Optimal strategy in games with chance nodes.

Article *in* Acta Cybernetica · January 2007

Source: DBLP

2 authors, including:

Benedek Nagy
Eastern Mediterranean University
212 PUBLICATIONS **1,251** CITATIONS

SEE PROFILE

Acta Cybernetica **18** (2007) 171–192.

Optimal strategy in games with chance nodes

Ervin Melkó* and Benedek Nagy†

Abstract

In this paper, games with chance nodes are analysed. The evaluation of these game trees uses the expectiminimax algorithm. We present pruning techniques involving random effects. The gamma-pruning aims at increasing the efficiency of expectiminimax (analogously to alpha-beta pruning and the classical minimax). Some interesting properties of these games are shown: for instance, a game without draw can be fair. A fair game may not be fair any more if it is played iteratively. To handle these phenomena, the use of additional indicators, such as the minimal guaranteed outcome value, is suggested.

Keywords: Game Theory, Artificial Intelligence, Game tree, Games with chance nodes, Expectiminimax algorithm, Pruning, Iterative games, Fair games

1 Introduction

Game Theory is an important field of Artificial Intelligence. Modern game theory was defined by von Neumann and Morgenstern [6, 7]. Game theory deals with decision problems in an environment where another agent or agents may have different aims. The theory of two player strategic games is well developed [5, 9, 10]. A subfield, the theory of fixed sum games, is equivalent to the family of zero-sum games, where two players have opposite aims. In games with perfect information players have all the information about the game to help to make their decisions (also called steps or moves). These games are represented by game trees. In these graphs there are two kinds of nodes representing the decision points of the two players. Game theory deals with well-defined games where players can choose among a fixed set of actions. NIM, Tic-Tac-Toe, Othello [1] and Chess are representative elements of this set. The first computer chess program was developed by Shannon and Turing in 1950. One of the main aims of the artificial intelligence research was to write a chess program that can beat the human world chess champion. It was a long process, but nowadays computer programs do beat human champions.

*University of Debrecen, Debrecen, Hungary, E-mail: melko.ervin@gmail.com

†University of Debrecen, Debrecen, Hungary and Rovira i Virgili University, Tarragona, Spain, E-mail: nbenedek@inf.unideb.hu

Game and decision theories are very important fields of Economics, as well. The Decision Theory deals, for instance, with problems represented by decision trees, in which a person has some decision points and there are some other (uncertain) events represented by (chance) points. Usually, these other points represent cases which occur randomly. The expected values can be computed to help in choosing a branch at decision points.

In this paper, we deal with fixed sum two player strategic games that include an element of chance. A well-known example for this type of game is the Backgammon. The order of players and possible random events with their probabilities are known by the players, therefore they can compute their optimal strategies. In the next section we recall some well-known elements of the theory of decision trees (one player "game" with random effects) and of the theory of two player non-random games. In Section 3.1, connecting these two theories, a method is shown to compute an optimal strategy in games with chance nodes. In Section 3.2 this algorithm, called *expectiminimax* is extended by introducing the pruning of the game trees. Because the game tree contains chance nodes, we introduce new pruning techniques, resulting in the algorithms of gamma-pruning. These methods give fast exact evaluation of the game, therefore one can find the optimal strategy faster than by analysing the full game-tree.

The games including chance nodes are more complicated than the games without chance. For example, a fair game without random effects remains fair if it is played repeatedly. In Section 4, we present an interesting example, where a fair game may cease to be fair if it is played repeatedly. We show variants of the previous algorithms that can help in the evaluation of such games by computing the minimal guaranteed outcome. Finally, in Section 5 we summarise the conclusions of the paper.

We note here that the topic of the paper has a strong relation with multiplayer games (or games with few players [4]).

2 Preliminaries

In this section we recall some basic definitions and basic algorithms, facts about the topic of decision trees and two player game trees based mostly on [10].

2.1 The theory of decision trees

Let A be a player who has some decision points. There are also some random events (their probabilities are known).

Formally, let the problem be the following. There is a tree with two kinds of nodes. At the so-called *decision nodes*, the player chooses a successor node (branch of the tree). At the nodes of the other type, the *chance nodes*, a random event happens: the successor node is chosen randomly with probabilities known in advance. The leaves of the tree represent situations where the utility function (the payoff, the value of the situation) is known. The question is the strategy, i.e.

what should the player's choice be at the decision points to achieve the maximal (expected) payoff.

The aim of the player is to maximise the outcome value and for this purpose she/he chooses the child-node (i.e. the branch of the tree) with the highest possible expected value.

Let $P(N)$ denote the probability of the event N (supposing that we are at the parent of the node representing N). In Figure 1, an example is shown. There are decision nodes, where the player chooses among the next possible nodes (represented by rectangles in the figure). At nodes marked by a circle a random event will determine the next node. (The probabilities are written on the edges.)

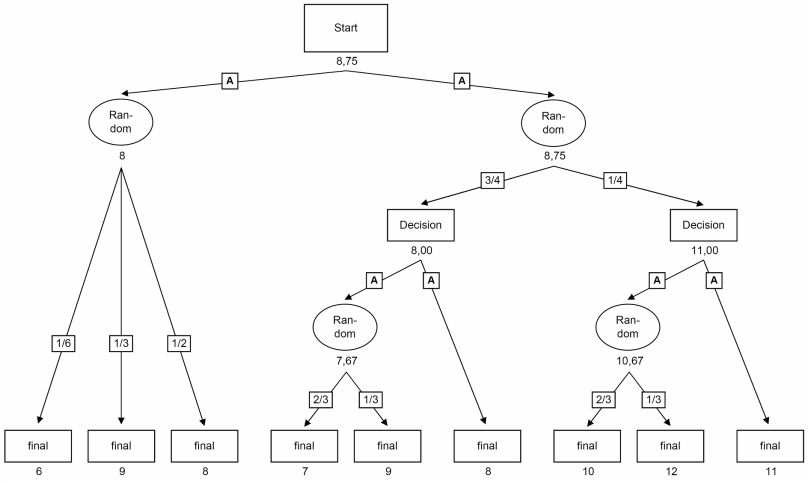

Figure 1: A decision tree

Algorithm 1. (DECISION)

```
1  function D(N) is
2  begin
3    if N is a leaf then
4      return the value of this leaf
5    else
6      Let N_1, N_2, ..., N_m be the successors of N
7      if N is an A node then
8        return max{D(N_1), ...,D(N_m)}
9      if N is a C node then
10       return P(N_1)D(N_1) + ... + P(N_m)D(N_m)
11 end D
```

In Figure 1 the nodes of the tree are assigned a numeric value (shown under the rectangles and circles representing the nodes of the tree), as described by Algorithm 1.

The technique presented here is called *expectimax*, since at decision points of the player (A nodes) it computes the maximum of the expected values of the successor nodes, while at the random effects (chance nodes or C nodes, for short) the expected value of the values of the successor nodes is computed.

Theorem 1. *If the decision tree is finite, Algorithm 1 gives the possible maximal expected value that the player can achieve.*

Note that every decision tree can be transformed to a *layered* (stratified) decision tree, i.e. a tree in which the nodes at the same depth are of the same type. If the player has to make multiple decisions in a sequence, then this sequence can be replaced by a complex decision. Similarly if random events follow each other, then this can be viewed as a single complex random effect (using probability theory).

In this paper we deal only with problems having a bounded set of possible outcomes. This is important to avoid some paradoxes of the probability theory. For instance the following game is known as the Saint-Petersburg paradox [8]. Drop a coin, if it is a head then drop again. If it is a tail, then you get \$ 2^{k+1} and the game ends, where k is the number of heads before. What is the value of this game? How much should you pay for a game if we want to do it in a fair way? On one hand, the expected value is $\frac{1}{2}\cdot 2^1+\left(1-\frac{1}{2}\right)\cdot\frac{1}{2}\cdot 2^2+\left(1-\frac{1}{2}\right)^2\cdot\frac{1}{2}\cdot 2^3+...+\left(1-\frac{1}{2}\right)^k\cdot\frac{1}{2}\cdot 2^{k+1}+... = \sum_{k=1}^{\infty}\left(\frac{1}{2^k}\cdot 2^k\right) = \sum_{k=1}^{\infty} 1 = \infty$. So, in principle, it is worth to buy the right to play for an arbitrarily large price. On the other hand, everybody has the feeling that it would not be worth to play for a very high price. In practice there are no people who would pay, let us say, \$ 1 000 000 000 for a game. The chance to win more than this amount is less than 1 : 500 000 000, since at least 29 consecutive heads are needed to begin the sequence. Thus the chance to loose a very high amount of money is very close to 1.

2.2 The theory of two player strategic games

A game is defined as follows. Let two players be given. They take turns in making moves, until the game is over. At every point of time the game has a state (e.g. the state of the board) and the player to make the next turn is known. In every state the possible moves form a well-defined (finite) set. Player A starts the game from its initial position. There are terminal states, in which the utility value of the state is given, for each player. In case of fixed sum games, when the sum of utility values is known, one utility value is enough to determine the result of the game at a given state.

In simple games, there are only two outcomes: A wins or B wins. In some games the result can be a draw. In more complex games a score is computed, and the final score is of importance. In these games the set of utility values can be very large. However, recall that we deal only with games which have a finite set of

utility values.

The following graphical representation of a game is called a game tree. The nodes of the tree represent the game states, while the arcs represent moves. There are two kinds of nodes representing the decision situations of player A and B, respectively. At *A nodes* player A chooses a successor node, while at *B nodes* the decision is B's. At the root node (the initial state) player A has the decision. The leaves represent terminal positions with separate utility values for the two players.

We only deal with fixed-sum games. Therefore it is enough to specify the utility value for player A, as the value for B can be easily computed. A terminal position where A wins is thus assigned a value which is always greater than the value of a position where B wins. We also allow draws, and arbitrary intermediate values.

We are thus considering two player, fixed-sum, perfect information finite games. We assume that the two players (A and B) take turns and try respectively to maximise and minimise the utility function, the value of the game.[1] Since players take turns, successive nodes represent positions where different players must move. We consider only games with a finite game tree.

In Figure 2 a part of the game tree of the well-known game Tic-Tac-Toe is shown [10]. In the upper part of the figure the states are shown, while in the lower part, in a more abstract way, the triangles show the player who can choose from the next states (if any). Triangles $\triangle$ represent decision points for player A, while triangles $\triangledown$ represent decision points for player B. The numbers in the figure represent so called minimax values, which are discussed below.

2.2.1 The minimax algorithm

Assuming a perfect opponent for deterministic games, a feasible strategy is as follows: choose a move to a position with the highest minimax value. In this way the best achievable payoff against best play can be reached.

The minimax game strategy for player A (B) is to select the move that leads to the successor node with the highest (lowest) value. The values are computed starting from the leaves of the tree and are propagated upwards to their predecessors in accordance with the minimax algorithm. Algorithm 2 below [2] is a concise formulation of the minimax principle which works properly, even if the nodes types do not alternate (similarly to Algorithm 1).

[1] If a game allows a sequence of steps performed by the same player, then this can be viewed as a single complex step. If, in the original game, only a fixed number of such step sequences is allowed, then the resulting game will be finite, as well. Such a transformation can help to use the theory described in the sequel.

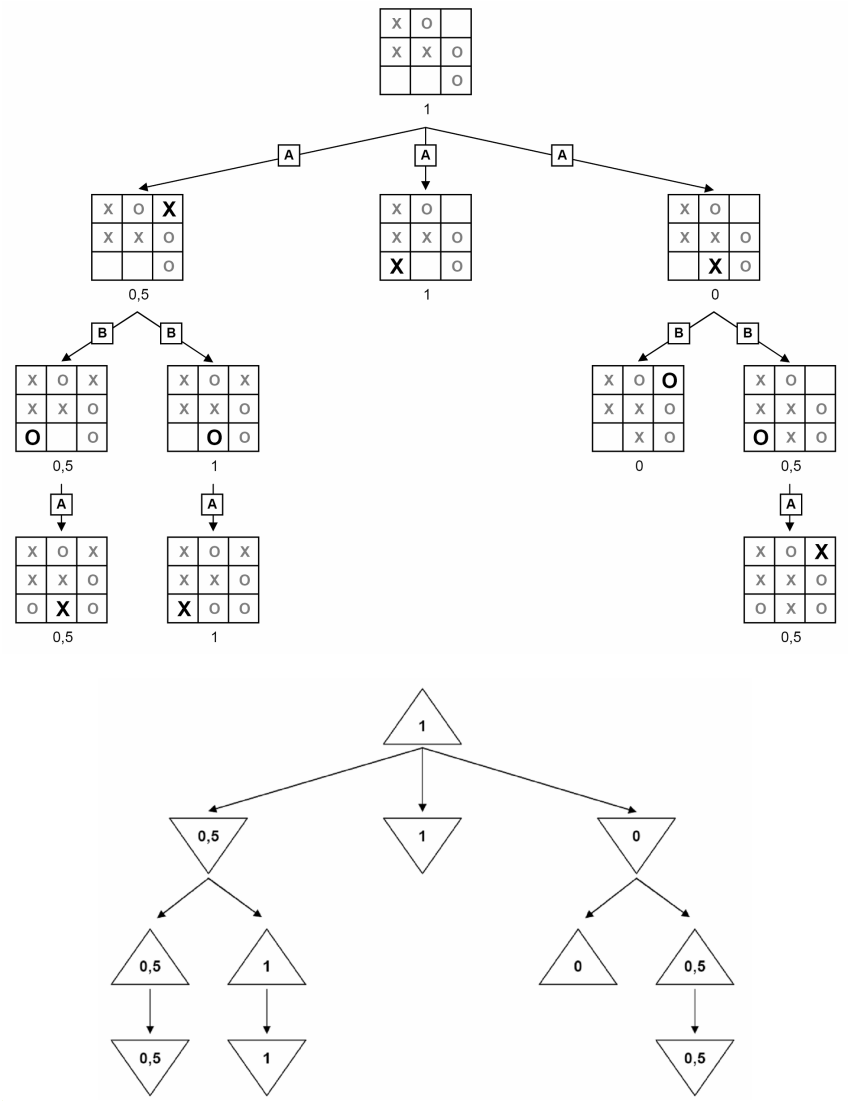

Figure 2: A part of the game tree of Tic-Tac-Toe

Algorithm 2. (MINIMAX)

```
1  function MM(N) is
2  begin
3    if N is a leaf then
4      return the value of this leaf
5    else
6      Let N_1, N_2, ..., N_m be the successors of N
7      if N is an A node then
8        return max{MM(N_1), ...,MM(N_m)}
9      if N is a B node then
10       return min{MM(N_1), ...,MM(N_m)}
11 end MM
```

The *value* of a (non-random) game is the payoff that both players can guarantee. So, if both players are playing well, the game will be finished in a state having the value of the game.

A (non-random) game is said to be *fair* if both players can guarantee at least the draw.

The following theorem is well-known about the correctness of the algorithm.

Theorem 2. *The minimax algorithm gives the exact value of any two player strategic game represented by a finite game tree.*

Theorem 2 is von Neumann's minimax theorem [6] for game trees (as formulated in [2]).

As a consequence of the previous theorem we can say that a two player strategic game cannot be fair, if draws are not allowed. In this case one of the players must have a winning strategy.

In Figure 2 the numbers beneath the nodes are the values assigned to the nodes by the minimax algorithm.

The problem with this algorithm is that it explores all nodes of the tree. In the next subsection we recall some methods which prune parts of the game tree and thus accelerate the evaluation.

2.2.2 Alpha and beta pruning

Alpha-beta cutoff is a method for reducing the number of nodes explored by the minimax algorithm described above. For each node it explores, it computes an alpha value and a beta value.

One can make the decision about the best choice without knowing the values of all successor nodes. The alpha and beta values, which are estimations of possible maximum and minimum values, can help in this.

As a first step we divide Algorithm 2 to define two separate functions for computing the value of A nodes and B nodes, respectively. This form of the algorithm can be found, for instance, in [10].

Algorithm 3. (MINI and MAX)

```
1  function maxvalue(N)
2   begin
3   if N is a leaf then
4      return the value of this leaf
5   else
6      let v = −∞
7      For every successor N_i of N do
8        let v = max{v,minvalue(N_i)}
9   return v
10  end maxvalue

1  function minvalue(N)
2   begin
3   if N is a leaf then
4      return the value of this leaf
5   else
6      let v = +∞
7      For every successor N_i of N do
8        let v = min{v,maxvalue(N_i)}
9   return v
10  end minvalue
```

Step 8 of the functions does the main operation, it calculates the maximal and the minimal value, respectively. (This new form of the minimax algorithm works only if the levels of A nodes and B nodes alternate in the tree.)

We now discuss how short-cuts can be used to eliminate some parts of this evaluation process. The alpha and beta pruning algorithm can abandon the evaluation of a step when at least one possibility has been found that proves the step to be worse than a previously examined step. Such moves need not be evaluated further. For this, we explain the role of alpha and beta values that are approximations of the real value of the node.

The *alpha value* of a node is a value, which is never greater than the real value of this node. Initially it is the value of that node if the node is a leaf, otherwise it is minus infinity. Then, at an A node, it is set to the largest of the values of its successors explored up to now, and at a B node, to the alpha value of its predecessor.

The *beta value* of a node is a value, which is never smaller than the real value of this node. Initially it is the value of that node if the node is a leaf, otherwise it is (plus) infinity. Then, at a B node, it is set to the smallest of the values of its successors explored up to now, and at an A node, to the beta value of its predecessor. It is guaranteed that:

The real value of a node will never be less than the alpha value and never greater than the beta value of that node. As the algorithm evolves, the alpha and beta

values of a node may change, but the alpha value will never decrease and the beta value will never increase. When a node is visited for the last time, its value is set to the alpha value of that node if it is an A node, otherwise it is set to the beta value. For more details we refer to [3, 10]. We formalise this method as Algorithm 4 below.

Algorithm 4. (ALPHA and BETA PRUNING)

```
1  function alphamax(N, α, β)
2   begin
3   if N is a leaf then
4      return the value of this leaf
5   else
6      let v = −∞
7      For every successor N_i of N do
8        let v = max{v,betamin(N_i, α, β)}
9        if v ≥ β then return v
10       let α = max{α, v}
11  return v
12  end alphamax

1  function betamin(N, α, β)
2   begin
3   if N is a leaf then
4      return the value of this leaf
5   else
6      let v = +∞
7      For every successor N_i of N do
8        let v = min{v,alphamax(N_i, α, β)}
9        if v ≤ α then return v
10       let β = min{β, v}
11  return v
12  end betamin
```

Observe that $\alpha \leq \beta$ for every node at every time.

The evaluation technique using Algorithm 4 is the so-called alpha-beta pruning (or alpha-beta minimax) algorithm.

The two functions call each other because the type of the nodes in the game tree alternate, so we can speak about A-layers and B-layers of the game tree.

If player A starts the game, then the alphamax function is called with the root of the game tree, as its first parameter, minus infinity ($-\infty$), as the value of α, and plus infinity ($+\infty$) as the value of β. We note here that usually (always in case of games with bounded set of outcomes) the value $-\infty$ (∞) is replaced by any number smaller (larger) than the lower (upper) bound of the outcomes.

The alpha-beta pruning algorithm is correct as the next theorem (based on [1]) states.

Theorem 3. *For finite non-random games Algorithm 4 results the same value as Algorithm 2.*

Thus, pruning does not affect the final result, so the same (exact) value can be computed in a much shorter time.

Note that the speed of the alpha-beta algorithm depends very much on the order in which the branches of the game tree are considered [3].

3 Two player games with random effects

We now introduce some extensions of the previous theories. Games with random events in the successor function (such as dropping a coin, rolling a dice) can be analysed using the methods described in this section. The so-called expectiminimax algorithm is presented, and some acceleration techniques are shown.

The problems we deal with are represented by trees. These trees are related to game trees and decision trees, as well. There are three types of nodes. At A nodes and B nodes player A and B makes the decision, respectively. The third type of node represents the random events. These new nodes, called the chance nodes (or type C nodes) play the same role as in decision trees.

Let us see an example.

Let the game be a modified version of Tic-Tac-Toe. Player A starts the game (by placing an 'X' in an empty cell), player B follows (by placing a 'O'), and then a random event happens, as Figure 3 shows. Places 1 and 4 will be interchanged (event w_1) with a probability $P(w_1) = \frac{1}{2}$, or places 2 and 6 will be interchanged (w_2) with a probability $P(w_2) = \frac{1}{3}$, or places 5 and 9 will be interchanged ($P(w_3) = \frac{1}{6}$). The game is continued for 4 iterations of steps A-B-random. Finally, player A wins if she/he has 3 'X' signs in a row (or in a column or diagonally), but B has no 3 'O'-s in such a way. B wins if there are 3 'O'-s in a row (in a column or diagonally), but there are no 3 'X'-s in this way. The result is a draw, if both or none of the players have 3 of their signs in a row (column or diagonal). Let us fix the value of the game as 1 point. The winner gets the whole point. At draw both players get a half point.

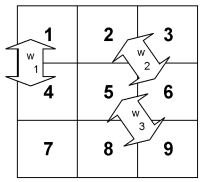

Figure 3: TIC-TAC-TOE with some random events

Because of the influence of random effects, the aim is to optimise the expected value of the game. (In games with no random effects, the game has an exact value;

in games of chance the expected value plays the same role.) The concept of *fair game* can be extended to these games: a game with chance nodes is fair if both players can guarantee the draw as the expected value.

Proposition 4. *If the game tree has no layers, then it can be modified to an equivalent one which has layers in a fixed order.*

This can be done by uniting consecutive nodes of the same type, and by introducing some new nodes where the branching factor of the tree is 1.

3.1 Modified minimax

In this section the basic evaluation algorithm is presented. While player A chooses the possible maximum, and player B chooses the possible minimum value, because of their aims, at chance nodes (of type C) there is a random event where the expected value (weighted average) is calculated. $P(N)$ denotes the probability of the fact that node N will be chosen as the successor of its type C parent node at a game. Now we formulate the general form of the so-called expectiminimax algorithm, adapted from [10].

Algorithm 5. (EXPECTIMINIMAX)

```
1  function EMM(N) is
2  begin
3    if N is a leaf then
4      return the value of this leaf
5    else
6      Let N_1, N_2, ..., N_m be the successors of N
7      if N is an A node then
8        return max{EMM(N_1), ...,EMM(N_m)}
9      if N is a B node then
10       return min{EMM(N_1), ...,EMM(N_m)}
11     if N is a C node then
12       return P(N_1)EMM(N_1) + ... + P(N_m)EMM(N_m)}
13 end EMM
```

We say that player A plays well, if she/he is playing with a strategy to maximise the expected payoff, while player B plays well, if she/he is minimising the expected payoff of the game.

Theorem 5. *If both players play well, then Algorithm 5 gives the expected value of the game.*

If any of the players follows another strategy, then her/his expected outcome will be less favourable.

Let us analyse the game shown in the previous section. For simplicity, let our start state be a Tic-Tac-Toe board shown at the root of Figure 4. The evaluation of this part of the game tree by Algorithm 5 can be seen in the figure.

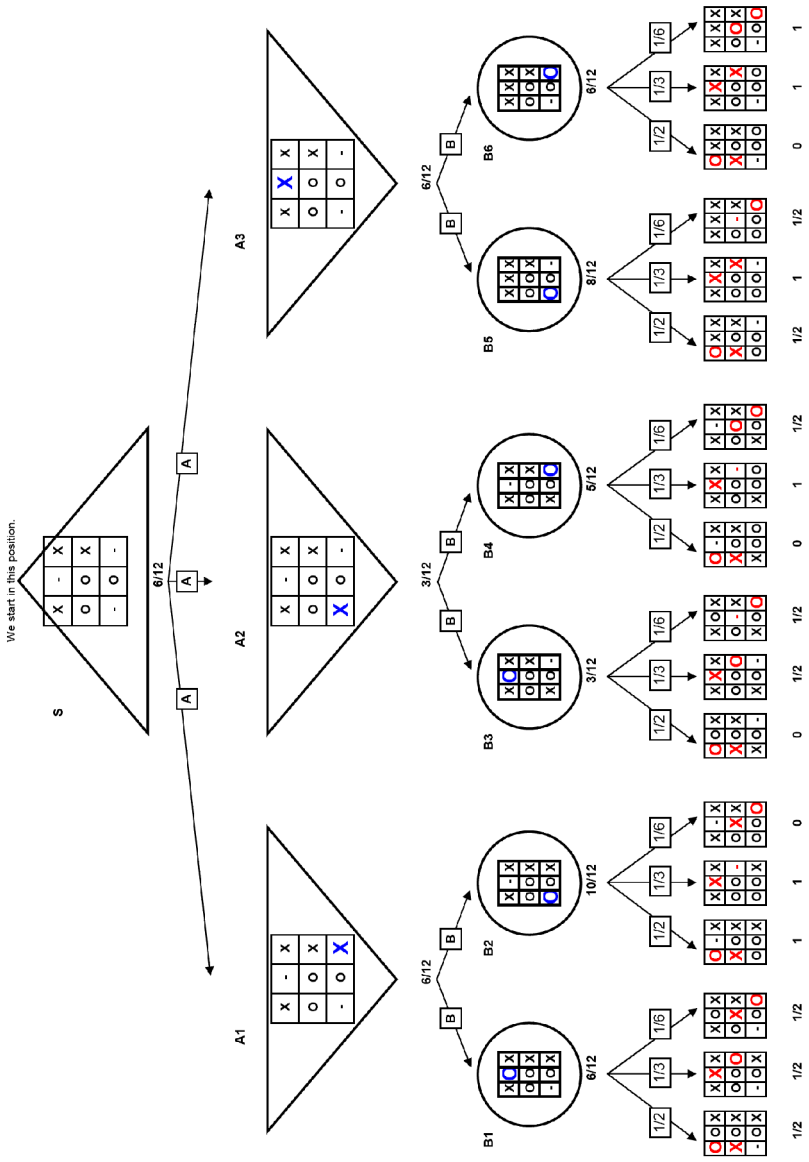

Figure 4: Evaluation of a part of the game tree of Tic-Tac-Toe with chance nodes

Note here that in non-random games players can choose a strategy so that the value of the game is always guaranteed. In games with chance nodes this is not true, only the expected value is optimised. In Section 4 we will show an interesting consequence of this fact.

If the nodes of the same type form layers in the tree, and these layers have a fixed order, then this algorithm can be written as three functions (in the same way as Algorithm 3 uses two functions), each of which computes the value of a different node type.

The fact that the optimal strategy depends not only on the order of the possible outcomes of the game, but on their values, leads to a phenomenon which is similar to the Saint-Petersburg paradox. The presence of a very improbable branch with a very high value may drastically change the situation. (We discuss this issue at the beginning of the next subsection.)

While [10] states that in the presence of random effects it is impossible and useless to develop strategies for the players, we believe that in some cases it is worth to compute the possibilities, and the game-evaluation can be accelerated by pruning techniques (when the set of the possible game values is bounded), as well.

In the following example we show that a game without a draw can be fair and correct if random events can happen.

An example for this phenomenon is the following game. The base (non-random) game is the well-known NIM [2]. (There are 3 groups of beans, two players take turns in removing some beans from an arbitrary group of beans. The player who removes the last bean wins the game.) The winner gets a point, the looser gets nothing. Let there be a random event with two outcomes, after every step of each player. The first outcome, w_1, where $P(w_1) = \frac{1}{2}$, implies that a bean is moved from the first group to the second group (provided there is a bean in the first group, otherwise nothing happens). At the second outcome, w_2, where $P(w_2) = \frac{1}{2}$, a bean is moved from the second group to the third group, if this is possible. If the starting position is $(0, 2, 1)$ then it can be easily seen that player A has an optimal strategy with the expected value of $\frac{1}{2}$. So, the game is fair. It cannot be a draw, but both players have the same chance to win.

It can also be shown that the value of this game is $v = \max\{P(w_1), P(w_2)\}$. Figure 5 presents the possible outcomes and the evaluation of the game for the probabilities $P(w_1) = \frac{1}{4}, P(w_2) = \frac{3}{4}$.

Figure 5 is a good example of a layered game tree, where on every branch the three types of nodes (A, C, and B) follow each other in a fixed order. When modifying Algorithm 5 to involve three separate functions one should care about the layers of the tree, and the functions should call each other according to this order. In each case the appropriate evaluating function, corresponding to the type of the successor nodes, must be applied.

3.2 Extensions of pruning techniques

Pruning techniques can be used to speed up the evaluation of games of chance, as well. In this section several variants of pruning are shown, most of them involve

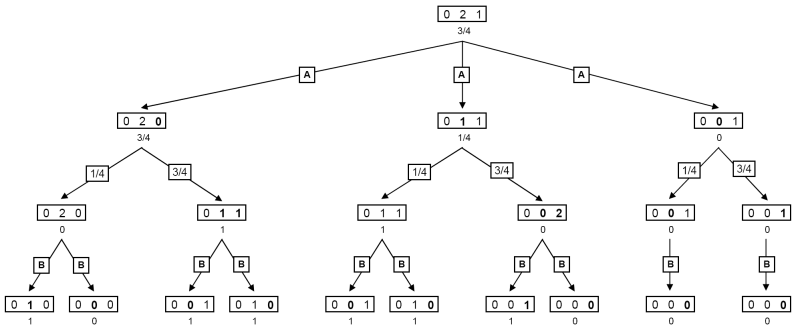

Figure 5: NIM with random events

chance nodes.

One could think that the branches of chance nodes, which have a very small probability can be ignored. However, doing this is a very rough estimation, because ignoring a branch with a small probability that has a very high or low value may drastically change the expected value. For instance, if there is a game with small leaf values (e.g. 0, 1, 2) in general, but also involving a leaf with a chance 10^{-8} that has a value 10^{12}, then this branch of the tree will be very significant. Observe here that changing the value of this leaf to 10^5, which is also much higher than the other values, causes this branch to lose its importance.

We think that the expected value itself may not be the correct quantity to evaluate the game, but it is still probably the most important value. Discussing this issue in detail is beyond the scope this paper. We only mention that situations similar to the Saint-Petersburg paradox can arise, in that very improbable effects may have a very big influence on the decision, if the latter is based (only) on the expected value. Note that in Section 4 we present some important values that can be used to improve the decision process.

The alpha (beta) pruning can be applied in a game of chance, if an A-layer is followed by a B-layer (a B layer followed by an A layer), exactly in the same way as for non-random games. This is because the layers below the immediate successors do not have any effect on the evaluation. So, in these cases the appropriate function of Algorithm 4 can be applied, but since the game tree has three kinds of nodes, rather than calling the other function (as in Algorithm 4), some parts of the expectiminimax algorithm should be used. (This means that at least line 8 of one of the functions must be changed, as required by the order of layers in the tree.)

We now present a new pruning technique, the so-called gamma pruning. It is based on the fact that knowing an estimated maximal or minimal possible outcome, one can easily decide whether a given branch can contribute to the solution, and so needs further exploration, or not.

The gamma pruning technique has two variants. The *gamma-B* (*gamma-A*) pruning can be applied if a B-layer (A-layer) of the tree is followed by a C-layer.

To introduce these techniques let us analyse two iterations of the expectimin-

imax algorithm together. So let us start from Algorithm 4 by replacing line 8 of both functions by the following:

```
8        let v = max{v,average(N_i, α, β)}
```

in alphamax and

```
8        let v = min{v,average(N_i, α, β)}
```

in betamin (where the function `average` computes the expected value at the node N_i as the weighted average of the values of its successors).

Now we discuss how one computes the average with pruning. If it is not necessary we do not compute the exact average, a lower or an upper estimation may still be of help in the decision making.

Let v_{max} and v_{min} be the maximal and minimal payoff of the game, respectively. These values will be used in the estimations.

Algorithm 6. (GAMMA PRUNING)

```
1  function gamma-A(N)
2   begin
3   let v = −∞
4   For every successor N_i of N do **** they are C nodes
5     let s = 0 and p = 0
6     For every successor N_{i,j} of N_i do
7       let the value of this node be v_{i,j}
8       let s = s + v_{i,j}P(N_{i,j}) and p = p + P(N_{i,j})
9       let α = s + (1 − p)v_max
10      if α ≤ v then exit from the loop
11    let v = max{α, v}
12  return v
13  end gamma-A

1  function gamma-B(N)
2   begin
3   let v = +∞
4   For every successor N_i of N do **** they are C nodes
5     let s = 0 and p = 0
6     For every successor N_{i,j} of N_i do
7       let the value of this node be v_{i,j}
8       let s = s + v_{i,j}P(N_{i,j}) and p = p + P(N_{i,j})
9       let β = s + (1 − p)v_min
10      if β ≥ v then exit from the loop
11    let v = min{β, v}
12  return v
13  end gamma-B
```

The idea of the algorithm is the following. The unknown (not calculated yet) values of the successors of a chance node are estimated as the best possible values for the player. If the expected value of this choice is less favourable (using this ideal estimation) than the known best option, then the evaluation of this option should not be continued.

In the special case of $v_{min} = 0$, which holds in our examples, the gamma-B function is evaluated in a faster way, because in this case s has the same value as β.

When computing the values $v_{i,j}$, pruning techniques can also be applied, depending on the order of layers of the game tree. If an A-layer and a B-layer follow each other, then the original alpha and beta pruning can be applied (depending on the order of layers). If an A-layer or a B-layer is followed by a C-layer, then the new technique, the gamma-pruning can be applied. For a given game, following the order of the layers, an appropriate recursive procedure can perform the evaluation. In this case, in line 7 of the gamma-pruning algorithm, the evaluation of the next layer can also be done recursively. For instance, if the order of the layers is A, B, C, then an alpha pruning calls a gamma-B pruning, etc.

Theorem 6. *The expected value of a game with chance nodes can be computed by using the functions of gamma-pruning algorithm. The expected value will be the same as the result of the expectiminimax algorithm.*

As an application of the gamma-B pruning, let us consider Figure 6, where a part of the game tree of the Tic-Tac-Toe game described previously (c.f. Figure 4) is shown. The figure also presents the values of the nodes. Observe that the part framed by a solid line can be pruned by gamma-B. The part framed by a dashed line can be eliminated by alpha pruning.

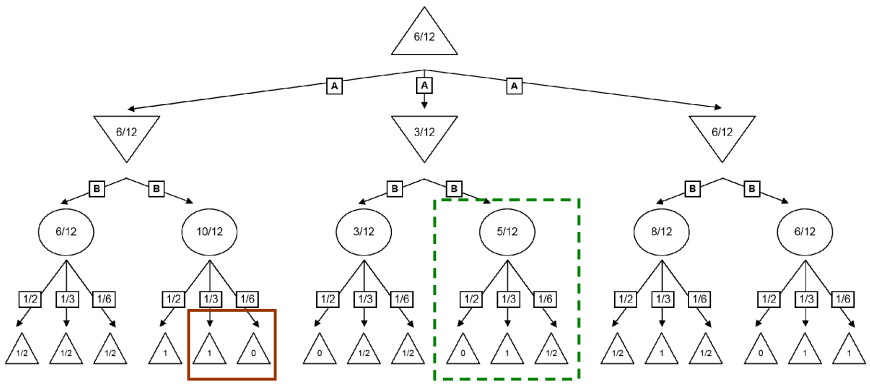

Figure 6: Accelerated evaluation by pruning: gamma-B pruning (solid box) and alpha pruning (dashed box)

Note that in the random NIM game both kinds of gamma pruning methods are applicable.

Of course, the speed-up due to these new pruning techniques also depends on the order of the branches. In case of gamma pruning, we believe that best results can be achieved when the successor nodes are tried in the order of decreasing probabilities (as it is shown in Figure 6). Usually the branches with higher probability have stronger effects on the computed values than the branches with lower probabilities.

4 A sequence of games as a two player game

In this section a new phenomenon is discussed. Let us consider a game as an iteration of a simple game as follows.

Let a simple base game be given (one can choose any of the games that can be described using the methods discussed in the previous sections). Now the players play several rounds of the base game, one after the other. The number of the base games can be fixed (say, for instance, n), or may depend on some events. The results of the games are accumulated. The winner of the iteration game will be the player who has collected more points than the other player at the end of the last game.

Let us consider a variant of the Tic-Tac-Toe game with random effects as shown in Figure 7. Here, the game starts from the state presented at the root of the tree, instead of the empty board, and so it involves only a single decision for each player. The expected value of this game is $\frac{1}{2}$, so the game seems to be fair.

Let the iteration game be a sequence of 4 base games of Figure 7. We show that player A has a strategy to win this game with a much higher probability than player B.

Let us first discuss the base game. Player A starts the game. She/he has three possibilities, but we can ignore branch A2, as this is bad for player A. Choice A1 (A3) involves placing an 'X' in the bottom-left corner (top-right corner) of the board, respectively. If player A chooses branch A3, she/he has a chance of $\frac{1}{2}$ to win, and the same chance to loose. If she/he chooses this strategy in every base game the outcomes of the iteration game can be the following (W stands for 'A wins', L for 'A looses', and D for a draw):

A wins: WWWW, LWWW, WLWW, WWLW, WWWL $P_A = \frac{5}{16}$
B wins: LLLL, WLLL, LWLL, LLWL, LLLW $P_B = \frac{5}{16}$
draw: WWLL, WLWL, WLLW, LLWW, LWLW, LWWL $P_{draw} = \frac{6}{16}$

If A chooses step A1 in each game, the draw can be guaranteed for both players.

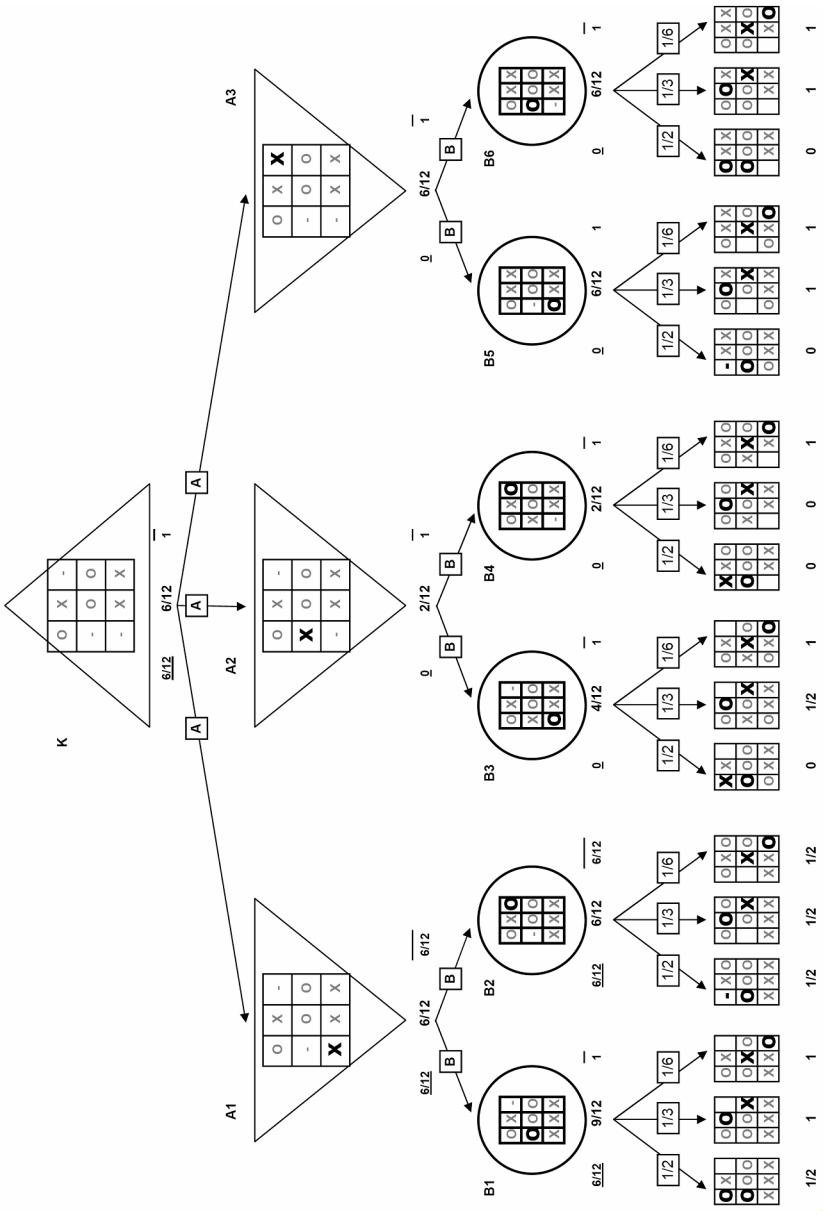

Figure 7: Evaluation with minimal guaranteed value in Tic-Tac-Toe with chance nodes

Let the strategy of player A in the iteration game be the following. In the first game she/he chooses the third possibility. In the subsequents games A plays as follows. If she/he has more points than player B has, then she/he chooses the step A1 to guarantee draw in every future game in the sequence. In this way A keeps her/his advantage. If A has not got more points than B, she/he chooses step A3 again in the next game.

It is easy to check that the possible outcomes of the sequence are the following:

A wins: WDDD, LWWD,	$P_A = \frac{10}{16}$
B wins: LLLL, LWLL, LLWL, LLLW	$P_B = \frac{4}{16}$
draw: LLWW, LWLW,	$P_{draw} = \frac{2}{16}$

This means that player A has a "probabilistic" winning strategy on this fair-like game. The techniques described in the previous sections do not show this phenomenon. This suggests that in the presence of iterative games we need something more than the expected value.

In our example we may introduce the concept of minimal guaranteed outcome. This is the value that the chosen strategy guarantees for A. It is impossible to have a smaller value for her/him if she/he plays well. (Thus it is the value that can be obtained in the case, when player A plays well, but the random events play against her/him.) At each node, in addition to the expected value, the minimal guaranteed value can also be computed for player A, and used in the decision process.

The computation of the minimal guaranteed outcome value can be done by a simple modification of the EXPECTIMINIMAX algorithm, by choosing the minimum at C nodes:

Algorithm 7. (MINIGMINIMAX)

```
1  function MGMM(N) is
2  begin
3    if N is a leaf then
4      return the value of this leaf
5    else
6      Let N_1, N_2, ..., N_m be the successors of N
7      if N is a A node then
8        return max{MGMM(N_1), ...,MGMM(N_m)}
9      if N is a B node then
10       return min{MGMM(N_1), ...,MGMM(N_m)}
11     if N is a C node then
12       return min{MGMM(N_1), ...,MGMM(N_m)}
13 end MGMM
```

It is easy to see that the algorithm is correct:

Theorem 7. *Algorithm 7 results in the (guaranteed) minimal possible payoff of player A if she/he plays well.*

This evaluation usually offers a different value than the expectiminimax algorithm, and can imply a different decision for player A.

Note that the evaluation for player B can be done in a similar way, computing the worst possibility for B at chance nodes. This requires replacing the function minimum by maximum in line 12. So the algorithm maxigminimax for player B is the same as Algorithm 7, except for line 12 being the following:

```
12        return max{MGMM(N_1), ...,MGMM(N_m)}
```

From A's point of view we can call this value the maximal possible value by choice (player B plays well, and the random events help A). Usually the evaluation for player B gives a possible game which differs from the game that Algorithm 7 suggests by evaluating the game for A.

In Figure 7 we have shown the minimal guaranteed values (underlined) and the maximal possible values by choice (overlined) on the left and right hand side of the expected values of the nodes, respectively.

This new type of information gives the possibility for player A to choose between the guaranteed draw (first option, A1 in the figure) and the random outcome (third option, A3). At step A1, the minimal guaranteed value is the same as the expected value at that node. This shows that the draw can be guaranteed. Therefore, using this additional information, A has a probabilistic winning strategy for the sequence of games, as detailed above.

Note that there is no problem regarding the fair outcome of the games in the sequence. If A employs the above winning strategy, and wins the sequence (which has a probability $P_{A,1} = \frac{5}{8}$), she/he wins exactly by 1 point (which is the minimal possible difference). There is a chance for draw as well ($P_{draw} = \frac{1}{8}$). At same time when B wins, she/he wins by 2 points ($P_{B,2} = \frac{3}{16}$) or by 4 points ($P_{B,4} = \frac{1}{16}$). In this way the expected value of the points achieved in the sequence of games shows a fair equilibrium between the two players. However, normally, the person who won matters and not the difference of scores.

Note that the algorithms for calculating the minimal guaranteed value and the minimal possible value by choice use only the maximum and minimum functions. Therefore alpha and beta pruning can be applied in a straightforward way in these algorithms. C nodes can be viewed as B nodes (in the evaluation for A) or as A nodes (in the evaluation for B). Using Proposition 4 these trees can be transformed to normal game trees, and on these new trees the evaluation can follow Algorithm 4.

5 Conclusions

Combining the theory of decision trees and two player strategic games, we have presented and analysed the theory of games with chance nodes. In these games, the expected value plays the role of the utility value of the non-random games.

In the presence of random events, a game without a draw outcome can still be fair, i.e. guarantee that the chance of winning for both players is the same (assuming

they play well).

The tree of games with chance nodes is related to both decision trees and non-random game trees. To handle the chance nodes appearing in these game trees, new pruning techniques have been developed. By using *gamma-A* and *gamma-B pruning* the exact value (expected value) of the game is computed more efficiently than without these. In several cases the expected value is only an average value, and there is no game which results in that outcome.

Games of chance are more complicated than non-random games, therefore further analysis is needed. As we have seen in Section 4, the expected value of a game with chance nodes does not provide enough information if, for instance, the game is played iteratively. Although only the whole distribution (containing the probabilities of all possible values) has all the information, we believe that some significant indicators can effectively help in finding an optimal strategy.

As important and useful examples of such indicators, we have introduced the *minimal guaranteed value* and the *maximal possible value by choice*, and have shown how can these help in making better decisions. Of course, the expected value is still the most important indicator, but other ones can help in finding the correct decision. There are several open questions in this area. Future research work is needed to develop appropriate indicators, together with efficient computation algorithms and usage scenarios.

Acknowledgements

The helpful comments and advice of the reviewers are gratefully acknowledged. This work is partly supported by the grant of the Hungarian National Foundation for Scientific Research (OTKA T049409), by a grant of Ministry of Education, Hungary and by the Öveges programme of the Agency for Research Fund Management and Research Exploitation (KPI) and National Office for Research and Technology (NKTH Nemzeti Kutatási és Technológiai Hivatal
A projekt a Nemzeti Kutatási és Technológiai Hivatal támogatásával valósult meg.).

References

[1] Fekete, I., Gregorics, T. and Nagy, S. Introduction to Artificial Intelligence: Heuristic Graph Search, Two Player Games, Automatic Theorem Proving (in Hungarian) LSI Oktatóközpont, Budapest, 1990.

[2] Futó, I., editor *Artificial Intelligence* (in Hungarian). Aula Kiadó, Budapest, 1999.

[3] Knuth, D. E. and Moore, R. W. An analysis of Alpha-Beta pruning. *Artificial Intelligence* 6:293–326, 1975.

[4] Lakatos, G. and Nagy, B. Games with few players. In Csőke, L., Olajos P., Szigetváry P. and Tómács T., editors *6th International Conference on Applied Informatics*, Volume II, pages 187–196, Eger, 2004.

[5] Luce, R. D. and Raiffa, H. *Games and Decisions: Introduction and Critical Survey*. Dover Publications, New York, 1989.

[6] von Neumann, J. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen* 10:295–320, 1928.

[7] von Neumann, J. and Morgenstern, O. *The Theory of Games and Economic Behavior*. Princeton University Press, 2nd edition, 1947.

[8] Resnik, M. D. *Choices: An Introduction to Decision Theory*. University of Minnesota Press, 1987.

[9] Rich, E. and Knight, K. *Artificial Intelligence*. McGraw-Hill, New York, 1991.

[10] Russell, S. and Norvig, P. *Artificial Intelligence, a Modern Approach*. Prentice-Hall, 1995. (2nd edition, 2003.)