Simulation of a Texas Hold'Em poker player

Bosilj, P; Palašek, P; Popovi, B; Štefi, D

Simulation of a Texas Hold'Em Poker Player

Petra Bosilj, Petar Palašek, Bojan Popović, Daria Štefić
Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb
e-mail: {name.surname, bojan.popovic2}@fer.hr

***Abstract*—Imperfect information environments are amongst common research subjects in the field of Artificial Intelligence. A game of poker is a good example of such an environment. As the popularity of the game grew, so did the interest in implementing a functioning automatized poker player. Approaches to this problem include various Machine Learning techniques like Bayesian decision networks, various Case-based reasoning (CBR) techniques and reinforcement learning. For a player to play well it is not enough to know just the probability estimates of one's own hand. A player must adjust his strategy according to his estimate of the opponents' strategies and an estimate of opponents' hand strength. This paper explores the usage of the k – Nearest Neighbors technique, an example of CBR techniques, in implementing an automatized poker player. As a result, an average player able to cope with most in-game situations was developed. The main difference from a model based on optimal mathematical play is that the developed player seems more human, which makes its actions harder to predict. Numerous simulations on the developed testing model show that a small but stable profit is gained by the implemented automatized player.**

I. Introduction

Poker playing became an interesting research subject in the field of Artificial Intelligence because it represents an imperfect information environment (during the game one lacks the information about the opponents' cards and the cards yet undealt). Good in-game decisions are based not only on statistical probabilities describing the strength of one's current hand but also on good estimates of the opponents' strategies. The reason lies in the nontransitivity of poker: there is always an adequate strategy that bests any other chosen strategy. Another example of a nontransitive game is *rock – paper – scissors*. It is known that rock bests scissors and that scissors best paper, so it could be presumed that rock bests paper, which is not the case and makes the game nontransitive. A good poker player, except for knowing the rules of the game and some basics of probabilities and statistics, has to be exceptionally patient and disciplined. Experience shows that a stable profit in this game is not easy to achieve.

After playing a large number of games, an experienced poker player often finds himself in situations very similar to the ones already seen. In many of these situations, gameplay patterns that maximize the player's profit emerge. These observations suggest the possibillity of constructing an automatized player, considering the facts that the game has a limited number of possible states, that one instance of the game is short and easy to follow and that it is possible to categorize the opponents based on their chosen strategy.

The goal is to construct an automatized player that has the ability to learn based on examples of games played by (good) human players and thus, combining this data with precise mathematical calculations, achieve an edge over human players. An automatized player's playing style (strategy) could further be attuned based on the strategies of his opponents. As concentration and self control play a major role in the success of a poker player, another advantage of the player built by using Machine Learning methods is that various factors such as fatigue, mood and temper have no impact on the gameplay.

The approach in which future actions are based on a large number of previously known examples of well played poker games is based on Machine Learning methods known as Case-based reasoning (*CBR*).

Related works and similar approaches to the problem are described in the next section (Section II). The general approach to the problem, the details of the chosen methods and a short overview of the implementation are given in Section III. Achieved results are presented in Section IV. The last section, Section V, offers the conclusion along with a few ideas for future work and possible improvements.

II. Related work

Lately there were many attempts in the field of automatized poker player construction. The resulting applications are popularly called *pokerbots*. Most of the developed systems are built to simulate Texas Hold'Em poker players. As Texas Hold'Em is one of todays most popular poker variants, this project also had the goal of constructing a pokerbot capable of playing this type of poker.

Prototypes of one of the first strong automatized poker playing programs that have played successfully against strong human opponents are known as Loki and Poki. These programs were developed as part of a project led by a group of researchers from University of Alberta[1]. The original implementation, Loki, based its decisions on expertly defined rules. This approach turned out to be unsuccessful when applied to Texas Hold'Em poker because the rules were too complex and numerous. For that reason, the approach used in the second version of the developed program, Poki, was based on simulating possible outcomes from a certain point in the game. These simulations were used to determine the best possible decision. Detailed explanations of the approaches used in these early prototypes developed as a part of this project can be found in [1] and [2].

Other attempts in constructing a poker playing program via Machine Learning methods are based on modelling of Bayesian networks [3], [4], reinforcement learning and Case-based reasoning tehniques. Works based on

[1]Information on this and many other projects from the field of game playing can be found on http://webdocs.cs.ualberta.ca/~games/.

Bayesian networks choose a fixed point in the game and construct a model with best defined variable dependencies (variables can represet the player's cards, community cards, opponent's cards etc.). The *human factor*, modeled by an estimate of the opponent's strategy is a variable of great significance; learning poker could otherwise be reduced to elementary probabilities and statistics. All papers exploring Bayesian networks approach to building an automatized poker player focus on constructing decision networks on which the developed systems base their actions, with possible limitations concerning the number of opponents. A very good example of a poker player prototype based on Bayesian networks is the Bayesian Poker Program described in [4]. The specificity of the Bayesian Poker Program, adapted to Texas Hold'Em Poker in [4], is an efficient model of bluffing embedded in the implementation. The negative sides of the Bayesian networks approach include poor performance against strong human opponents and long adaptation time in gameplay against such an opponent.

The second most common approach used in many similar game playing problems is reinforcement learning. The goal of this method is maximizing the agent's profit based on awards and penalties for traversing between states. This kind of environments can typically be modeled by Markov decision processes [5]. Application of this approach applied to 1-Card Poker can be seen in [6]. While the variant of poker used in [6] is much simpler than Texas Hold'Em poker, it still models an imperfect information environment and competitive play against an opponent. The constructed agent showed promising results against simple player models. Another example of a poker agent developed by the means of reinforcement learning, although using a different approach than [6] can be seen in [7]. The modeled player's performance is evaluated separately during the (rather lengthy) training process and when playing against a model of a player used for training the adaptive poker player [7]. The main significance of the work presented in [7] regarding the automatized poker player developed for this project is the evaluation method based on modeling four basic opponent types. A similar evaluation model is used here to estimate the developed prototype's quality of play, with details presented in Subsection IV-A.

Case-based reasoning techniques choose the appropriate action based on a database of memorized situations with complete information. The solution developed as a part of this project also relies on one of these techniques, k – Nearest Neighbors (*k-NN*) method, for decision making. The database usually consists of all the situations known to the system with corresponding actions. The system can then search the database with its current state as a parameter and decide on the next best action depending on the most similar states in the database [8]. The large database of played hands available for this project eliminated the need for constructing the *learning set* by letting the system play against itself using random strategies, which is the approach used in [8] for extracting gameplay examples. The data collected for this project's database is categorized based on the betting round and the number of community cards visible to the player in the *preflop*, *flop*, *turn* and *river* examples in contrast to the database in [8] which is divided only on *preflop* and *postflop* learning examples. Another point of difference from the approach used in [8] is a good evaluation of the player's hand, independent of the gameplay factor. CASPER: a Case-Based Poker-Bot [9], the latest pokerbot from the makers of Loki and Poki, is another example of a CBR poker bot which shows some similarities to the approach used in this work. It also uses the k-NN algorithm to make decisions, and a database of previous playing scenarios. The main difference is that CASPER has a database derived from playing versus play-money opponents and other poker bots, which, according to its authors [9], is the main reason for it not playing profitably against real-money opponents. Our project uses a database of real-money hands played by live players to avoid problems that occurred in [9]. Also, CASPER is modeled for the Limit Texas Hold'em variant which simplifies decision making. No Limit poker games, which are modeled in this work, induce a variety of guessing space (and therefore classification features) by introducing betting and raising patterns which don't exist in a Limit game.

III. Approach

A. Input data parsing and feature extraction

Descriptions of the played hands in XML format were obtained from the SQL database acquired for this project. For the purpose of extracting features of the training set examples, an XML parser has been built for the input data.

A very good *measurement* (*feature*) of a player's hand before the community cards are shown or any of the opponents' actions observed, is the strength of cards in one's posession. Equity Value (EV) of a hand provides a quality description of its strength. Each card pair is associated with a numerical value calculated from the sample of 115 million hands using *data mining* methods[2]. The Equity Value is actually a percentage representation of the expected gain in terms of the player's total stake. EV can also be interpreted in terms of payoff probability, and is negative if the probability is less than $1/(num\_current\_opponents + 1)$.

As the game progresses, both human and mathematical factors play an equally important role in making decisions about the next move. A basic and most important mathematical factor is the payoff probability of a hand. Unfortunately, calculations of this factor cause a combinatory explosion since a large number of game scenarios is possible (scenarios are affected by opponents' cards and actions as well as by yet unknown community cards). To get an adequate approximation of the mathematical parameters, an estimate based on simulating 100000 different scenarios is used instead. Important features such as win probability, pobability of improvement for a player's hand and for the opponents' hands are also calculated using simulation methods. The number of *outs*, cards that improve a player's hand, is obtained from simulation as well. Important human factors describe opponents'

[2]Data acquired from http://www.tightpoker.com/poker_hands.html.

aggressiveness, their raises and bets as well as their reactions to the raises and bets of other players.

In the last betting round, after the last community card, *the river*, has been dealt, estimations of a hand's worth are no longer needed: for a certain hand, it can be determined exactly which hands it bests. Win probability of a player's hand can thus be precisely calculated in this betting round.

B. Case-based reasoning methods

As previously described in Section II, the core idea of all CBR methods is to search the available knowledge database for all the states (situations) similar to the current system state and base the final decision for the current state on the decisions made in the related states.

While playing poker, the decisions made in four different betting rounds can be distinguished. In every betting round, different kinds of relevant information are introduced. Decisions in later phases of the game are based on more attributes because all the past states and actions during the game must be considered, and the number of unknown cards grows smaller. The database can be viewed as split into four different parts, each part containing the examples concerning a specific betting round with a known classification. In this instance, the action taken in a certain point in the game is considered as the classification of an example. When determining the next action, the system bases its decision solely on the knowledge contained in the database for a phase corresponding to the current phase in the game, comparing the current state only to the states belonging to that phase.

This paper considers using the k-NN algorithm for determining the most appropriate action for the current state. This simple algorithm searches the database for k examples nearest (most similar) to the example representing the current system state, and classifies the current example in the most common class among k nearest examples found [10]. All the features used in the classification process are described with numeric or binary values. The simple Manhattan distance is then used on the normalised feature values to define similarity between samples. The optimal parameter k has been determined by extensive testing and evaluation for every game phase separately and corresponds to considering the 30 nearest examples in the preflop phase, 40 nearest examples for the flop and turn phases and 60 for the river phase.

Time complexity of k-NN classification is linear in the size of the training set. Since the performance speed for a single decision was almost instant on the databases available for this project and can be expected to remain satisfactory fast even if the database sizes increase multiple times, no further analysis of the upper limit of the database size was made. Training speeds are higher, but not as significant, considering the training process is a one time event occurring before the simulation or actual game.

C. Training set and database description

A training set for this project used for building the knowledge database is comprised of poker hands played by several *good* human players[3]. The data was gathered using the Hold'Em Manager[4], an in-game program for calculating opponents' statistics which enables saving of played hands in a local database for later analysis and review.

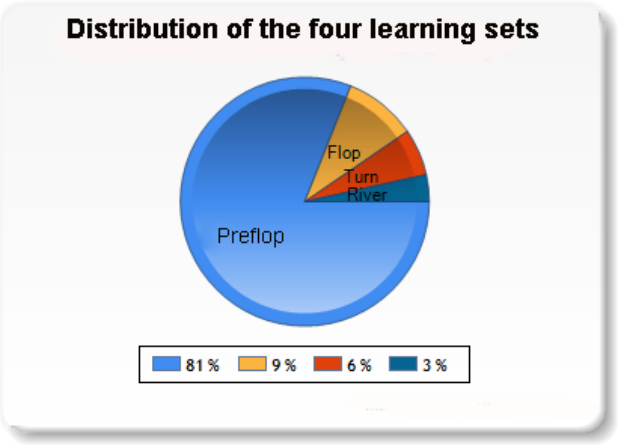

Fig. 1: Distribution of learning sets

The final version of the implemented system uses around 75000 different examples in the learning phases and is showing satisfactory classification results. A single example consists of all the actions undertaken on the table from the moment the *blinds* (obligatory bets) have been posted until the winner collects the money won in the current hand. It also holds information regarding player positioning at the table, betting amounts and even chat logs. Four training sets (divided by the game phases) consist roughly of 57000 examples in the preflop database, 7000 examples in the flop database, 3600 examples in the turn database and 2400 examples in the river database. Each phase has an unique set of important details which are later used to derive classification features for the k-NN algorithm. For example, a preflop hand sample has information consisting of the player's hand strength, the current state of the pot (unopened, opened, raised), the player's position at the table, etc. Distribution of the examples in the training set amongst phase-specific databases can be seen in Figure 1. A rapid decrease in the number of examples through the game phases can be explained by the fact that the human player rarely advances to the later phases of the game, which is natural for Hold'Em poker. A system a database of played hands is less predictible than the system basing its actions only on mathematical simulations.

D. Implementation

Parsing of the examples (input data) in XML format has been done using the implemented XML parser written in Java programming language. The implemented parser and feature extractor uses simulation methods to calculate certain example measurements (Subsection III-A). Game simulations used in the parser were developed using the finished implementations from the Poker Prophesier

[3]For the details on the quality of the human players whose hands were used, please contact the authors of this paper.

[4]For more information, visit http://www.holdemmanager.com.

library[5]. For the development of the classifier itself, the programming language Python was used in combination with the open source package Orange[6]. This classifier is then used by the player prototype implemented in the C++ programming language. A simple command line game simulator (also written in the C++ programming language) has been implemented in order to test and evaluate the player prototype. The developed simulator allows various implementations of players to "sit" at the table and join the game, simulating the basic mechanics of the gameplay (models of tables and hands). The simulator provides the means to follow the details of the game play and makes the overview of basic data for each player available at any given moment (e.g. the current amount of money in possesion).

IV. Results overview

A. Evaluation methods

The way in which the player prototype has been designed makes *cross validation* one of the most compelling evaluation methods. It is important to notice that this evaluation method does not relate directly to the quality of the developed system. The results can be better interpreted in terms of similarity with the human player whose hands were the foundation of the training set. The best evaluation results are achieved by balancing the training set, i.e. seemingly attempting to even out the number of training examples for each class (or in this case, player action). The package Orange, used for building the classifier, provides an easy way of doing this.

To evaluate the quality of play for the developed system, simulation of real game conditions has to be conducted, i.e. the system has to play against the realistic models of opponent players. This paper considers an environment in which the implemented system plays against simple mathematical models of players that base their decisions on win probabilities for each hand as a *good enough* testing environment. The mathematical model of a player is implemented in two different adaptations: in the first the player model plays each hand mathematically optimal, while the second one allows ajusting of different parameters in the attempt to make rough models of four basic player types.

A mathematical model of an optimal player makes its decision about the next move based on win probability obtained from:

- the *normalized EV value* of the current hole cars for the preflop phase,
- the *simulator* for the later phases.

The amount of money that this model of player is prepared to invest in proportion to the total possible gain (i.e. the sum of money already on the table) is equal to the win percentage of the hand. If the appraised value is less then the amount required to stay in the game, the player yields, or in poker terms, *folds* his hand. However, if the value is roughly the same as the required amount, the player pays the opponent's bet, and raises if the value is significantly larger than the minimal required amount. By separately shifting the threshold value that the matematically optimal player *bases his decisions on* and the quantity that the optimal player *is willing to invest*, four distinct models describing a player class emerge based on two different criteria:

- players *passiveness* or *agressiveness* and
- players *tightness* or *looseness*.

In combination, models describe a rough strategy classification of a player.

A model of the player that bases its actions on the keyboard input has also been implemented, enabling the testing of the developed player prototype against the human player. This method of evaluation was unfortunately proven uneffective because of its lengthiness: the amount of data required to produce valid conclusions would require the tester to play an impractically large number of games against the designed prototype.

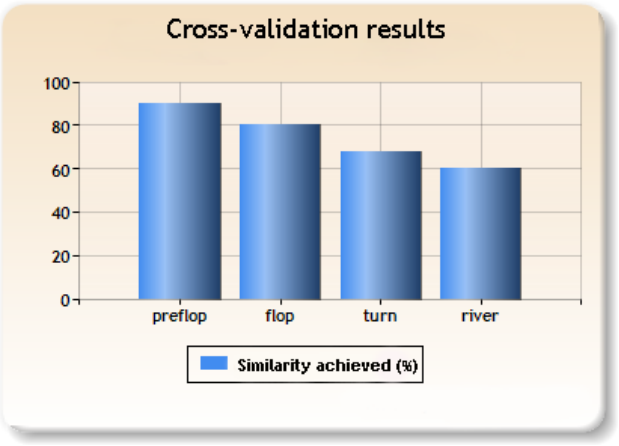

Fig. 2: Results of cross validation by phase

B. Results analysis and overview

For the evaluation of the developed classifier, 10-fold cross validation was employed. This method, as previously mentioned in the Subsection IV-A, describes only the similarity between the developed automatized player and the human player used as a basis for learning. As the knowledge database for this problem is split into four training sets, the process of cross validation has been carried out separetely for each training set. The results of the cross validation process for each training set can be seen in Figure 2.

Due to the increasing complexity of the in-game situations in the later phases of a hand (a combinatorial explosion of parameters occurs and the representation of the information about the opponents to the computer becomes intrinsically harder), the accuracy determined by cross validation is highest for the preflop phase and drops in subsequent phases. These results arise from the fact that the mathematical features and those referring only to the player's hole cards become less influential in determining the player's next action. Here, the complexity of poker and the unpredictability of a hand's outcome come

[5]More about this toolkit can be found at http://www.javaflair.com/pp/docs/manual.shtml.

[6]Documentation and usage instructions for this package can be found at http://www.ailab.si/orange/.

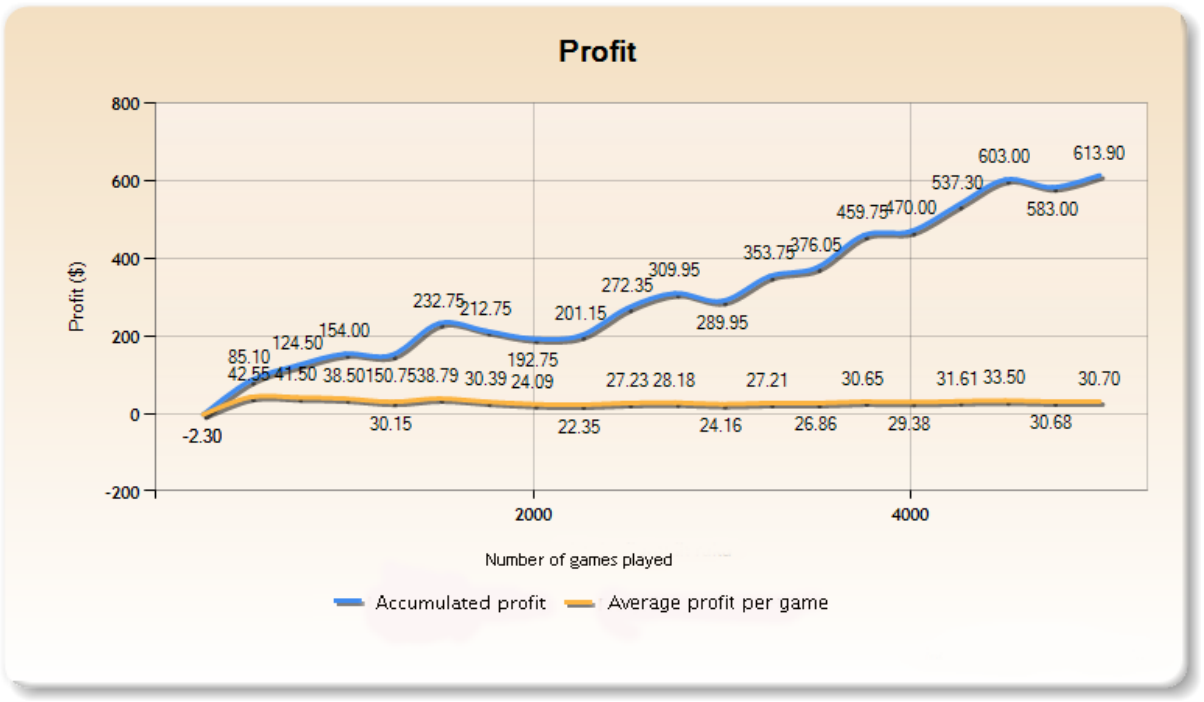

Fig. 3: Profit made in 5000 hands played

into play. To get better evaluation results, opponents' habits, patterns and strategies should be considered. A good poker player adapts his own strategy based on the perceived opponents' strategies, considers the amounts of the opponents' bets and tries to observe and determine their *betting patterns*.

Experienced online poker players usually play up to 2500 hands in one daily sesson, playing simultaneously on eight to twelve tables. This number of hands has therefore been taken as a reference for evaluating the developed player. After simulating a series of 200 sessions of 250 hands each, the performance of the player prototype is commensurable with the performance (and profit) of the human player in a month's time. The total and average profit is noted after each session. The testing environment assumes tables with the maximum of six players (the designed player prototype plays against a maximum of five opponents) with the starting deposit of \$20. The *blinds* are \$0.10/\$0.20. The opponents are variations of the mathematically modeled optimal player with different aggressiveness and tightness parameters. The conducted tests start with the total of six players per table and if a player loses all of his money in a session, he leaves the table (*re-buys* are not possible) which makes observing and following any player's profit much easier.

The total and average profit accumulated in 5000 hands, the equivalent of two evening sessions, is displayed in Figure 3. The image shows a constant profit growth despite the fact that in some sessions the player lost all of his money. By joining a total of 20 tables, the player invested 400 dollars of his own money. After the game, he left the tables with a total of 1013 dollars, making a net profit of 613 dollars. This means he finished the session doubling the amount of money he started out with. Apparently, the designed player is far superior than our mathematical player models, seeing as none of them matched his profit, although they are objectively not up to par with intermediate human players. On a monthly basis, the average results dropped slightly, making the average profit 12 dollars per session.

The implemented player prototype would be able to perform even better if he would to have actual information on his opponents, which is, as stated earlier, also a very important factor when contemplating the next move. There are four types of players, described in Subsection IV-A. Each of these strategies has an optimal counter-strategy, so possessing information on an opponent's type would give the player ability to adapt his strategy to match his opponent's type of play.

Going over the details of the evaluation games played by the developed prototype, certain key lapses in the gameplay have been noticed. A good example is unsatisfactory play of *overpairs*, i.e. a situation where the player's hole cards represent a strong *pocket pair* (e.g. a couple of aces) when the highest possible pair that can be made from the community cards is weaker (e.g. the flop is $J82$). The developed protoype plays too passively in these situations, allowing the opponents to wait for a possible straight or flush. This problem arises from the lack of features that would distinguish between this situation and the situation where the player has only a high pair (which does not guarantee a strong hand).

V. Conclusion

The goal of this project was to build an automatized poker player that would know how to make decisions in the situations with limited information, based on the results of the similar actions in the past. The developed prototype has an advantage against the human players because it acts on a combination of experience gained

from the knowledge database (training set) and precise mathematical evaluation of the hands value.

The goal of the paper was to combine two perspectives, the unpredictability and strategy of the human player with the precision and computing power that provides an accurate probability calculation. With a simplified set of features and a limited knowledge database, a very good performance of our player on the developed test model was achieved. Of course, our player is in no way superior since it is very difficult to accurately simulate human thought patterns while taking in consideration only the basic features of the decision making process. However, from the results of evaluation, it can be presumed that the developed prototype would profit against human players, which makes the results satisfactory. By applying Case-based reasoning techniques, a somewhat successful emulation of a decision making player in a space with a combinatory explosion of possible consequences regarding these decisions was achieved with satisfactory quality.

Possible improvements include further division of the training set for each phase according to the assessment of the opponents' types and their playing strategies. During the game, the automatized player notes his opponents' actions and after a certain number of hands is able to estimate their respective strategies. This would include the information about the opponent's strategy in the strategy of the developed player, allowing him to base his decisions on the examples from a more specific training set, and would add a model of another important factor disregarded in the current implementation to the decision making process.

GRATITUDE AND THANKS

The authors of the paper would like to thank everyone whose suggestions helped to make this paper better and more understandable. We would also like to thank a number of friends who provided their hand histories and advice.

REFERENCES

[1] D. Billings, L. Peña, J. Schaeffer, and D. Szafron, *Learning to play strong poker*, 2001.
[2] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artif. Intell.*, vol. 134, pp. 201–240, January 2002.
[3] A. E. Nicholson, K. B. Korb, and D. Boulton, "Using bayesian decision networks to play texas hold'em poker," 2006.
[4] K. Tretyakov and L. Kamm, "Modeling texas hold'em poker strategies with bayesian networks," 2009.
[5] F. A. Dahl, "The lagging anchor algorithm: Reinforcement learning in two-player zero-sum games with imperfect information," 2002.
[6] M. R. Wahab, "A reinforcement learning agent for 1-card poker," 2008.
[7] G. Kendall and M. Willdig, "An investigation of an adaptive poker player," 2001.
[8] A. Sandve and B. Tessem, "A case-based learner for poker," 2008.
[9] I. Watson and J. Rubin, "Casper: a case-based poker-bot," 2008.
[10] L. Gyergyek, N. Pavešić, and S. Ribarić, *Uvod u raspoznavanje uzoraka*, 1988.