$\mathcal{R}$minimax: An Optimally Randomized MINIMAX Algorithm

Silvia García Díez, Jérôme Laforge, and Marco Saerens

***Abstract*—This paper proposes a simple extension of the celebrated MINIMAX algorithm used in zero-sum two-player games, called $\mathcal{R}$minimax. The $\mathcal{R}$minimax algorithm allows controlling the strength of an artificial rival by randomizing its strategy in an optimal way. In particular, the randomized shortest-path framework is applied for biasing the artificial intelligence (AI) adversary toward worse or better solutions, therefore controlling its strength. In other words, our model aims at introducing/implementing bounded rationality to the MINIMAX algorithm. This framework takes into account all possible strategies by computing an optimal tradeoff between exploration (quantified by the entropy spread in the tree) and exploitation (quantified by the expected cost to an end game) of the game tree. As opposed to other tree-exploration techniques, this new algorithm considers complete paths of a tree (strategies) where a given entropy is spread. The optimal randomized strategy is efficiently computed by means of a simple recurrence relation while keeping the same complexity as the original MINIMAX. As a result, the $\mathcal{R}$minimax implements a nondeterministic strength-adapted AI opponent for board games in a principled way, thus avoiding the assumption of complete rationality. Simulations on two common games show that $\mathcal{R}$minimax behaves as expected.**

***Index Terms*—MINIMAX, randomized shortest paths (RSPs), two-player zero-sum perfect-information games.**

I. General Introduction

ARTIFICIAL INTELLIGENCE (AI) techniques [18], [23], [31] are widely used in realistic-behavior video games [20], [36]. These methods aim at finding paths for motion planning, collaborating between computer entities, learning from past experience, proposing game strategies, etc. The main focus of this paper is on finding strategies for two-player perfect-information zero-sum games [21], [24], [27], such as chess and draughts. These games can be seen as a succession of plays that alternate from one player to another and where the profit is maximized for the current player and therefore minimized for the opponent. They are often solved due to the well-known MINIMAX algorithm [18], [20], [23], [31], [36], which is straightforwardly or indirectly used in most board games (see Section I-A for a more detailed introduction to MINIMAX).

Manuscript received September 20, 2010; revised July 31, 2011; accepted June 4, 2012. Date of publication August 6, 2012; date of current version January 11, 2013. This work was supported in part by the *Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture* under Grant F3/5/5-MCF/ROI/BC-21716, by the *Région Wallonne,* and by the Belgian *Politique Scientifique Fédérale*. This paper was recommended by Associate Editor J. Shamma.

The authors are with the Department of Information Systems, Université catholique de Louvain, 1348 Louvain-la-Neuve, Belgium (e-mail: silvia.garciadiez@uclouvain.be; jrlaforge@gmail.com; marco.saerens@uclouvain.be).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2012.2207951

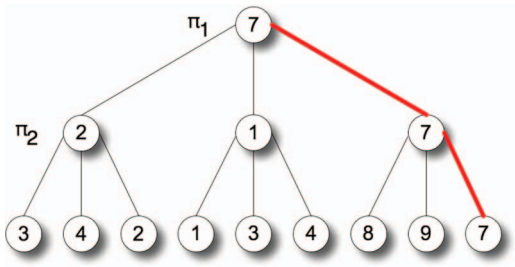

Fig. 1. Example of the MINIMAX algorithm for a game tree with a depth of 3 and players π_1 (which plays max) and π_2 (which plays min). (Circles) Utility values. (Red) Optimal strategy for π_1.

From its inception, MINIMAX assumes perfect rationality for both players, and therefore, it is completely deterministic, i.e., the player will adopt the same deterministic strategy when encountering the same situation. Since the behavior of the AI player is completely predictable, the game might become annoying for the rival. Such problem is tackled in this paper by proposing a simple way to randomize the strategy while still remaining optimal. The main idea is to control the spread randomness in the game tree, quantified through its Shannon entropy, and to select the optimal minimum expected-cost strategy for this entropy. In this way, good (low-cost) randomized strategies are favored, whereas bad ones (high cost) are discarded. Adjusting the tradeoff between the exploitation and the exploration of the game tree and, therefore, the strength of the player is achieved by varying the entropy. In other words, our model aims at introducing/implementing bounded rationality (see [29] and [43] for a survey) to the MINIMAX algorithm. The proposed method, called $\mathcal{R}$minimax, is the application of the randomized shortest-path (RSP) framework [32] to game trees.

In summary, the $\mathcal{R}$minimax contributions are to model nonrational players, to control the strength of a player, and to avoid the total predictability of a player.

A. MINIMAX Algorithm

The MINIMAX algorithm [18], [20], [23], [31], [36] computes the optimal strategy for two-player zero-sum games, provided that the opponent is fully rational, i.e., it will also play according to its optimal strategy. In order to illustrate the MINIMAX principle, let us assume a game tree such as the one shown in Fig. 1, i.e., a MAX player π_1 and a MIN player π_2

who want to maximize and minimize, respectively, their utility value (cost or reward). The utility value is initially defined for the leaf states of the game tree (winning states) as a low value for winning states of π_2 and higher values for winning states of π_1. Intermediate utility values indicate the advantage of one of the players over its opponent. Once the leaf utility values are defined, the MINIMAX algorithm recursively operates on the game tree, iterating between the π_1 player, which takes the maximum of its children's utility values (on odd-depth states), and the π_2 player, which takes the minimum of its children's utility values (on even-depth states). As a result of this, the utility values are backed up to the current state (root node), and it reflects the utility value of the optimal strategy's next move (colored in bold red in Fig. 1 and corresponds to the child with a utility value of 7).

B. Related Work

As it is the nature of MINIMAX to search the whole game tree, much attention has been paid to reducing the search space. The simplest technique consists of bounding the depth of the tree with an *n-ply lookahead* strategy [18], where n is the number of explored levels of the tree. Also, very common are the *alpha–beta* (AB) pruning techniques [22]. The AB algorithm prunes irrelevant subtrees, which will never be part of the MINIMAX strategy by using a window of two plies. An AB multiplayer version is proposed in [39]. The *Negascout* algorithm [28] reduces even further this window size, which allows performing a faster pruning than the AB. Nonetheless, the tree may be massively pruned, leading to the elimination of good strategies. Similarly, the *Memory-enhanced Test Driver* (MTD (f)) [26] limits the AB window size to zero. Although this pruning is faster, an initial "guess," i.e., f, of the MINIMAX position is required. This method is also based on *transposition tables*, which are used in games with a vast search space where recurrent states appear. In this case, it is more efficient to "remember" the decision taken the first time the state was observed than redoing the entire search. Despite the MTD outperforming the Negascout in the number of searched nodes, it suffers from stability issues, it depends on the transposition tables, and it is also very sensitive to the initial guess. Eventually, a pruning technique that computes the expected value of the continued search is proposed in [30]. It has been shown [30] that this method also suffers from numerical instabilities.

Opening books [5] are another improvement technique applied to huge search space games. Efficient "opening" and "ending" game strategies that are often used by expert players are stored in these books. It is proven that initial strategies are critical for reducing the search space, as well as guiding the game toward the winning states. However, even when the search space is reduced, the interaction time remains a key feature that must be also taken into consideration. *Iterative deepening* techniques may be useful in cases where the calculation time is unknown *a priori*. In this way, a strategy is available to interact at any time, but its quality will depend on the depth of the last explored tree. Often, this technique is used to choose a few good strategies obtained with a small depth and validated by extending them further. *Quiescence pruning* [12] avoids searching the branches of the tree whose heuristic function values are stable and, therefore, with no leadership changes. MINIMAX has been also extended for chance games such as Backgammon. A version of the game tree with a new type of "chance" nodes representing the probabilistic states of the game (i.e., where a dice is thrown) is proposed in [19]. Eventually, a stochastic approach that computes the probabilities of correctly scoring the following moves, via a heuristic function, is given in [2].

A different approach involving randomized strategies can be found in the Game Theory literature [21], [24], [27]. *Mixed strategies* are an alternative to *pure strategies* in games where several decision makers interact in order to maximize their payoffs. Players must choose among a set of possible actions where each action has an associated cost or reward. In contrast to pure strategies where a player takes a deterministic action, i.e., $p_{\text{action}} = \{0, 1\}$, mixed strategies allow players taking an action with a given probability $p_{\text{action}} \in [0, 1]$. These probabilities are usually computed via the Nash equilibrium of the game, which corresponds to the best strategy (expected payoff) that player A can adopt while taking into consideration player B's decision. Although the exact play remains unknown for the opponent, the probabilities of his actions are known in advance, letting the game be pseudorandom. An extension of these strategies for two-player turn-based random games are *stochastic games* [33]. This technique tries to maximize the expected payoff for a player by choosing an optimal strategy, and its computation has been the subject of several studies [25], [37].

Nevertheless, little attention has been paid to modeling the strength of an adversary in two-player zero-sum games in the AI community. A basic approach consists in using the n-ply lookahead algorithm [18] in order to vary the capacity of a rival. Unfortunately, n may be tough to tune as it depends on the game and the branching factor, i.e., for low values of n (i.e., in chess a small $n < 6$), the AI-based opponent can be easily outperformed by the user (who normally plans six or eight plies ahead), whereas for very high values $(n > 8)$, it may become extremely difficult to win. Other frequently used techniques are *ϵ-greedy* [41], where the optimal branch is taken with probability $1 - \epsilon$ and a random branch with uniformly distributed probability $\epsilon/$number_branches; *Boltzmann exploration* [41], where the probability of taking a branch follows a Boltzmann distribution with an inverse temperature, which depends on the state-specific exploration coefficient; and techniques based on the addition of noise to the evaluation function. However, such techniques focus on the current state, limiting their strategy to local decisions and failing to find an optimal global strategy over the whole game tree for a given entropy [1].

This idea of *bounded rationality* (see, e.g., [11], [29], and [43]) has been already applied in a large number of fields from Psychology [34] to AI [17]. In this last category, we find the work from [43], where the author presents the link between game theory and statistical physics. In this context, he shows how Shannon's information theory provides a framework for bounded rational game theory. In particular, when we know both players' mixed strategy and their expected cost, the probability distribution of possible actions should follow a

Boltzmann distribution. However, the author does not provide a precise algorithm implementation for his ideas. This paper can be considered as a concrete instantiation of these ideas for two-player zero-sum games. Nested Monte Carlo search [7] provides another bounded rational algorithm over a game tree, which combines nested calls with randomness and memorization of the best sequence of moves.

The proposed approach of this paper does not only focus on modeling the strength of an adversary but also on ameliorating the AI of the MINIMAX by adding probabilistic, more human-like, while still optimal strategies.

II. Randomized MINIMAX

MINIMAX has been widely applied for emulating an opponent in two-player zero-sum games. While being very useful in most situations, it however suffers from some drawbacks. First, the assumption of perfect rationality for both players is unrealistic, as human players often incur into error. Second, it does not address the issue of vast search space for certain games, and therefore, the use of a heuristic function is often necessary, which is usually hard to define. Third, the behavior of the player is deterministic and thus predictable. Fourth, in its basic form, controlling the strength of the player is not feasible. The developed approach of this section overcomes some of these shortcomings.

It can be observed from the game tree that a deterministic strategy leads to a path from the root node (initial state) to a leaf node (end game, i.e., winning/losing state). MINIMAX chooses the path that maximizes the gain of the current player while minimizing the gain of the adversary. A variant of MINIMAX, which will randomize the choice among all possible paths of the game tree, is introduced. The advantage of this technique is threefold: first, deterministic strategies are avoided, therefore eliminating the predictability of the game; second, the perfect rationality of the player is not assumed; and third, control over the strength of the player is allowed. Although the issue of the search space is not tackled in this paper, as for MINIMAX, any of the existing techniques could be applied in order to reduce the size of the explored tree.

A. RSP Framework

Since the $\mathcal{R}$minimax algorithm introduced in this paper heavily relies on the RSP model, in order to make this paper as self-contained as possible, a short description of the RSP framework is provided in this subsection. The interested reader is invited to consult the original paper [10], [32] for details.

The original RSP framework was inspired by a stochastic transportation model [3] (see [32] or [10] for an alternative derivation in the special case of acyclic graphs). Let G be a directed graph containing a source node with index 1 and a destination or goal node with index n. Moreover, the goal node is absorbing; once node n is reached, the path stops, i.e., there is no outgoing arc from n. A nonnegative local cost $c_{kk'} \geq 0$ is associated to each of the arcs. If there are many destination nodes, the following trick can be used: a dummy node n is created, and a zero-cost arc between each destination node and the dummy node n is added. The set of all paths (including cycles) that go from 1 to n is denoted as $\mathcal{P}_{1n}$. Each path $\wp \in \mathcal{P}_{1n}$ is composed by a sequence of arcs $k \to k'$ that ties the source to the destination node. Moreover, let the total cost $C(\wp)$ of path $\wp$ be the sum of these local costs along $\wp$. The path randomization will be driven by global performance; a probability will be assigned to each path, favoring nearly optimal paths having a low cost $C(\wp)$. Therefore, optimal or slightly nearly optimal paths will be assigned a high probability, whereas paths leading to a high cost will be penalized. The *Shannon entropy*, i.e.,

$$H_0 = -\sum_{\wp \in \mathcal{P}_{1n}} P(\wp) \ln P(\wp) \tag{1}$$

will be used for controlling this penalization of expensive paths via the assigned probability distribution. Parameter H_0 controls the degree of randomness or exploration in the graph and is related to the *inverse temperature* of the graph, i.e., $\theta > 0$. Formally, our problem can be stated as

$$\left| \begin{array}{ll} \underset{P(\wp)}{\text{minimize}} & \displaystyle\sum_{\wp \in \mathcal{P}_{1n}} P(\wp) C(\wp) \\ \text{subject to} & \displaystyle -\sum_{\wp \in \mathcal{P}_{1n}} P(\wp) \ln P(\wp) = H_0 \end{array} \right. \tag{2}$$

where $P(\wp)$ is the probability of the following path $\wp$. It can be shown from this optimization problem [32] that the probability of each path is

$$\boxed{P(\wp) = \frac{\exp[-\theta C(\wp)]}{\mathcal{Z}}} \tag{3}$$

where $\mathcal{Z} = \sum_{\wp \in \mathcal{P}_{1n}} \exp[-\theta C(\wp)]$ is the partition function and $P(\wp)$ therefore follows a *Boltzmann distribution*, as imposed by the problem constraint. This equation defines the optimal global policy for reaching the goal node n from the initial node 1. As shown in [43], the Boltzmann distribution is obtained by solving the maxent Lagrangian when both players play a mixed strategy and we know their expected costs, although their approach focus on the possible strategies and not on the paths. In this way, the possible actions at a certain state are favored when the expectation of the cost is low. Furthermore, the link between the information theory, the game theory, and the statistical physics is presented in [43]. It must be noted that, when $\theta \to \infty$, the entropy is zero, and thus, the probability distribution is peaked on optimal minimum-cost paths. On the other hand, when $\theta \to 0$, an almost equal probability is assigned to all paths, leading to a blind randomized exploration of the graph. In this way, θ can be seen as the parameter that controls the entropy [15]. Indeed, θ is the inverse temperature that is inversely related to the entropy, allowing the tradeoff between the exploration and the exploitation of the graph. It should be noted that this model assumes the independence between the various paths, which is not always realistic.

Let us now show how to efficiently compute the local strategy or policy (represented by state transition probabilities) in terms of forward and backward variables. The expected number of

passages per node, i.e., n_k, and the expected number of passages per link $k \to k'$, i.e., $n_{kk'}$, can be defined as

$$n_k = \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k) \frac{\exp\left[-\theta C(\wp)\right]}{\mathcal{Z}} \tag{4}$$

$$n_{kk'} = \sum_{\wp \in \mathcal{P}_{1n}} \delta(\wp; k, k') \frac{\exp\left[-\theta C(\wp)\right]}{\mathcal{Z}} \tag{5}$$

where $\delta(\wp; k, k')$ ($\delta(\wp; k)$) is an indicator variable that counts how many times link $k \to k'$ (node k) belongs to path $\wp$. These equations simply sum the probabilities of passing through the node or the link. By rewriting (4) and (5) (see [32] for details) in terms of forward (z_k^f) and backward (z_k^b) variables, the following are obtained:

$$n_{kk'} = \frac{z_k^f \exp[-\theta c_{kk'}] z_{k'}^b}{z_{1n}} \tag{6}$$

$$n_k = \frac{n_{kk'}}{\sum\limits_{k' \in \mathrm{Succ}(k)} n_{kk'}} = \frac{z_k^f z_k^b}{z_{1n}} \tag{7}$$

where $\mathrm{Succ}(k)$ is the set of successors of state k, $z_{1n} = \mathcal{Z} = z_n^f$. The *forward variable* z_k^f and the *backward variable* z_k^b are defined as [10], [32] $z_k^f = \sum_{\wp \in \mathcal{P}_{1k}} \exp[-\theta C(\wp)]$ and $z_k^b = \sum_{\wp \in \mathcal{P}_{kn}} \exp[-\theta C(\wp)]$, where $\mathcal{P}_{1k}$ is the set of paths starting in node 1 and ending in node k, and $\mathcal{P}_{kn}$ is the set of paths starting in node k and ending in node n. They can be computed [10], [32] due to the recurrence equations

$$\begin{cases} z_n^b = 1 \\ z_k^b = \sum\limits_{k' \in \mathrm{Succ}(k)} \exp[-\theta c_{kk'}]\, z_{k'}^b, & \text{for } k \neq n \end{cases} \tag{8}$$

$$\begin{cases} z_1^f = 1 \\ z_{k'}^f = \sum\limits_{k \in \mathrm{Pred}(k')} \exp[-\theta c_{kk'}]\, z_k^f, & \text{for } k' \neq 1 \end{cases} \tag{9}$$

where $\mathrm{Succ}(k)$ and $\mathrm{Pred}(k)$ represent the set of successors and predecessors of state k, respectively. In the general case, (4) and (5) represent two systems of linear equations that have to be solved in the function of the forward and backward variables. However, if the graph is acyclic (as is the case in a game tree), these linear equations simply define recurrence relations allowing us to compute the forward/backward variables by simple back-substitution.

When the game tree is too large to solve the linear system completely, an n-ply lookahead technique can be used, limiting the tree to the first n levels of the game. The same equations can be then applied to the extracted subtree, limiting the computational complexity of the method at the price of an approximate solution.

The local state transition probabilities corresponding to the probability distribution of (3) can be computed from (6) and (7) as

$$\boxed{p_{kk'} = \frac{n_{kk'}}{n_k} = \frac{z_{k'}^b}{z_k^b} \exp[-\theta c_{kk'}]} \tag{10}$$

which only depend on the backward variables and the local costs. Denominator z_k^b is a normalization factor ensuring that the transition probabilities sum to one. These transition probabilities define the strategy, also called policy, of a random walker on the graph sampling the paths to the destination state according to (3). The random walker explores the graph with a fixed entropy while minimizing the expected cost to the destination state, assuming the independence of the paths. This RSP framework will be now applied to our game tree in order to provide an optimal strategy.

B. $\mathcal{R}$minimax Algorithm

The application of the RSP framework to the game tree will allow biasing the transition probabilities toward better or worse solutions as θ increases or decreases. In that case, graph G is a tree, and it is therefore acyclic. Equation (8) therefore defines the recurrence relation allowing us to compute the backward variables z_k^b from the destination node n to each intermediary node k.

Assume that π_1 is our AI player and π_2 is the opponent. We will randomize π_1's strategy while still assuming that π_2 plays rationally. The set of winning/losing states indicating the end of the game will be denoted by $\mathcal{N}$, and the set of paths is now $\mathcal{P}_{1\mathcal{N}}$. By applying the RSP framework to this situation, the backward variables [see (8)] are redefined in terms of the following recurrence relations:

$$\boxed{\begin{cases} z_n^b = 1, & \text{for } n \in \mathcal{N} \\ z_k^b = \begin{cases} \sum\limits_{k' \in \mathrm{Succ}(k)} \exp[-\theta c_{kk'}]\, z_{k'}^b, & \text{if } k \text{ is in } \pi_1\text{'s turn} \\ \min\limits_{k' \in \mathrm{Succ}(k)} \exp[-\theta c_{kk'}]\, z_{k'}^b, & \text{if } k \text{ is in } \pi_2\text{'s turn} \end{cases} \end{cases}} \tag{11}$$

where $k \notin \mathcal{N}$ is assumed. It can be observed that, when π_1 (the AI player) plays, it takes into account the costs of all successors of state k for randomizing its future strategy, while π_2 plays the best strategy by considering only one branch of the tree. Indeed, since the transition probabilities (the policy followed by player π_1) are proportional to $\exp[-\theta c_{kk'}]\, z_{k'}^b$ [see (10)], according to (11), π_2 chooses the action corresponding to the lowest (min) transition probability, i.e., the move that is *least favorable* to his opponent π_1, all the other moves being dismissed; the game tree is accordingly pruned. As π_1 and π_2 play in turn, the value of the backward variables is computed by alternating both equations. It must be also noticed that, in order to avoid overflow or underflow problems, the standard formula for the logarithm of a sum (see, e.g., [13]) could be applied when computing z_k^b.

Although it is not immediately obvious from (11), player π_1 *minimizes* the expected cost to the end game by following the optimal policy provided by (10) [this directly follows from the RSP framework; see (8)], while player π_2 tries to *maximize* it. Indeed, let us take $-(1/\theta)\log$ of each member of the recurrence relation for player π_2 in (11), i.e.,

$$-\frac{1}{\theta}\log\left(z_k^b\right) = -\frac{1}{\theta}\log\left(\min_{k' \in \mathrm{Succ}(k)} \exp[-\theta c_{kk'}] z_{k'}^b\right). \tag{12}$$

TABLE I
EXAMPLE OF TRANSITION PROBABILITIES p_{ij} (TRANSITION PROBABILITY FROM NODE i TO CHILD j) FOR A SIMPLE BINARY GAME TREE WITH A DEPTH OF 3, WHEN VARYING θ

	$\theta = 3$	$\theta = 0.5$	$\theta = 0.001$
p_{12}	0.998	0.728	0.5001
p_{13}	0.002	0.268	0.4999

By using $-\log(\min(x, y)) = \max(-\log(x), -\log(y))$ and defining $v_k = -(1/\theta)\log(z_k^b)$, we obtain the following for player π_2:

$$v_k = \begin{cases} 0, & \text{for } k \in \mathcal{N} \\ \max\limits_{k' \in \text{Succ}(k)} (c_{kk'} + v_{k'}), & \text{if } k \text{ occurs during } \pi_2\text{'s turn.} \end{cases} \tag{13}$$

Now, this is exactly the recurrence equation, akin to the Bellman equation (see, e.g., [4]), allowing us to compute the maximal-cost path to the end-game states. Therefore, player π_2 consistently tries to maximize the cost. If player π_1 would only consider the best move, as does player π_2, we recover the standard MINIMAX. The $\mathcal{R}$minimax algorithm is summarized in Algorithm 1.

Algorithm 1 $\mathcal{R}$minimax.
Require:
• G: The generated game tree obtained with the MINIMAX algorithm. The root of the game is $k \in \pi_1$.
• $\theta > 0$: The degree of randomization of the tree (∞ for a perfect rational player; $\simeq 0$ for an almost completely random player).
• $c_{kk'} \geq 0$: The cost of each arc of the tree.
1. Assign $z_n^b = 1$ for each $n \in \mathcal{N}$.
2. Recursively compute z_l^b according to (11).
3. Compute the corresponding $p_{kk'}$ value according to (10).
4. **return** $p_{kk'}$: the transition probabilities for the next play.

Note that, when θ takes a high value, near-optimal strategies are chosen by the AI player π_1, while for small values, he will model a weak rival with a poor strategy. As an example of the effect of the different θ on the transition probabilities, let us consider the following case: assume a trivial binary game tree with only three levels where the current node is the root node, and the aim is to reach a winning node, associated to a reward (see the simulation methodology in Section III for more details), while playing with strength θ. The cost of each play is $+1$. Once all quantities have been computed, the results shown in Table I are obtained. It must be noticed that, when $\theta \to \infty$, the optimal strategy given by the MINIMAX algorithm is recovered. As θ decreases, the transition probabilities are less biased toward the optimal solution. In the case of $\theta \to 0$, the assigned costs become irrelevant, and therefore, the strategy is utterly random (the transition probabilities p_{12} and p_{13} are almost uniformly distributed).

Similar approaches (although not optimal) can be found in reinforcement learning [6], [14], [35], [41]. Typical exploration methods are incorporated to reinforcement learning in multi-agent systems in [6]. One such method is the *Softmax* [40] technique (also called *Boltzmann exploration*) based on applying a Boltzmann distribution on each of the possible actions (branches of the tree) based on a utility function. As stated in [42], Boltzmann distributions provide a way to combine random exploration with exploitation, and the likelihood of picking an action is exponentially weighted by its utility. Boltzmann exploration is also used as a bandit strategy in order to avoid the lookahead pathology and shows competitive results [16]. Another bandit-based method that performs efficient "cuts" of suboptimal branches with high confidence is proposed in [9]. This technique also allows controlling the tradeoff between the exploration and the exploitation of the tree. Eventually, [8] proposes the application to the game of Go of a bandit technique, which biases the exploration of the tree in order to find the most suitable strategy to be explored according to previous information (such as the number of times a state has been explored or the number of times that it led to a victory). Yet, most of these techniques eventually find an optimal policy and stop exploring the graph, therefore loosing their stochastic behavior. On the other hand, we may find the reinforcement learning technique proposed in [14], which continually explores the graph. However, the convergence to the optimal policy can no longer be proven.

III. SIMULATION RESULTS

In order to illustrate the proposed method, systematic simulations on two-player zero-sum games have been performed. Two common well-known games such as *Tic-Tac-Toe* and *Connect-4* are tested.

Tic-Tac-Toe is a popular game that takes place on a horizontal 3×3 grid where two players position a token (circle or cross) alternatively. The aim of the game is to be the first to place three consecutive tokens in a row, column, or diagonal. The aim of the Connect-4 game is similar, although the grid is 4×4, and it is placed in a vertical position where the players drop their colored disks alternatively, letting the bottom rows be filled first.

Game trees for both games have been generated with both the MINIMAX and AB algorithms. Two AI opponents have been simulated, each with a different strength θ_i for testing the behavior of our method when confronting different heterogeneous players. The simulation methodology is given below.

1) The game tree for the current node k is computed with the MINIMAX algorithm.
   a) The tree can either be fully generated or limited to a five-ply lookahead (depth $= 5$).[1]
   b) The tree can be pruned with the AB algorithm.
2) A reward is assigned to each transition to a winning node. Cost are computed below.
   a) In the case of a full game tree, lower costs are assigned to winning nodes and higher ones to losing nodes. Tie

[1]It must be noted that, at this stage, any of the previously explained techniques for reducing the search space could be applied (transposition tables, pruning techniques, etc.). However, only the case of pruning is showed here, as our purpose is merely illustrative.

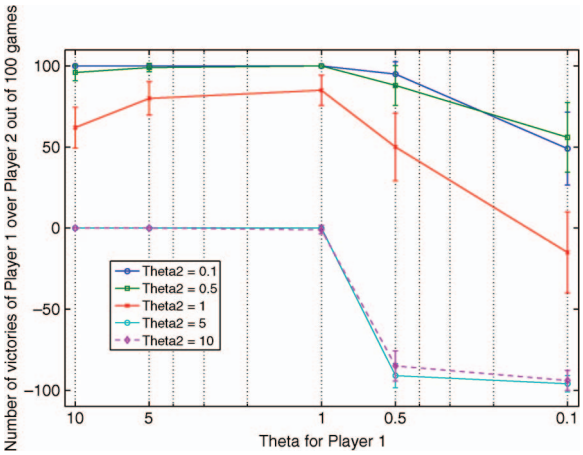

Fig. 2. Resulting curves for the $\mathcal{R}$minimax algorithm applied to the game Tic-Tac-Toe. The algorithm is applied to the full game tree generated by the MINIMAX algorithm. The horizontal axis represents the variation of θ_1 for player π_1, while the vertical axis shows the number of victories of π_1 over π_2, out of 100 games. Each curve corresponds to a value of θ_2 for player π_2 with its 95% confidence intervals.

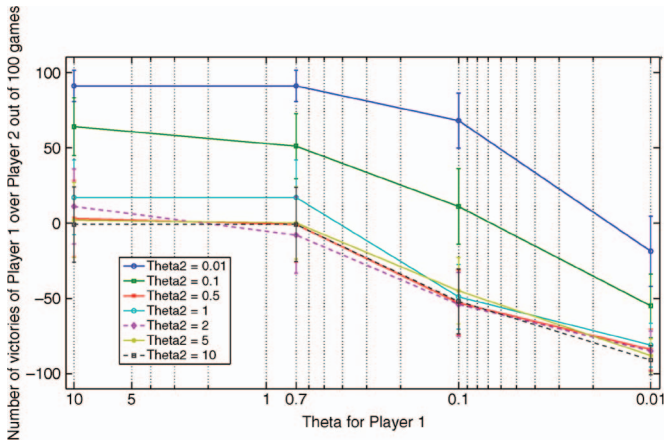

Fig. 3. Resulting curves for the $\mathcal{R}$minimax algorithm applied to the game Connect-4. The algorithm is applied to a game tree with a depth of 5 generated by the MINIMAX algorithm, combined with heuristics. The horizontal axis represents the variation of θ_1 for player π_1, while the vertical axis shows the number of victories of π_1 over π_2, out of 100 games. Each curve corresponds to a value of θ_2 for player π_2.

nodes are assigned a value between those of winning and losing nodes.

b) In the case of a five-ply lookahead, the heuristic described in Section III-B is used.

c) All other internal arcs are assigned a cost of $c_{kk'} = 1$ so that short-winning paths will be preferred over long-winning paths. It is the length of the path and the final transitions' costs that matter when choosing a certain strategy.

3) For both players, apply the $\mathcal{R}$minimax algorithm as described in Algorithm 1 with strength θ_i for player i.
4) Choose the next state k' among all successor of k with probability $p_{kk'}$.
5) If k' is a winning/losing end-game state, the result is increased/decreased by one unit according to the winner, and a new game is started. Otherwise, it is the next player's turn, and return to step 1.

This whole procedure is repeated 100 times (different runs) and returns result r, which takes its values in $r \in [-100, 100]$, which indicates the number of victories of both players. If $r > 0$, player π_1 has $|r|$ out of 100 victories over π_2. Otherwise, the winner will be π_2 with $|r|$ victories out of 100, and $r = 0$ represents result parity.

A. $\mathcal{R}$minimax With Full Game Tree

For getting a better insight about $\mathcal{R}$minimax's behavior, it is first applied to Tic-Tac-Toe on the full game tree generated by the MINIMAX algorithm. In order to visualize the performance of our method when two players of different strengths interact, 100 runs have been performed between two players of varying strength θ. According to our simulation methodology previously stated, the performance of both players is recorded when applying the $\mathcal{R}$minimax. Tested values of θ are $\theta_1 = \theta_2 = \{0.1, 0.5, 1, 5, 10\}$. The resulting curves are shown in Fig. 2.

As it can be observed, all curves have a similar shape but start at different levels. This can be translated into a high resemblance in the behavior of the AI players; when $\theta_1 \gg \theta_2$, player 1 wins, while for $\theta_1 \ll \theta_2$, it is player 2 who leads the game. Such behavior fulfills what we expected, as for $\theta \to \infty$, the entropy is 0, and thus, the player chooses an optimal strategy and vice versa. In the case of $\theta = \infty$, the game reduces to the MINIMAX strategy. The level at which a curve begins depends on the difference between both θ values.

On the other hand, the slope of the curves reflect the effect of the relative advantage of π_1 over π_2. Indeed, when π_1 moves first, it has an advantage over π_2. This can be observed in Fig. 2, where a lower slope is shown for low values of θ_2 for π_2.

B. $\mathcal{R}$minimax With Five-Ply Lookahead and Heuristics

Another frequent tools used in AI are heuristics and evaluation functions [31]. The performance of our method when using a partial game tree combined with the use of heuristics is studied in this section. In this experiment, the investigated game is Connect-4. As generating the full game tree would be computationally expensive, a five-ply lookahead method is implemented here and combined with the use of a heuristic function for scoring the final transitions. The applied heuristics is the one proposed in [38] and corresponds to the sum of two quantities, i.e., the number of winning lines that may still be done for each following move plus a fixed quantity that corresponds to the goodness of the empty positions that are left (some positions are more versatile than others). Tested values of θ are $\theta_1 = \{0.01, 0.1, 0.7, 10\}$ and $\theta_2 = \{0.01, 0.1, 0.5, 1, 2, 5, 10\}$. Results are shown in Fig. 3.

These resulting curves and the ones of the previous section are alike. Yet, as the game tree is limited to a certain depth and Connect-4 has a wider set of initial positions than Tic-Tac-Toe, the relative advantage of π_1 is not as clear as in the former case. Indeed, for observing the same effect, significantly lower values of θ are needed (than those of the Tic-Tac-Toe).

C. $\mathcal{R}$minimax With Five-Ply Lookahead and AB Pruning

For this simulation, a partial game tree with a depth of 5 has been generated for the game of Connect-4. This

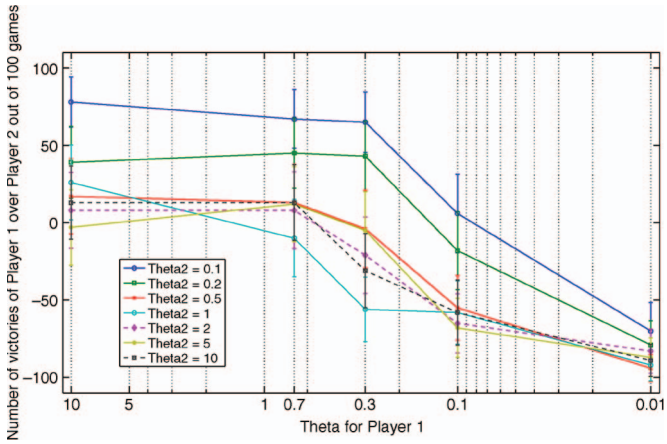

Fig. 4. Resulting curves for the $\mathcal{R}$minimax algorithm applied to the game Connect-4. The algorithm is applied to a game tree with a depth of 5 generated by the AB algorithm combined with heuristics. The horizontal axis represents the variation of θ_1 for player π_1, while the vertical axis shows the number of victories of π_1 over π_2, out of 100 games. Each curve corresponds to a value of θ_2 for player π_2.

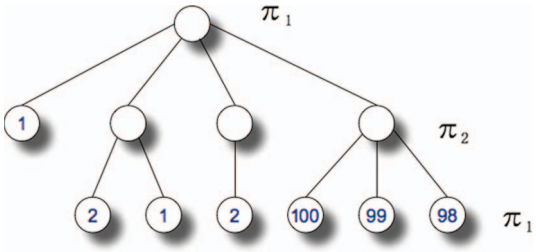

Fig. 5. Game tree with a depth of 3 composed of 11 nodes for comparison between the $\mathcal{R}$minimax and the ϵ-greedy. Leaf nodes contain the utility values. π_1 plays MIN and π_2 plays MAX.

time, a pruning algorithm reducing the search space is applied. The objective is to observe the behavior of the $\mathcal{R}$minimax algorithm combined with a technique that reduces not only the depth of the tree but also the search space. Tested values of θ are $\theta_1 = \{0.01, 0.1, 0.3, 0.7, 10\}$ and $\theta_2 = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$. Results are shown in Fig. 4.

These results are consistent with those of previous sections. However, in this case, the relative advantage of π_1 is even smaller. In contrast to the former case, a smaller θ_2 allows π_2 winning as θ_1 decreases. This is due to the pruning of the AB, as it restrains the set of explored branches.

IV. Comparison With ϵ-Greedy

Although the $\mathcal{R}$minimax is proved to be optimal for a fixed entropy, this section illustrates its optimality when compared with another popular bounded-rational algorithm, i.e., the ϵ-greedy [41].

In order to illustrate the behavior of both algorithms, two game trees have been used (see Figs. 5 and 6). As the ϵ-greedy algorithm makes local decisions (at a state level) and the $\mathcal{R}$minimax makes strategic decisions (at a path level), a fixed entropy for both algorithms has been fixed on the tree, so that the expectation of the cost can be compared under similar conditions.

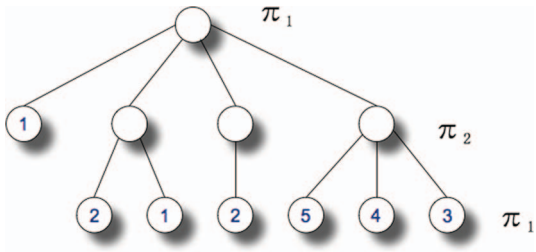

Fig. 6. Game tree with a depth of 3 composed of 11 nodes for comparison between the $\mathcal{R}$minimax and the ϵ-greedy. Leaf nodes contain the utility values. π_1 plays MIN and π_2 plays MAX.

TABLE II
Comparison of $\mathcal{R}$ Minimax and ϵ-Greedy Algorithms on the Tree From Fig. 5 for Different Levels of Entropy H. Parameter Estimations ($\hat{\theta}$ and $\hat{\epsilon}$) and Expected Costs Are Reported

H	$\hat{\theta}$	$\mathbb{E}[C]_{Rminimax}$	$\hat{\epsilon}$	$\mathbb{E}[C]_{\epsilon-greedy}$
1.0	0.4469	1.9000	0.6679	12.5116
0.9	0.6397	1.7150	0.7187	10.7515
0.8	0.7984	1.5766	0.7665	9.0930
0.7	0.9474	1.4624	0.8080	7.6545
0.6	1.0987	1.3636	0.8456	6.3514
0.5	1.2574	1.2785	0.8798	5.1667
0.4	1.4356	1.2035	0.9106	4.1005
0.3	1.6456	1.1385	0.9384	3.1359
0.2	1.9239	1.0818	0.9628	2.2897
0.1	2.3634	1.0348	0.9838	1.5619

In order to estimate both $\hat{\theta}$ and $\hat{\epsilon}$, a binary search algorithm has been used as follows: 1) the value of the entropy over tree H_0 is fixed; 2) two initial values of θ (or ϵ) are used to compute the probabilities of the different paths or strategies, i.e., $\wp$, of the tree (see Algorithm 1); 3) the obtained entropy of tree H_t is computed due to (1); 4) if the obtained $H_t = H_0$, we consider the value of θ as $\hat{\theta}$ (or respectively ϵ); and 5) otherwise, we apply the binary search and continue searching on a subinterval of the initial values until $H_t = H_0$.

Table II shows the result of this experiment for the tree from Fig. 5 with utility values {1, 2, 98, 99, 100}. Each arc is supposed to have a unit cost $c_{kk'} = 1$. For different levels of the entropy (the constraint of our optimization problem), parameters are estimated ($\hat{\theta}$ and $\hat{\epsilon}$), letting us compute the expected cost (the variable to optimize) when following a $\mathcal{R}$minimax or ϵ-greedy strategy.

The implemented $\mathcal{R}$minimax corresponds with Algorithm 1, and the implementation of the ϵ-greedy combines the typical ϵ-greedy[2] for π_1 (odd levels) with a standard MINIMAX for π_2 (even-levels). The same experiment has been repeated with the tree from Fig. 6 with utility values {1, 2, 3, 4, 5} in order to analyze the impact of the chosen utility values on the comparison between both algorithms. Results are presented in Table III.

Results show that the expected cost in the case of the $\mathcal{R}$minimax is always below those values for the ϵ-greedy regardless of the utility values. This behavior was expected, as the $\mathcal{R}$minimax is indeed optimal for a given entropy. It must

[2]Let us remind that ϵ-greedy takes the optimal action with probability (ϵ/n) and other actions with probability $(1-\epsilon)/m$, where n is the number of optimal actions, and m is the number of nonoptimal actions.

TABLE III
COMPARISON OF $\mathcal{R}$MINIMAX AND ϵ-GREEDY ALGORITHMS ON THE TREE FROM FIG. 6 FOR DIFFERENT LEVELS OF ENTROPY H. PARAMETER ESTIMATIONS ($\hat{\theta}$ AND $\hat{\epsilon}$) AND EXPECTED COSTS ARE REPORTED

H	$\hat{\theta}$	$\mathbb{E}[C]_{Rminimax}$	$\hat{\epsilon}$	$\mathbb{E}[C]_{\epsilon-greedy}$
1.0	0.6349	1.8311	0.6679	1.9962
0.9	0.7496	1.6877	0.7187	1.8439
0.8	0.8692	1.5628	0.7665	1.7004
0.7	0.9937	1.4550	0.8080	1.5759
0.6	1.1280	1.3601	0.8456	1.4631
0.5	1.2745	1.2773	0.8798	1.3606
0.4	1.4454	1.2031	0.9106	1.2683
0.3	1.6505	1.1385	0.9384	1.1848
0.2	1.9239	1.0821	0.9628	1.1116
0.1	2.3634	1.0348	0.9838	1.0486

be also noted that the more they differ the utility values, i.e., the quality of the goals, the bigger the difference between the expected cost of both algorithms.

V. CONCLUSION

This paper has presented a randomized version of the MINIMAX algorithm, which turns a zero-sum perfect-information two-player game into a nondeterministic game adapted to the player's level. By using the RSP framework, it is not only able to compute the probabilities of each play through dynamic programming techniques but also able to optimally vary the strength of the AI by adjusting the entropy through the θ parameter. There is a clear relation between the $\mathcal{R}$minimax algorithm and mixed strategies in the game theory and the methods used in reinforcement learning. Yet, these methods provide either a stochastic behavior at a local level (mixed strategies and reinforcement learning) or a global stochastic behavior at a global level (reinforcement learning with *online learning*) but fail to find an optimal policy. The presented method has provided a global optimal strategy (for the depth of the computed game tree) given a level of entropy while still simulating a stochastic behavior and following an optimal policy (for a degree of entropy θ) at the same complexity than simpler techniques (such as ϵ-greedy or local Boltzmann exploration).

The main drawback of this method is that paths are assumed to be independent and the opponent is assumed to be fully rational, both of which are not realistic for some problems.

Simulation experiments have led to the conclusion that the $\mathcal{R}$minimax algorithm behaves as expected. The compound of the $\mathcal{R}$minimax with pruning techniques, as well as techniques for reducing the search space, has been demonstrated to be effective.

Future work will focus on two main areas, i.e., the investigation the extension of the $\mathcal{R}$minimax to multiplayer games, as well as online or dynamic games; the estimation of a real player's θ parameter in order to mimic users' behavior and follow a similar learning curve; and the application of this framework to nested Monte Carlo search techniques.

ACKNOWLEDGMENT

The authors would like to thank *Région Wallonne*and the Belgian *Politique Scientifique Fédérale*for the opportunity to conduct both fundamental and applied research, and the reviewers for their constructive comments and input.

REFERENCES

[1] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens, "Tuning continual exploration in reinforcement learning: An optimality property of the Boltzmann strategy," *Neurocomputing*, vol. 71, no. 13–15, pp. 2507–2520, Aug. 2008.
[2] G. M. Adelson-Velsky, V. L. Arlazarov, and M. V. Donskoy, *Algorithms for Games*. New York: Springer-Verlag, 1988.
[3] T. Akamatsu, "Cyclic flows, Markov process and stochastic traffic assignment," *Transp. Res. B, Methodological*, vol. 30, no. 5, pp. 369–386, Oct. 1996.
[4] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Belmont, MA: Athena Scientific, 2000.
[5] M. Buro, "Toward opening book learning," in *Proc. Workshop Comput. Games*, Nagoya, Japan, 1997, pp. 1–5.
[6] D. Carmel and S. Markovitch, "Exploration strategies for model-based learning in multi-agent systems: Exploration strategies," *Auton. Agents Multi-Agent Syst.*, vol. 2, no. 2, pp. 141–172, Jun. 1999.
[7] T. Cazenave, "Nested Monte-Carlo search," in *Proc. Int. Joint Conf. Artif. Intell.*, 2009, pp. 456–461.
[8] L. Chatriot, S. Gelly, J. Hoock, J. Pérez, A. Rimmel, and O. Teytaud, "Introduction de connaissances expertes en Bandit-Based Monte-Carlo planning avec application au Computer-Go," in *Proc. J. Francophones Planification, Décision Apprentissage Conduite Syst.*, Metz, France, 2008.
[9] P. Coquelin and R. Munos, "Bandit algorithms for tree search," INRIA Futurs, Orsay, France, INRIA-00150207, 2007.
[10] S. García-Díez, F. Fouss, M. Shimbo, and M. Saerens, "A sum-over-paths extension of edit distances accounting for all sequence alignments," *Pattern Recognit.*, vol. 44, no. 6, pp. 1172–1182, 2011.
[11] G. Gigerenzer and R. Selten, *Bounded Rationality: The Adaptive Toolbox*. Cambridge, MA: MIT Press, 2002.
[12] L. R. Harris, "The heuristic search and the game of chess: A study of quiescence, sacrifices, and plan oriented play," in *Proc. 4th Int. Joint Conf. Artif. Intell.*, San Francisco, CA, 1975, pp. 334–339.
[13] X. Huang, Y. Ariki, and M. Jack, *Hidden Markov Models for Speech Recognition*. New York: Columbia Univ. Press, 1990.
[14] G. H. John, "When the best move isn't optimal: Q-learning with exploration," in *Proc. 12th Nat. Conf. Artif. Intell.*, 1994, p. 1464.
[15] J. N. Kapur and H. K. Kesavan, *Entropy Optimization Principles With Applications*. Boston, MA: Academic, 1992.
[16] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.*, 2006, pp. 282–293.
[17] C. F. Lee and D. H. Wolpert, "Product distribution theory for control of multi-agent systems," in *Proc. 3rd Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2004, vol. 2, pp. 522–529, IEEE Computer Society, Washington, DC.
[18] G. F. Luger, *Artificial intelligence: Structures and Strategies for Complex Problem Solving*, 6th ed. Upper Saddle River, NJ: Pearson Int., 2009.
[19] D. Michie, "Game-playing and game-learning automata," in *Adv. Programming Non-Numer. Comput.*, L. Fox, Ed. New York: Pergamon, 1966, pp. 183–196.
[20] I. Millington, *Artificial Intelligence for Games*. San Francisco, CA: Morgan Kaufmann, 2006.
[21] P. Morris, *Introduction to Game Theory*. New York: Springer-Verlag, 1994.
[22] A. Newell, J. Shaw, and H. Simon, "Chess playing programs and the problem of complexity," *IBM J. Res. Develop.*, vol. 4, no. 2, pp. 320–335, 1958.
[23] N. J. Nilsson, *Artificial Intelligence: A New Synthesis*. San Francisco, CA: Morgan Kaufmann, 1998.
[24] M. J. Osborne, *An Introduction to Game Theory*. London, U.K.: Oxford Univ. Press, 2002.
[25] S. D. Patek and D. P. Bertsekas, "Stochastic shortest path games," *SIAM J. Control Optim.*, vol. 37, no. 3, pp. 804–824, Feb. 1999.
[26] A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin, "Best-first fixed-depth game-tree search in practice," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, 1995, pp. 273–279, Morgan Kaufmann, San Francisco, CA.
[27] E. Rasmusen, *Games and Information: An Introduction to Game Theory*. Oxford, U.K.: Blackwell, 1989.
[28] A. Reinefeld, J. Schaeffer, and A. Marsland, "Information acquisition in minimal window search," in *Proc. 9th Int. Joint Conf. Artif. Intell.*, 1985, vol. 2, pp. 1040–1043, Morgan Kaufmann, San Francisco, CA.

[29] A. Rubinstein, Ed., *Modeling Bounded Rationality*. Cambridge, MA: MIT Press, 1998.
[30] S. Russell and E. Wefald, "Do the right thing: Studies in limited rationality," in *Artificial Intelligence*. Cambridge, MA: MIT Press, 1991.
[31] S. J. Russell and Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2003.
[32] M. Saerens, Y. Achbany, F. Fouss, and L. Yen, "Randomized shortest-path problems: Two related models," *Neural Comput.*, vol. 21, no. 8, pp. 2363–2404, Aug. 2009.
[33] L. Shapley, "Stochastic games," in *Proc. Nat. Acad. Sci. United States Amer.*, 1953, vol. 39, pp. 1095–1100.
[34] H. A. Simon, "Bounded rationality and organizational learning," *Org. Sci.*, vol. 2, no. 1, pp. 125–134, Feb. 1991.
[35] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Mach. Learn.*, vol. 39, no. 3, pp. 287–308, 2000.
[36] J. Smed, *Algorithms and Networking for Computer*. New York: Wiley, 2006.
[37] R. Somla, "New algorithms for solving simple stochastic games," *Electron. Notes Theor. Comput. Sci.*, vol. 119, no. 1, pp. 51–65, 2005.
[38] M. Stenmark, "Synthesizing board evaluation functions for connect4 using machine learning techniques," M.S. thesis, Dept. Comput. Sci., Østfold Univ. College, Fredrikstad, NO, 2005.
[39] N. R. Sturtevant and R. E. Korf, "On pruning techniques for multi-player games," in *Proc. 17th Nat. Conf. Artif. Intell.*, 2000, pp. 201–207.
[40] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proc. 7th Int. Conf. Mach. Learn.*, 1990, pp. 216–224.
[41] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[42] S. Thrun, "Exploration in active learning," *The Handbook of Brain Theory and Neural Networks*, pp. 381–384, 1998.
[43] D. Wolpert, "Information theory—The bridge connecting bounded rational game theory and statistical physics," *Complex Eng. Syst.*, vol. 14, pp. 262–290, 2006.

Authors' photographs and biographies not available at the time of publication.