Mazurkiewicz Jacek
Wroclaw University of Science and Technology, Faculty of Electronics, Poland

Intelligent system for card game analysis and prediction

Keywords

poker game, expert system, softcomputing, picture analysis, card game rules verification

Abstract

The paper presents a system which to make proper poker decision in situations provided by end user via image of the online poker table. The system provides feedback and reasoning behind decision made. The main goal is to minimize influences of errors and unwanted factors on each step so final decision could be accurate and useful in as many cases as possible. The idea is to use softcomputing technologies as neural networks for image recognition and expert system for decision making process. The system - able to parse poker table image - could be used by poker player for self-study on example on his/her own past in-game situations. Image is screenshot of the interface that is provided by online poker room to a player, so all information available to a player will also be available for further processing. The proposed solution can be an essential tool for the monitoring and verification of card game rules systems and to point the incorrect or illegal situations based on video data.

1. Introduction

The aim of this work is to check if it is possible to utilize professional poker player knowledge to provide automatic feedback about Texas Hold'em game situation. Seems it is possible to develop a system which makes proper poker decision in situations provided by end user via image of the online poker table. What is more such system would have to provide feedback and reasoning behind decision made. The main goal is to minimize influences of errors and unwanted factors on each step so final decision could be accurate and useful in as many cases as possible. The idea is to use softcomputing technologies as neural networks for image recognition and expert system for decision making process [5],[1].

Mentioned system could be used by poker player for self-study on example on his/her own past in-game situations. Such application should to be able to parse poker table image. Image is screenshot of the interface that is provided by online poker room to a player, so all information available to a player will also be available for further processing. These information have to be gathered from image. This is where first problem appears – image recognition. Unnecessary data should be removed so only important information will be left. The idea is to use neural networks and check whether one can be trained to recognize important elements properly. Because of expert knowledge utilization there have to be a way for introducing it to the system. This is where second problem appears – performing automatic reasoning according to knowledge provided. Expert systems were already successfully used in medicine and other areas where expert knowledge could be gathered for performing decision in automatic way at some level. This is why the solution will be to develop the expert system for solving mentioned problem. It will be used for coming up with a proper decision according to rules provided. As mentioned before in most cases decision in poker game cannot be ultimately wrong or right. Because of that expert knowledge will be treated as reference to output provided. The measurements will be provided to help to answer important questions: How well developed rules could reflect expert knowledge? How credible such system could become for other poker players? Finally system will have to provide end user interface where poker table images and strategy can be provided and after processing output is presented to user. Such output will have to include decision and justification behind it, so user could understand experts reasoning [6].

2. System modules

The application will be split into modules in a way that each module will have one responsibility. *Figure 1*

presents overall look on how those modules will interfere with each other.

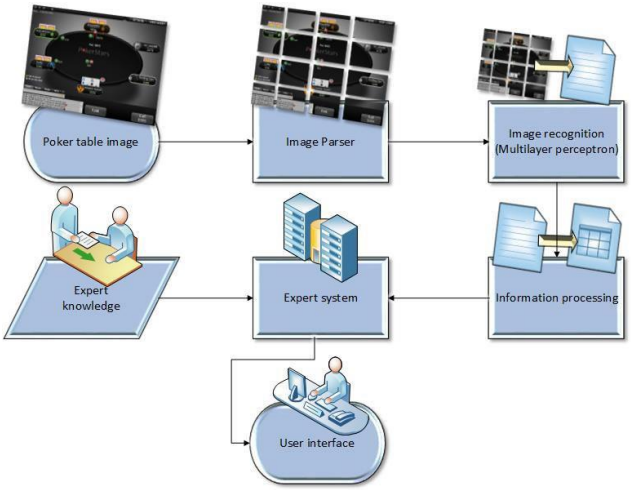

Figure 1. Application data flow diagram

In the first step poker table image will be provided as the application input. First module to process it will be image parser. At that point image will be split into pieces, so only important parts that will be used in further recognition are left. Next, those small images will be processed by image recognition module. At that step all information held by images provided will be extracted to data form. This will be done via multilayer perceptron when necessary and some lighter custom solutions when possible. After that extracted data will be delivered to information processing module where it will be converted to a form where all unnecessary information is dropped and only data essential for decision making process is left. These information will be directly inserted to expert system module where decision process will take place. However in order to perform a decision process firstly a strategy in form of rules have to be provided. To acquire a strategy in required form another process have to take place. Poker expert will be questioned about his strategy. At that point set of points describing how the game should be played according to expert will be created. Further those points will have to be translated to rule form understood by expert system. With rules provided expert system will be performing backward chaining trying to prove raise decision for every poker table image provided. If system succeeds to prove raise, then set of rules proven during backward chaining will be displayed to end user along with final raise decision. In other case, when system won't be able to prove raise by using rules provided it will automatically assume that player should fold and display according information to end user [2].

2.1. Image parser

After providing poker image to the system the first step is to cut out only important parts of it. This is done in image parser module. Information about what information is important was acquired via interview with poker expert. During it he was questioned about what he looks at the table while making a decision. Conclusion was which elements have to be extracted to be able to get full necessary knowledge from them. *Figure 2* presents an example poker table with marked essential parts.

Figure 2. Poker table image - essential parts marked

First important part is the fold button (marked by white box). It indicates possibility to fold which is always available where person playing is to make a decision. What is more it is available only then, so it's appearance could be used as a validation that a proper image with decision pending have been provided to the system. Second thing are player cards (marked by red box). This is without a doubt part providing essential information for further decision making process. Because whole card images are not needed only a small square containing both cards figures and suits is cut out. Next important thing to track on poker table is dealer button. Its positions says who is the dealer in current hand, but also much more information is acquired basing on it. On the image above green boxes marks spots where button will appear if corresponding player is a dealer. To avoid unnecessary increasing of images those six spots are cut out and merged to a single image before further processing. Undoubtedly one of more important thing is to know amount of each player's chips. However there are two spots for each player that needs to be combined in order to acquire that information. First is under each player's name and second near his/her avatar on the table (both spots marked with grey box for each player). That is because when someone makes a bet its amount is instantly subtracted from value displaying under his/her name while not being in pot yet. This value is visualized

by chips stack and corresponding number near the player. Information about chips we have can be read from spot marked with blue boxes. In that case there is also need for reading spot on the table even we are always before making any decision. That is because there is a possibility of us being at blind position. When that's the case the blind value is displayed in same manner as a bet. Next we need to check position of cards near each of our opponents. If they are near one it means he/she is still playing current hand, otherwise not. For that a spot marked with yellow box is cut out for each opponent. Another interesting spot is each opponent border colour (marked with brown boxes). That is to be used for further application tuning. Poker software allows for marking opponents with colours and notes so colours can correspond to player evaluation poker expert made. Such information could surely affect final decision. Last important information is to know how many players are left at the table. This one can be simply acquired by checking for each opponent if his/her avatar is displaying in specific spot at the table. For that spot where player avatar appears is cut out for each opponent [5].

2.2. Image recognition

It was decided to recognize this information using neural networks because images contain different colours (unlike number of chips for example) and they have limited domain, so it will be easier to teach the network. Recognizing an image using artificial neural networks forced preprocessing to be considered as the process of learning is quite long. In order to make it shorter, but still reliable, neural network teaching user interface was created. It allowed not only setting network parameters but also specify image scaling and to convert image to pure black and white or grayscale. Thanks to created UI decision what to do with the particular type of the image (card figures, colours or dealer position) can be put on hold till network teaching phase because everything can be changed on the fly. Under the hood there is a teaching module that handles this process. It makes necessary image conversions allowed by the UI, then converts such data to flattened array of doubles (numbers from range [0.0, 1.0]). The array consists of normalized, serialized RGB values. *Figure 3.* describes process of acquiring such data structure. The flattened array of the image is used as input to the neural network. The chosen neural network is multilayer perceptron (MLP) with back propagation learning method, as the outcome of the images was known and it was possible to specify the output of corresponded input. Back propagation with momentum was used in order to enhance the neural network by avoiding oscillation problem (when the error surface has a very narrow minimum area).

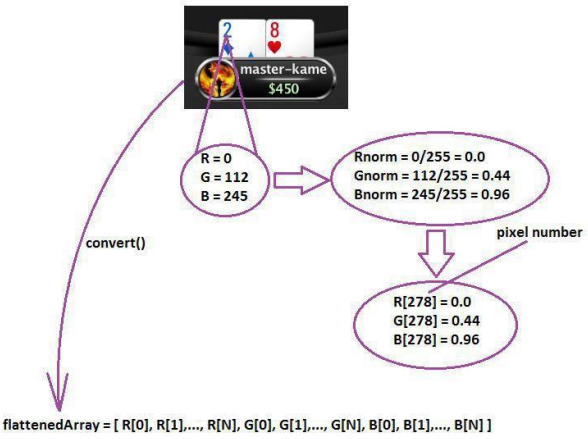

Figure 3. Image to flattened array conversion

Neural networks used consist of three layers: input layer, hidden layer and output layer. In the input layer number of neurons is equal to number of elements in flattened array, so it is dependent on provided image size. Because image size vary on scaling parameter number of input layer neurons will change during testing. Output layer has different values depends on type of the image - if this is image that contains card figure, it has 4 neurons, as the figure you can describe binary in 4 bits (there are 13 values from 2 to Ace. Analogically card suits and dealer button position have respectively 2 and 3 output neurons. Finally three MLP were used: one for card figure recognition, one for card suit recognition and one for dealer position recognition. All network configuration was same – with sigmoid activation function - in each case with exception of number of neurons. For network teaching process group of tables selected in a way that every card figure from 2 to Ace, every card suit within spades, clubs, diamonds and hearts and every dealer position appear at least once within them . That means as for the cards there was no requirement that every of 52 different cards must appear. It is because card is not considered as a whole but as the set of two independent information: suit and figure. Also because card figures were decided to be converted to black and white during preprocessing there is no reason of having each figure image in each possible colour. To be sure enough within samples that are provided in final set each card figure appears at least twice in at least two different suits – conversion to black and white may not produce exactly same image for different colours on pixel level. Final teaching set consisted of 44 poker table images, with two cards on each one giving a total of 88 card figure, 88 card suit and 44 dealer position samples. For each of recognized types data set is created. Such data set

consist of number of rows equal to number of samples provided. Each row consist of flattened image array and output is should provide. It is fed along with teaching parameters to MLP and train function is invoked. During its execution MLP iterates over data set and updates weights between each neuron layer accordingly to anticipated results provided in each data set row. Exact teaching parameters used were determined by research [3].

Table 1: Output of recognition

Recognized information	Value
Figure of first and second card	2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, A, where T=10, J=Jack, K=King, A=Ace
Suit of first and second card	h, d, s, c, where h=Hearts, d=Diamonds, s=Spades, c=Clubs
Dealer button position	integer within range [1,6]

2.3. Information processing

After all operations of image recognition module are done we are left with simple objects containing all gathered data in variables, so there is no need for further usage of images. This is the point where these information needs to be processed within information processing module. The reason for this is that raw data in most cases can be compressed, for example information about suits of first and second card are useless by themselves. The only thing that is important while making a decision is whether those suits are the same or not (if our cards are suited or not). After consultation with expert in this area list of information that has to be extracted from gathered data was created.

Data delivered to module: is fold button available (Boolean value), dealer position (1 to 6 numeric value), number of players (3 to 6 numeric value), first card figure (text value), first card suit (text value), second card figure (text value), second card suit (text value), total chips for each player (6 element list of numeric values), chips on table for each player (6 element list of numeric values), opponent border colour (6 element list of textual values).

By using information provided various important measures are calculated. Final output information is: player position, first raise position, player stack in big blinds, effective stack in big blinds, higher card figure, lower card figure, are cards suited, border rating, number of players, number of raisers, number of limpers [4].

Player position is represented by number from 1 to 6. It is assumed that person on Big Blind (2 places clockwise from dealer position) is on position 6 and number decreases while moving counter clockwise. When number of players decreases positions start to disappear from lowest ones so for example when there are only 3 players left the positions are 4, 5 and 6.

To calculate player stack in big blinds player stack have to be divided by value of Big Blind with standard rounding (example: stack = 850, big blind = 200, stack in bb = 4). Result value is player stack in big blinds.

To calculate effective stack at first each stack have to be calculated (player and opponents) in big blinds. This is done same way as calculation of player stack in big blinds but this time for each opponent. When each stack is expressed in big blinds formula have to be calculated: *Min(player stack, Max(opponents left in play stacks))*. So at first only opponents that are still in play (are still to move or already lipped or raised) have to be checked and highest stack value in big blinds have to be taken. Then it have to be compared to player stack in big blinds. Smaller of those two values is effective stack, it represent how much player can lose if he/she will play this hand.

To calculate those two both figures that were read from table have to be compared to each other. The higher one is returned as higher card figure and lower one as Lower card figure. Possible figures from lowest to highest are: 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K and A, where T is ten, J is Jack, Q is Queen, K is King and A is Ace. In case when both card figures are same for example 7 and 7 there is no difference which card will be returned as higher and lower.

To tell if cards are suited suits of two cards that player got have to be compared. If they are the same true is returned, otherwise false [4].

To calculate border rating border colours that were read are mapped to player rating. Currently following ratings are used: Random player, Regular player, Weak limper and Good limper. Borders are mapped in following manner: no label, yellow, black, blue, red, pink, light orange – Random, orange, lime, green, cyan, dark blue, white, – Regular, purple – Good limper, grey – Weak limper.

To calculate number of limpers/raisers firstly chips lying on table next to each opponent that already made action and haven't folded have to be looked up. If that amount is same as value of big blind it means that such person is a limper and if this amount is greater than value of big blind then this player is considered to be a raiser. After it both number of raisers and number of limpers are counted.

If some opponent raised before we were to make a decision, only information needed is first raise position, even if there is more than one raiser. For each player that moves before us the amount of chips on table is checked. This is done from earliest to latest position, if someone is detected to be a raiser – his/her amount of chips on table is greater than BB current opponent position is returned. In case of

number of players there is no need for any conversion, value is forwarded further as it came [5].

2.4. Expert system

Expert system was used as a solution for making a decision at poker table. Because no solutions that suited application specific needs in 100% decision was made to develop custom expert system engine. This engine is slightly simplified but not closed for extending. In poker games that are considered within this work players will fold much more often than raise. Because of that the easiest way to develop strategy is to specify when player should raise and assuming than he/she should fold in every other situation. By doing that knowledge base can be greatly optimized. It can consist of only rules that either have raise decision as their consequent or that can be used as a part of a chain for proving it. Regarding to thesis limitations decision is binary – player either has to fold or go all in. According to that hypothesis was made that player should go all in and engines will try to prove it. In other words engine have a goal and it will try to achieve. That analysis resulted in choosing backward chaining as it is goal-driven reasoning [5] as mechanism which will be used within engine. After engine finishes its work it gives information if it was able to prove hypothesis with rules provided along with textual information on rules used in case of success.

Whole knowledge base is represented by list of rules. Each rule can be logically divided into IF and THEN parts. On implementation level THEN is represented as single fact object and IF part is list of such facts. THEN - the consequent is an action which is executed when it's IF part – the antecedents list is proven. As was mentioned before my expert system is slightly simplified and because of that there is currently no support for OR statement and negations in Antecedent - only AND statement is supported. In other words all facts that are on Antecedents list have to be proven true in order to action to be fired. Knowledge base are stored in **.rgp* files in textual form. Table 2 below presents part of example rgp file.

Table 2: Example rules within rgp file

1	Fact: Bubble R1; Number of players=3, Number of raisers 1, Number of limpers 0
2	Raise; Pb FTA Ps9+BB, Player position UTG,A7s+
3	Raise; Pb FTA Ps9+BB, Player position UTG,ATo+

Database is a simple list of facts that are true. In this system case this list is only filled once before trying to prove hypothesis and not modified afterwards. This is because of using backward chaining method which is not modifying database during execution. This is also where a concept of mine called Knowledge scheme was introduced. After questioning expert it came out that simply inserting processed information to database might not be a good idea. It is because exact information is not always useful for the expert. For example there is no difference whether player have 9, 10, 11 or more big blinds, the only thing that matters is that he/she have 9 or more. More specific information will not have impact on final decision. That is the reason why small sub-module was introduced in process of insertion of knowledge to database. It allows user to provide filter – knowledge scheme. This module will not insert information provided directly to database but process it according to scheme. Scheme allow to parse and aggregate various type of information provided. *Table 3* provides part of knowledge scheme used as an example [4],[5],[6].

Table 3: Example entries in knowledge scheme file

1	Number of players > 3; Numeric; numberOfPlayers; Greater than; 3
2	Player position BB; Numeric; playerPosition; Equal; 6
3	First raise position SB; Numeric; firstRaisePosition; Equal; 5
4	Pair; Text; higherCardFigure; Equal; lowerCardFigure
5	Effective stack 5-6BB; Numeric; effectiveStackInBb; Between; 4; 7

Developed backward chaining algorithm works as follows:

1. Check if hypothesis is in database.
2. If yes then return true - current hypothesis is proven.
3. For each rule check if hypothesis is in rules consequent.
4. If Consequent from current rule matches hypothesis then iterate over antecedent list.
5. Take current element as sub-hypothesis and try to prove it - go back to point 1 and use it as hypothesis.
6. If any list element cannot be proven it means whole antecedent cannot be proven so
7. Move to next rule if possible - go back to point 4.
8. If you are out of rules then return false.

As it can be seen this is recursive algorithm. If at any moment during its execution every element of antecedent of rule that will prove main hypothesis is proven right algorithm finishes and returns true which means that hypothesis was proven. If algorithm is out of rules on highest step - while looking for rules that could prove main hypothesis, then it returns false which means that hypothesis cannot be proven with obtained facts.

2.5. Expert knowledge

In order to use expert knowledge in expert system it have to be expressed in form of rules. To do that many sessions with poker expert were performed. Long process ended up with statement of 135 points describing expert's strategy. Each of those points was brief of description of conditions for some specific situation that have to be met in order for making a raise decision. After that set of rules had to be prepared in order to feed expert engine. For testing purposes 314 poker table images were prepared. Each of them was labelled with proper decision thanks to expert's help so it could be used as reference point during measurements. Strategy points created along with expert were grouped and sorted from most common situations to most specific ones. Thanks to that it was possible to measure how each part will influence recognition rate and time. As mentioned earlier each point consists of conditions for raise decision for some specific in-game situation. Despite those points describes fully functional strategy and that they could be used in this exact form by someone to play they are still not useful for expert system. Those points could be easily translated to valid expert system rules by simply dividing each of them to groups of conditions which all have to be proven in order to prove raise decision. However that wouldn't be a proper solution. Whole next part of rule development was focused of finding groups and repetitions within points. Let's say there is *99+* in range description in some point. Even considering all other conditions have to be simply proven true, this means duplication of them in many rules for each of pairs: 99, TT, JJ, QQ, KK and AA and this is only considering 99+ in range section. Those rules would have to be multiplied each time new hand appears in range section. We can see that there would be minimum of 6 rules for 99+. When adding for example AJ+ we get 6x3 combinations. This pattern would follow for each additional hand with + suffix and 23+ would get us 156 combinations alone. Because of that first big set of rules described chain proving each possible hand, so for example mentioned 99+ could be used in rule and be understood by expert system. To accomplish that 680 rules have to be made. That may seem much in fact it greatly reduces future number of rules. Let's say expert system is processing a table where player have Ace and Jack suited and should raise according to strategy. Engine is trying to prove that and now is processing hand ranges in rule that will eventually prove raise decision. Rule says that player hand have to meet condition T9s+ [4],[5].

3. Results

3.1. Image recognition analysis

Image recognition module was developed in such manner that it is possible to tweak both MLP specific parameters which are maximum number of iterations, learning rate, error rate and momentum and also image preprocessing parameters: conversion to black and white, conversion to grayscale and image scaling. Firstly commonly used MLP parameters were selected in order to check how far using such presets could go. For image preprocessing firstly scaling image was set to 0,7 of its size and it was converted to black and white, because in case of images used colour was not giving any important information. Rest of values used were mentioned common settings for MLP and fortunately it was possible to get 100% correct recognition rate. First tests were made with reducing scale. Graphs below show how reducing image scale influenced success rate and average time per table recognition. First tests were made with reducing scale. *Figure 4* shows how reducing image scale influenced success rate.

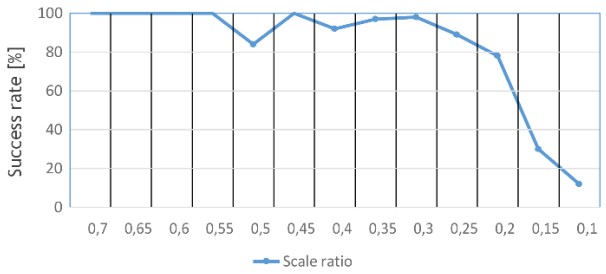

Figure 4. Success rate to scale ratio

As described earlier number of neurons in input and hidden layers are directly dependent on number of elements in flattened image table. Because of that this is not a surprise that table recognition time decreases as scale gets smaller because neural network gets simpler. As it goes for success rate it can be seen that it is stable till scale is 0.5. After that point some fluctuations can be seen and correct recognition rate starts to drop significantly after scale reaches 0.3 point. When image is scaled down its resolution decreases, so each pixel in scaled image is calculated as weight average of group of pixels from original image. After reaching 0.55 fluctuations are caused by this averaging. For some scales averages can make pictures more or less distinguishable. As an example we can see a drop at 0.5 scale but it gets better again at 0.45. Significant drop after 0.3 happens simply because images get scaled down so much that each scale reduction cause them to be more and more undistinguishable. Because 0.55 seems to be last stable point it was used for updating base values.

After dealing with scale learning rate was next. This value have to be carefully selected so weights converge to a response fast enough. Tests started with value lower than base one and then were gradually increased (*Figure 5*). It seems correct success rate drops when learning rate goes below 0.3

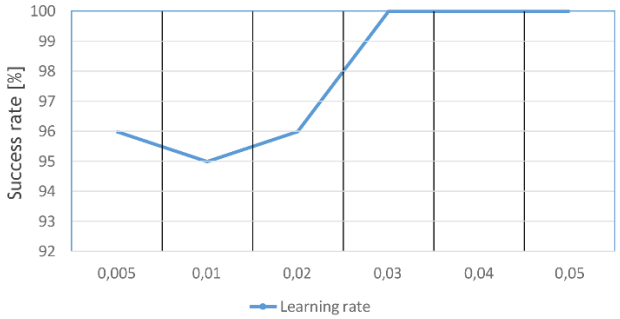

Figure 5. Success rate to learning rate

Next measured value was momentum. It determines how much each individual weight change depending to its previous change. In this case base value was used as starting point which was 0.9 and after that it was gradually decreased (*Figure 6*). Changes that appear looks more like caused by random error rather than parameter value change.

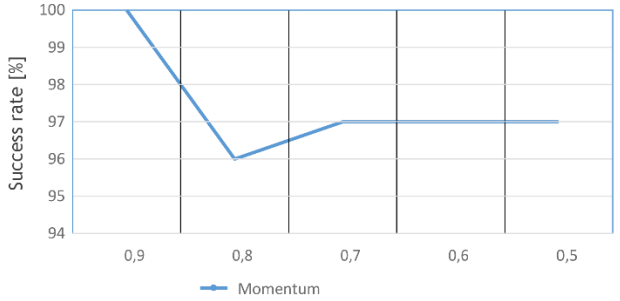

Figure 6. Success rate to momentum

Because base value for max iterations was fairly big measurements were started with 5000, then moved to 2000 and 1000. Later on success rate started to change so measurement was made every 100 iterations. After going below 100 success rate changed more rapidly so additional measurements were made for 50, 25 10 and 5 iterations (*Figure 7*).

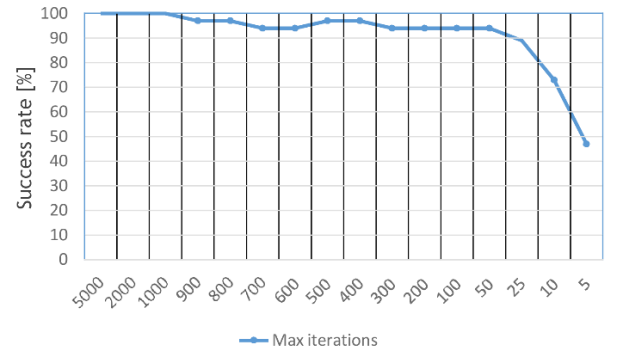

Figure 7. Success rate to max number of iterations

It can be seen that success rate is stable at 100% till passing 1000 iterations border. After that it fluctuates within 10% range and starts drastically dropping when number of iterations is less than 100. Those results seems to be reasonable and therefore 1000 max iterations were chosen for redoing learning rate and momentum measurements.

3.2. Expert system analysis

Strategy developed in form of rules along with expert system itself turned to be a great success. Finally it was possible to correctly recognize decision from all but one test table images in average time ~0.6 seconds per table. Including all utility rules it took exactly 1960 rules to accomplish that. As mentioned earlier rules were developed in iterative process following strategy points delivered by the expert. Thanks to that it was possible to perform measurements and evaluations all along the way.

In its current state engine is set so it tries to prove raise decision and assumes fold if is unable to. Because of that first test was performed with no rules at all. This can be considered also as "always fold strategy". Results weren't surprising. 221 out of 314 tables were recognized correctly which is about 70%. Average recognition time was 265ms per table. 70% may sound like a lot but in fact it is not reliable measurement in this situation. That was the reason two additional measurements were made: raises recognized correctly, and of course none of them was recognized correctly and folds recognized correctly with correct recognition rate of 100%. We can observe that basing on available sample player should raise in only 30% of situations and fold in other 70%. Summing all that up 70% correct recognition rate isn't telling anything about usefulness of application by itself. One cannot win if he/she won't play so with 0% correct raise recognition rate "always fold strategy" is useless. However this measurement will serve as reference point for following ones.

Next measurement was performed for basic strategy used previously for engine testing purposes. This is very simple strategy, but still a valid one. Point of this measurement is to get another reference point and be able to compare correct recognition rate, correct raise recognition rate and correct fold recognition rate of it to target strategy during further iterative development. Results were not as bad as predicted, 258 out of 314 tables were recognized correctly (82%), with 60/93 (64%) raises and 198/221 (89%) folds recognized correctly. This strategy consists of only 88 rules and average time of recognition increased only slightly by 24ms to 289ms total per table. Tab 4. presents results for two reference and target strategies after two iterations.

Table 4: Results - two reference strategies

	Correct	ms/ table	No. rules	Raises	Folds
Always fold strategy	221/314 (70%)	265	0	0/93 (0%)	221/221 (100%)
Basic strategy	258 /314 (82%)	289	88	60/93 (64%)	198/221 (89%)
Target strategy #1	240/314 (76%)	481	724	20/93 (21%)	220/221 (99%)
Target strategy #2	250/314 (79%)	981	835	33/94 (35%)	217/220 (98%)

At that point average recognition time per table seemed to be considerably big. If number of rules from #1 iteration were subtracted from #2 we see that it increased by 111. Assuming that "always fold strategy" average time is used for all other processing but rules, it can be calculated that 724 rules from #1 iteration take 216ms (481ms–265ms) to process while #2 iteration 835 rules take 716ms (918ms–265ms) on average to process. With those numbers it is easy to calculate that time increased by 332% while number of rules increased by 111 (15%). Those are highly alarming values considering that #2 iteration of target strategy consists of about 9% of whole expert knowledge.

4. Conclusions

It was possible to develop a system in form of desktop application which is capable of providing valid poker decision basing on poker table image and expert knowledge provided as an input. The average time which takes to perform whole process from the start to output decision and reasoning information is less than a second which makes whole system much more useful for end user who doesn't have to wait while performing his/her analysis. It was possible to minimize internal image recognition errors to the point that every of 314 table images used during testing was recognized correctly. What is more work done with expert was very fruitful. It took 1960 rules to express expert knowledge in form understood by expert system and fortunately system was able to achieve 99% accurate (according to the expert) correct decision making rate. Only one decision (rounded up to 1%) made was incorrect. As mentioned in previous sections it was caused by very rare and unexpected situation in game which conflicted with the way number of opponents were recognized. Summing up it was possible to solve main problems which were: image recognition and performing automatic reasoning according to knowledge provided along with minor ones which appeared along the way. System is very useful to end user which is intermediate poker player as he is able to get deep feedback to real situations he encountered as long as we assume expert knowledge provided as correct, but it was one of the main assumptions of this work.
System created can be considered a complete working product but it is not closed for extensions. In the future first think might be to deal with one situation which was recognized incorrectly. To do so way of opponent recognition have to be analysed and implemented again. Other than that future work might focus on dealing with system limitations. It was proven that it is possible for such system to work in limited environment, so next step could be to research whether it would be possible achieve similar results for other than first phases of poker hand were we have to deal not only with additional cards dealt on the table (which greatly increases complexity of the situation) but we also have to take into account previous decisions made. Because system was made in modular way it would be also possible to extend image recognition module to work with various online poker clients and their table layouts so it could also be used for players playing on other platforms. Finally if restricting to only first decision of each hand limitation was removed system could be extended to work with other Texas Hold'em (and probably other poker variations) game formats. This limitation was only one which was refraining from doing so as system was developed in the way that expert knowledge is delivered as separate file, so there is nothing else preventing from developing rules for different strategies and for different formats. All the time system could be kept in its modular form as it greatly improves both possibilities for future improvements and extensions.

References

[1] Bishop, Ch. M. (1995). Neural Networks for Pattern Recognition. *Birmingham, UK: Clarendon Press Oxford*.

[2] Butler, Ch. & Caudill, M. (1994). Understanding Neural Networks. *MA, USA: MIT Press Cambridge*.

[3] Damiani, E. (2004). Soft Computing in Software Engineering. *Berlin, Germany: Springer*.

[4] Joey P., *Skill or Luck.* http://realmoney.durrrrchallenge.com/skill-luck-paradox/.

[5] O'Meara, A. F., *Online No-Limit Texas Hold'em For Beginners*. http://www.gamblingsystem.biz/books/Online%20No-Limit%20Texas%20Hold'em%20Poker%20For%20Beginners%20(August%20O'%20Meara).pdf.

[6] Waterman, D.A. (1985). A Guide to Expert Systems. *Boston, USA: Addison Wesley Publishing Company*.