations. However, it is an important distinction in Hold'em if the opponent has hole cards that are "suited," which means two cards of the same suit, versus "off-suited," which means the two cards are of different suits. Having suited cards provide the advantage of increasing the chance of obtaining a flush by the end of the game. As a result, the particular suit does not matter, but whether the cards are suited or off-suited is an important distinction to make. Thus, the initial bucketing tracks pairs, suited non-pairs, and off-suited non-pairs, thereby substantially reducing the number of combinations. This in turn limits the search space of the problem.

The probability of transitioning from any initial state to any other set of cards is again just an application of entry level probability and is known prior to any particular hand. This probability depends only on the known cards in play and the number of cards remaining from which to choose. Thus, the structure and transitional probabilities associated with the HMM are known before the game begins and never vary from hand to hand.

It is important to note that though the structure does not vary from hand to hand, it alters in shape throughout the course of any given hand. As mentioned before, the particular suits do not matter in the preflop stage of the game. However, after the flop hits, the suits of the opponent's hole cards are needed so that an evaluation with the community cards can be performed to see what type of poker hand he/she could potentially have. While this leads to an increased number of possible states, the number of states is reduced by continuing to bucket the potential outcomes into poker hands instead of modeling all the combinations that could occur with the ultimate 7 cards.

Furthermore, the model of the opponent is updated as new information about the state of the game becomes available. One such new information source is the appearance of the community cards. If a card shows up in the community cards and only one deck is being used, as is the case for a standard Hold'em game, then the card clearly cannot be one of the opponent's hole cards. As a result, any initial state containing any of the community cards can be eliminated as a possibility.

Another source of new information is the observations of the opponent, which are his/her bets. Again, the opponent's actions can be bucketed to limit the search space. The opponent's actions will be categorized as passive, normal, and aggressive.

As stated above, the structure and transitional probabilities associated with the HAM are known ahead of time and do not change. However, what does change from opponent to opponent, or even from hand to hand, is the observational probability, or the likelihood of the opponent taking a certain action at any state in a hand. The observations are the only information present that are directly related to what a particular opponent has for hole cards. As a result, the problem of inferring an opponent's hand is essentially one of conditional probability: given the current state of the game (the community cards and the opponent observations), what is the probability that the opponent has any particular pair of hole cards. With HMMs, this then becomes a problem of hypothesis testing. Each state can be evaluated for consistency with the opponent's previous actions and the community cards that are available. If the actions seem inconsistent with a particular hole card pair, then this pair is not a likely candidate for the opponent's cards.

The observational probabilities are clearly of vital importance to the inference process. If this method were to be selected, the Bayesian Nets would be used to populate the observational probabilities. This could be done by, on a high level, embedding a Bayesian Net into a particular state to determine the probability of an action given a number of discrete and continuous variables identified as important. The Net could then output the probability of seeing any particular observation for a given state.

A very similar method is to use Matlab's Stateflow charts to control the flow of the game. Many of the same development discussed above applies directly to this method. For instance, the hole cards would form the initial state and then transitions would occur based upon the cards shown in the community cards. These transitions would have predefined probabilities. The reason that this method is distinguished is that it is not just graphically-based, but is in fact graphical in implementation as well. The user could follow the flow of the game care of a GUI. By contrast, the HMM would be based on the graph, but would still be implemented in code, requiring an additional GUI to be written in order to achieve actual visualization.

The final method evaluated was a code-based model. In this method, the state of the game is tracked solely through variables stored in the code. If a player bets, the pot is increased, their active money is increased, the amount of money required to call is increased, the number of active players is incremented, and so forth. This method allows the greatest flexibility in that any change that needed to be made to the model could theoretically be added with as few as a couple of lines of code. However, this method has the disadvantage of being the most abstract, requiring users to interact over a console instead of a GUI. However, a final major advantage is that the code-based method can easily incorporate code written previously written by other people that has been made freely available to the public. Examples of such code include hand evaluators. Numerous poker hand evaluators can be downloaded from the internet. All of these tools allow the user to specify known cards with the result being the best possible hand available from the cards. These can then be modified to enumerate the different possibilities a villain could have, which would be quite useful in the hypothesis testing of the inference.

The code-based model was selected as the implementation method for the representation of the state of the game. Several factors went into this selection. Perhaps the most dominant reason is that the two graphical methods suffer from the curse of dimensionality. Even if the aforementioned bucketing is performed preflop and then subsequent hands are bucketed based on poker hand type, the number of states is tremendously high. Furthermore, bucketing causes the model to be inherently less accurate than if bucketing was not employed. It was deemed infeasible to physically draw all the states in Matlab, let alone connect them with the necessary logic. Furthermore, the code-based method ties in well with the specific parser choice for information extraction. With the added benefit of employing hand evaluators at our discretion, the code-based model was a clear choice.

Statistical Inference

In order to perform inference tasks in Hold'em with Bayesian Nets, we needed a method for generating the nodes, structure and parameters of these graphical models.

The choice of variables of interest must be done by an expert, since the game transcripts do not have any raw statistical data from which to define the nodes of our Bayesian Net. For example, one variable that factors into human play is the notion of pot odds. This variable is a ratio calculated by dividing the total pot size by the amount a player must currently put forward to play. Therefore, we needed a concise selection of variables from domain knowledge to be incorporated into a representation. For the topology, we also had the choice of learning or specifying the topology. Since the variable definitions nearly capture the notion of a correlation with other variables, using expert knowledge here would be nearly trivial. For example, hand-strength and actual hole cards are connected, while hole cards and pot odds are not. However, there are a number of packages for learning the structure of Bayesian Nets, thus providing an attractive and more optimal alternative to expert knowledge. When increasing the number of variables of interest it appears that learning the structure may be more desirable, especially if there is a high density of possible connections. Most of these structure learning algorithms involve the use of search(i.e. hill climbing) for the best structure, given a scoring metric. Furthermore, the parameters in the form of Conditional Probability Distributions, of a given structure must also be specified. Again the choice must be made between expert knowledge and statistical learning. Because the estimation of these parameters involves a great deal of calculation, this part of the project seemed more suitable for an algorithm. Finally, given the fully specified Bayes Net, one must be able to generate inferences given some observed data. To perform this action, a number 'inference engines' are available. These inference engines are useful for different problem specifications, since the choice of engine involves trading off a number of factors including speed, accuracy, and restrictions on node types.

Implementation

This section of the paper describes the design and implementation of a poker inference tool, and our evaluation of the tradeoffs associated with each decision made during the process.

Database Generation

With the hand history-specific parser and code-based model selected for the design, a critical decision needed to be made regarding which programming language to use to implement the design. Given that this was deemed to be Brian's section of the project, the real options for programming languages were C/C++ (heretofore just referred to as C), Matlab, or Python, though his experience with Python was extremely limited. Brian decided to implement the database generation code in Python with time intensive tasks to be done in C.

A number of factors went into this decision. The condensed timeline of the project eliminated C from contention. While C is the language Brian is probably most comfortable with, writing a parser in a language as low level as C would have taken substantially longer than writing in a higher level language such as Python. As stated earlier, the database was deemed to be important to produce as quickly as possible so that the Bayesian Net could be tweaked for a maximum amount of time. Producing a quick parser in C seemed impractical given the scope of the parser required, especially if a famous hidden C bug appeared. Plus, since the project had no realtime requirement for the data, there was no need to suffer through the pains of C just to get the performance enhancement.

The time savings of Python was not just anticipated due to the high level nature of the language, but also through the option of using regular expressions to aid in the parsing. While Brian had little to no experience with either Python or regular expressions, he decided that it would be faster to teach himself Python and regular expressions than to write a parser in C. This was based on his experience with writing a C parser while interning at Lockheed Martin two summers ago. This gamble, which probably admittedly took a fair amount of hubris to make, ended up paying off in the end. As for Matlab, we decided that we did not want everything to be limited to having a version of Matlab installed on the computer. Instead, we wanted as generic a program as possible. Plus, the benefits of Matlab code similarly be achieved in Python.

The best way to describe the high-level detail of how the created is to walk through an example of parsing a sample hand history for one particular game of Hold'em. A sample hand history file is showed below. It will be referenced throughout the next section.

```
#Game No : 1597832386
***** Hand History for Game 1597832386 *****
$3/$6 Hold'em - Tuesday, February 15, 23:32
Table Table  11280 (Real Money)
Seat 8 is the button Total num of players : 10
Seat 1: AA_Killer ( $131.5 )
Seat 2: T_Furgeson ( $229 )
Seat 3: Kasugai ( $145.5 )
Seat 4: billpokerwon ( $100 )
Seat 5: Tallpower ( $259.75 )
Seat 7: nychig ( $17.5 )
Seat 8: Grinning_Dog ( $101 )
Seat 9: thirddan1 ( $150 )
Seat 10: Zestaa ( $154 )
Seat 6: teraldino ( $150 )
Zestaa posts small blind [$1].
AA_Killer posts big blind [$3].
** Dealing down cards **
T_Furgeson folds.
Kasugai folds.
billpokerwon calls [$3].
Tallpower folds.
nychig calls [$3].
Grinning_Dog folds.
Zestaa calls [$2].
AA_Killer checks.
** Dealing Flop ** [ 6h, 4d, 3h ]
Zestaa bets [$3].
AA_Killer calls [$3].
billpokerwon calls [$3].
nychig calls [$3].
** Dealing Turn ** [ 9c ]
Zestaa checks.
AA_Killer checks.
billpokerwon checks.
nychig bets [$6].
Zestaa calls [$6].
AA_Killer folds.
billpokerwon calls [$6].
** Dealing River ** [ 8h ]
Zestaa checks.
billpokerwon checks.
nychig is all-In  [$5.5]
Zestaa folds.
billpokerwon calls [$5.5].
billpokerwon shows [ Ah, Qc ] high card ace.
nychig doesn't show [ Jc, Qs ] high card queen.
billpokerwon wins $50.5 from  the main pot
with high card ace.
```

The first step in encoding a hand was to see if it contained any useful information to the problem. Since Hold'em is inherently a game of hidden information, unless the game goes to showdown, the villain's cards remain hidden. As a result, in order for a given hand history to be of use in providing information from which learning can be performed, it must be of a game that either went to showdown or had a player present whose cards are available in the history. This example was selected because two players went to showdown. This can be ascertained by searching the hand history for the word ”show” or the string ”doesn't show.” The reason that ”doesn't show” is an option here is that in this particular game, if a player goes to showdown and loses, they can select to not graphically display their cards to the rest of the table. The information still obviously is placed in the hand history, though. Since these strings were matched, the game is of use to us and parsing continues.

Next, the players are populated in a list based upon the seat order. This is clearly seen in the hand history as ”Seat x: playername.” While doing this, the player who is on the button (the player that is last to act) is read in and stored. The blinds are also collected and the pot amount is updated to reflect this.

At the beginning of every betting round, the position is established for each player. In the non-preflop rounds, the player's position is simply the order in which they act. This can be most easily seen by looking at the betting round associated with the flop. In this round, Zestaa has a position of 1, AA_Killer has a position of 2, billpokerwon has a position of 3, and nychig has a position of 4. This information is updated before each betting round. The only difference with preflop is that the player posting the small blind is said to be in position 1 and the player posting big blind is in position 2.

Finally, the actual action is parsed. It is important to note that only the players who show their cards have their information written to the database, but all the actions affect the current state of the game. Instead of going through the entire hand, only a representative sample of the preflop will be demonstrated. The variables being stored for the state of the game include pot size, number of active players, amount of money required to call, and board cards. Then, information for each player is stored, including if they are active, their position, how much money they contributed to the pot in the current round, and, if they show down in the end, their hole cards.

Before the betting starts, the pot has a value of 4.00, which is the value of the blinds and the amount to call is 3.00, which is the value of the big blind. Also, the board cards are empty, the number of players equals 8 (though 10 are at the table, only 8 are actually playing, which you can tell be looking at who acts),

To start off the action T_Furgeson folds. As a result, the active variable is set to false and the number of active players is decremented. All the other variables remain constant. The same process is followed when Kasugai folds next. Next, billpokerwon calls for $3. The number of active players stays the same this time, but the pot size and amount of active money for billpokerwon is increased according to the value of the bet.

However, billpokerwon is one of the two players who ultimately goes to show down. As a result, his current information is written to the database before it is updated. This is to save the state of the game so that the Bayesian Net can learn the necessary parameters to perform a future inference. All the variables of interest as specified in the Bayesian Net are then written to a database file, with tabs delimiting the fields. In order to get the required hand score and hand score by river values needed by the Bayesian Net, a hand evaluator written in C is called with billpokerwon's known cards of Ah Qc used as arguments.

The same process is repeated for all the subsequent actions in the game. As stated earlier, every action updates the appropriate variables in the state of the game (a raise increases the pot, active money of the player, and amount to call while the number of active players remains the same) while only the state of players who eventually show down are written to the database.

Learning with Matlab Bayenet Toolbox

We had a number of choices for representation of the structure and parameters of our Bayesian Net, and the algorithms used to perform inference on that net. Our primary choice was to use pre-existing software packages to do the learning and inference, or to write our own. Although having the freedom of a customized Bayes Net implementation was attractive, for the goals of the project we decided to leverage existing work. However, even within the space of available packages, we had to narrow down our choices. In looking at the features of each of our choices, we narrowed it to the Matlab Bayes Net toolbox, and the CMU AutonLab Bayes Net Learner (Andrew Moore), each freely available and portable to our development environment. Between the two, we chose the Matlab option because of the extensibility of the representation, and the overall user-friendliness of coding in Matlab. However, although the interface and representation was simple, this package provided little documentation and support for problems. Therefore, the process of learning how to properly specify and/or learn a Bayesian Net structure and parameters involved a time-consuming learning curve. In hindsight, it may have been better to im-

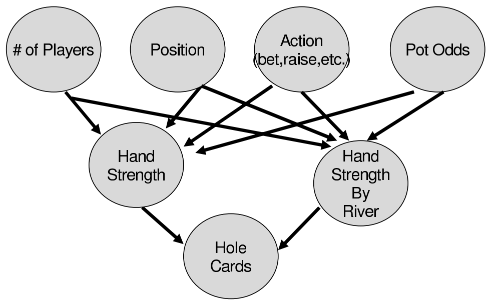

Figure 1: Expert Bayes Net Structure for Hold'em inference

plement the Bayes Net Learner from scratch, but that may have been a project all in its own.

Once we chose the Bayes Net Learner, we used expert knowledge to specify the parameters of interest. Based upon knowledge of the game, we chose pot odds, position, number of players in the hand, actual action, hand strength, hand strength by river(potential). These variables needed to be extracted from the transcripts of hands and incorporated into the database generation to provide this step with the correct data.

In addition, we needed to specify the topology of our variables of interest. Our topology choice, shown in Figure 1 captures what we feel is a range for the top four most likely influencing variables on an opponents actions. Although this choice is a simplification of the overall number of variables, it demonstrates the ability for the Bayes Net to decouple these complex relationships into parent-child connections, and then subsequently learn the parameters of the Conditional Probability Densities using Maximum Likelihood Estimation.

Using the 476,468 database entries generated during our training set extraction, we ran Matlab Bayes Net toolbox sequential Bayesian parameter updating algorithm. On a Windows Pentium 4 3.6GHz system with 2GB of RAM, the learning takes nearly 30 seconds. Considering the size of this database, these performance metrics are acceptable.

Component Integration

The different components of the project were all brought together through the tremendous coding-glue that is Python. The database generation was done through the use of two python scripts, dbgen.py and driverDBgen.py. dbgen.py performed the generation of the database information for any given hand history file while driverDBgen.py acted as a shell script that called dbgen.py repeatedly for all available hand histories. Inside dbgen.py, the C hand evaluator and other functions were called as necessary.

The learning and evaluation of the Bayesian Net were also controlled through the use of Python. First the test cases were generated by a script entitled testgen.py, which was driven by driverTestsGen.py. Another script was added to the repertoire in the form of driverInference.py. This is truly the most overachieving script. It kicks off driverTestGen.py in order to generate all of the test cases to evaluate. It then calls the Matlab executable of the Bayesian Net to perform the inference based upon the supplied evidence. It then takes the guesses supplied by Matlab and writes the results along with the actual cards to a file entitled results.txt.

Evaluation

To test the accuracy of our trained Bayesian Net, we ran 20 randomly selected tests witheld from learning as a validation set. These 20 tests were generated using driverInference.py script and passed to our Matlab executable. For each test, this executable generates a Probability Mass Function across our distribution of a handstrength variable. Recall that our handstrength variable is discretized into 10 buckets, so at best our current implementation will make a correct inference to within a specific decile of hands. We use this PMF to make an estimate of actual hole cards by enumerating a weighted contribution from the two deciles corresponding to highest distribution of probability mass. We weight the contributions such that that 75

On these examples, our inference mechanism overestimated the value of an opponent's holding on all but two trials. On the successful runs, our system accurately predicted the final showdown hand of a particular opponent to within the top 10 possibilities. Consdering the fact that there are 1326 possible preflop combinations to guess from, we con-

Variable	Description	Discretized Domain
action	action taken by player	{fold, check/call, bet/raise }
pot_odds	size of bet:size of pot	{[0-5),[5-10), ...,[20-25), inf }
position	player position wrt dealer	{blinds, early, middle, late }
num_players	number of active players	{shorthanded, mid, large, full }
hand_value	current evaluation of hand	{[0-10), [10-20), ..., [90-100)}
hand_riverval	hand potential by river	{[0-10), [10-20), ..., [90-100)}
hole_cards	specif. of hole cards	{0,1,2..1325}

Table 1: Discretization policy for variables.

sider these results promising in that we may begin to see useful results from only a few improvements to the system.

Conclusions and Future Work

In conclusion, we implemented a inference tool for Texas Hold'em Poker that produced a top 20 list of possible hole card values for an opponent given the evidence of the villain's actions. This inference was produced through the use of a Bayesian Net that was implemented using a freely available Bayesian Net toolbox for Matlab. The results produced by the inference did not yield a high rate of accuracy.

The lack of accuracy of the predictions should not be interpreted as a failure of this method for this particular problem. Given the condensed time frame of the assignment, accuracy was purposely sacrificed to accommodate the necessarily small scope of the project. We knew that a 1-semester development cycle would result in this low accuracy.

Several modifications to the methods could result in drastically improved results and the authors fully intend to continue this work into the summer and beyond. One such improvement would be to use a much more complex Bayesian Net. The design of this Net was actually completed back in early April, but we decided its implementation would take longer than the given time frame. It was at this point that we simplified the Net to the version presented in this paper. Dramatic improvements could also be seen by employing either a dynamic Bayesian Net or an HMM in conjunction with the Bayesian Net to track the changes in information throughout the game. Doing show would allow us to alter the parameters of the Bayesian Net depending on the stage of the game. This is important because the variables of interest have the potential to change drastically throughout the game. For instance, 7:1 pot odds preflop is drastically different than 7:1 pot odds on the river. To simplify the implementation, the current model has no sense of altering these parameters. It also has no sense of memory. Said differently, it does not perform the hypothesis testing referred to earlier over an entire path of actions, but instead just gives a guess based upon a single action. Again, this was a conscious choice to reduce the scope of the problem, but in doing so, accuracy was sacrificed. Finally, these and other changes will be made in an attempt to steer the project in the direction of achieving a Texas Hold'em strategy for the program. The goal of this is to ultimately have a human competition Texas Hold'em player.

References

Billings, D.; Burch, N.; Davidson, A.; Holte, R.; Schaeffer, J.; Schauenberg, T.; and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. In Gottlob, G., and Walsh, T., eds., *IJCAI*, 661–668. Morgan Kaufmann.

Davidson, A. 1999. Using artifical neural networks to model opponents in texas hold'em. http://spaz.ca/aaron/poker/nnpoker.pdf.

Davies, S., and Moore, A. 2002. Interpolating conditional density trees. In *Conference on Uncertainty in Artificial Intelligence*.

Davies, S. 2002. *Fast Factored Density Estimation and Compression with Bayesian Networks*. Ph.D. Dissertation, Carnegie Mellon University.

Freitag, D., and McCallum, A. 2000. Information extraction with HMM structures learned by stochastic optimization. In *AAAI/IAAI*, 584–589.

Koza, J. R.; Kean, M. A.; Streeter, M. J.; Mydlowec, W.; Yu, J.; and Lanza, G. 2003. *Genetic Programming IV:Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.