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 \(R\)-minimax. The \(R\)-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 [27] is applied for biasing the AI adversary towards worse or better solutions, therefore controlling its strength. This framework takes into account all possible strategies by computing an optimal trade-off between exploration (quantified by the spread entropy 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 \(R\)-minimax implements a non-deterministic, 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 \(R\)-minimax behaves as expected.

Index Terms—minimax, randomized shortest-paths, two-player zero-sum perfect-information games.

I. GENERAL INTRODUCTION

Artificial intelligence (AI) techniques [14], [19], [26] are widely used in realistic-behavior video games [16], [30]. These methods aim, i.e., 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, such as chess and draughts. These games can be seen as a succession of plays which alternate from one player to another, and where the profit is maximized for the current player – therefore, minimized for the opponent. They are often solved thanks to the well-known MINIMAX algorithm [14], [16], [19], [26], [30] which is straightforwardly or indirectly used in most board games.

From its inception, MINIMAX assumes perfect rationality for both players and, therefore, it is completely deterministic – 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-cost strategy for this entropy. In this way, good (low-cost) randomized strategies are favored, while bad ones (high-cost) are discarded.

Adjusting the trade-off between exploitation and exploration of the game tree, and therefore the strength of the player, is achieved by varying the entropy. The proposed method, called \(R\)-minimax, is the application of the randomized shortest-path (RSP) framework [27] to game trees.

In summary, the \(R\)-minimax contributions are (i) to model non-rational players, (ii) to control the strength of a player, and (iii) to avoid the total predictability of a player.

A. 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 on bounding the depth of the tree with a \(n\)-ply look-ahead strategy [14], where \(n\) is the number of explored levels of the tree. Another very common family of strategies are the alpha-beta (AB) pruning techniques [18]. The AB algorithm prunes irrelevant subtrees which will never be part of the MINIMAX strategy by using a window of two plies. An AB multi-player version is proposed in [33]. The Negascout algorithm [24] reduces even further this window size which allows to perform a faster pruning than AB. Nonetheless, the tree may be massively pruned leading to the elimination of good strategies. Similarly, the Memory-enhanced Test Driver (MTD) [22] limits the AB window size to zero. Although this pruning is faster, an initial “guess”, \(f\), 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 that MTD outperforms the Negascout in the number of searched nodes, it suffers from stability issues, it depends on the transposition tables, and is also very sensitive to the initial guess. Eventually, a pruning technique which computes the expected value of the continued search is proposed by [25]. It has been shown [25] that this method suffers also from numerical instabilities.

Opening books [4] are another improvement technique applied to huge search space games. Efficient “opening” as well as “ending” game strategies that are often used by expert players are stored in these books. It is proved that initial strategies are critic for reducing the search space, as well as guiding the game towards the winning states. However, even when the search space is reduced, interaction time remains a key feature that must also be 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. *Qui
cesence pruning* [9] avoids searching the branches of the tree whose heu
ritic function values are stable and, therefore, with no leadership
changes. MINIMAX has also been 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
[15]. Eventually, a stochastic approach which computes the
probabilities of correctly scoring the following moves, via a
heuristic function, is given in [2].

A different approach to the randomized strategies can be
found in the Game Theory literature [17], [20], [23]. 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,
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 pseudo-random. An extension of these strategies
for two-player turn-based random games are *stochastic games*
[28]. 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 [21], [31].

Nevertheless, little attention has been paid to modeling the
strength of an adversary in two-player zero-sum games in the
artificial intelligence (AI) community. A basic approach
consists in using the n-ply look-ahead algorithm [14] 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 easily be outperformed by the user
(who normally plans 6 or 8 plies ahead), while for very high
values (n > 8) it may become extremely difficult to win.
Other frequently used techniques are ε-greedy [35], where the
optimal branch is taken with probability 1 − ε and a random
branch with uniformly distributed probability 1/n.

*Boltzmann exploration* [35], 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 for a given entropy
[1].

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

II. A 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. Firstly, the assumption of perfect rationality for
both players is unrealistic, as human players often incur into
error. Secondly, 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.
Thirdly, the behavior of the player is deterministic and thus
predictable. Fourthly, 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 − winning/losing state). MINIMAX
chooses the path which 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, perfect rationality of the player is not assumed; 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. The randomized shortest-path framework

In order to allow randomization of the game strategies we
will briefly introduce the randomized shortest-path framework
(RSP) – inspired by a stochastic transportation model [3] –; see
[27], or [8] for an alternative derivation in the special case of
acyclic graphs. Let \( G \) be a graph containing a source node with
index 1 and a destination or goal node with index n. A non-
negative 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 \( \mathcal{P}_{1n} \). Assume that each path \( \varphi \in \mathcal{P}_{1n} \) is composed by a
sequence of arcs \( k \to k' \) that ties the source to the destination
node. Moreover, let the total cost \( C(\varphi) \) of a path \( \varphi \) be the sum
of these local costs along \( \varphi \). The path randomization will be

\[
\text{minimize } \sum_{\varphi \in \mathcal{P}_{1n}} P(\varphi) C(\varphi)
\]

subject to \( \sum_{\varphi \in \mathcal{P}_{1n}} P(\varphi) \ln P(\varphi) = H_0 \)
It can be shown from this optimization problem [27] that the probability of each path is
\[
P(\psi) = \frac{\exp[-\theta C(\psi)]}{Z}
\]  
where \( Z = \sum_{\psi \in P_n} \exp[-\theta C(\psi)] \) is the partition function and \( P(\psi) \) therefore follows a Boltzmann distribution, as imposed by the problem constraint. 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, \( \theta \) can be seen as the parameter that controls the entropy [12]. Indeed, \( \theta \) is the inverse temperature which is inversely proportional to the entropy, allowing the trade-off 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 generally not fulfilled in practice.

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, \( n_k \), and the expected number of passages per link \( k \to k' \), \( n_{kk'} \), can be defined as
\[
n_k = \sum_{\psi \in P_n} \delta(\psi; k) \frac{\exp[-\theta C(\psi)]}{Z}
\]  
\[
n_{kk'} = \sum_{\psi \in P_n} \delta(\psi; k, k') \frac{\exp[-\theta C(\psi)]}{Z}  
\]
where \( \delta(\psi; k, k') \) (\( \delta(\psi; k') \)) is an indicator variable that counts how many times the link \( k \to k' \) (the node \( k \)) belongs to the path \( \psi \). These equations simply sum the probabilities of passing through the node or the link. By rewriting Equations (3) and (4) (see [27] for more details) in terms of forward (\( z_k^f \)) and backward (\( z_k^b \)) variables, the following are obtained
\[
n_{kk'} = z_k^f \exp[-\theta c_{kk'}] z_k^b
\]  
\[
n_k = \sum_{k' \in \text{Succ}(k)} \frac{n_{kk'}}{n_{kk'}} = z_k^f z_k^b
\]
where Succ(\( k \)) is the set of successors of state \( k \), and where the forward variable \( z_k^f \) and the backward variable \( z_k^b \) are defined as [8] \( z_k^f = \sum_{\psi \in P_n} \exp[-\theta C(\psi)] \) and \( z_k^b = \sum_{\psi \in P_n} \exp[-\theta C(\psi)] \) where \( P_{n1} \) is the set of paths starting in node 1 and ending in node \( k \), and \( P_{kn} \) is the set of paths starting in node \( k \) and ending in node \( n \). They can be computed [27] thanks to the recurrence equations
\[
\begin{align*}
  z_n^b &= 1, & n \in \mathcal{N} \\
  z_k^b &= \sum_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}] z_k^b, \text{ for } k \neq n
\end{align*}
\]
which only depend on the backward variables and the local costs. The denominator \( z_k^b = \sum_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}] z_k^b \) 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 Equation (2). The random walker explores the graph with a fixed entropy while minimizing the expected cost to the destination state, assuming independence of the paths. This RSP framework will now be applied to our game tree in order to provide an optimal strategy.

B. The Rminimax algorithm

The application of the RSP framework to the game tree will allow to bias the transition probabilities towards better or worse solutions as \( \theta \) increases or decreases. In that case, the graph \( G \) is a tree and it is therefore acyclic. Equation (7) defines therefore the recurrence relations allowing to compute the backward variables \( z_k^b \) from the destination node \( n \) to each intermediary node \( k \).

Assume that \( \pi_1 \) is our AI player, and \( \pi_2 \) is the opponent. We will randomize \( \pi_1 \)'s strategy while still assuming that \( \pi_2 \) 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 \( P_{1N} \). By applying the RSP framework to this situation, the backward variables (Equation (7)) are redefined in terms of the following recurrence relations
\[
\begin{align*}
  z_n^b &= 1, \text{ for } n \in \mathcal{N} \\
  z_k^b &= \sum_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}] z_k^b, \text{ for } k \neq n \\
  z_k^b &= \min_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}] z_k^b, \text{ for } k \in \mathcal{N} \text{'s turn}
\end{align*}
\]
where \( k \notin \mathcal{N} \) is assumed. It can be observed that when \( \pi_1 \) plays, it takes into account the costs of all successors of state \( k \) for randomizing its future strategy, while \( \pi_2 \) plays the best strategy by considering only one branch of the tree. Indeed, since the transition probabilities (the policy followed by player
$\pi_1$ are proportional to $\exp[-\theta c_{kk'}] z^b_{kk'}$ (see Equation (9)). $\pi_2$ chooses the action corresponding to the lowest transition probability, i.e. the move that is least favorable to his opponent $\pi_1$, all the other moves being dismissed – the game tree is pruned accordingly. As $\pi_1$ and $\pi_2$ play in turn, the value of the backward variables is computed by alternating both equations. It must also be noticed that in order to avoid overflow or underflow problems, the standard formula for the logarithm of a sum (see, e.g., [10]) could be applied when computing $z^b_{kk'}$.

Although it is not immediately obvious from Equation (10), player $\pi_1$ minimizes the expected cost to the end-game by following the optimal policy provided by Equation (9) (this directly follows from the RSP framework, see Equation (7)) while player $\pi_2$ tries to maximize it. Indeed, let us take $-\frac{1}{\theta} \log$ of each member of the recurrence relation for player $\pi_2$ in Equation (10),

$$-\frac{1}{\theta} \log(z^b_{kk'}) = -\frac{1}{\theta} \log\left(\min_{k' \in \text{Succ}(k)} \exp[-\theta c_{kk'}] z^b_{kk'}\right)$$

By using $-\log(\min(x,y)) = \max(-\log(x), -\log(y))$ and defining $v_k = -\frac{1}{\theta} \log(z^b_{kk})$, we obtain for player $\pi_2$

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

Now, this is exactly the recurrence equation, akin to the Bellman equation, allowing to compute the maximal-cost path to the end-game states. Therefore, player $\pi_2$ consistently tries to maximize the cost. If player $\pi_1$ would only consider the best move, as does player $\pi_2$, we recover the standard minimax. The $R$minimax algorithm is summarized in Algorithm 1.

### Algorithm 1 $R$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 ($\infty$ for a perfect rational player, $\approx 0$ for an almost completely random player).
- $c_{kk'}$: The cost of each arc of the tree.
  1. Assign $z^b_{nn} = 1$ for each $n \in \mathcal{N}$.
  2. Compute recursively the $z^b_n$ according to Equation (10).
  3. Compute the corresponding $p_{kk'}$ according to Equation (9).
  4. Return $p_{kk'}$: the transition probabilities for the next play.

Note that, when $\theta$ has a high value, near-optimal strategies are chosen by the AI player $\pi_1$, while for small values, he will model a weak rival with a poor strategy. As an example of the effect of the different $\theta$ 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 a strength $\theta$. 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 $\theta$ decreases, the transition probabilities are less biased towards 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).

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$p_{12}$</th>
<th>$p_{13}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.998</td>
<td>0.002</td>
</tr>
<tr>
<td>0.5</td>
<td>0.728</td>
<td>0.268</td>
</tr>
<tr>
<td>0.001</td>
<td>0.5001</td>
<td>0.4999</td>
</tr>
</tbody>
</table>

**TABLE I**

**EXAMPLE OF TRANSITION PROBABILITIES $p_{ij}$ (TRANSITION PROBABILITY FROM NODE $i$ TO CHILD $j$) FOR A SIMPLE BINARY GAME TREE OF DEPTH 3, WHEN VARYING $\theta$.

Similar approaches (although not optimal) can be found in reinforcement learning [5], [11], [29], [35]. Typical exploration methods are incorporated to reinforce learning in multi-agent systems by [5]. One such method is the Softmax [34] technique (also called Boltzmann exploration) based on applying a Boltzmann distribution on each of the possible actions (branches of the tree) based on an utility function. As stated in [36], 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 bandit strategy in order to avoid the lookahead-pathology, and shows competitive results [13]. Another bandit based method which performs efficient “cuts” of sub-optimal branches with high confidence is proposed by [7]. This technique allows, as well, to control the trade-off between exploration and exploitation of the tree. Eventually, [6] propose the application to the game of Go of a bandit technique which bias the exploration of the tree so as 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 reinforcement learning techniques eventually find an optimal policy and stop exploring the graph, therefore losing their stochastic behavior. On the other hand we may find the reinforcement learning technique proposed in [11], which continually explores the graph. However, the convergence to the optimal policy can no longer be proved.

### 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. Game trees for both games have been generated with both the MINIMAX as well as the alpha-beta (AB) algorithm. Two AI opponents have been simulated, each with a different strength, $\theta_i$, for testing the behavior of our method when confronting different heterogeneous players. The simulation methodology is as follows:

1. The game tree for the current node, $k$, is computed with the MINIMAX algorithm:

   ```
   Algorithm 1 Rminimax.
   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 ($\infty$ for a perfect rational player, $\approx 0$ for an almost completely random player).
   - $c_{kk'}$: The cost of each arc of the tree.
   1. Assign $z^b_{nn} = 1$ for each $n \in \mathcal{N}$.
   2. Compute recursively the $z^b_n$ according to Equation (10).
   3. Compute the corresponding $p_{kk'}$ according to Equation (9).
   4. Return $p_{kk'}$: the transition probabilities for the next play.
   ```
a) the tree can be either fully generated, or limited to
a 5-ply lookahead (depth = 5).1  
b) the tree can be be pruned with the alpha-beta
algorithm.

2) A reward is assigned to each transition to a winning
node. Cost are computed as follows:

a) in the case of a full game tree, lower costs are
assigned to winning nodes and higher ones to los-
ing nodes. Tie nodes are assigned a value between
those of winning and losing nodes.

b) in the case of a 5-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 to
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 $R$minimax algorithm as
described in Algorithm (1) with strength $\theta_i$ 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 a result $r$ which takes its values in $r \in [-100, 100]$, 
which indicates the number of victories of both players. If $r >
0$, player $\pi_1$ has $|r|$ out of 100 victories over $\pi_2$. Otherwise,
the winner will be $\pi_2$ with $|r|$ victories out of 100, and $r = 0$
represents result parity.

A. $R$minimax with full game tree

For getting a better insight about $R$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 per-
formance of our method when two players of different strengths
interact, 100 runs have been performed between two players of
varying strength $\theta$. According to our simulation methodology
stated above, the performance of both players is recorded
when applying the $R$minimax. Tested values of $\theta$ are $\theta_1 =
\{0.1, 0.3, 0.5, 0.7, 1, 5, 10\}$ and $\theta_2 = \{0.2, 0.5, 1, 2.5, 10\}$.
The resulting curves are shown in Figure 1.

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 $\theta_1 >> \theta_2$,
player 1 wins, while for $\theta_1 << \theta_2$, 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 $\theta$’s.

1It 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.

On the other hand, the steep of the curves reflect the effect of
the relative advantage of $\pi_1$ over $\pi_2$. Indeed, when $\pi_1$ moves
first, it has an advantage over $\pi_2$. This can be observed in
Figure 1 where a lower steep is shown for low values of $\theta_2$
for $\pi_2$.

B. $R$minimax with 5-ply lookahead and heuristics

Another frequent tool used in AI are heuristics. The perfor-
mance 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 5-ply
lookahead method is here implemented and combined with the
use of a heuristic function for scoring the final transitions. The
applied heuristics is the one proposed by [32], and corresponds
to the sum of two quantities: the number of winning lines
that may still be done for each following move, plus a fixed
quantity which corresponds to the goodness of the empty
positions which are left (some positions are more versatile than
others). Tested values of $\theta$ 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 Figure
2.

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 $\pi_1$ is not as clear as in the former
case. Indeed, for observing the same effect, significantly lower
values of $\theta$ are needed (than those of the Tic-Tac-Toe).

C. $R$minimax with 5-ply lookahead and alpha-beta pruning

For this simulation, a partial game tree of depth 5 has been
generated for the game of Connect-4. This time, a pruning
algorithm reducing the search space is applied. The objective is
to observe the behavior of the $R$minimax algorithm combined
with a technique which reduces not only the depth of the
tree, but also the search space. Tested values of $\theta$ are $\theta_1 =
\{0.01, 0.1, 0.3, 0.7, 10\}$ and $\theta_2 = \{0.1, 0.2, 0.5, 1, 2, 5, 10\}$.
The results are shown in Figure 3.
also to optimally vary the strength of the AI by adjusting the number of each play through dynamic programming techniques, but in this case, the relative advantage of $\pi_1$ is even smaller. In contrast to the former case, a smaller $\theta_2$ allows $\pi_2$ winning as $\theta_1$ decreases. This is due to the pruning of the alpha-beta, as it restrains the set of explored branches while our method considers all paths.

**IV. Conclusion**

This work presented a randomized version of the MIN-MAX algorithm which turns a zero-sum perfect-information two-player game into a non-deterministic game adapted to the player’s level. By using the randomized shortest-path framework, it is not only able to compute the probabilities of each play through dynamic programming techniques, but also to optimally vary the strength of the AI by adjusting the entropy through the $\theta$ parameter. There is a patent relation between the $R$minimax algorithm and mixed strategies in game theory and the methods used in reinforcement learning. Yet these methods provide either a stochastic behavior at a local level (mixed strategies, reinforcement learning), or they provide a global stochastic behavior at a global level (reinforcement learning with online learning) but fail to find an optimal policy. The presented method gives 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 $\theta$) at the same complexity than simpler techniques (such as $\epsilon$-greedy, or Boltzmann exploration).

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

Future work will focus on two main areas: (i) investigating the extension of the $R$minimax to multi-player games as well as online or dynamic games, and (ii) to the estimation of a real player’s $\theta$ parameter in order to mimic users’ behavior and follow a similar learning curve.

**Acknowledgments**

This work is partially supported by the Fonds pour la formation à la Recherche dans l’Industrie et dans l’Agriculture (F.R.I.A.) under grant reference F3/5/5-MCF/ROI/BC-21716. Part of this work was also funded by projects with the Région Wallonne and the Belgian Politique Scientifique Fédérale. We thank these institutions for giving us the opportunity to conduct both fundamental and applied research.

**References**