Symulacja w Badaniach i Rozwoju
Vol. 1, No. 2/2010

Katarzyna KOŚCIUK [1], Marek DRUŻDŻEL [1,2]
[1] Faculty of Computer Science, Białystok Technical University, Białystok
E-mail: k.kosciuk@pb.edu.pl , m.druzdzel@pb.edu.pl

[2] Decision Systems Laboratory, School of Information Sciences,
University of Pittsburgh, Pittsburgh, PA 1526
E-mail: marek@sis.pitt.edu

Player Modeling Using Bayesian Networks

1 Introduction

Typically programs for game playing use the Minimax strategy [1], which assumes perfectly rational opponent, who always performs optimal actions. However, most human opponents depart from rationality [2]. In this case, the best move at any given step may not be one that is indicated by Minimax but rather by an algorithm that takes into consideration human imperfection. In order to consider a player's weaknesses, it is necessary to model the player – learn and know his/her strategies.

In this paper, we describe our attempt to model players of a variant of the Tic-Tac-Toe game. The underlying assumption, supported by both anecdotal and empirical evidence, is that games can benefit from adapting to a particular opponent, rather than using the same general strategy against all players. We implement a program playing Tic-Tac-Toe using (1) conventional Minimax strategy, (2) the Minimax with Alpha-beta pruning, that searches the game tree in order to choose the best move for the computer, and (3) a variant of (1) that plays until the very end, even if the game appears to be lost. We add (4) an algorithm that makes decisions based on a model of opponent's weaknesses. The algorithm includes a learning module that observes the opponent carefully and learns his/her strategies and weaknesses. We study the improvements of the algorithm over time and test it against the Minimax strategy. We build a Bayesian network to model the player. We learn the conditional probability tables in the network from data collected in the course of the game.

This paper is organized as follows. Section 2 describes the Minimax algorithm. Section 3 presents the structure of our Bayesian network model. Section 4 describes our program. The results of experiments we presented in Section 5. Finally, Section 6 proposes some ideas of future work.

2 The Minimax algorithm

One way to pick the best move in a game is to search the game tree using the Minimax algorithm or one of its variants. The game tree is a directed graph in which nodes are positions in a game and edges are the moves. There is a value associated with each position or state of the game, and this value is computed by means of an evaluation function. This value defines how good it would be for a player to achieve this position.

The Minimax is a recursive algorithm for choosing the next move in an *n*-player game. At each step of the game, each player, by assumption, maximizes his or her gain. Player A, following the Minimax strategy, will pick a move that maximizes A’s chances of winning. When considering player B's move, A assumes that B will maximize B's chances of winning. Without any loss in generality, we will call the player A – MAX, the player B – MIN, and we assume that MAX starts the game.

A technique called alpha-beta pruning is an improvement on the basic Minimax strategy. It reduces the number of nodes that need to be evaluated, and stops evaluating a move as soon as it can prove that the move cannot be better than the best move found so far. In this way, alpha-beta pruning eliminates some branches of the search tree and leads to significant computational savings.

Minimax assumes that the opponent always chooses the best move. This assumption seems to have little ground in practice, as human players often lack the cognitive ability to find the optimal move. Effectively, they may significantly depart from what Minimax would prescribe as the optimal or rational move. People can choose an inferior move because they do not know or do not notice a better move. An algorithm that takes this into consideration should perform better than Minimax.

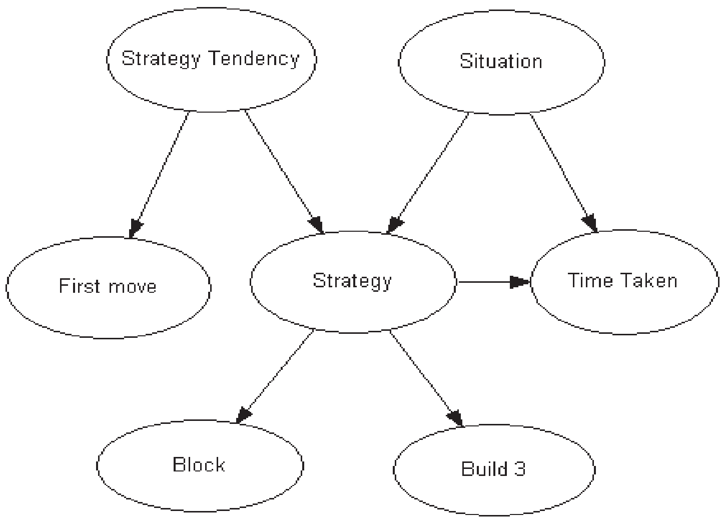
3 Bayesian Networks

A Bayesian network [4] is an acyclic directed graph, whose nodes represent random variables, such as observable quantities, latent variables, unknown parameters or hypotheses. The edges represent direct influences. Nodes that are not connected represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability distribution over the variable represented by the node.

Our program stores the following information about the opponent:

- the place where the opponent puts the first move,
- the opponent's long-term strategy,
- types of moves,
- situation on the board,
- how long the player thinks before a move.

We capture this information in the Bayesian network shown in Figure 1.



Rys. 1. Sieć Bayesowska
Fig. 1. Bayesian Network

The node *First_move* represents the place where the player puts the first mark. It has three possible states: *corner*, *side*, and *center*. It might seem that there are 16 possible positions (corresponding to the 16 free places on the board). However, because of the symetry of the board, many of them are equivalent (e.g. placing the mark in any corner).

The node *Block* has two states: *yes* and *no*. It assumes the state yes when the player puts a mark 'X' in order to block two 'O's in row, otherwise it assumes the state *no*.

The node *Build_3* has two states: *yes* where the player puts two 'X' in row, and has a opportunity to put the third 'X' in row, otherwise the state is *no*.

The node *Time_taken* represents how long the player thinks before his/her move.

The node *Strategy* corresponds with the player's strategy in the particular move:

- *Attack* – the player attacks and sometimes does not see the necessity of defense.
- *Defense* – the player blocks 'O's (even unnecessary, when he/she has a opportunity to win).
- *Attack_Defense* – the player chooses the move that block 'O's and build three 'X' in row, or the move that does not change the situation on the board.

The node *Strategy_Tendency* represents the player's general strategy:

- *Attack* – the player has a tendency to attack more often than to defend.
- *Defense* – the player has a tendency to block more often than to attack.
- *Attack_Defense* – the player sometimes blocks, sometimes attacks

The node *Situation* corresponds with the situation on the board. We distinguish seven situations:

- *UnderAttack* – two unblocked 'O's in row,

- *BeforeWin* – two unblocked 'X's in row,
- *Trap* (fork) - an opportunity where 'O' can win in two ways,
- *Initial* – before the first move,
- *Scarce* – a lot of free places n the board,
- *Dense* – a few free places on the board,
- *AttackedBeforeWin* – two unblocked 'O's in row and two unblocked 'X's in row (player has an opportunity to win, but sometimes does not see it and blocks 'O's instead).

4 The Program

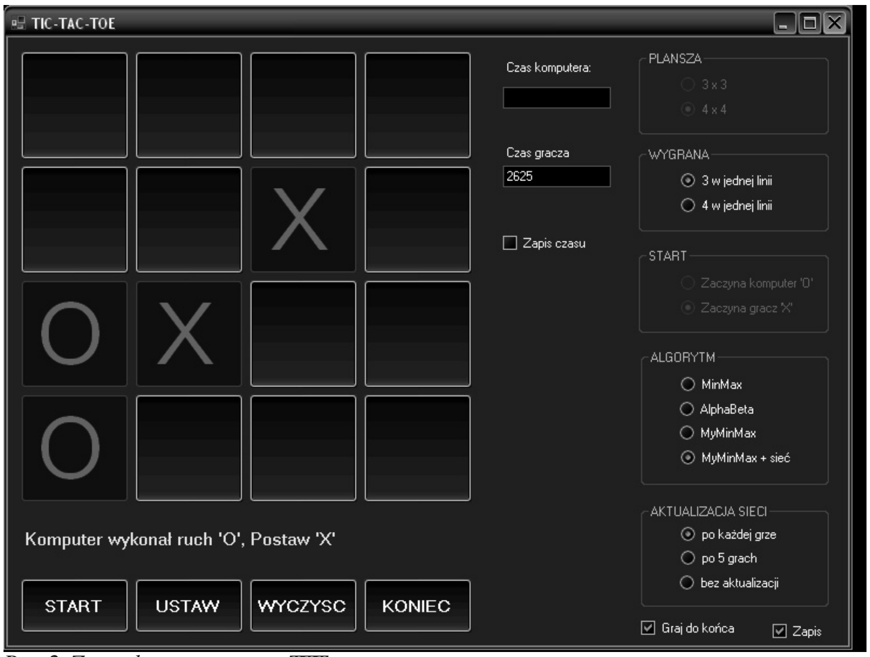
Our program consists of the four algorithms mentioned in the introduction: (1) the Minimax algorithm, (2) the Minimax algorithm enhanced with Alpha-beta pruning, (3) Minimax that does not surrender, and (4) Minimax + Bayesian Network. The program plays a variant of the Tic-Tac-Toe game. We choose the board dimension 4x4. The winning position are three 'X' or three 'O' in the same line. For simplification, we assume that the computer plays 'O' and the human player plays 'X'. We establish the following evaluation function F:

- F=0 a draw,
- F=1 computer wins,
- F=-1 computer looses.

In such variant of the Tic-Tac-Toe, the player who starts the game always wins, at least when she/he always chooses optimal moves. We noticed, consistently with our original assumption, that people sometimes lose because they do not choose the best move.

In our prior work [3], we tested a simple modification of the Minimax algorithm, which never gave up playing, even if the situation on the board appeared lost (we call this modification "persistent Minimax"). When there are several moves with the evaluation function equal 1, the persistent Minimax choses the move that lets the computer win fastest. When there are more than one move with the same F value and the same end-depth value, the persistent Minimax algorithm returnes a random move. Original Minimax always returns the first or the last move in such situations (depending on the implementation).

In the current paper, we improve on this algorithm – when there are several moves with the same F value and the same end-depth value, our program consults the Bayesian network about the opponent's strategy.



Czas komputera:

Czas gracza: 2625

[ ] Zapis czasu

PLANSZA: ( ) 3 x 3 [✓ 4 x 4]

WYGRANA: [✓ 3 w jednej linii] ( ) 4 w jednej linii

START: ( ) Zaczyna komputer 'O' [✓ Zaczyna gracz 'X']

ALGORYTM: ( ) MinMax ( ) AlphaBeta ( ) MyMinMax [✓ MyMinMax + sieć]

AKTUALIZACJA SIECI: [✓ po każdej grze] ( ) po 5 grach ( ) bez aktualizacji

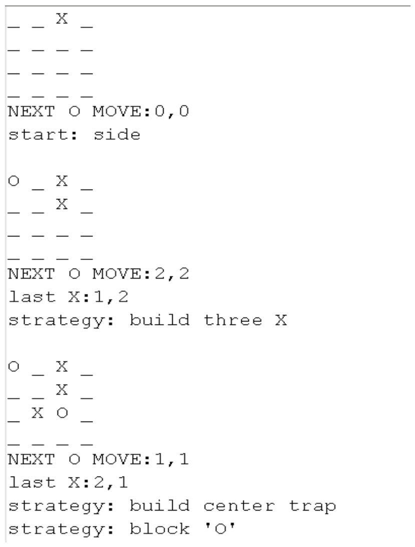
[✓] Graj do końca [✓] Zapis

Komputer wykonał ruch 'O', Postaw 'X'

Rys. 2. Zrzut ekranu programu TTT
Fig. 2. A screen shot of the TTT program

We can actualize the network after every game or after five games. Every move is written to the file, every game is analyzed.

We assume that a player with the offensive strategy often tries to put three 'X' in row or build a trap. The player with the defensive strategy often puts 'X' near 'O', trying to block rather than build three 'X's. The examples of data suggesting the specific strategies are shown below.

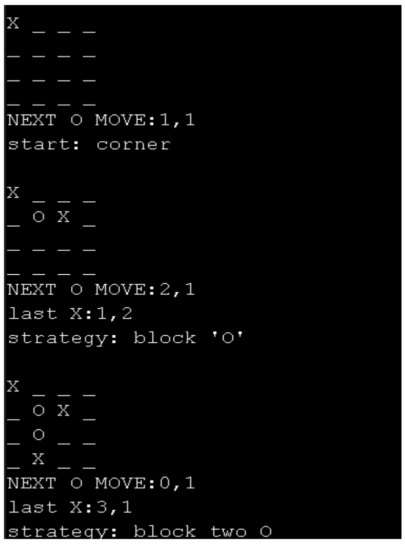


```
_ _ X _
_ _ _ _
_ _ _ _
_ _ _ _
NEXT O MOVE:0,0
start: side

O _ X _
_ _ X _
_ _ _ _
_ _ _ _
NEXT O MOVE:2,2
last X:1,2
strategy: build three X

O _ X _
_ _ X _
_ X O _
_ _ _ _
NEXT O MOVE:1,1
last X:2,1
strategy: build center trap
strategy: block 'O'
```

Rys. 3. Przykład danych sugerujących strategię ofensywną
Fig. 3. The example of data suggesting the offensive strategy



```
X _ _ _
_ _ _ _
_ _ _ _
_ _ _ _
NEXT O MOVE:1,1
start: corner

X _ _ _
_ O X _
_ _ _ _
_ _ _ _
NEXT O MOVE:2,1
last X:1,2
strategy: block 'O'

X _ _ _
_ O X _
_ O _ _
_ X _ _
NEXT O MOVE:0,1
last X:3,1
strategy: block two O
```

Rys. 4. Przykład strategii defensywnej
Fig. 4. The example of defensive strategy

5 The experiments

The experiments were carried out with the 7-old boy. The Human player ‘X’ always started the game, because there were no chance of winning in another situations. Here are the average results:

After 20 games:

- Minimax + Bayesian Network won in the 14 cases.
- Conventional Minimax won in 1 case.

After 60 games:

- Minimax + Bayesian Network won in the 25 games,
- Conventional Minimax won in 1 game.

After about 30-40 games the human player learn how to win the game.

The results are similar to the previous results [3], because of the simplicity of the game.

6 Conclusions and future work

We believe that games (understood in broad terms) can benefit from adapting to a particular opponent, rather than using the same general strategy against all players.

Algorithms that take into consideration human’s suboptimality can be expected to perform better than Minimax.

The gain from modeling the opponent in tic-tac-toe does not seem to be large – our previous program, modified Minimax, performs just as good. One of the reasons for this modest gain may be simplicity of the game. We plan to experiment with more complex games, such as bridge.

Referencesa

1. von Neumann J.,Morgenstern O.: *Theory of Games and Economic Behavior*, Princeton University Press, Princeton NJ., United States 1944
2. Simon A. H.: A Behavioral Model of Rational Choice. *The Quarterly Journal of Economics*, Vol. 69, No. 1, pp. 99-118, The MIT Press, United States 1955
3. Kościuk K., Drużdżel M.: Exploring opponent’s weaknesses as an alternative to the Minimax strategy. XV Warsztaty Naukowe PTSK : *Symulacja w Badaniach i Rozwoju*, Poland 2008
4. Pearl, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, San Mateo, CA: Morgan Kaufmann Publishers 1988

Summary

Typically programs for game playing use the Minimax strategy, which assumes a perfectly rational opponent whose actions are performed optimally. However, most human opponents depart from rationality. In this case, the best move at any given step may not be one that is indicated by MiniMax and an algorithm that takes into consideration humans imperfection will perform better. In order to consider player's weaknesses, it is necessary to model the opponent – learn and know his/her strategies. We build a Bayesian network to model the player. We learn the conditional probability tables in the network from data collected in the course of the game.

Katarzyna KOŚCIUK, Marek DRUŻDŻEL

Modelowanie gracza przy użyciu sieci Bayesowskiej

Streszczenie

Algorytmy grające w gry zazwyczaj używają strategii Minimax zakładającej perfekcyjność przeciwnika, który wybiera zawsze najlepsze ruchy w grze. Gracze jednakże mogą nie działać całkiem racjonalnie. Algorytm, który weźmie to pod uwagę, może dawać lepsze wyniki niż Minimax.

Aby wykorzystać słabości przeciwnika, należy stworzyć jego model. W tym celu zbudowaliśmy sieć bayesowską, w której tworzymy tablicę prawdopodobieństw z danych zbieranych w trakcie gry.