ESAIM: PS 23 (2019) 176–216
https://doi.org/10.1051/ps/2018006

ESAIM: Probability and Statistics
www.esaim-ps.org

ON MONTE-CARLO TREE SEARCH FOR DETERMINISTIC GAMES WITH ALTERNATE MOVES AND COMPLETE INFORMATION[☆]

Sylvain Delattre[1,*] and Nicolas Fournier[2]

Abstract. We consider a deterministic game with alternate moves and complete information, of which the issue is always the victory of one of the two opponents. We assume that this game is the realization of a random model enjoying some independence properties. We consider algorithms in the spirit of Monte-Carlo Tree Search, to estimate at best the minimax value of a given position: it consists in simulating, successively, n well-chosen matches, starting from this position. We build an algorithm, which is optimal, step by step, in some sense: once the n first matches are simulated, the algorithm decides from the statistics furnished by the n first matches (and the *a priori* we have on the game) how to simulate the $(n+1)$th match in such a way that the increase of information concerning the minimax value of the position under study is maximal. This algorithm is remarkably quick. We prove that our step by step optimal algorithm is not globally optimal and that it always converges in a finite number of steps, even if the *a priori* we have on the game is completely irrelevant. We finally test our algorithm, against MCTS, on Pearl's game [Pearl, *Artif. Intell.* **14** (1980) 113–138] and, with a very simple and universal *a priori*, on the game Connect Four and some variants. The numerical results are rather disappointing. We however exhibit some situations in which our algorithm seems efficient.

Mathematics Subject Classification. 91A05, 68T20, 60J80.

Received June 2, 2017. Accepted January 15, 2018.

1. Introduction

1.1. Monte-Carlo Tree Search algorithms

Monte-Carlo Tree Search (MCTS) are popular algorithms for heuristic search in two-player games. Let us mention the book of Munos [17] and the survey paper of Browne *et al.* [4], which we tried to briefly summarize here and to which we refer for a much more complete introduction on the topic.

We consider a deterministic game with complete information and alternate moves involving two players, that we call J_1 and J_0. We think of Go, Hex, Connect Four, etc. Such a game can always be represented as a discrete

[☆] We warmly thank Bruno Scherrer for his invaluable help. We also thank the anonymous referee for his numerous fruitful comments.

Keywords and phrases: Monte Carlo tree search, deterministic games with alternate moves and complete information, minimax values, finite random trees, branching property.

[1] Laboratoire de Probabilités et Modèles Aléatoires, UMR 7599, Université Paris Diderot, Case courrier 7012, 75205 Paris Cedex 13, France.
[2] Laboratoire de Probabilités et Modèles Aléatoires, UMR 7599, Université Pierre-et-Marie Curie, Case 188, 4 place Jussieu, 75252 Paris Cedex 5, France.

[*] Corresponding author: sylvain.delattre@univ-paris-diderot.fr

tree, of which the nodes are the positions of the game. Indeed, even if a single position can be thought as the child of two different positions, we can always reduce to this case by including the whole history of the game in the position. Also, we assume that the only possible outcomes of the game, which are represented by values on the leaves of the tree, are either 1 (if J_1 wins) or 0 (if J_0 wins). If draw is a possible outcome, we *e.g.* identify it to a victory of J_0.

Let r be a configuration in which J_1 has to choose between several moves. The problem we deal with is: how to select at best one of these moves with a computer and a given amount of time.

If having a huge amount of time, such a question can classically be completely solved by computing recursively, starting from the leaves, the minimax values $(R(x))_{x\in\mathcal{T}}$, see Remark 2.1. Here $\mathcal{T}$ is the tree (with root r and set of leaves $\mathcal{L}$) representing the game when starting from r, and for each $x \in \mathcal{T}$, $R(x)$ is the value of the game starting from x. In other words, $R(x) = 1$ if J_1 has a winning strategy when starting from x and 0 else. So we compute $R(x)$ for all the children x of r and choose a move leading to some x such that $R(x) = 1$, if such a child exists.

In practice, this is not feasible, except if the game (starting from r) is very small. One possibility is to cut the tree at some reasonable depth K, to assign some estimated values to all positions of depth K, and to compute the resulting (approximate) minimax values on the subtree above depth K. For example if playing Connect Four, one can assign to a position the value *remaining number of possible alignments for* J_1 *minus remaining number of possible alignments for* J_0. Of course, the choice of such a value is highly debatable, and heavily depends on the game.

A more universal possibility, introduced by Abramson [1], is to use some Monte-Carlo simulations: from each position with depth K, we handle a certain number N of uniformly random matches (or matches with a simple *default policy*), and we evaluate this position by the number of these matches that led to a victory of J_1 divided by N. Such a procedure is now called *Flat MCTS*, see Coquelin and Munos [8], Browne *et al.* [4], see also Ginsberg [13] and Sheppard [19].

Coulom [9] introduced the class of MCTS algorithms. Here are the main ideas: we have a *default policy* and a *selection procedure*. We make J_1 play against J_0 a certain number of times and make grow a subtree of the game. Initially, the subtree $\mathcal{T}_0$ consists of the root and its children. After n steps, we have the subtree $\mathcal{T}_n$ and some statistics $(C(x), W(x))_{x\in\mathcal{T}_n}$ provided by the previous matches: $C(x)$ is the number of times the node x has been crossed and $W(x)$ is the number of times this has led to a victory of J_1. Then we select a leave y of $\mathcal{T}_n$ using the selection procedure (which relies on the statistics $(C(x), W(x))_{x\in\mathcal{T}_n}$) and we end the match (from y) by using the default policy. We then build $\mathcal{T}_{n+1}$ by adding to $\mathcal{T}_n$ the children of y, and we increment the values of $(C(x), W(x))_{x\in\mathcal{T}_{n+1}}$ according to the issue of the match (actually, it suffices to compute these values for x in the branch from r to y and for the children of y). Once the given amount of time is elapsed, we choose the move leading to the child x of r with the highest $W(x)/C(x)$.

Actually, this procedure throws away a lot of data: $C(x)$ is not exactly the number of times x has been crossed, it is rather the number of times it has been crossed since $x \in \mathcal{T}_n$, and a similar fact holds for $W(x)$. In practice, this limits the memory used by the algorithm.

The most simple and universal default policy is to choose each move at uniform random and this is the case we will study in the present paper. It is of course more efficient to use a simplified strategy, depending on the game under study, but this is another topic.

Another important problem is to decide how to select the leave y of $\mathcal{T}_n$. Kocsis and Szepesvári [15] proposed to use some bandit ideas, developed (and shown to be optimal, in a very weak sense, for bandit problems) by Auer *et al.* [2], see Bubeck and Cesa-Bianchi [5] for a survey paper. They introduced a version of MCTS called UCT (for UCB for trees, UCB meaning Upper Confidence Bounds), in which the selection procedure is as follows. We start from the root r and we go down in $\mathcal{T}_n$: when in position x where J_1 (resp. J_0) has to play, we choose the child z of x maximizing $W(z)/C(z) + \sqrt{c(\log n)/C(z)}$ (resp. $(C(z) - W(z))/C(z) + \sqrt{c(\log n)/C(z)}$). At some time we arrive at some leave y of $\mathcal{T}_n$, this is the selected leave. Here $c > 0$ is a constant to be chosen empirically. Kocsis and Szepesvári [15] proved the convergence of UCT.

Chaslot *et al.* [7] have broaden the framework of MCTS. Also, they proposed different ways to select the best child (after all the computations): either the one with the highest W/C, the one with the highest C, or something intermediate.

Gelly *et al.* [12] experimented MCTS (UCT) on Go. They built the program MoGo, which also uses some pruning procedures, and obtained impressive results on a 9×9 board.

The early paper of Coquelin and Munos [8] contains many results. They showed that UCT can be inefficient on some particular trees and proposed a modification taking into account some possible smoothness of the tree and outcomes (in some sense). They also studied Flat MCTS.

Lee *et al.* [16] studied the problem of fitting precisely the parameters of the selection process. Of course, W/C means nothing if $C = 0$, so they empirically investigated what happens when using $(W + a)/(C + b) + \sqrt{c(\log n)/(C + 1)}$, for some constants $a, b, c > 0$. They conclude that $c = 0$ is often the best choice. This is not so surprising, since the logarithmic term is here to prevent from large deviation events, which do asymptotically not exist for deterministic games. MoGo uses $c = 0$ and *ad hoc* constants a and b. This version of MCTS is the one presented in Appendix A and the one we used to test our algorithm.

Let us mention the more recent theoretical work by Buşoniu *et al.* [6], as well as the paper of Garivier *et al.* [11] who study in details a bandit model for a two-round two-player random game.

The survey paper of Browne *et al.* [4] discusses many tricks to improve the numerical results and, of course, all this has been adapted with very special and accurate procedures to particular games. As everybody knows, AlphaGo [20] became the first Go program to beat a human professional Go player on a full-sized board. Of course, AlphaGo is far from using only MCTS, it also relies on deep-learning and many other things.

1.2. Our goal

We would like to study MCTS when using a probabilistic model for the game. To simplify the problem as much as possible, we only consider the case where the default policy is *play at uniform random*, and we consider the modified version of MCTS described in Appendix A, where we keep all the information. This may cause memory problems in practice, but we do not discuss such difficulties. So the modified version is as follows, for some constants $a, b > 0$ to be fitted empirically. We make J_1 play against J_0 a certain number of times and make a subtree of the game grow. Initially, the subtree $\mathcal{T}_0$ consists in the root r and its children. After n steps, we have the subtree $\mathcal{T}_n$ and some statistics $(C(x), W(x))_{x\in\mathcal{T}_n}$ provided by the previous matches: $C(x)$ is the number of times the node x has been crossed and $W(x)$ the number of times this has led to a victory of J_1. The $(n + 1)$th step is as follows: start from r and go down in $\mathcal{T}_n$ by following the highest values of $(W + a)/(C + b)$ (resp. $(C - W + a)/(C + b)$) if it is J_1's turn to play (resp. J_0's turn to play), until we arrive at some leave z of $\mathcal{T}_n$. From there, complete the match at uniform random until we reach a leave y of $\mathcal{T}$. We then build $\mathcal{T}_{n+1}$ by adding to $\mathcal{T}_n$ the whole branch from z to y together with all the brothers of the elements of this branch, and we compute the values of $(C(x), W(x))_{x\in\mathcal{T}_{n+1}}$ (actually, it suffices to compute these values for x in the branch from r to y). Once the given amount of time is elapsed, we choose the move leading to the child x of r with the highest $C(x)/W(x)$.

As shown by Coquelin and Munos [8], one can build games for which MCTS is not very efficient. So it would be interesting to know for which class of games it is. This seems to be a very difficult problem. One possibility is to study if MCTS works well *in mean*, *i.e.* when the game is chosen at random. In other words, we assume that the tree and the outcomes are the realization of a random model. We use a simple toy model enjoying some independance properties, which is far from convincing if modeling true games but for which we can handle a complete theoretical study.

We consider a class of algorithms resembling the above mentioned modified version of MCTS. After n simulated matches, we have some information $\mathcal{F}_n$ on the game: we have visited n leaves, we know the outcomes of the game at these n leaves, and we have the explored tree $\mathcal{T}_n = B_n \cup D_n$, where B_n is the set of all crossed positions, and D_n is the boundary of the explored tree (roughly, D_n consists of uncrossed positions of which the father belongs to B_n).

So we can approximate $R(r)$, which is the quantity of interest, by $\mathbb{E}[R(r)|\mathcal{F}_n]$ (if the latter can be computed). Using only this information $\mathcal{F}_n$ (and possibly some *a priori* on the game furnished by the model), how to select $z \in D_n$ so that, simulating a uniformly random match starting from z and updating the information, $\mathbb{E}[R(r)|\mathcal{F}_{n+1}]$ is as close as possible (in L^2) to $R(r)$?

We need a few assumptions. In words, (a) the tree and outcomes enjoy some independence properties, (b) we can compute, at least numerically, for $x \in D_n$, $m(x) = \textit{mean value of } R(x)$ and $s(x) = \textit{mean quantity of information that a uniformly random match starting from } x \textit{ will provide.}$ See Section 2.8 for precise definitions.

Under such conditions, we show that $\mathbb{E}[R(r)|\mathcal{F}_n]$ can indeed be computed (numerically), and z can be selected as desired. The procedure resembles in spirit MCTS, but is of course more complicated and requires more computations.

This is extremely surprising: the computational cost to find the best $z \in D_n$ does not increase with n, because this choice requires to compute some values of which the update does not concern the whole tree $\mathcal{T}_n$, but only the last visited branch, as MCTS. (Actually, we also need to update all the brothers of the last visited branch, but this remains rather reasonable). It seems miraculous that this *theoretical* algorithm behaves so well. Any modification, such as taking draws into account or changing the notion of optimality, seems to lead to drastically more expensive algorithms, that require to update some values on the whole visited tree $\mathcal{T}_n$. We believe that this is the most interesting fact of the paper.

The resulting algorithm is explained in details in the next section. We will prove that this algorithm is convergent (in a finite number of steps) even if the model is completely irrelevant. This is not very surprising, since all the leaves of the game are visited after a finite number of steps. We will also prove on an example that our algorithm is *myopic*: it is not *globally* optimal. There is a theory showing that for a class of problems, a step by step optimal algorithm is *almost* globally optimal, *i.e.* up to some reasonable factor, see Golovin and Kraus [14]. However, it is unclear whether this class of problems includes ours.

1.3. Choice of the parameters

We will show, on different classes of models, how to compute the functions m and s required to implement our algorithm. We studied essentially two possibilities.

In the first one, we assume that the tree $\mathcal{T}$ is the realization of an inhomogeneous Galton–Watson tree, with known reproduction laws, and that the outcomes of the game are the realizations of i.i.d. Bernoulli random variables of which the parameters depend only on the depths of the involved (terminal) positions. Then $m(x)$ and $s(x)$ depend only on the depth of the node x. We can compute them numerically once for all, using rough statistics we have on the *true game* we want to play (*e.g.* Connect Four), by handling a high number of uniformly random matches.

The second possibility is much more universal and adaptive and works better in practice. At the beginning, we prescribe that $m(r) = a$, for some fixed $a \in (0, 1)$. Then each time a new litter $\{y_1, \dots, y_d\}$ (with father x) is created by the algorithm, we set $m(y_1) = \cdots = m(y_d) = m(x)^{1/d}$ if x is a position where it is J_0's turn to play, and $m(y_1) = \cdots = m(y_d) = 1 - (1 - m(x))^{1/d}$ else. Observe here that d is discovered by the algorithm at the same time as the new litter. Concerning s, we have different possibilities (see Sects. 2.14 and 6), more or less justified from a theoretical point of view, among which $s(x) = 1$ for all x does not seem to be too bad.

The first possibility seems more realistic but works less well than the second one in practice and requires some preliminary fitting. The second possibility relies on a symmetry modeling consideration: assuming that all the individuals of the new litter behave similarly necessarily leads to such a function m. This is much more universal in that once the value of $a = m(r)$ is fixed (actually, $a = 0.5$ does not seem to be worse than another value), everything can be computed in a way not depending on the true game. Of course, the choice of $a = m(r)$ is debatable, but does not seem to be very important in practice.

1.4. Comments and a few more references

Our class of models generalize a lot the Pearl model [18], which simply consists of a regular tree with degree d, depth K, with i.i.d. Bernoulli(p) outcomes on the leaves. However, we still assume a lot of independence. This is not fully realistic and is probably the main reason why our numerical experiments are rather disappointing.

The model proposed by Devroye-Kamoun [10] seems much more relevant, as they introduce correlations between the outcomes as follows. They consider that each edge of the tree has a value (*e.g.* Gaussian). The value of the leave is then 1 if the sum of the values along the corresponding branch is positive, and 0 else. This

more or less models that a player builds, little by little, his game. But from a theoretical point of view, it is very difficult to study, because we only observe the values of the leaves, not of the edges. So this unfortunately falls completely out of our scope. See also Coquelin and Munos [8] for a notion of smoothness of the tree, which seems rather relevant from a modeling point of view.

Finally, our approach is often called *Bayesian*, because we have an *a priori* law for the game. This has already been studied in the artificial intelligence literature. See Baum and Smith [3] and Tesauro, Rajan and Segal [22]. Both introduce some conditional expectations of the minimax values and formulas resembling (2.2) already appear.

1.5. Pruning

Our algorithm automatically proceeds to some pruning, as AlphaBeta, which is a clever version to compute exactly the minimax values of a (small) game. This can be understood if reading the proof of Proposition 2.13. The basic idea is as follows: if we know from the information we have that $R(x) = 0$ for some internal node $x \in \mathcal{T}$ and if the father v of x is a position where J_0 plays, then there is no need to study the brothers of x, because $R(v) = 0$. And indeed, our algorithm will never visit these brothers.

Some versions of MCTS with some additional pruning procedures have already been introduced empirically. See Gelly *et al.* [12] for MoGo, as well as many other references in [4]. It seems rather nice that our algorithm automatically prunes and this holds even if the *a priori* we have on the game is completely irrelevant. Of course, if playing a large game, this pruning will occur only near the leaves.

1.6. Numerical experiments

We have tested our *general* algorithm against some *general* versions of MCTS. We would not have the least chance if using some versions of MCTS modified in such a way that it takes into account some symmetries of a particular game, with a more clever default policy, etc. But we hope our algorithm might also be adapted to particular games.

Next, let us mention that our algorithm is subjected to some numerical problems, in that we have to compute many products, that often lead numerically to 0 or 1. We overcome such problems by using some logarithms, which complicates and slows down the program.

Let us now briefly summarize the results of our experiments, see Section 8.

We empirically observed on various games that, very roughly, each iteration of our algorithm requires between 2 and 4 times more computational time than MCTS.

When playing Pearl's games, our algorithm seems rather competitive against MCTS (with a given amount of time per move), which is not very surprising, since our algorithm is precisely designed for such games.

We also tried to play various versions of Connect Four. Globally, we are clearly beaten by MCTS. However, there are two situations where we win.

The first one is when the game is so large, or the amount of time so small, that very few iterations can be performed by the challengers. This is quite natural, because (a) our algorithm is only optimal *step by step*, (b) it relies on some independence properties that are less and less true when performing more and more iterations.

The second one is when the game is so small that we can hope to find the winning strategy at the first move, and where our algorithm finds it before MCTS. We believe this is due to the automatic pruning.

1.7. Comparison with AlphaBeta

Assume that $\mathcal{T}$ is a finite balanced tree, *i.e.* that all the nodes with the same depth have the same degree, and that we have some i.i.d. outcomes on the leaves. This slightly generalizes Pearl's game. Then Tarsi [21] showed that AlphaBeta is optimal in the sense of the expected number of leaves necessary to perfectly find $R(r)$. For such a game, Bruno Scherrer told us that our algorithm visits the leaves precisely in the same order as AlphaBeta, up to some random permutation (see also Sect. 8.11 for a rather convincing numerical indication in

this direction). The advantage is that we provide an estimated value during the whole process, while AlphaBeta produces nothing before it really finds $R(r)$.

This strict similarity with AlphaBeta does not hold generally, because our algorithm takes into account the degrees of the nodes. On the one hand, a player is happy to find a node with more possible moves than expected. On the other hand, the value of such a node may be difficult to determine. So the way our algorithm takes degrees into account is complicated and not very transparent.

1.8. Organization of the paper

In the next section, we precisely state our main results and describe our algorithm. Section 3 is devoted to the convergence proof. In Sections 4 and 5, we establish our main result. Section 6 is devoted to the computation of the functions m and s for particular models. In Section 7, we show on an example that global optimality fails. We present numerical experiments in Section 8. In Appendix A, we precisely describe the versions of MCTS and its variant we used to test our algorithms.

2. Notation and main results

2.1. Notation

We first introduce once for all the whole notation we need concerning trees.

Let $\mathbb{T}$ be the complete discrete ordered tree with root r and infinite degree. An element of $\mathbb{T}$ is a finite word composed of letters in $\mathbb{N}_*$. The root r is the empty word. If *e.g.* $x = n_1n_2n_3$, this means that x is the n_3th child of the n_2th child of the n_1th child of the root. We consider *ordered trees* to simplify the presentation, but the order will not play any role.

The depth (or generation) $|x|$ of $x \in \mathbb{T}$ is the number of letters of x. In particular, $|r| = 0$.

We say that $y \in \mathbb{T}$ is the father of $x \in \mathbb{T} \setminus \{r\}$ (or that x is a child of y) if there is $n \in \mathbb{N}_*$ such that $x = yn$. We denote by $f(x)$ the father of x.

For $x \in \mathbb{T}$, we call $\mathbb{C}_x = \{y \in \mathbb{T} : f(y) = x\}$ the set of all the children of x and $\mathbb{T}_x$ the whole progeny of x: $\mathbb{T}_x$ is the subtree of $\mathbb{T}$ composed of x, its children, its grandchildren, etc.

We say that $x, y \in \mathbb{T}$ are brothers if they are different and have the same father. For $x \in \mathbb{T} \setminus \{r\}$, we denote by $\mathbb{H}_x = \mathbb{C}_{f(x)} \setminus \{x\}$ the set of all the brothers of x. Of course, $\mathbb{H}_r = \emptyset$.

For $x \in \mathbb{T}$ and $y \in \mathbb{T}_x$, we denote by B_{xy} is the branch from x to y. In other words, $z \in B_{xy}$ if and only if $z \in \mathbb{T}_x$ and $y \in \mathbb{T}_z$.

For $x \in \mathbb{T}$, we introduce $\mathbb{K}_x = \{r\} \cup \bigcup_{y \in B_{rx} \setminus \{x\}} \mathbb{C}_y = \cup_{y \in B_{rx}} (\{y\} \cup \mathbb{H}_y)$, which consists of x, its brothers, its father and uncles, its grandfather and granduncles, etc.

For $\mathbf{x} \subset \mathbb{T}$, we introduce $B_{\mathbf{x}} = \cup_{x \in \mathbf{x}} B_{rx}$, the finite subtree of $\mathbb{T}$ with root r and set of leaves $\mathbf{x}$.

For $\mathbf{x} \subset \mathbb{T}$, we also set $\mathbb{D}_{\mathbf{x}} = (\cup_{x \in B_{\mathbf{x}}} \mathbb{H}_x) \setminus B_{\mathbf{x}}$, the set of all the brothers of the elements of $B_{\mathbf{x}}$ that do not belong to $B_{\mathbf{x}}$. Observe that $B_{\mathbf{x}} \cup \mathbb{D}_{\mathbf{x}} = \cup_{x \in \mathbf{x}} \mathbb{K}_x$.

Let $\mathcal{S}_f$ be the set of all finite subtrees of $\mathbb{T}$ with root r. For T a finite subset of $\mathbb{T}$, it holds that $T \in \mathcal{S}_f$ if and only if for all $x \in T$, $B_{rx} \subset T$.

For $T \in \mathcal{S}_f$ and $x \in T$, we introduce $C_x^T = T \cap \mathbb{C}_x$ the set of the children of x in T, $H_x^T = T \cap \mathbb{H}_x$ the set of the brothers of x in T, $T_x = T \cap \mathbb{T}_x$ the whole progeny of x in T, and $K_x^T = T \cap \mathbb{K}_x$ which contains x, its brothers (in T), its father and uncles (in T), its grandfather and granduncles (in T), etc. See Figure 1.

We denote by $L_T = \{x \in T : T_x = \{x\}\}$ the set of the leaves of $T \in \mathcal{S}_f$. We have $B_{L_T} = T$.

Finally, for $T \in \mathcal{S}_f$ and $\mathbf{x} \subset T$, we introduce $D_{\mathbf{x}}^T = T \cap \mathbb{D}_{\mathbf{x}}$, the set of all the brothers (in T) of the elements of $B_{\mathbf{x}}$ not belonging to $B_{\mathbf{x}}$ (observe that $B_{\mathbf{x}} \subset T$), see Figure 2. It holds that $B_{\mathbf{x}} \cup D_{\mathbf{x}}^T = \cup_{x \in \mathbf{x}} K_x^T$.

2.2. The general model

We have two players J_0 and J_1. The game is modeled by a finite tree $\mathcal{T} \in \mathcal{S}_f$ with root r and leaves $\mathcal{L} = L_{\mathcal{T}}$. An element $x \in \mathcal{T}$ represents a configuration of the game. On each node $x \in \mathcal{T} \setminus \mathcal{L}$, we set $t(x) = 1$ if it is J_1's turn to play when in the configuration x and $t(x) = 0$ else. The move is alternate and the player J_1 starts, so

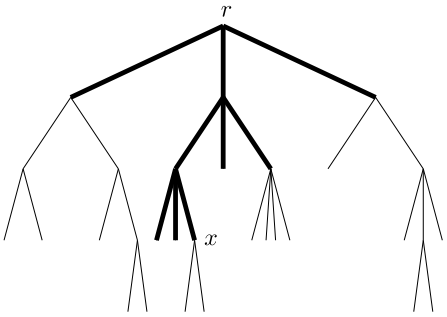

FIGURE 1. The subtree K_x^T of T is thick.

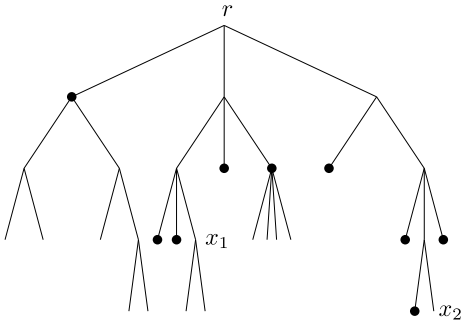

FIGURE 2. $D^T_{\{x_1,x_2\}}$ consists of the bullets.

that we have $t(x) = \mathbf{1}_{\{|x| \text{ is even}\}}$, where $|x|$ is the depth of x. We have some outcomes $(R(x))_{x\in\mathcal{L}}$ in $\{0,1\}$. We say that $x \in \mathcal{L}$ is a winning outcome for J_1 (resp. J_0) if $R(x) = 1$ (resp. $R(x) = 0$).

So J_1 starts, he chooses a node x_1 among the children of r, then J_0 chooses a node x_2 among the children of x_1, and so on, until we reach a leave $y \in \mathcal{L}$, and J_1 is the winner if $R(y) = 1$, while J_0 is the winner if $R(y) = 0$.

2.3. Notation

For $x \in \mathcal{T}$, we set $\mathcal{C}_x = C_x^{\mathcal{T}}$, $\mathcal{H}_x = H_x^{\mathcal{T}}$ and $\mathcal{K}_x = K_x^{\mathcal{T}}$ and, for $\mathbf{x} \subset \mathcal{T}$, $\mathcal{D}_{\mathbf{x}} = D_{\mathbf{x}}^{\mathcal{T}}$.

2.4. Minimax values

Given the whole tree $\mathcal{T}$ and the outcomes $(R(x))_{x\in\mathcal{L}}$, we can theoretically completely solve the game. We classically define the minimax values $(R(x))_{x\in\mathcal{T}}$ as follows. For any $x \in \mathcal{T}$, $R(x) = 1$ if J_1 has a winning strategy when starting from x and $R(x) = 0$ else (in which case J_0 necessarily has a winning strategy when starting from x).

Remark 2.1. For $x \in \mathcal{L}$, the value of $R(x)$ is prescribed. For $x \in \mathcal{T} \setminus \mathcal{L}$,

$$R(x) = \mathbf{1}_{\{t(x)=0\}} \min\{R(y) : y \in \mathcal{C}_x\} + \mathbf{1}_{\{t(x)=1\}} \max\{R(y) : y \in \mathcal{C}_x\}. \tag{2.1}$$

It is thus possible to compute $R(x)$ for all $x \in \mathcal{T}$ by backward induction, starting from the leaves.

This is easily checked: for $x \in \mathcal{T}$ with $t(x) = 0$, we have $R(x) = 0$ if x has at least one child $y \in \mathcal{T}$ such that $R(y) = 0$ (because J_0 can choose y from which J_1 has no winning strategy) and $R(x) = 1$ else (because any choice of J_0 leads to a position y from which J_1 has a winning strategy). This rewrites $R(x) = \min\{R(y) : y \in \mathcal{C}_x\}$.

If now $t(x) = 1$, then $R(x) = 1$ if x has at least one child $y \in \mathcal{T}$ such that $R(y) = 1$ (because J_1 can choose y, from where it has a winning strategy) and $R(x) = 0$ else (because any choice of J_1 leads to a position from which he has no winning strategy). This can be rewritten as $R(x) = \max\{R(y) : y \in \mathcal{C}_x\}$.

2.5. The goal

Our goal is to estimate at best $R(r)$ with a computer and a given amount of time.

In practice, we (say, J_1) are playing at some true game such as *Connect Four* or any deterministic game with alternate moves, against a true opponent (say, J_0). As already mentioned in the introduction, we may always consider that such a game is represented by a tree, and we may remove draws by identifying them to victories of J_0 (or of J_1).

We are in some given configuration (after a certain number of true moves of both players). We call this configuration r. We have to decide between several possibilities. We thus want to estimate at best from which of these possibilities there is a winning strategy for J_1. In other words, given a position r (which will be the root of our tree), we want to know at best $R(r) = \max\{R(y) : y \in \mathcal{C}_r\}$: if our estimate suggests that $R(r) = 0$, then any move is similarly desperate. If our estimate suggests that $R(r) = 1$, this necessarily relies on the fact that we think that some (identified) child y_0 of r satisfies $R(y_0) = 1$. Then in the true game, we will play y_0.

Of course, except for very *small* games, it is not possible in practice to compute $R(r)$ as in Remark 2.1, because the tree is too large.

The computer knows nothing about the true game except the rules: when it sees a position (node) $x \in \mathcal{T}$, it is able to decide if x is terminal position (*i.e.* $x \in \mathcal{L}$) or not; if x is a terminal position, it knows the outcome (*i.e.* $R(x)$); if x is not a terminal position, it knows who's turn it is to play (*i.e.* $t(x)$) and the possible moves (*i.e.* $\mathcal{C}_x$).

The true game is deterministic and our study does not apply *at all* to games of chance such as *Backgammon* (because games of chance are more difficult to represent as trees, their minimax values or not clearly well-defined, etc.). However, it is a very large game and, in some sense, unknown, so one might hope it resembles the realization of a random model. We will thus assume that $\mathcal{T}$, as well as the outcomes $(R(x))_{x\in\mathcal{L}}$, are given by the realization of some random model satisfying some independence properties. It is not clear whether such an assumption is reasonable. In any case, our theoretical results completely break down without such a condition.

2.6. A class of algorithms

We consider a large class of algorithms resembling the Monte Carlo Tree Search algorithm, of which a version is recalled in details in Appendix A. The idea is to make J_1 play against J_0 a large number of times: the first match is completely random, but then we use the statistics of the preceding matches. MCTS makes J_1 and J_0 rather play some moves that often led them to victories. At the end, this provides some ratings for the children of r. In the true game, against the true opponent, we then play the move leading to the child of r with the highest rating.

Definition 2.2. We call a *uniformly random match* starting from $x \in \mathcal{T}$, with $y \in \mathcal{L}$ as a *resulting leave*, the following procedure. Put $y_0 = x$. If $y_0 \in \mathcal{L}$, set $y = y_0$. Else, choose y_1 uniformly among $\mathcal{C}_{y_0}$. If $y_1 \in \mathcal{L}$, set $y = y_1$. Else, choose y_2 uniformly among $\mathcal{C}_{y_1}$. If $y_2 \in \mathcal{L}$, set $y = y_2$. Etc. Since $\mathcal{T}$ is finite, this always ends up.

The class of algorithms we consider is the following.

Definition 2.3. An *admissible algorithm* is a procedure of the following form.

Step 1. Simulate a uniformly random match from r, call x_1 the resulting leave and set $\mathbf{x}_1 = \{x_1\}$. Keep track of $R(x_1)$, of $B_{\mathbf{x}_1} = B_{rx_1}$ and of $\mathcal{D}_{\mathbf{x}_1} = \cup_{y\in B_{rx_1}} \mathcal{H}_y$.

Step n+1. Using only the knowledge of $B_{\mathbf{x}_n}$, $\mathcal{D}_{\mathbf{x}_n}$ and $(R(x))_{x\in\mathbf{x}_n}$, choose some (possibly randomized) $z_n \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$.

If $z_n \in \mathbf{x}_n$, set $x_{n+1} = z_n$, $\mathbf{x}_{n+1} = \mathbf{x}_n$, $B_{\mathbf{x}_{n+1}} = B_{\mathbf{x}_n}$ and $\mathcal{D}_{\mathbf{x}_{n+1}} = \mathcal{D}_{\mathbf{x}_n}$.

If $z_n \in \mathcal{D}_{\mathbf{x}_n}$, simulate a uniformly random match starting from z_n and call x_{n+1} the resulting leave. Set $\mathbf{x}_{n+1} = \mathbf{x}_n \cup \{x_{n+1}\}$ and keep track of $R(x_{n+1})$, of $B_{\mathbf{x}_{n+1}} = B_{\mathbf{x}_n} \cup B_{rx_{n+1}}$ and of $\mathcal{D}_{\mathbf{x}_{n+1}} = (\mathcal{D}_{\mathbf{x}_n} \setminus \{z_n\}) \cup \bigcup_{y \in B_{z_n x_{n+1}} \setminus \{z_n\}} \mathcal{H}_y$.

Conclusion. Stop after a given number of iterations n_0 (or after a given amount of time). Choose some (possibly randomized) *best child* x_* of r, using only the knowledge of $B_{\mathbf{x}_{n_0}}$, $\mathcal{D}_{\mathbf{x}_{n_0}}$ and $(R(x))_{x \in \mathbf{x}_{n_0}}$.

After n matches, $B_{\mathbf{x}_n}$ represents the explored part of $\mathcal{T}$, while $\mathcal{D}_{\mathbf{x}_n}$ represents its boundary.

Note that we assume we that know $\mathcal{D}_{\mathbf{x}_n}$ (the set of all the brothers, in $\mathcal{T}$, of the elements of $B_{\mathbf{x}_n}$ that are not in $B_{\mathbf{x}_n}$) after the simulation some matches leading to the set of leaves $\mathbf{x}_n$. This is motivated by the following reason. Any $y \in \mathcal{D}_{\mathbf{x}_n}$ has its father in $B_{\mathbf{x}_n}$. Thus at some point of the simulation, we visited $f(y)$ for the first time and we had to decide (at random) between all its children: we are aware of the fact that $y \in \mathcal{T}$.

Also note that we assume that the best thing to do, when visiting a position for the first time (*i.e.* when arriving at some element of $\mathcal{D}_{\mathbf{x}_n}$), is to simulate from there a uniformly random match. This models that fact that we know nothing of the game, except the rules.

The randomization will allow us, in practice, to make some *uniform* choice in case of equality.

Finally, it seems stupid to allow x_{n+1} to belong to $\mathbf{x}_n$, because this means we will simulate *precisely* a match we have already simulated: this will not give us some new information. But this avoids many useless discussions. Anyway, a *good* algorithm will always, or almost always, exclude such a possibility.

Remark 2.4.

(i) In *Step n+1*, by *"using only the knowledge of $B_{\mathbf{x}_n}$, $\mathcal{D}_{\mathbf{x}_n}$ and $(R(x))_{x \in \mathbf{x}_n}$ choose some (possibly randomized) $z_n \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$"*, we mean that $z_n = F(B_{\mathbf{x}_n}, \mathcal{D}_{\mathbf{x}_n}, (R(x))_{x \in \mathbf{x}_n}, X_n)$, where $X_n \sim \mathcal{U}([0,1])$ is independent of everything else and where F is a deterministic measurable application from A to $\mathbb{T}$, where A is the set of all $w = (B, D, (\rho(x))_{x \in \mathbf{x}}, u)$, with $B \in \mathcal{S}_f$, with $\mathbf{x} = L_B$, with $D \subset \mathbb{D}_{\mathbf{x}}$ finite, with $(\rho(x))_{x \in \mathbf{x}} \in \{0,1\}^{\mathbf{x}}$, and with $u \in [0,1]$, such that $F(w) \in \mathbf{x} \cup D$.

(ii) In *Conclusion*, by *"choose some (possibly randomized)* best child x_* *of r, using only the knowledge of $B_{\mathbf{x}_{n_0}}, \mathcal{D}_{\mathbf{x}_{n_0}}, (R(x))_{x \in \mathbf{x}_{n_0}}$"*, we mean that $x_* = G(B_{\mathbf{x}_{n_0}}, \mathcal{D}_{\mathbf{x}_{n_0}}, (R(x))_{x \in \mathbf{x}_{n_0}}, X_{n_0})$, where $X_{n_0} \sim \mathcal{U}([0,1])$ is independent of everything else and where G is a deterministic application from A to $\mathbb{T}$ such that, for $w \in A$ as above, $G(w) \in C_r^{B \cup D}$.

(iii) The two applications F, G completely characterize an admissible algorithm.

2.7. Assumption

Except for the convergence of our class of algorithms, the proof of which being purely deterministic, we will suppose at least the following condition.

Assumption 2.5. The tree $\mathcal{T}$ is a random element of $\mathcal{S}_f$. We denote by $\mathcal{L} = L_{\mathcal{T}}$ and, as already mentioned, we set $\mathcal{C}_x = C_x^{\mathcal{T}}$, $\mathcal{H}_x = H_x^{\mathcal{T}}$ and $\mathcal{K}_x = K_x^{\mathcal{T}}$ for $x \in \mathcal{T}$ and $\mathcal{D}_{\mathbf{x}} = D_{\mathbf{x}}^{\mathcal{T}}$ for $\mathbf{x} \subset \mathcal{T}$. Conditionally on $\mathcal{T}$, we have some random outcomes $(R(x))_{x \in \mathcal{L}}$ in $\{0,1\}$. We assume that for any $T \in \mathcal{S}_f$ with leaves L_T, the family

$$((\mathcal{T}_x, (R(y))_{y \in \mathcal{L} \cap \mathcal{T}_x}), x \in L_T)$$

is independent conditionally on $A_T = \{T \subset \mathcal{T} \text{ and } \mathcal{D}_{L_T} = \emptyset\}$ as soon as $\Pr(A_T) > 0$.

Observe that $A_T = \{T \subset \mathcal{T} \text{ and } x \in T \text{ implies } \mathcal{H}_x \subset T\} = \{T \subset \mathcal{T} \text{ and } \forall\, x \in T,\ \mathcal{K}_x = K_x^T\}$.

This condition is a branching property: knowing A_T, *i.e.* knowing that $T \subset \mathcal{T}$ and that all the brothers (in $\mathcal{T}$) of $x \in T$ belong to T, we can write $\mathcal{T} = T \cup \bigcup_{x \in L_T} \mathcal{T}_x$, and the family $((\mathcal{T}_x, (R(y))_{y \in \mathcal{L} \cap \mathcal{T}_x}), x \in L_T)$ is independent. A first consequence is as follows.

Remark 2.6. Suppose Assumption 2.5. For $T \in \mathcal{S}_f$ such that $\Pr(A_T) > 0$ and $z \in L_T$, we denote by $G_{T,z}$ the law of $(\mathcal{T}_z, (R(y))_{y \in \mathcal{L} \cap \mathcal{T}_z})$ conditionally on A_T. We have $G_{T,z} = G_{K_z^T, z}$.

Indeed, put $S = K_z^T \subset T$ and observe that $A_T = A_S \cap \bigcap_{x \in L_S \setminus \{z\}} A'_x$, where we have set $A'_x = \{T_x \subset \mathcal{T}_x, \mathcal{D}_{L_T} \cap \mathbb{T}_x = \emptyset\}$. But for each $x \in L_S \setminus \{z\}$, $A'_x \in \sigma(\mathcal{T}_x)$. It thus follows from Assumption 2.5 that $(\mathcal{T}_z, (R(y))_{y \in \mathcal{L} \cap \mathcal{T}_z})$ is independent of $\bigcap_{x \in L_S \setminus \{z\}} A'_x$ knowing A_S. Hence its law knowing A_T is the same as knowing A_S.

Assumption 2.5 is of course satisfied if $\mathcal{T}$ is deterministic and if the family $(R(x))_{x \in \mathcal{L}}$ is independent. It also holds true if $\mathcal{T}$ is an inhomogeneous Galton–Watson tree and if, conditionally on $\mathcal{T}$, the family $(R(x))_{x \in \mathcal{L}}$ is independent and (for example) $R(x)$ is Bernoulli with some parameter depending only on the depth $|x|$. But there are many other possibilities, see Section 6 for precise examples of models satisfying Assumption 2.5.

2.8. Two relevant quantities

Here we introduce two mean quantities necessary to our study.

Definition 2.7. Suppose Assumption 2.5. Let $T \in \mathcal{S}_f$ such that $\Pr(A_T) > 0$, and $z \in L_T$. Observe that on A_T, $z \in \mathcal{T}$.

(i) We set $m(T, z) = \Pr(R(z) = 1 | A_T)$. By Remark 2.6, $m(T, z) = m(K_z^T, z)$, because $R(z)$ is of course a deterministic function of $(\mathcal{T}_z, (R(y))_{y \in \mathcal{L} \cap \mathcal{T}_z})$, see Remark 2.1.

(ii) Simulate a uniformly random match starting from z, denote by y the resulting leave. We put $\mathcal{K}_{zy} = \mathcal{K}_y \cap \mathbb{T}_z$ and introduce $\mathcal{G} = \sigma(y, \mathcal{K}_{zy}, R(y))$. We set

$$s(T, z) = \mathbb{E}\Big[\Big(\Pr(R(z) = 1 | \mathcal{G} \vee \sigma(A_T)) - m(T, z)\Big)^2 \Big| A_T\Big].$$

By Remark 2.6, $s(T, z) = s(K_z^T, z)$.

Since $\mathbb{E}[\Pr(R(z) = 1 | \mathcal{G} \vee \sigma(A_T)) | A_T] = m(T, z)$, $s(T, z)$ is a conditional variance.

We will see in Section 6 that for some particular classes of models, m and s can be computed.

Recall that our goal is to produce some admissible algorithm. Assume we have explored n leaves $x_1, \dots, x_n$ and set $\mathbf{x}_n = \{x_1, \dots, x_n\}$. Recall that $B_{\mathbf{x}_n}$ is the explored tree and that $\mathcal{D}_{\mathbf{x}_n}$ is, in some sense, its boundary. For $z \in \mathbf{x}_n$, we perfectly know $R(z)$. But for $z \in \mathcal{D}_{\mathbf{x}_n}$, we only know that $z \in \mathcal{T}$ and we precisely know $\mathcal{K}_z$, since $\mathcal{K}_z = K_z^{\mathcal{T}} = K_z^{B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}}$. Thus the best thing we can say is that $R(z)$ equals 1 with (conditional) probability $m(\mathcal{K}_z, z)$. Also, $s(\mathcal{K}_z, z)$ quantifies some mean amount of information we will get if handling a uniformly random match starting from z.

2.9. The conditional minimax values

From now on, we work with the following setting.

Fix $n \geq 1$. Using an *admissible* algorithm, we have *simulated* n matches, leading to the leaves $\mathbf{x}_n = \{x_1, \dots, x_n\} \subset \mathcal{L}$. Hence the σ-field $\mathcal{F}_n = \sigma(\mathbf{x}_n, \mathcal{D}_{\mathbf{x}_n}, (R(x))_{x \in \mathbf{x}_n})$ represents our knowledge of the game. Observe that $B_{\mathbf{x}_n}$ is of course $\mathcal{F}_n$-measurable. Also, for any $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, $\mathcal{K}_x$ and $\mathcal{H}_x$ are $\mathcal{F}_n$-measurable, as well as $\mathcal{C}_x$ if $x \in B_{\mathbf{x}_n} \setminus \{\mathbf{x}_n\}$.

This last assertion easily follows from the fact that for any $\mathbf{x} \subset \mathcal{T}$, any element of $B_{\mathbf{x}} \cup \mathcal{D}_{\mathbf{x}}$ has all its brothers (in $\mathcal{T}$) in $B_{\mathbf{x}} \cup \mathcal{D}_{\mathbf{x}}$.

We first want to compute $R_n(r) = \mathbb{E}[R(r) | \mathcal{F}_n] = \Pr(R(r) = 1 | \mathcal{F}_n)$, which is, in some obvious sense, the best approximation of $R(r)$ knowing $\mathcal{F}_n$. Of course, we will have to compute $R_n(x)$ on the whole explored subtree of $\mathcal{T}$. We will check the following result in Section 5.

Proposition 2.8. *Grant Assumption 2.5 and Setting 2.9. For all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, define $R_n(x) = \Pr(R(x) = 1 | \mathcal{F}_n)$. They can be computed by backward induction, starting from $\mathbf{x}_n \cup \mathcal{D}_{\mathbf{x}_n}$, as follows. For all $x \in \mathbf{x}_n$, $R_n(x) =$*

$R(x)$. For all $x \in \mathcal{D}_{\mathbf{x}_n}$, $R_n(x) = m(\mathcal{K}_x, x)$. For all $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$,

$$R_n(x) = \mathbf{1}_{\{t(x)=0\}} \prod_{y \in \mathcal{C}_x} R_n(y) + \mathbf{1}_{\{t(x)=1\}} \Big(1 - \prod_{y \in \mathcal{C}_x} (1 - R_n(y))\Big). \tag{2.2}$$

2.10. Main result

We still work under Setting 2.9. We want to simulate a $(n+1)$th match. We thus want to choose some $z \in \mathcal{D}_{\mathbf{x}_n}$ and then simulate a uniformly random match starting from z, in such a way that the increase of information concerning $R(r)$ is as large as possible. We unfortunately need a few more notation.

Notation 2.1. *Adopt Setting 2.9.*

(i) For $x \in (B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}) \setminus \{r\}$, we set

$$U_n(x) = \mathbf{1}_{\{t(f(x))=0\}} \prod_{y \in \mathcal{H}_x} R_n(y) + \mathbf{1}_{\{t(f(x))=1\}} \prod_{y \in \mathcal{H}_x} (1 - R_n(y)). \tag{2.3}$$

(ii) Define $Z_n(x)$, for all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, by backward induction, starting from $\mathbf{x}_n \cup \mathcal{D}_{\mathbf{x}_n}$, as follows. If $x \in \mathbf{x}_n$, set $Z_n(x) = 0$. If $x \in \mathcal{D}_{\mathbf{x}_n}$, set $Z_n(x) = s(\mathcal{K}_x, x)$. If $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$, set

$$Z_n(x) = \max\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_x\}. \tag{2.4}$$

(iii) Fix $z \in \mathcal{D}_{\mathbf{x}_n}$, handle a uniformly random match starting from z, with resulting leave y_z, set $\mathbf{x}^z_{n+1} = \mathbf{x}_n \cup \{y_z\}$ and denote by $\mathcal{F}^z_{n+1} = \sigma(\mathbf{x}^z_{n+1}, \mathcal{D}_{\mathbf{x}^z_{n+1}}, (R(x))_{\mathbf{x}^z_{n+1}})$ the resulting knowledge. Set $R^z_{n+1}(x) = \Pr(R(x) = 1|\mathcal{F}^z_{n+1})$ for all $x \in B_{\mathbf{x}^z_{n+1}} \cup \mathcal{D}_{\mathbf{x}^z_{n+1}}$.

Our main result reads as follows.

Theorem 2.9. *Suppose Assumption 2.5 and adopt Setting 2.9 and Notation 2.1. Define $z_* \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$ as follows. Put $y_0 = r$ and set $y_1 = \operatorname{argmax}\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_{y_0}\}$. If $y_1 \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$, set $z_* = y_1$. Else, put $y_2 = \operatorname{argmax}\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_{y_1}\}$. If $y_2 \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$, set $z_* = y_2$. Else, put $y_3 = \operatorname{argmax}\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_{y_2}\}$, etc. This procedure necessarily stops since $\mathcal{T}$ is finite. Each time we use* argmax*, we choose* e.g. *at uniform random in case of equality.*

On the event $\{R_n(r) \notin \{0,1\}\}$, we have $z_* \in \mathcal{D}_{\mathbf{x}_n}$ and

$$z_* = \operatorname{argmin}\Big\{\mathbb{E}\Big[(R^z_{n+1}(r) - R(r))^2 \Big| \mathcal{F}_n\Big] : z \in \mathcal{D}_{\mathbf{x}_n}\Big\}. \tag{2.5}$$

Observe that if $R_n(r) \in \{0,1\}$, then $R(r) = R_n(r)$, because conditionally on $\mathcal{F}_n$, $R(x)$ is Bernoulli with parameter $R_n(r) = \Pr(R(r) = 1|\mathcal{F}_n)$. Hence on the event $\{R_n(r) \in \{0,1\}\}$, we perfectly know $R(r)$ from $\mathcal{F}_n$ and thus the $(n+1)$th match is useless.

When $R_n(r) \notin \{0,1\}$, we have the knowledge $\mathcal{F}_n$, and the theorem tells us how to choose $z_* \in \mathcal{D}_{\mathbf{x}_n}$ such that, after a uniformly random match starting from z_*, we will estimate at best, in some L^2-sense, $R(r)$. In words, z_* can be found by starting from the root, getting down in the tree $B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$ by following the maximum values of $U_n^2 Z_n$, until we arrive in $\mathcal{D}_{\mathbf{x}_n}$.

As noted by Bruno Scherrer, z_* is also optimal if using a L^1-criterion.

Remark 2.10. With the assumptions and notation of Theorem 2.9, it also holds that

$$z_* = \operatorname{argmin}\Big\{\mathbb{E}\Big[|R^z_{n+1}(r) - R(r)| \Big| \mathcal{F}_n\Big] : z \in \mathcal{D}_{\mathbf{x}_n}\Big\}. \tag{2.6}$$

This is easily deduced from (2.5), noting that conditionally on $\mathcal{F}^z_{n+1}$ (which contains $\mathcal{F}_n$), $R(r)$ is Bernoulli($R^z_{n+1}(r)$)-distributed, and that for $X \sim$Bernoulli(p), $\mathbb{E}[|X-p|] = 2\mathbb{E}[(X-p)^2]$.

2.11. The algorithm

The resulting algorithm is as follows.

Algorithm 2.11. Each time we use argmax, we *e.g.* choose at uniform random in case of equality.

Step 1. Simulate a uniformly random match from r, call x_1 the resulting leave and set $\mathbf{x}_1 = \{x_1\}$.

During this random match, keep track of $R(x_1)$, of $B_{\mathbf{x}_1} = B_{rx_1}$ and of $\mathcal{D}_{\mathbf{x}_1} = \cup_{y\in B_{rx_1}} \mathcal{H}_y$ and set $R_1(x) = m(\mathcal{K}_x, x)$ and $Z_1(x) = s(\mathcal{K}_x, x)$ for all $x \in \mathcal{D}_{\mathbf{x}_1}$.

Set $x = x_1$, $R_1(x) = R(x_1)$ and $Z_1(x) = 0$.

Do $\{x{=}f(x)$, compute $R_1(x)$ using (2.2), $(U_1(y))_{y\in\mathcal{C}_x}$ using (2.3) and $Z_1(x)$ using (2.4)$\}$ until $x = r$.

Step n+1. Put $z{=}r$. Do $z = \text{argmax}\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_z\}$ until $z \in \mathcal{D}_{\mathbf{x}_n}$. Set $z_n = z$.

Simulate a uniformly random match from z_n, call x_{n+1} the resulting leave, set $\mathbf{x}_{n+1} = \mathbf{x}_n \cup \{x_{n+1}\}$.

During this random match, keep track of $R(x_{n+1})$, of $B_{\mathbf{x}_{n+1}} = B_{\mathbf{x}_n} \cup B_{rx_{n+1}}$ and of $\mathcal{D}_{\mathbf{x}_{n+1}} = (\mathcal{D}_{\mathbf{x}_n} \setminus \{z_n\}) \cup \bigcup_{y\in B_{z_n x_{n+1}}\setminus\{z_n\}} \mathcal{H}_y$ and set $R_{n+1}(x) = m(\mathcal{K}_x, x)$ and $Z_{n+1}(x) = s(\mathcal{K}_x, x)$ for all $x \in \bigcup_{y\in B_{z_n x_{n+1}}\setminus\{z_n\}} \mathcal{H}_y$.

For all $x \in (B_{\mathbf{x}_{n+1}} \cup \mathcal{D}_{\mathbf{x}_{n+1}}) \setminus B_{rx_{n+1}}$, put $R_{n+1}(x) = R_n(x)$ and $Z_{n+1}(x) = Z_n(x)$.

For all $x \in (B_{\mathbf{x}_{n+1}} \cup \mathcal{D}_{\mathbf{x}_{n+1}}) \setminus \mathcal{K}_{x_{n+1}}$, put $U_{n+1}(x) = U_n(x)$.

Set $x = x_{n+1}$, put $R_{n+1}(x) = R(x_{n+1})$ and $Z_{n+1}(x) = 0$.

Do $\{x = f(x)$, compute $R_{n+1}(x)$ using (2.2), compute $(U_{n+1}(y))_{y\in\mathcal{C}_x}$ using (2.3) and $Z_{n+1}(x)$ using (2.4)$\}$ until $x = r$.

If $R_{n+1}(r) \in \{0, 1\}$, go directly to the conclusion.

Conclusion. Stop after a given number of iterations n_0 (or after a given amount of time). As *best child* of r, choose $x_* = \text{argmax}\{R_{n_0}(x) : x \in \mathcal{C}_r\}$.

2.12. The update is rather quick

For *e.g.* a (deterministic) regular tree $\mathcal{T}$ with degree d and depth K, the cost to achieve n steps of the above algorithm is of order nKd, because at each step, we have to update the values of $R_n(x)$ and $Z_n(x)$ for $x \in B_{rx_n}$ (which concerns K nodes) and the values of $U_n(x)$ for $x \in \mathcal{K}_{x_{n+1}}$ (which concerns Kd nodes).

Observe that MCTS algorithms (see Sects. 1 and A.2) enjoy a cost of order Kn, since the updates are done only on the branch B_{rx_n} (or even less than that, but in any case we have at least to simulate a random match, of which the cost is proportional to K, at each step).

The cost in Kdn for Algorithm 2.11 seems miraculous. We have not found any deep reason, but calculus, explaining why this *theoretically optimal* (in a loose sense) behaves so well. It would have been more natural, see Remark 5.2, that the update would concern the whole explored tree $B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, which contains much more than Kd elements. Very roughly, its cardinal is of order Kdn, which would lead to a cost of order Kdn^2 to achieve n steps (Fig. 3). We did not write it down in the present paper, which is technical enough, but we also studied two variations of Theorem 2.9:

- First, we considered the very same model, but we tried minimize $\Pr(\mathbf{1}_{\{R^z_{n+1}(r)\}>1/2} \neq R(r)|\mathcal{F}_n)$ instead of (2.5). This is more natural, since in practice, one would rather estimate $R(r)$ by $\mathbf{1}_{\{R_n(r)>1/2\}}$ than by $R_n(r)$ (because $R(r)$ takes values in $\{0, 1\}$). It is possible to extend our theory, but this leads to an algorithm with a cost of order Kdn^2 (at least, we found no way to reduce this cost).
- We also studied what happens in the case where the game may lead to draw. Then the outcomes $(A(x))_{x\in\mathcal{T}}$ can take three values, 0 (if J_0 wins), 1 (if J_1 wins) and 1/2 (if the issue is a draw). For any $x \in \mathcal{T}$, we can define the minimax rating $A(x)$ as 0 (if J_0 has a winning strategy), 1 (if J_1 has a winning strategy) and 1/2 (else). The family $(A(x))_{x\in\mathcal{T}}$ satisfies the backward induction relation (2.1). A possibility is to identify a draw to a victory of J_0 (or of J_1). Then, under Assumption 2.5 with $R(x) = \mathbf{1}_{\{A(x)=1\}}$, we can

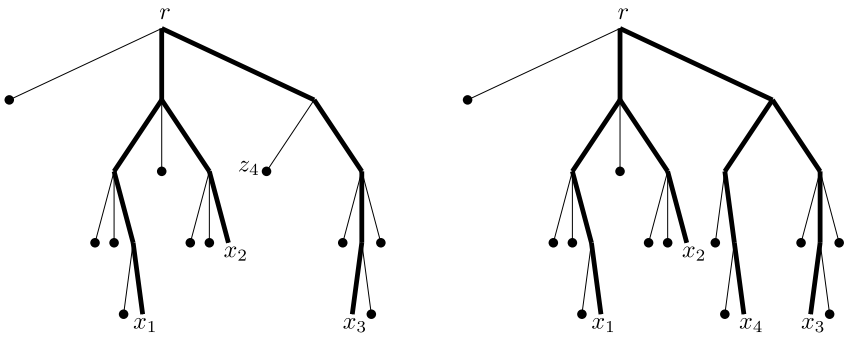

FIGURE 3. After Step 3, we have the (thick) explored tree $B_{\mathbf{x}_3}$, its boundary (bullets) $\mathcal{D}_{\mathbf{x}_3}$, and the values of R_3, Z_3 and U_3 on the whole picture. 1. Select z_4 using the (U_3, Z_3)'s. 2. Simulate a uniformly random match from z_3,leading to a leave x_4. This builds a new thick branch and new bullets. 3. Observe $R(x_4)$ and compute R_4, Z_4, U_4 on $\mathcal{K}_{x_4}$. Everywhere else, set $(R_4, Z_4, U_4) = (R_3, Z_3, U_3)$.

apply directly Theorem 2.9. However, this leads to an algorithm that tries to find a winning move, but gives up if it thinks it cannot win: the algorithm does not make any difference between a loss and a draw. It is possible to adapt our theory to overcome this default by trying to estimate both $R(x) = \mathbf{1}_{\{A(x)=1\}}$ and $S(x) = \mathbf{1}_{\{A(x)=0\}}$, *i.e.* to minimize something like $\mathbb{E}[a(R^z_{n+1}(r) - R(r))^2 + b(S^z_{n+1}(r) - S(r))^2 | \mathcal{F}_n]$ for some $a, b \geq 0$. However, this leads, again, to an algorithm of which the cost is of order Kdn^2, unless $b = 0$ (or $a = 0$), which means that we identify a draw to a victory of J_0 (or of J_1). Technically, this comes from the fact that in such a framework, nothing like Observation (5.1) does occur, see also Remark 5.2.

In practice, one can produce an algorithm taking draws into account as follows: at each step, we compute $(R_n(x), Z_n(x))_{x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}}$ identifying draws to victories of J_0 and, with obvious notation, $(R'_n(x), Z'_n(x))_{x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}}$ identifying draws to victories of J_1. We use our algorithm with $(R_n(x), Z_n(x))_{x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}}$ while $R_n(r)$ is not too small, and we then switch to $(R'_n(x), Z'_n(x))_{x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}}$. We have no clear idea of how to choose the threshold.

Finally, the situation is even worse for games with a large number (possibly infinite) of game values (representing the gain of J_1). This could for example be modeled by independent Beta priors on the leaves. As first crippling difficulty, Beta laws are not stable by maximum and minimum.

2.13. Convergence

It is not difficult to check that, even with a completely wrong model, Algorithm 2.11 always converges in a finite (but likely to be very large) number of steps.

Remark 2.12.

(i) Forgetting everything about theory, we can use Algorithm 2.11 with any pair of functions m and s, both defined on $\{(S, x) : S \in \mathcal{S}_f, x \in L_S\}$, m being valued in $[0, 1]$ and s being valued in $[0, \infty)$.
(ii) For any constant $\lambda > 0$, the algorithm using the functions m and λs is precisely the same than the one using m and s.

Note that we allow s to be larger than 1, which is never the case from a theoretical point of view. But in view of (ii), it is very natural. We will prove the following result in Section 3.

Proposition 2.13. *Consider any fixed tree $\mathcal{T} \in \mathcal{S}_f$ with $\mathcal{L}$ its set of leaves and any fixed outcomes $(R(x))_{x \in \mathcal{L}}$. Denote by $(R(x))_{x \in \mathcal{T}}$ the corresponding minimax values. Apply Algorithm 2.11 with any given pair of functions m and s on $\{(S, x) : S \in \mathcal{S}_f, S \subset \mathcal{T}, x \in L_S\}$ with values in $[0, 1]$ and $[0, \infty)$ respectively and satisfying the following*

condition: for any $S \in \mathcal{S}_f$ *with* $S \subset \mathcal{T}$ *and* $\mathcal{D}_{L_S} = \emptyset$, *any* $x \in L_S$, $s(S,x) = 0$ *if and only if* $m(S,x) \in \{0,1\}$, *and in such a case,* $m(S,x) = R(x)$.

(i) For every $n \geq 1$, $R_n(r) \notin \{0,1\}$ *implies that* $x_{n+1} \notin \mathbf{x}_n$.
(ii) There is $n_0 \geq 1$ *finite such that* $R_{n_0}(r) \in \{0,1\}$ *and we then have* $R_{n_0}(r) = R(r)$.

Let us emphasize this proposition assumes nothing but the fact that $\mathcal{T}$ is finite. Assumption 2.5 is absolutely not necessary here. The condition on m and s is very general and obviously satisfied if *e.g.* m is $(0,1)$-valued and if s is $(0,\infty)$-valued.

Once a sufficiently large part of the tree is explored (actually, almost all the tree up to some pruning), the algorithm knows perfectly the minimax value of r, even if m and s are meaningless. Thus, the structure of the algorithm looks rather nice. In practice, for a large game, the algorithm will never be able to explore such a large part of the tree, so that the choice of the functions m and s is very important. However, Proposition 2.13 is reassuring: we hope that even if the modeling is approximate, so that the choices of m and s are not completely convincing, the algorithm might still behave well.

Lemma 2.14. *Under Assumption 2.5, the functions* m *and* s *introduced in Definition 2.7 satisfy the condition of Proposition 2.13: for any* $S \in \mathcal{S}_f$ *such that* $\Pr(A_S) > 0$, *any* $x \in L_S$, $s(S,x) = 0$ *if and only if* $m(S,x) \in \{0,1\}$, *and in such a case,* $R(x) = m(S,x)$ *a.s. on* A_S.

Consequently, we can apply Proposition 2.13 under Assumption 2.5 with the functions m and s introduced in Definition 2.7: for any fixed realization of $\mathcal{T}$ and $(R(x))_{x\in\mathcal{L}}$, if $S \in \mathcal{S}_f$ with $S \subset \mathcal{T}$ and $\mathcal{D}_{L_S} = \emptyset$, then A_S is realized, so that indeed, for any $x \in L_S$, $s(S,x) = 0$ if and only if $m(S,x) \in \{0,1\}$, and in such a case, $m(S,x) = R(x)$.

2.14. Practical choice of the functions m and s

In Section 6, we will describe a few models satisfying our conditions and for which we can compute the functions m ans s. As seen in Definition 2.7 (see also Algorithm 2.11), it suffices to be able to compute $m(\mathcal{K}_x, x)$ and $s(\mathcal{K}_x, x)$ for all $x \in \mathcal{T}$ (actually, for all x in the boundary $\mathcal{D}_{\mathbf{x}_n}$ of the explored tree). Let us summarize the two main examples.

(i) First, assume that $\mathcal{T}$ is an inhomogeneous Galton–Watson tree, with maximum depth K and known reproduction laws, and that conditionally on $\mathcal{T}$, the outcomes $(R(x))_{x\in\mathcal{L}}$ are independent Bernoulli random variables with parameters depending only on the depths of the involved nodes. Then we will show that the functions $m(\mathcal{K}_x, x)$ and $s(\mathcal{K}_x, x)$ depend only on the depth of x and can be computed numerically, once for all, from the parameters of the model. See Section 6.1 for precise statements. Let us mention that the parameters of the model can be fitted to some real game such as Connect Four (even if it is not clear at all that this model is reasonable) by handling a high number of uniformly random matches, see Section 8.2 for a few more explanations.

(ii) Second, assume that $\mathcal{T}$ is some given random tree to be specified later. Fix some values $a \in (0,1)$ and $b \in \mathbb{R}$. Define $m(\{r\}, r) = a$, $s(\{r\}, r) = 1$ (or any other positive constant, see Rem. 2.12) and, recursively, for all $x \in \mathcal{T}$, define $m(\mathcal{K}_x, x)$ and $s(\mathcal{K}_x, x)$ from $m(\mathcal{K}_{f(x)}, f(x))$, $s(\mathcal{K}_{f(x)}, f(x))$ and $|\mathcal{C}_{f(x)}|$ by the formulas

$$m(\mathcal{K}_x, x) = \mathbf{1}_{\{t(f(x))=0\}}[m(\mathcal{K}_{f(x)}, f(x))]^{1/|\mathcal{C}_{f(x)}|} + \mathbf{1}_{\{t(f(x))=1\}}(1 - [1 - m(\mathcal{K}_{f(x)}, f(x))]^{1/|\mathcal{C}_{f(x)}|}),$$
$$s(\mathcal{K}_x, x) = \Big(\mathbf{1}_{\{t(f(x))=0\}} m(\mathcal{K}_{f(x)}, f(x)) + \mathbf{1}_{\{t(f(x))=1\}}[1 - m(\mathcal{K}_{f(x)}, f(x))]\Big)^{b(|\mathcal{C}_{f(x)}|-1)} s(\mathcal{K}_{f(x)}, f(x)).$$

The formula for m is a modeling symmetry assumption rather well-justified and we can treat the following cases, see in Section 6.3 for more details.

(ii)-(a) If we consider Pearl's model [18], *i.e.* $\mathcal{T}$ is the deterministic d-regular tree with depth K and the outcomes are i.i.d. Bernoulli random variables with parameter p (explicit as a function of a, d, K), then the above formula for s with $b = 2$ is theoretically justified, see Remark 6.5.

(ii)-(b) Assume next that $\mathcal{T}$ is a finite homogeneous Galton–Watson tree with reproduction law $(1-p)\delta_0 + p\delta_d$ (with $pd \leq 1$) and that conditionally on $\mathcal{T}$, the outcomes $(R(x))_{x\in\mathcal{L}}$ are independent Bernoulli random variables with parameters $(m(\mathcal{K}_x, x))_{x\in\mathcal{L}}$ (that do not need to be computed). Then if $a = a_0$ is well-chosen (as a function of p and d), the above formula for s with $b = 0$ (*i.e.* $s \equiv 1$) is theoretically justified, see Remark 6.7.

We also experimented, without theoretical justification, other values of b. But we obtained so few success in this direction that we will not present the corresponding numerical results.

Let us mention that while (i) requires to fit precisely the functions m and s using rough statistics on the true game, (ii) is rather universal. In particular, it seems that the choice $a = 0.5$ and $b = 0$ works quite well in practice, and this is very satisfying. Also, the implementation is very simple, since each time a new node x is visited by the algorithm, we can compute $m(\mathcal{K}_x, x)$ from $m(\mathcal{K}_{f(x)}, f(x))$ and the number of children $|\mathcal{C}_{f(x)}|$ of $f(x)$.

Finally observe that for any tree $\mathcal{T}$ and any choices of $a \in (0,1)$ and $b \in \mathbb{R}$, m is $(0,1)$-valued and s is $(0,\infty)$-valued, so that Proposition 2.13 applies: the algorithm always converges in a finite number of steps.

2.15. Global optimality fails

By Theorem 2.9, Algorithm 2.11 is optimal, in a loose sense, *step by step*. That is, if knowing, for some $n \geq 1$, $\mathcal{D}_{\mathbf{x}_n}$ and the values of $R(x)$ for $x \in \mathbf{x}_n \subset \mathcal{L}$, it tells us how to choose the next leave $x_{n+1} \in \mathcal{L}$ so that $\mathbb{E}[(R_{n+1}(r) - R(r))^2]$ is as small as possible. However, it is not *globally* optimal.

Remark 2.15. Let $\mathcal{T}$ be the (deterministic) binary tree with depth 3 and assume that the outcomes $(R(x))_{x\in\mathcal{L}}$ are i.i.d. and Bernoulli(1/2)-distributed. Then Assumption 2.5 is satisfied and we can compute the functions m and s introduced in Definition 2.7. We thus may apply Algorithm 2.11, producing some random leaves $x_1, x_2, \ldots$. We set $\mathcal{F}_n = \sigma(\{B_{\mathbf{x}_n}, \mathcal{D}_{\mathbf{x}_n}, (R(x))_{x\in\mathbf{x}_n}\})$ and $R_n(r) = \mathbb{E}[R(r)|\mathcal{F}_n]$.

There is another *admissible* algorithm, producing some random leaves $\tilde{x}_1, \tilde{x}_2, \ldots$, such that, for $\tilde{\mathcal{F}}_n = \sigma(\{B_{\tilde{\mathbf{x}}_n}, \mathcal{D}_{\tilde{\mathbf{x}}_n}, (R(x))_{x\in\tilde{\mathbf{x}}_n}\})$ and $\tilde{R}_n(r) = \mathbb{E}[R(r)|\tilde{\mathcal{F}}_n]$, we have

$$\mathbb{E}[(\tilde{R}_2(r) - R(r))^2] > \mathbb{E}[(R_2(r) - R(r))^2] \quad \text{but} \quad \mathbb{E}[(\tilde{R}_3(r) - R(r))^2] < \mathbb{E}[(R_3(r) - R(r))^2].$$

It looks very delicate to determine the globally optimal algorithm. Moreover, it is likely that such an algorithm will be very intricate and will not enjoy the *quick update* property discussed in Section 2.12.

3. General convergence

We first show that Algorithm 2.11 is convergent with *any* reasonable parameters m and s.

Proof of Proposition 2.13. We consider some fixed tree $\mathcal{T} \in \mathcal{S}_f$ with $\mathcal{L}$ its set of leaves, some fixed outcomes $(R(x))_{x\in\mathcal{L}}$ and we denote by $(R(x))_{x\in\mathcal{T}}$ the corresponding minimax values. We also consider any pair of functions m and s on $\{(S,x) : S \in \mathcal{S}_f, x \text{ leave of } S\}$ with values in $[0,1]$ and $[0,\infty)$ respectively and we apply Algorithm 2.11. We assume that for any $S \in \mathcal{S}_f$ of which x is a leave, $s(S,x) = 0$ if and only if $m(S,x) \in \{0,1\}$, and that in such a case, $m(S,x) = R(x)$.

Step 1. After the nth step of the algorithm, we have some values $R_n(x) \in [0,1]$, $U_n(x) \in [0,1]$ and $Z_n(x) \geq 0$ for all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, for some $\mathbf{x}_n = \{x_1, \ldots, x_n\} \subset \mathcal{L}$. These quantities can generally not be interpreted in

terms of conditional expectations, but they always obey, by construction, the following rules:

(a) If $x \in \mathbf{x}_n$, then $R_n(x) = R(x)$ and $Z_n(x) = 0$.
(b) If $x \in \mathcal{D}_{\mathbf{x}_n}$, then $R_n(x) = m(\mathcal{K}_x, x)$ and $Z_n(x) = s(\mathcal{K}_x, x)$.
(c) If $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$, $R_n(x) = \mathbf{1}_{\{t(x)=0\}} \prod_{y\in\mathcal{C}_x} R_n(y) + \mathbf{1}_{\{t(x)=1\}}[1 - \prod_{y\in\mathcal{C}_x}(1 - R_n(y))]$.
(d) If $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$, $Z_n(x) = \max\{U_n^2(y)Z_n(y) : y \in \mathcal{C}_x\}$.
(e) If $x \in (B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}) \setminus \{r\}$, $U_n(x) = \mathbf{1}_{\{t(v(x))=0\}} \prod_{y\in\mathcal{H}_x} R_n(y) + \mathbf{1}_{\{t(v(x))=1\}} \prod_{y\in\mathcal{H}_x}(1 - R_n(y))$.
We finally recall that, by Proposition 2.1,
(f) if $x \in \mathcal{T} \setminus \mathcal{L}$, $R(x) = \mathbf{1}_{\{t(x)=0\}} \min\{R(y) : y \in \mathcal{C}_x\} + \mathbf{1}_{\{t(x)=1\}} \max\{R(y) : y \in \mathcal{C}_x\}$.

Step 2. Here we prove that for all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$,

$$R_n(x) \in \{0, 1\} \text{ and } Z_n(x) = 0 \text{ are equivalent and imply that } R_n(x) = R(x). \tag{3.1}$$

In the whole step, the notions of *child* (of $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$) and *brother* (of $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$) refer to the tree $\mathcal{T}$ or, equivalently, to the tree $B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$.

First, (3.1) is obvious if $x \in \mathbf{x}_n$ by point (a) (then $R_n(x) = R(x) \in \{0, 1\}$ and $Z_n(x) = 0$) and if $x \in \mathcal{D}_{\mathbf{x}_n}$ by point (b) and our assumption on m and s. We next work by backward induction: we consider $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$, we assume that all its children satisfy (3.1), and we prove that x also satisfies (3.1). We assume for example that $t(x) = 0$, the case where $t(x) = 1$ being treated similarly.

If $R_n(x) = 0$, then by (c), x has (at least) one child y such that $R_n(y) = 0$ whence, by induction assumption, $R(y) = 0$ and thus $R(x) = 0$ by (f). Furthermore, $R_n(y) = 0$ implies that $U_n(z) = 0$, whence $U_n^2(z)Z_n(z) = 0$, for all z brother of y by (e). And by induction assumption, we have $Z_n(y) = 0$, whence $U_n^2(y)Z_n(y) = 0$. We conclude, by (d), that $Z_n(x) = 0$, and we have seen that $R_n(x) = 0 = R(x)$.

If $R_n(x) = 1$, then by (c), all the children y of x satisfy $R_n(y) = 1$, whence, by induction assumption, $R(y) = 1$ and thus $R(x) = 1$ by (f). Still by induction assumption, $Z_n(y) = 0$ for all the children y of x, whence $Z_n(x) = 0$ by (d), and we have seen that $R_n(x) = 1 = R(x)$.

Assume now that $Z_n(x) = 0$, whence $U_n^2(y)Z_n(y) = 0$ for all the children y of x by (d). If there is (at least) one child y of x for which $U_n(y) = 0$, this means that there is another child z of x for which $R_n(z) = 0$ by (e), whence $R_n(x) = 0$ by (c). Else, we have $Z_n(y) = 0$ for all the children y of x, so that $R_n(y) \in \{0, 1\}$ by induction assumption, and thus $R_n(x) \in \{0, 1\}$ by (c).

We have shown that $R_n(x) \in \{0, 1\}$ implies $Z_n(x) = 0$ and $R_n(x) = R(x)$, and we have verified that $Z_n(x) = 0$ implies $R_n(x) \in \{0, 1\}$. Hence x satisfies (3.1), which completes the step.

Step 3. We now prove that if $x_{n+1} \in \mathbf{x}_n$, then $R_n(r) \in \{0, 1\}$ and this will prove point (i). Looking at Algorithm 2.11, we see that $x_{n+1} \in \mathbf{x}_n$ means that the procedure

$$\text{put } z = r \text{ and do } z = \operatorname{argmax}\{U_n^2(y)Z_n(y) : y \in \mathcal{C}_z\} \text{ until } z \in \mathbf{x}_n \cup \mathcal{D}_{\mathbf{x}_n}$$

returns some $z \in \mathbf{x}_n$. But then, $Z_n(z) = 0$ by (a). From (d) and the way z has been built, one easily gets convinced that this implies that $Z_n(r) = 0$, whence $R_n(r) \in \{0, 1\}$ by Step 1.

Step 4. By Step 3 and since $\mathcal{T}$ has a finite number of leaves, $n_0 = \inf\{n \geq 1 : R_n(r) \in \{0, 1\}\}$ is well-defined and finite. Finally, we know from Step 2 that $R_{n_0}(r) = R(r)$. □

4. Preliminaries

We first establish some general formulas concerning the functions m and s. They are not really necessary to understand the proof of our main result, but we need them to show Lemma 2.14. Also, we will use them to derive more tractable expressions in some particular cases in Section 6.

Lemma 4.1. *Suppose Assumption 2.5 and recall Definition 2.7. For any $S \in \mathcal{S}_f$ such that $\Pr(A_S) > 0$ and any $x \in L_S$,*

$$m(S,x) = \Pr(x \in \mathcal{L}, R(x) = 1 | A_S) + \sum_{\mathbf{y} \subset \mathbb{C}_x, |\mathbf{y}| \geq 1} \Pr(\mathcal{C}_x = \mathbf{y} | A_S) \Theta(S,x,\mathbf{y}),$$

$$s(S,x) = \sum_{k \in \{0,1\}} \Pr(x \in \mathcal{L}, R(x) = k | A_S)[k - m(S,x)]^2 + \sum_{\mathbf{y} \subset \mathbb{C}_x, |\mathbf{y}| \geq 1} \Pr(\mathcal{C}_x = \mathbf{y} | A_S) \sum_{y \in \mathbf{y}} \frac{\Gamma(S,x,\mathbf{y},y)}{|\mathbf{y}|},$$

where

$$\Theta(S,x,\mathbf{y}) = \mathbf{1}_{\{t(x)=0\}} \prod_{y \in \mathbf{y}} m(S \cup \mathbf{y}, y) + \mathbf{1}_{\{t(x)=1\}} \Big(1 - \prod_{y \in \mathbf{y}} (1 - m(S \cup \mathbf{y}, y)) \Big),$$

$$\begin{aligned} \Gamma(S,x,\mathbf{y},y) = {} & \mathbf{1}_{\{t(x)=0\}} \Big(s(S \cup \mathbf{y}, y) \prod_{u \in \mathbf{y} \setminus \{y\}} [m(S \cup \mathbf{y}, u)]^2 + \Big[\prod_{u \in \mathbf{y}} m(S \cup \mathbf{y}, u) - m(S,x) \Big]^2 \Big) \\ & + \mathbf{1}_{\{t(x)=1\}} \Big(s(S \cup \mathbf{y}, y) \prod_{u \in \mathbf{y} \setminus \{y\}} [1 - m(S \cup \mathbf{y}, u)]^2 + \Big[\prod_{u \in \mathbf{y}} (1 - m(S \cup \mathbf{y}, u)) - (1 - m(S,x)) \Big]^2 \Big). \end{aligned}$$

Proof. We fix $S \in \mathcal{S}_f$ such that $\Pr(A_S) > 0$ and $x \in L_S$. We first observe that for any $\mathbf{y} \subset \mathbb{C}_x$ with $|\mathbf{y}| \geq 1$, $A_S \cap \{\mathcal{C}_x = \mathbf{y}\} = A_{S \cup \mathbf{y}}$ (recall that A_S is the event on which $S \subset \mathcal{T}$ and all the brothers (in $\mathcal{T}$) of all the elements of S also belong to S).

We now study $m(S,x) = \Pr(R(x) = 1 | A_S)$, starting from

$$m(S,x) = \Pr(x \in \mathcal{L}, R(x) = 1 | A_S) + \sum_{\mathbf{y} \subset \mathbb{C}_x, |\mathbf{y}| \geq 1} \Pr(\mathcal{C}_x = \mathbf{y}, R(x) = 1 | A_S).$$

Hence the only difficulty is to verify that $\Pr(\mathcal{C}_x = \mathbf{y}, R(x) = 1 | A_S) = \Pr(\mathcal{C}_x = \mathbf{y} | A_S) \Theta(S,x,\mathbf{y})$ or, equivalently, that

$$\Pr(R(x) = 1 | A_S \cap \{\mathcal{C}_x = \mathbf{y}\}) = \Theta(S,x,\mathbf{y}). \tag{4.1}$$

Since $A_S \cap \{\mathcal{C}_x = \mathbf{y}\} = A_{S \cup \mathbf{y}}$ and since $\mathbf{y} \subset L_{S \cup \mathbf{y}}$, we know from Assumption 2.5 that the family $(\mathcal{T}_y, (R(u))_{u \in \mathcal{L} \cap \mathcal{T}_y})_{y \in \mathbf{y}}$ is independent conditionally on $A_S \cap \{\mathcal{C}_x = \mathbf{y}\}$. Consequently, the family $(R(y))_{y \in \mathbf{y}}$ is independent conditionally on $A_S \cap \{\mathcal{C}_x = \mathbf{y}\}$ (because $R(y)$ depends only on $\mathcal{T}_y$ and $(R(u))_{u \in \mathcal{L} \cap \mathcal{T}_y}$, recall Rem. 2.1). We assume *e.g.* $t(x) = 1$. Since $R(x) = \max\{R(y) : y \in \mathcal{C}_x\}$, we may write

$$\Pr(R(x) = 1 | A_S \cap \{\mathcal{C}_x = \mathbf{y}\}) = 1 - \prod_{y \in \mathbf{y}} \Pr(R(y) = 0 | A_S \cap \{\mathcal{C}_x = \mathbf{y}\}).$$

But for $y \in \mathbf{y}$, $\Pr(R(y) = 0 | A_S \cap \{\mathcal{C}_x = \mathbf{y}\}) = \Pr(R(y) = 0 | A_{S \cup \mathbf{y}}) = 1 - m(S \cup \mathbf{y}, y)$, whence (4.1), because $t(x) = 1$.

We next study s. Knowing A_S, we handle a uniformly random match starting from x, with resulting leave v and we set $\mathcal{G} = \sigma(v, R(v), \mathcal{K}_{xv})$, where $\mathcal{K}_{xv} = \mathcal{K}_v \cap \mathbb{T}_x$. We recall that $s(S,x) = \mathbb{E}[(R_1(x) - m(S,x))^2 | A_S]$, where $R_1(x) = \Pr(R(x) = 1 | \mathcal{G} \vee \sigma(A_S))$. If $x \in \mathcal{L}$, then $v = x$, whence $R(x)$ is $\mathcal{G}$-measurable and thus $R_1(x) = R(x)$. Consequently,

$$s(S,x) = \mathbb{E}[(R(x) - m(S,x))^2 \mathbf{1}_{\{x \in \mathcal{L}\}} | A_S] + \sum_{\mathbf{y} \subset \mathbb{C}_x, |\mathbf{y}| \geq 1} \mathbb{E}[(R_1(x) - m(S,x))^2 \mathbf{1}_{\{\mathcal{C}_x = \mathbf{y}\}} | A_S].$$

We have $\mathbb{E}[(R(x)-m(S,x))^2\mathbf{1}_{\{x\in\mathcal{L}\}}|A_S]=\sum_{k\in\{0,1\}}\Pr(x\in\mathcal{L},R(x)=k|A_S)[k-m(S,x)]^2$. We thus only have to check that $\mathbb{E}[(R_1(x)-m(S,x))^2\mathbf{1}_{\{\mathcal{C}_x=\mathbf{y}\}}|A_S]=\Pr(\mathcal{C}_x=\mathbf{y}|A_S)\sum_{y\in\mathbf{y}}|y|^{-1}\Gamma(S,x,\mathbf{y},y)$ or, equivalently, that $\mathbb{E}[(R_1(x)-m(S,x))^2|A_S\cap\{\mathcal{C}_x=\mathbf{y}\}]=\sum_{y\in\mathbf{y}}|y|^{-1}\Gamma(S,x,\mathbf{y},y)$.

On $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}$, let w be the child of x belonging to B_{xv}. Since v is obtained by handling a uniformly random match starting from x, $\Pr(w=y|A_S\cap\{\mathcal{C}_x=\mathbf{y}\})=|\mathbf{y}|^{-1}$ for all $y\in\mathbf{y}$. We thus only have to verify that

$$\mathbb{E}[(R_1(x)-m(S,x))^2|A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}]=\Gamma(S,x,\mathbf{y},y). \tag{4.2}$$

But $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}=A_{S\cup\mathbf{y}}$, so that, by Assumption 2.5 (and since the random match is independent of everything else), the family $(\mathcal{T}_u,(R(z))_{z\in\mathcal{L}\cap\mathcal{T}_u})_{u\in\mathbf{y}}$ is independent conditionally on $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}$. Hence the family $(R(u))_{u\in\mathbf{y}\setminus\{y\}}$ is independent and independent of $(\mathcal{T}_y,(R(z))_{z\in\mathcal{L}\cap\mathcal{T}_y})$ conditionally on $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}$. In particular, $(R(u))_{u\in\mathbf{y}\setminus\{y\}}$ is independent of $\mathcal{G}$ conditionally on $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}$.

From now on, we assume *e.g.* that $t(x)=0$.

We have $R(x)=\min\{R(u):u\in\mathbf{y}\}=\prod_{u\in\mathbf{y}}R(u)$ on $\{\mathcal{C}_x=\mathbf{y}\}$ and we conclude from the above independence property that, conditionally on $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}=A_{S\cup\mathbf{y}}\cap\{w=y\}$,

$$R_1(x)=\Pr\Big(\prod_{u\in\mathbf{y}}R(u)=1\Big|\mathcal{G}\vee\sigma(A_{S\cup\mathbf{y}})\Big)=\Pr(R(y)=1|\mathcal{G}\vee\sigma(A_{S\cup\mathbf{y}}))\prod_{u\in\mathbf{y}\setminus\{y\}}\Pr(R(u)=1|A_{S\cup\mathbf{y}}).$$

But $\Pr(R(u)=1|A_{S\cup\mathbf{y}})=m(S\cup\mathbf{y},u)$. Adopting the notation $R_1(y)=\Pr(R(y)=1|\mathcal{G}\vee\sigma(A_{S\cup\mathbf{y}}))$, we deduce that $R_1(x)=R_1(y)\prod_{u\in\mathbf{y}\setminus\{y\}}m(S\cup\mathbf{y},u)$ on $A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}$, whence

$$R_1(x)-m(S,x)=[R_1(y)-m(S\cup\mathbf{y},y)]\prod_{u\in\mathbf{y}\setminus\{y\}}m(S\cup\mathbf{y},u)+\Big[\prod_{u\in\mathbf{y}}m(S\cup\mathbf{y},u)-m(S,x)\Big].$$

Recall that $t(x)=0$. To conclude that (4.2) holds true, it only remains to verify that:

(a) $\mathbb{E}[(R_1(y)-m(S\cup\mathbf{y},y))^2|A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}]=s(S\cup\mathbf{y},y)$,
(b) $\mathbb{E}[R_1(y)|A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}]=m(S\cup\mathbf{y},y)$.

By definition, we have $s(S\cup\mathbf{y},y)=\mathbb{E}[(R_1(y)-m(S\cup\mathbf{y},y))^2|A_{S\cup\mathbf{y}}]$ conditionally on $\{w=y\}$, because on $\{w=y\}$, $R_1(y)=\Pr(R(y)=1|\mathcal{G}\vee\sigma(A_{S\cup\mathbf{y}}))$ is indeed the conditional probability that $R(y)=1$ knowing the information provided by a uniformly random match starting from w (with resulting leave v). Point (a) follows.

For (b), we write

$$\begin{aligned}\mathbb{E}[R_1(y)|A_S\cap\{\mathcal{C}_x=\mathbf{y}\}\cap\{w=y\}]&=\mathbb{E}[\Pr(R(y)=1|\mathcal{G}\vee\sigma(A_{S\cup\mathbf{y}}))|A_{S\cup\mathbf{y}}\cap\{w=y\}]\\&=\Pr(R(y)=1|A_{S\cup\mathbf{y}}\cap\{w=y\})\\&=\Pr(R(y)=1|A_{S\cup\mathbf{y}})\\&=m(S\cup\mathbf{y},y).\end{aligned}$$

For the second equality, we used that $\{w=y\}\in\mathcal{G}\vee\sigma(A_{S\cup\mathbf{y}})$. For the third equality, we used that w is of course independent of $R(y)$ knowing $A_{S\cup\mathbf{y}}$. □

We next give the

Proof of Lemma 2.14. We fix $S\in\mathcal{S}_f$ such that $\Pr(A_S)>0$ and $z\in L_S$.

If first $m(S,z)=\Pr(R(z)=1|A_S)=0$, then of course $R(z)=0$ a.s. on A_S, whence also $\Pr(R(z)=1|\mathcal{G}\vee\sigma(A_S))=0$ a.s. on A_S (recall Def. 2.7) and thus $s(S,z)=0$.

Similarly, $m(S,z)=1$ implies that $R(z)=1$ a.s. on A_S and that $s(S,z)=0$.

It only remains to prove that $s(S,z)=0$ implies that $m(S,z)\in\{0,1\}$.

If $\Pr(z\in\mathcal{L}|A_S)>0$, then either $\Pr(z\in\mathcal{L},R(z)=0|A_S)>0$ or $\Pr(z\in\mathcal{L},R(z)=1|A_S)>0$. If $s(S,z)=0$, we deduce from Lemma 4.1 that either $[m(S,z)]^2=0$ or $[1-m(S,z)]^2=0$, whence $m(S,z)\in\{0,1\}$.

If $\Pr(z\in\mathcal{L}|A_S)=0$, we consider a finite tree T with root z such that $\Pr(\mathcal{T}_z=T|A_S)>0$. We set $U_z=S$ and, for all $x\in T\setminus\{z\}$, $U_x=S\cup\bigcup_{y\in B_{zx}\setminus\{x\}}C_y^T\in\mathcal{S}_f$. It holds that $x\in L_{U_x}$ for all $x\in T$ and, if $x\in T\setminus L_T$, $U_x\cup C_x^T=U_y$ for all $y\in C_x^T$.

We now prove by backward induction that for any $x\in T$, $s(U_x,x)=0$ implies that $m(U_x,x)\in\{0,1\}$. Applied to $x=z$, this will complete the proof.

If first $x\in L_T$, then $\Pr(x\in\mathcal{L}|A_{U_x})>0$, because $A_S\cap\{\mathcal{T}_z=T\}\subset A_{U_x}\cap\{x\in\mathcal{L}\}$, because $\Pr(\mathcal{T}_z=T|A_S)>0$ and because $\Pr(A_S)>0$. We thus have already seen that $s(U_x,x)=0$ implies that $m(U_x,x)\in\{0,1\}$.

If next $x\in T\setminus L_T$, we introduce $\mathbf{y}=C_x^T$ and we see that $\Pr(\mathcal{C}_x=\mathbf{y}|A_{U_x})>0$, because $A_S\cap\{\mathcal{T}_z=T\}\subset A_{U_x}\cap\{\mathcal{C}_x=\mathbf{y}\}$, because $\Pr(\mathcal{T}_z=T|A_S)>0$ and because $\Pr(A_S)>0$. We deduce from Lemma 4.1 that if $s(U_x,x)=0$, then $\Gamma(U_x,x,\mathbf{y},y)=0$ for all $y\in\mathbf{y}$. If *e.g.* $t(x)=0$, this implies that for all $y\in\mathbf{y}$ (recall that $U_x\cup\mathbf{y}=U_y$),

$$\Gamma(U_x,x,\mathbf{y},y)=s(U_y,y)\prod_{u\in\mathbf{y}\setminus\{y\}}m(U_u,u)+\Big[\prod_{u\in\mathbf{y}}m(U_u,u)-m(U_x,x)\Big]^2=0.$$

Thus we always have $m(U_x,x)=\prod_{u\in\mathbf{y}}m(U_u,u)$ and either (i) $s(U_u,u)=0$ for all $u\in\mathbf{y}$ or (ii) there is $u\in\mathbf{y}$ such that $m(U_u,u)=0$. In case (i), we deduce from the induction assumption that $m(U_u,u)\in\{0,1\}$ for all $u\in\mathbf{y}$, whence $m(U_x,x)\in\{0,1\}$. In case (ii), we of course have $m(U_x,x)=0$. □

We next study the information provided by some admissible algorithm. Here, Assumption 2.5 is not necessary. The following result is intuitively obvious, but we found no short proof.

Lemma 4.2. *Recall Setting 2.9: an admissible algorithm provided some leaves $\mathbf{x}_n=\{x_1,\dots,x_n\}$ together with the objects $\mathcal{D}_{\mathbf{x}_n}$ and $(R(x))_{x\in\mathbf{x}_n}$. For any (deterministic) $\mathbf{y}_n=\{y_1,\dots,y_n\}\subset\mathbb{T}$, any $D_n\subset\mathbb{D}_{\mathbf{y}_n}$ and any $(a(y))_{y\in\mathbf{y}_n}\subset\{0,1\}^{\mathbf{y}_n}$, the law of $(\mathcal{T},(R(y))_{y\in\mathcal{L}})$ knowing*

$$A_n=\{\mathbf{x}_n=\mathbf{y}_n,\mathcal{D}_{\mathbf{y}_n}=D_n,(R(y))_{y\in\mathbf{y}_n}=(a(y))_{y\in\mathbf{y}_n}\}$$

is the same as knowing

$$A_n'=\{\mathbf{y}_n\subset\mathcal{L},\mathcal{D}_{\mathbf{y}_n}=D_n,(R(y))_{y\in\mathbf{y}_n}=(a(y))_{y\in\mathbf{y}_n}\}$$

as soon as $\Pr(A_n')>0$.

Proof. We work by induction on n.

Step 1. We fix $y_1\in\mathbb{T}$, we set $\mathbf{y}_1=\{y_1\}$, we consider $D_1\subset\mathbb{D}_{\mathbf{y}_1}$ and $a(y_1)\in\{0,1\}$. In this step we prove that the law of $(\mathcal{T},(R(y))_{y\in\mathcal{L}})$ knowing $A_1=\{x_1=y_1,\mathcal{D}_{\mathbf{y}_1}=D_1,R(y_1)=a(y_1)\}$ is the same as knowing $A_1'=\{y_1\in\mathcal{L},\mathcal{D}_{\mathbf{y}_1}=D_1,R(y_1)=a(y_1)\}$.

To this aim, we consider $T\in\mathcal{S}_f$ such that $y_1\in L_T$, $D_{\mathbf{y}_1}^T=D_1$ and $(\alpha(y))_{y\in L_T}\in\{0,1\}^{L_T}$ such that $\alpha(y_1)=a(y_1)$. We have to check that

$$\frac{\Pr(\mathcal{T}=T,(R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T},x_1=y_1)}{\Pr(\mathcal{D}_{\mathbf{y}_1}=D_1,R(y_1)=a(y_1),x_1=y_1)}=\frac{\Pr(\mathcal{T}=T,(R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T})}{\Pr(y_1\in\mathcal{L},\mathcal{D}_{\mathbf{y}_1}=D_1,R(y_1)=a(y_1))}$$

or, equivalently, that $p=q$, where

$$p=\Pr(x_1=y_1|\mathcal{T}=T,(R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}),$$

$$q = \Pr(x_1 = y_1 | y_1 \in \mathcal{L}, \mathcal{D}_{\mathbf{y}_1} = D_1, R(y_1) = a(y_1)).$$

Recalling that x_1 is the leave resulting from a uniformly random match starting from r, one easily gets convinced that $p = \Pr(x_1 = y_1 | \mathcal{T} = T) = \prod_{z \in B_{ry_1} \setminus \{y_1\}} |C_z^T|^{-1}$. By the same way, $q = \Pr(x_1 = y_1 | y_1 \in \mathcal{L}, \mathcal{D}_{\mathbf{y}_1} = D_1) = \prod_{z \in B_{ry_1} \setminus \{y_1\}} |C_z^{B_{y_1} \cup D_1}|^{-1}$. Since $D^T_{\mathbf{y}_1} = D_1$, we have $C_z^T = C_z^{B_{y_1} \cup D_1}$ for all $z \in B_{ry_1} \setminus \{y_1\}$ and the conclusion follows.

Step 2. Assume that the statement holds true with some $n \geq 1$. Consider some deterministic $\mathbf{y}_{n+1} = \{y_1, \dots, y_{n+1}\} \subset \mathbb{T}$, $D_{n+1} \subset \mathbb{D}_{\mathbf{y}_{n+1}}$ and $(a(y))_{y \in \mathbf{y}_{n+1}} \subset \{0,1\}^{\mathbf{y}_{n+1}}$, as well as the events

$$A_{n+1} = \{\mathbf{x}_{n+1} = \mathbf{y}_{n+1}, \mathcal{D}_{\mathbf{y}_{n+1}} = D_{n+1}, (R(y))_{y \in \mathbf{y}_{n+1}} = (a(y))_{y \in \mathbf{y}_{n+1}}\}$$
$$A'_{n+1} = \{\mathbf{y}_{n+1} \subset \mathcal{L}, \mathcal{D}_{\mathbf{y}_{n+1}} = D_{n+1}, (R(y))_{y \in \mathbf{y}_{n+1}} = (a(y))_{y \in \mathbf{y}_{n+1}}\}.$$

Recall that x_{n+1} is chosen as follows: for some deterministic function F as in Remark 2.4 and some $X_n \sim \mathcal{U}([0,1])$ independent of everything else, we set $z_n = F(\mathbf{x}_n, \mathcal{D}_{\mathbf{x}_n}, (R(x))_{x \in \mathbf{x}_n}, X_n)$, which belongs to $\mathbf{x}_n \cup \mathcal{D}_{\mathbf{x}_n}$. If $z_n \in \mathbf{x}_n$, we set $x_{n+1} = z_n$, else, we handle a uniformly random match starting from z_n and denote by x_{n+1} the resulting leave.

If $y_{n+1} \in \mathbf{y}_n$, then we have $A_{n+1} = A_n \cap \{F(\mathbf{y}_n, D_n, (a(x))_{x \in \mathbf{y}_n}, X_n) = y_{n+1}\}$ and $A'_{n+1} = A'_n$, where $D_n = D_{n+1}$, where $A_n = \{\mathbf{x}_n = \mathbf{y}_n, \mathcal{D}_{\mathbf{y}_n} = D_n, (R(y))_{y \in \mathbf{y}_n} = (a(y))_{y \in \mathbf{y}_n}\}$ and where $A'_n = \{\mathbf{y}_n \subset \mathcal{L}, \mathcal{D}_{\mathbf{y}_n} = D_n, (R(y))_{y \in \mathbf{y}_n} = (a(y))_{y \in \mathbf{y}_n}\}$. By induction assumption, we know that the law of $(\mathcal{T}, (R(y))_{y \in \mathcal{L}})$ knowing A_n is the same as knowing A'_n. Since X_n is independent of $(\mathcal{T}, (R(y))_{y \in \mathcal{L}}), A_n$, the law of $(\mathcal{T}, (R(y))_{y \in \mathcal{L}})$ knowing A_{n+1} is the same as knowing A_n and thus the same as knowing A'_{n+1} (which equals A'_n).

If $y_{n+1} \notin \mathbf{y}_n$, let x be the element of $B_{ry_{n+1}} \cap B_{\mathbf{y}_n}$ the closest to y_{n+1} and let z_n be the child of x belonging to $B_{ry_{n+1}}$. We set $D_n = (D_{n+1} \setminus \mathbb{T}_{z_n}) \cup \{z_n\}$. Then $A_{n+1} = A_n \cap B_1 \cap B_2$ and $A'_{n+1} = A'_n \cap B'_2$, where A_n and A'_n are as in the statement and

$$B_1 = \{F(\mathbf{y}_n, D_n, (a(x))_{x \in \mathbf{y}_n}, X_n) = z_n\},$$
$$B_2 = \{x_{n+1} = y_{n+1}, \mathcal{D}_{\{y_{n+1}\}} \cap \mathbb{T}_{z_n} = D_{n+1} \cap \mathbb{T}_{z_n}, R(y_{n+1}) = a(y_{n+1})\},$$
$$B'_2 = \{y_{n+1} \in \mathcal{L}, \mathcal{D}_{\{y_{n+1}\}} \cap \mathbb{T}_{z_n} = D_{n+1} \cap \mathbb{T}_{z_n}, R(y_{n+1}) = a(y_{n+1})\}.$$

First, since X_n is independent of everything else, the law of $(\mathcal{T}, (R(y))_{y \in \mathcal{L}})$ knowing A_{n+1} is the same as knowing $A_n \cap B_2$ (from now on, we take the convention that in B_2, x_{n+1} is the leave resulting from a uniformly random match starting from z_n). We thus only have to prove that the law of $(\mathcal{T}, (R(y))_{y \in \mathcal{L}})$ knowing $A_n \cap B_2$ is the same as knowing $A'_n \cap B'_2$. Consider $T \in \mathcal{S}_f$ and $(\alpha(y))_{y \in L_T} \in \{0,1\}^{L_T}$, such that $\mathbf{y}_{n+1} \subset L_T$, $D^T_{\mathbf{y}_{n+1}} = D_{n+1}$ and $(\alpha(y))_{y \in \mathbf{y}_{n+1}} = (a(y))_{y \in \mathbf{y}_{n+1}}$. We have to prove that

$$\Pr(\mathcal{T} = T, (R(y))_{y \in L_T} = (\alpha(y))_{y \in L_T} | A_n \cap B_2) = \Pr(\mathcal{T} = T, (R(y))_{y \in L_T} = (\alpha(y))_{y \in L_T} | A'_n \cap B'_2).$$

We start from

$$\Pr(\mathcal{T} = T, (R(y))_{y \in L_T} = (\alpha(y))_{y \in L_T} | A_n \cap B_2) = \frac{\Pr(\{\mathcal{T} = T, (R(y))_{y \in L_T} = (\alpha(y))_{y \in L_T}\} \cap B_2 | A_n)}{\Pr(B_2 | A_n)}$$
$$= \frac{\Pr(\{\mathcal{T} = T, (R(y))_{y \in L_T} = (\alpha(y))_{y \in L_T}\} \cap B_2 | A'_n)}{\Pr(B_2 | A'_n)}$$

thanks to our induction assumption. On the one hand, exactly as in Step 1, we have

$$\Pr(\{\mathcal{T} = T, (R(y))_{y \in L_T} = (\alpha(y))_{y \in L_T}\} \cap B_2 | A'_n)$$

$$= \Pr(\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}, x_{n+1}=y_{n+1}|A'_n)$$
$$= \Pr(\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}|A'_n) \prod_{u\in B_{z_n y_{n+1}}\setminus\{y_{n+1}\}} |C_u^T|^{-1}.$$

On the other hand,

$$\Pr(B_2|A'_n) = \Pr(B'_2\cap\{x_{n+1}=y_{n+1}\}|A'_n) = \Pr(B'_2|A'_n) \prod_{u\in B_{z_n y_{n+1}}\setminus\{y_{n+1}\}} \left|C_u^{B_{z_n y_{n+1}}\cup(D_{n+1}\cap\mathbb{T}_{z_n})}\right|^{-1}.$$

Since $C_u^T = C_u^{B_{z_n y_{n+1}}\cup(D_{n+1}\cap\mathbb{T}_{z_n})}$ for all $u\in\setminus\{y_{n+1}\}$ (because $D^T_{\mathbf{y}_{n+1}} = D_{n+1}$), we conclude that

$$\Pr(\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}|A_n\cap B_2) = \frac{\Pr(\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}|A'_n)}{\Pr(B'_2|A'_n)}.$$

Since finally $\{\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}\}\subset B'_2$, we conclude that

$$\Pr(\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}|A_n\cap B_2) = \Pr(\mathcal{T}=T, (R(y))_{y\in L_T}=(\alpha(y))_{y\in L_T}|A'_n\cap B'_2),$$

which was our goal. □

We deduce the following observation, that is crucial to our study.

Lemma 4.3. *Suppose Assumption 2.5 and recall Setting 2.9: an admissible algorithm provided some leaves* $\mathbf{x}_n=\{x_1,\dots,x_n\}$ *together with the objects* $\mathcal{D}_{\mathbf{x}_n}$ *and* $(R(x))_{x\in\mathbf{x}_n}$ *and we define* $\mathcal{F}_n=\sigma(\mathbf{x}_n,\mathcal{D}_{\mathbf{x}_n},(R(x))_{x\in\mathbf{x}_n})$. *Recall also Remark 2.6.*

(i) Knowing $\mathcal{F}_n$, *for all* $x\in\mathcal{D}_{\mathbf{x}_n}$, *the conditional law of* $(\mathcal{T}_x,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x})$ *is* $G_{\mathcal{K}_x,x}$.
(ii) Knowing $\mathcal{F}_n$, *for all* $x\in B_{\mathbf{x}_n}\setminus\mathbf{x}_n$, *the family* $((\mathcal{T}_u,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_u},u\in\mathcal{C}_x)$ *is independent.*

Recall that for all $x\in\mathcal{D}_{\mathbf{x}_n}$, $\mathcal{K}_x$ is $\mathcal{F}_n$-measurable and that for all $x\in B_{\mathbf{x}_n}\setminus\mathbf{x}_n$, $\mathcal{C}_x$ is $\mathcal{F}_n$-measurable. Hence this statement is meaningful.

Proof. We observe that $\mathcal{F}_n$ is generated by the events of the form

$$A_n=\{\mathbf{x}_n=\mathbf{y}_n,\mathcal{D}_{\mathbf{y}_n}=D_n,(R(y))_{y\in\mathbf{y}_n}=(a(y))_{y\in\mathbf{y}_n}\}$$

as in Lemma 4.2. Let $A'_n=\{\mathbf{y}_n\subset\mathcal{L},\mathcal{D}_{\mathbf{y}_n}=D_n,(R(y))_{y\in\mathbf{y}_n}=(a(y))_{y\in\mathbf{y}_n}\}$. We see that on A'_n (which contains A_n), $\mathcal{K}_x=K_x^T$ for all $x\in D_n$ and $\mathcal{C}_x=C_x^T$ for all $x\in B_{\mathbf{x}_n}\setminus\mathbf{x}_n$, where $T=B_{\mathbf{y}_n}\cup D_n$.

To check (i), it thus suffices to prove that knowing A_n, for all $x\in D_n$, the law of $(\mathcal{T}_x,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ is $G_{K_x^T,x}$. We fix $x\in D_n$. By Lemma 4.2, it suffices to verify that the law of $(\mathcal{T}_x,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ knowing A'_n is $G_{K_x^T,x}$. Recalling that $A_{K_x^T}=\{K_x^T\subset\mathcal{T},\mathcal{D}_{K_x^T}=\emptyset\}$, we write $A'_n=A_{K_x^T}\cap\bigcap_{u\in L_{K_x^T}\setminus\{x\}}E_u$, where

$$E_u=\{\mathbf{y}_n\cap\mathbb{T}_u\subset\mathcal{L},\mathcal{D}_{\mathbf{y}_n}\cap\mathbb{T}_u=D_n\cap\mathbb{T}_u,(R(y))_{y\in\mathbf{y}_n\cap\mathbb{T}_u}=(a(y))_{y\in\mathbf{y}_n\cap\mathbb{T}_u}\}.$$

By Assumption 2.5, $(\mathcal{T}_x,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ is independent of $\bigcap_{u\in L_{K_x^T}\setminus\{x\}}E_u$ knowing $A_{K_x^T}$. Thus the law of $(\mathcal{T}_x,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ knowing A'_n equals the law of $(\mathcal{T}_x,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ knowing $A_{K_x^T}$, which is $G_{K_x^T,x}$ by definition.

For (ii), we show that for any $x\in B_{\mathbf{y}_n}\setminus\mathbf{y}_n$, the family $((\mathcal{T}_u,(R(y))_{y\in\mathcal{L}\cap\mathcal{T}_u},u\in C_x^T)$ is independent conditionally on A_n, or equivalently, conditionally on A'_n. To this aim, we introduce $S=T\setminus\cup_{y\in C_x^T}(\mathbb{T}_y\setminus\{y\})$ (*i.e.* S is the tree T from which we have removed all the subtrees strictly below the children of x). We write $A'_n=A_S\cap\bigcap_{u\in L_S}E_u$ with E_u as in the proof of (i). We know from Assumption 2.5 that the family

$((\mathcal{T}_u, (R(y))_{y\in\mathcal{L}\cap\mathbb{T}_u}, u \in L_S)$ is independent conditionally on A_S. Observing that $E_u \in \sigma(\mathcal{T}_u, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_u})$ for all $u \in L_S$, we conclude that the family $((\mathcal{T}_u, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_u}, u \in L_S)$ is independent conditionally on A'_n. (Here we used that if a family of random variables $(X_i)_{i\in I}$ is independent conditionally on some event A and if we have some events $E_i \in \sigma(X_i)$, for $i \in I$, then the family $(X_i)_{i\in I}$ is independent conditionally on $A \cap \bigcap_{i\in I} E_i$). Since $C_x^T \subset L_S$, the conclusion follows. □

5. Proof of the main result

In the whole section, we take Assumption 2.5 for granted. We first compute the conditional minimax values.

Proof of Proposition 2.8. We work under Setting 2.9. If first $x \in \mathbf{x}_n$, then $R(x)$ is $\mathcal{F}_n$-measurable, so that $R_n(x) = \Pr(R(x) = 1|\mathcal{F}_n) = R(x)$.

Next, for $x \in \mathcal{D}_{\mathbf{x}_n}$, we know from Lemma 4.3 that the law of $(\mathcal{T}_x, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ knowing $\mathcal{F}_n$ is $G_{\mathcal{K}_x,x}$. Recalling Definition 2.7 and Remark 2.6, we see that $R_n(x) = \Pr(R(x) = 1|\mathcal{F}_n) = m(\mathcal{K}_x, x)$.

Finally, for $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$, Lemma 4.3 tells us that the family $((\mathcal{T}_y, (R(u))_{u\in\mathcal{L}\cap\mathcal{T}_y}, y \in \mathcal{C}_x)$ is independent conditionally on $\mathcal{F}_n$. But for $y \in \mathcal{C}_x$, $R(y)$ is of course $\sigma(\mathcal{T}_y, (R(u))_{u\in\mathcal{L}\cap\mathcal{T}_y})$-measurable (recall Rem. 2.1). Thus the family $(R(y), y \in \mathcal{C}_x)$ is independent conditionally on $\mathcal{F}_n$. If $t(x) = 0$, we may write, by (2.1),

$$R_n(x) = \Pr(R(x) = 1|\mathcal{F}_n) = \Pr(\min\{R(y) : y \in \mathcal{C}_x\} = 1|\mathcal{F}_n) = \prod_{y\in\mathcal{C}_x} \Pr(R(y) = 1|\mathcal{F}_n),$$

which equals $\prod_{y\in\mathcal{C}_x} R_n(y)$ as desired. If now $t(x) = 1$, we find similarly

$$R_n(x) = \Pr(R(x) = 1|\mathcal{F}_n) = \Pr(\max\{R(y) : y \in \mathcal{C}_x\} = 1|\mathcal{F}_n) = 1 - \prod_{y\in\mathcal{C}_x} \Pr(R(y) = 0|\mathcal{F}_n),$$

which is nothing but $1 - \prod_{y\in\mathcal{C}_x}(1 - R_n(y))$. □

We now study the different possibilities for the $(n + 1)$th step.

Proposition 5.1. *Adopt Setting 2.9 and Notation 2.1. For $z \in \mathcal{D}_{\mathbf{x}_n}$ and $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, we set*

$$\Delta_n^z(x) = \mathbb{E}[(R_{n+1}^z(x) - R_n(x))^2|\mathcal{F}_n].$$

(i) We have $\operatorname{argmin}\{\mathbb{E}[(R_{n+1}^z(r) - R(r))^2|\mathcal{F}_n] : z \in \mathcal{D}_{\mathbf{x}_n}\} = \operatorname{argmax}\{\Delta_n^z(r) : z \in \mathcal{D}_{\mathbf{x}_n}\}$.
(ii) For all $z \in \mathcal{D}_{\mathbf{x}_n}$, we have $\Delta_n^z(z) = s(\mathcal{K}_z, z)$.
(iii) For all $z \in \mathcal{D}_{\mathbf{x}_n}$, all $x \in B_{rz} \setminus \{r\}$, we have $\Delta_n^z(f(x)) = (U_n(x))^2\Delta_n^z(x)$.
(iv) For all $z \in \mathcal{D}_{\mathbf{x}_n}$, $\Delta_n^z(r) = s(\mathcal{K}_z, z)\prod_{y\in B_{rz}\setminus\{r\}}(U_n(y))^2$.

Proof. Point (i) is not difficult: for all $z \in \mathcal{D}_{\mathbf{x}_n}$,

$$\begin{aligned}\mathbb{E}[(R_n(r) - R(r))^2|\mathcal{F}_n] = \mathbb{E}[(R_{n+1}^z(r) - R(r))^2|\mathcal{F}_n] + \mathbb{E}[(R_{n+1}^z(r) - R_n(r))^2|\mathcal{F}_n] \\ + 2\mathbb{E}[(R_{n+1}^z(r) - R(r))(R_n(r) - R_{n+1}^z(r))|\mathcal{F}_n].\end{aligned}$$

Using that $\mathcal{F}_n \subset \mathcal{F}_{n+1}^z$ and that $R_{n+1}^z(r) = \mathbb{E}[R(r)|\mathcal{F}_{n+1}^z]$, we conclude that

$$\mathbb{E}[(R_n(r) - R(r))^2|\mathcal{F}_n] = \mathbb{E}[(R_{n+1}^z(r) - R(r))^2|\mathcal{F}_n] + \mathbb{E}[(R_{n+1}^z(r) - R_n(r))^2|\mathcal{F}_n].$$

Since the left hand side does not depend on z, minimizing $\mathbb{E}[(R_{n+1}^z(r) - R(r))^2|\mathcal{F}_n]$ is equivalent to maximizing $\mathbb{E}[(R_{n+1}^z(r) - R_n(r))^2|\mathcal{F}_n]$.

Also, point (iv) immediately follows from points (ii) and (iii).

To check points (ii) and (iii), we fix $z \in \mathcal{D}_{x_n}$. We recall that $\mathcal{F}_{n+1}^z = \mathcal{F}_n \vee \mathcal{G}$, where $\mathcal{G} = \sigma(y, \mathcal{K}_{zy}, R(y))$, where y is the leave resulting from a uniformly random match starting from z and where $\mathcal{K}_{zy} = \mathcal{K}_y \cap \mathbb{T}_z$. We also recall that $R_{n+1}^z(x) = \Pr(R(x) = 1 | \mathcal{F}_{n+1}^z)$ for all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$.

We know from Lemma 4.3 that the law of $(\mathcal{T}_z, (R(y))_{y \in \mathcal{L} \cap \mathcal{T}_z})$ knowing $\mathcal{F}_n$ is $G_{\mathcal{K}_z, z}$. Recalling Definition 2.7 and Remark 2.6, we immediately deduce that $R_n(z) = m(\mathcal{K}_z, z)$ and that

$$\Delta_n^z(z) = \mathbb{E}[(R_{n+1}^z(z) - R_n(z))^2 | \mathcal{F}_n] = s(\mathcal{K}_z, z).$$

This proves (ii). To prove (iii), we fix $x \in B_{rz} \setminus \{r\}$ and we set $v = f(x)$. By Lemma 4.3, the family $((\mathcal{T}_y, (R(u))_{u \in \mathcal{L} \cap \mathcal{T}_y}), y \in \mathcal{C}_v)$ is independent conditionally on $\mathcal{F}_n$. Furthermore, $\mathcal{G}$, which only concerns $(\mathcal{T}_x, (R(u))_{u \in \mathcal{L} \cap \mathcal{T}_x})$ is independent of the family $((\mathcal{T}_y, (R(u))_{u \in \mathcal{L} \cap \mathcal{T}_y}), y \in \mathcal{H}_x)$. Finally, we recall that for all $y \in \mathcal{C}_v$, $R(y)$ is $\sigma(\mathcal{T}_y, (R(u))_{u \in \mathcal{L} \cap \mathcal{T}_y})$-measurable.

If $t(v) = 0$,

$$R_{n+1}^z(v) = \Pr(R(v) = 1 | \mathcal{F}_n \vee \mathcal{G}) = \Pr(\min\{R(y) : y \in \mathcal{C}_v\} = 1 | \mathcal{F}_n \vee \mathcal{G})$$

whence, by conditional independence, $R_{n+1}^z(v) = \prod_{y \in \mathcal{C}_v} \Pr(R(y) = 1 | \mathcal{F}_n \vee \mathcal{G})$ and thus

$$R_{n+1}^z(v) = \Big(\prod_{y \in \mathcal{H}_x} \Pr(R(y) = 1 | \mathcal{F}_n) \Big) \Pr(R(x) = 1 | \mathcal{F}_n \vee \mathcal{G}) = \Big(\prod_{y \in \mathcal{H}_x} R_n(y) \Big) R_{n+1}^z(x),$$

whence $R_{n+1}^z(v) = U_n(x) R_{n+1}^z(x)$. Since now $R_n(v) = \prod_{y \in \mathcal{C}_v} R_n(y) = U_n(x) R_n(x)$ by Proposition 2.8, we conclude that $(R_{n+1}(v) - R_n(v))^2 = U_n^2(x)(R_{n+1}^z(x) - R_n(x))^2$, whence, taking expectations conditionally on $\mathcal{F}_n$, $\Delta_n^z(v) = U_n^2(x) \Delta_n^z(x)$.

If next $t(v) = 1$,

$$R_{n+1}^z(v) = \Pr(R(v) = 1 | \mathcal{F}_n \vee \mathcal{G}) = \Pr(\max\{R(y) : y \in \mathcal{C}_v\} = 1 \Big| \mathcal{F}_n \vee \mathcal{G}),$$

so that $R_{n+1}^z(v) = 1 - \prod_{y \in \mathcal{C}_v} \Pr(R(y) = 0 | \mathcal{F}_n \vee \mathcal{G})$. We conclude that

$$\begin{aligned} R_{n+1}^z(v) &= 1 - \Big(\prod_{y \in \mathcal{H}_x} \Pr(R(y) = 0 | \mathcal{F}_n) \Big) \Pr(R(x) = 0 | \mathcal{F}_n \vee \mathcal{G}) \\ &= 1 - \Big(\prod_{y \in \mathcal{H}_x} (1 - R_n(y)) \Big) (1 - R_{n+1}^z(x)), \end{aligned}$$

so that $R_{n+1}^z(v) = 1 - U_n(x)(1 - R_{n+1}^z(x))$. Since finally $R_n(v) = 1 - \prod_{y \in \mathcal{C}_v} (1 - R_n(y)) = 1 - U_n(x)(1 - R_n(x))$ by Proposition 2.8, $(R_{n+1}^z(v) - R_n(v))^2 = U_n^2(x)(R_{n+1}^z(x) - R_n(x))^2$, whence, taking expectations conditionally on $\mathcal{F}_n$, $\Delta_n^z(v) = U_n^2(x) \Delta_n^z(x)$. □

We now have all the weapons to give the

Proof of Theorem 2.9. Recall that we work under Setting 2.9 and that we adopt Notation 2.1 in which U_n, Z_n and $\mathcal{F}_z^{n+1}$ are defined. For $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$, we set $\bar{U}_n(x) = \prod_{B_{rx} \setminus \{r\}} U_n(y)$, with the convention that $\bar{U}_n(r) = 1$.

Step 1. In view of the explicit formula for $\Delta_n^z(r)$ checked in Proposition 5.1-(iv), the natural way to find z_* maximizing $\Delta_n^z(r)$ is to start from r and to go down in $B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$ following the maximum values of $(N_n(x))_{x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}}$ defined as follows. Set $N_n(x) = 0$ for $x \in \mathbf{x}_n$, set $N_n(x) = (\bar{U}_n(x))^2 s(\mathcal{K}_x, x)$ for $x \in \mathcal{D}_{\mathbf{x}_n}$ and put $N_n(x) = \max\{N_n(y) : y \in \mathcal{C}_x\}$ for $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$.

We claim that $N_n(x) = (\bar{U}_n(x))^2 Z_n(x)$ for all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$.

Indeed, set $\tilde{N}_n(x) = (\bar{U}_n(x))^2 Z_n(x)$ and recall Notation 2.1-(ii). We obviously have $\tilde{N}_n(x) = N_n(x)$ for $x \in \mathbf{x}_n$ (because then $Z_n(x) = 0$) and for $x \in \mathcal{D}_{\mathbf{x}_n}$ (because then $Z_n(x) = s(\mathcal{K}_x, x)$). And for $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$, we have $\tilde{N}_n(x) = (\bar{U}_n(x))^2 \max\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_x\} = \max\{\tilde{N}_n(y) : y \in \mathcal{C}_x\}$, because for $y \in \mathcal{C}_x$, we have $\bar{U}_n(x) U_n(y) = \bar{U}_n(y)$. The claim follows by (backward) induction.

Step 2. We define $z^* \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$ as follows. Put $y_0 = r$. Find $y_1 = \operatorname{argmax}\{N_n(y) : y \in \mathcal{C}_{y_0}\}$. If $y_1 \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$, set $z^* = y_1$. Else, put $y_2 = \operatorname{argmax}\{N_n(y) : y \in \mathcal{C}_{y_1}\}$. If $y_2 \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$, set $z^* = y_2$. Else, put $y_3 = \operatorname{argmax}\{N_n(y) : y \in \mathcal{C}_{y_2}\}$, etc.

By construction, $z^* = \operatorname{argmax}\{N_n(z) : z \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n\}$. Also, $N_n(x) = N_n(r)$ for all $x \in B_{rz^*}$.

Step 3. We recall that $z_* \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$ was defined, similarly, as follows: put $y_0 = r$ and find $y_1 = \operatorname{argmax}\{U_n^2(y) Z_n(y) : y \in \mathcal{C}_{y_0}\}$. If $y_1 \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$, set $z_* = y_1$. Else, put $y_2 = \operatorname{argmax}\{U_n^2(y) Z_n(y) : y \in \mathcal{C}_{y_1}\}$. If $y_2 \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n$, set $z_* = y_2$. Else, put $y_3 = \operatorname{argmax}\{U_n^2(y) Z_n(y) : y \in \mathcal{C}_{y_2}\}$, etc.

Step 4. We now prove that if $N_n(r) > 0$, then $z_* = z^*$.

First observe that for any $x \in B_{\mathbf{x}_n} \setminus \mathbf{x}_n$ such that $\bar{U}_n(x) > 0$,

$$\begin{aligned} \operatorname{argmax}\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_x\} &= \operatorname{argmax}\{(\bar{U}_n(x))^2 (U_n(y))^2 Z_n(y) : y \in \mathcal{C}_x\} \\ &= \operatorname{argmax}\{(\bar{U}_n(y))^2 Z_n(y) : y \in \mathcal{C}_x\} \\ &= \operatorname{argmax}\{N_n(y) : y \in \mathcal{C}_x\}. \end{aligned} \tag{5.1}$$

Furthermore, we have $N_n(x) = N_n(r)$ for all $x \in B_{rz^*}$. Hence if $N_n(r) > 0$, then $\bar{U}_n(x) > 0$ for all $x \in B_{rz^*}$ (recall that $N_n(x) = (\bar{U}_n(x))^2 Z_n(x)$). Consequently, (5.1) holds true during the whole computation of z^*, so that $z_* = z^*$.

Step 5. By Steps 2 and 4, $z_* = z^* = \operatorname{argmax}\{N_n(z) : z \in \mathcal{D}_{\mathbf{x}_n} \cup \mathbf{x}_n\}$ on $\{N_n(r) > 0\}$. And we know from Proposition 2.13 and Lemma 2.14 that on $\{R_n(r) \notin \{0,1\}\}$, $z_* \in \mathcal{D}_{\mathbf{x}_n}$. We also know that $R_n(r) \notin \{0,1\}$ implies that $Z_n(r) > 0$ (see Step 2 of the proof of Prop. 2.13), whence $N_n(r) = Z_n(r) > 0$ (recall Step 1 and that $\bar{U}_n(r) = 1$). Thus on $\{R_n(r) \notin \{0,1\}\}$, we have $z_* \in \mathcal{D}_{\mathbf{x}_n}$ and $z_* = \operatorname{argmax}\{N_n(z) : z \in \mathcal{D}_{\mathbf{x}_n}\} = \operatorname{argmax}\{(\bar{U}_n(z))^2 s(\mathcal{K}_z, z) : z \in \mathcal{D}_{\mathbf{x}_n}\}$ by definition of N_n, see Step 1. By Proposition 5.1-(iv), we conclude that on $\{R_n(r) \notin \{0,1\}\}$

$$z_* = \operatorname{argmax}\{\mathbb{E}[(R_{n+1}^z(r) - R_n(r))^2 | \mathcal{F}_n] : z \in \mathcal{D}_{\mathbf{x}_n}\},$$

which equals $\operatorname{argmin}\{\mathbb{E}[(R_{n+1}^z(r) - R(r))^2 | \mathcal{F}_n] : z \in \mathcal{D}_{\mathbf{x}_n}\}$ by Proposition 5.1-(i). We have verified that on $\{R_n(r) \notin \{0,1\}\}$, $z_* \in \mathcal{D}_{\mathbf{x}_n}$ and $z_* = \operatorname{argmin}\{\mathbb{E}[(R_{n+1}^z(r) - R(r))^2 | \mathcal{F}_n] : z \in \mathcal{D}_{\mathbf{x}_n}\}$, which was our goal. □

Remark 5.2. As seen in the proof, the natural way to find z_* would be to start from r and to go down in the tree following the highest values of N_n. Recalling the discussion of Section 2.12, this would lead to an algorithm with cost of order Kdn^2: since (generally) $R_{n+1}(x) \neq R_n(x)$ for $x \in B_{rx_{n+1}}$, this (generally) modifies the value of $\bar{U}_n(x)$ for all $x \in B_{\mathbf{x}_n} \setminus B_{rx_{n+1}}$ (actually, except for $x \in B_{\mathbf{x}_n} \cap B_{rx_{n+1}}$) and thus the values of $N_n(x)$ for all x on the whole explored tree $B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$. The observation (5.1), which asserts that $\operatorname{argmax}\{N_n(y) : y \in \mathcal{C}_x\} = \operatorname{argmax}\{(U_n(y))^2 Z_n(y) : y \in \mathcal{C}_x\}$ is thus crucial, as well as the fact that U_n and Z_n enjoy a quick update property.

6. Computation of the parameters for a few specific models

Here we present a few models for the tree $\mathcal{T}$ and the outcomes $(R(x))_{x \in \mathcal{L}}$ where our assumptions are met and where we can compute, at least numerically, the functions m and s introduced in Definition 2.7. Recall that these functions are necessary to implement Algorithm 2.11.

6.1. Inhomogeneous Galton–Watson trees

We assume that $\mathcal{T}$ is the realization of an inhomogeneous Galton–Watson tree with reproduction laws $\mu_0, \dots, \mu_K$: the number of children of the root r follows the law $\mu_0 \in \mathcal{P}(\mathbb{N})$, the number of children of these children are independent and μ_1-distributed, etc. We assume that $\mu_K = \delta_0$, so that any individual of generation K is a leave and thus K is the maximal depth of $\mathcal{T}$.

We also consider a family $q_0, \dots, q_K$ of numbers in $[0,1]$. Conditionally on $\mathcal{T}$, we assume that the family $(R(x))_{x\in\mathcal{L}}$ is independent and that $R(x) \sim \text{Bernoulli}(q_{|x|})$ for all $x \in \mathcal{L}$.

Example 6.1. With such a model, Assumption 2.5 is fulfilled, and for any $S \in S_f$ and $x \in L_S$ such that $\Pr(A_S) > 0$, we have $m(S,x) = \mathfrak{m}(|x|)$ and $s(S,x) = \mathfrak{s}(|x|)$, where

(i) $\mathfrak{m}$ is defined by backward induction by $\mathfrak{m}(K) = q_K$ and, for $k = 0, \dots, K-1$,

$$\mathfrak{m}(k) = \mu_k(0)q_k + \sum_{\ell\geq 1}\mu_k(\ell)\Big(\mathbf{1}_{\{k \text{ is odd}\}}(\mathfrak{m}(k+1))^\ell + \mathbf{1}_{\{k \text{ is even}\}}(1-[1-\mathfrak{m}(k+1)]^\ell)\Big),$$

(ii) $\mathfrak{s}$ is defined by backward induction by $\mathfrak{s}(K) = q_K(1-q_K)$ and, for $k = 0, \dots, K-1$,

$$\begin{aligned}\mathfrak{s}(k) =& \mu_k(0)[q_k(1-\mathfrak{m}(k))^2 + (1-q_k)(\mathfrak{m}(k))^2] \\ &+ \mathbf{1}_{\{k \text{ is odd}\}}\sum_{\ell\geq 1}\mu_k(\ell)\Big[(\mathfrak{m}(k+1))^{2\ell-2}\mathfrak{s}(k+1) + \big((\mathfrak{m}(k+1))^\ell - \mathfrak{m}(k)\big)^2\Big] \\ &+ \mathbf{1}_{\{k \text{ is even}\}}\sum_{\ell\geq 1}\mu_k(\ell)\Big[(1-\mathfrak{m}(k+1))^{2\ell-2}\mathfrak{s}(k+1) + \big((1-\mathfrak{m}(k+1))^\ell - (1-\mathfrak{m}(k))\big)^2\Big].\end{aligned}$$

These quantities can be computed once for all if one knows the parameters $\mu_0, \dots, \mu_K$ and $q_0, \dots, q_K$ of the model. If unknown, as is generally the case, these parameters can be evaluated numerically quite precisely, handling an important number of uniformly random matches. From these evaluations, we can derive some approximations of $\mathfrak{m}$ and $\mathfrak{s}$. However, the main problem is of course that in general, assuming that the true game is the realization of such a model is not very realistic.

Proof. First, Assumption 2.5 is satisfied, thanks to the classical branching property of Galton–Watson trees. Indeed, consider $S \in \mathcal{S}_f$ and $x \in L_S$ such that $\Pr(A_S) > 0$. Conditionally on A_S, we can write $\mathcal{T} = S \cup \bigcup_{x\in L_S}\mathcal{T}_x$ and the family $((\mathcal{T}_x, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}, x \in L_S)$ is independent by construction. Furthermore, for any $x \in L_S$, the law $G_{S,x}$ of $(\mathcal{T}_x, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ knowing A_S depends only on the depth $|x|$.

Consequently, there are $(\mathfrak{m}(k))_{k=0,\dots,K}$ and $(\mathfrak{s}(k))_{k=0,\dots,K}$ such that for $S \in \mathcal{S}_f$ and $x \in L_S$ with $\Pr(A_S) > 0$, $m(S,x) = \mathfrak{m}(|x|)$ and $s(S,x) = \mathfrak{s}(|x|)$.

If $|x| = K$, then x is necessarily a leave, so that $\Pr(x \in \mathcal{L}|A_S) = 1$, whence, by Lemma 4.1, $\mathfrak{m}(K) = m(S,x) = \Pr(x \in \mathcal{L}, R(x) = 1|A_S) = q_K$ and

$$\mathfrak{s}(K) = s(S,x) = \Pr(x \in \mathcal{L}, R(x) = 1|A_S)(1-m(S,x))^2 + \Pr(x \in \mathcal{L}, R(x) = 0|A_S)(m(S,x))^2,$$

which equals $q_K(1-q_K)^2 + (1-q_K)q_K^2 = q_K(1-q_K)$ as desired.

Finally, the proof can be completed by using Lemma 4.1 and that if $|x| = k \in \{0, \dots, K-1\}$ and if for example $t(x) = 0$ (*i.e.* k is odd), for any $\ell \geq 1$,

- $\Pr(x \in \mathcal{L}, R(x) = 1|A_S) = \mu_k(0)q_k$ and $\Pr(x \in \mathcal{L}, R(x) = 0|A_S) = \mu_k(0)(1-q_k)$,
- $\sum_{\mathbf{y}\subset\mathbb{C}_x, |\mathbf{y}|=\ell}\Pr(\mathcal{C}_x = \mathbf{y}|A_S) = \mu_k(\ell)$,
- $\Theta(S,x,\mathbf{y})$ and $\Gamma(S,x,\mathbf{y},y)$ depend only on k and on $\ell = |\mathbf{y}|$ and (since $t(x) = 0$), $\Theta(S,x,\mathbf{y}) = [\mathfrak{m}(k+1)]^\ell$ and $\Gamma(S,x,\mathbf{y},y) = \mathfrak{s}(k+1)[\mathfrak{m}(k+1)]^{2\ell-2} + [(\mathfrak{m}(k+1))^\ell - \mathfrak{m}(k)]^2$.

□

Remark 6.2. In Pearl's model [18], $\mathcal{T}$ is the deterministic regular tree with degree $d \geq 2$ and depth $K \geq 1$ and the family $(R(x))_{x\in\mathcal{L}}$ is i.i.d. Bernoulli(p)-distributed. This is a particular case of Example 6.1 with $\mu_0 = \cdots = \mu_{K-1} = \delta_d$ and $\mu_K = \delta_0$ and $q_K = p$ (the values of $(q_k)_{k=0,\dots,K-1}$ being irrelevant). One thus finds $\mathfrak{m}(K) = p$, $\mathfrak{s}(K) = p(1-p)$ and, for $k = 0, \dots, K-1$,

$$\begin{aligned}\mathfrak{m}(k) =& \mathbf{1}_{\{k \text{ is odd}\}}(\mathfrak{m}(k+1))^d + \mathbf{1}_{\{k \text{ is even}\}}(1-[1-\mathfrak{m}(k+1)]^d),\\ \mathfrak{s}(k) =& \mathbf{1}_{\{k \text{ is odd}\}}(\mathfrak{m}(k+1))^{2d-2}\mathfrak{s}(k+1) + \mathbf{1}_{\{k \text{ is even}\}}(1-\mathfrak{m}(k+1))^{2d-2}\mathfrak{s}(k+1).\end{aligned}$$

6.2. Inhomogeneous Galton–Watson trees of order two

Here we mention that we can also deal with random trees that enjoy some independence properties without being Galton–Watson trees. For example, the following model *of order* 2 allows one to build a broad class of random trees with non-increasing degree (along each branch), which might be useful for real games. It is possible to treat some models of higher order, but the functions m and s then become really tedious to compute theoretically and to approximate in practice.

We consider a family of probability measures on $\mathbb{N}$: μ_0 and $\mu_{k,d}$ for $k = 1, \dots, K$ and $d \geq 1$. We assume that $\mu_{K,d} = \delta_0$ for all $d \geq 1$ and K will represent the maximum depth of the tree.

We build the random tree $\mathcal{T}$ as follows: the root has $D_r \sim \mu_0$ children. Conditionally on D_r, all the children x of the root produce, independently, a number $D_x \sim \mu_{1,D_r}$ of children. Once everything is built up to generation $k \in \{0, \dots, K-1\}$, all the individuals x with $|x| = k$ produce, independently (conditionally on what is already built), a number $D_x \sim \mu_{k,D_{f(x)}}$ of children.

We also consider a family $q_0, \dots, q_K$ of numbers in $[0,1]$. Conditionally on $\mathcal{T}$, we assume that the family $(R(x))_{x\in\mathcal{L}}$ is independent and that $R(x) \sim \text{Bernoulli}(q_{|x|})$ for all $x \in \mathcal{L}$.

Example 6.3. With such a model, Assumption 2.5 is fulfilled, and for any $S \in S_f$ (with $\{r\} \subsetneq S$) such that $\Pr(A_S) > 0$ and any $x \in L_S$, we have we have $m(S,x) = \mathfrak{m}(|x|, |C^S_{f(x)}|)$ and $s(S,x) = \mathfrak{s}(|x|, |C^S_{f(x)}|)$, where $\mathfrak{m}$ and $\mathfrak{s}$ can be computed by backward induction as follows:

(i) $\mathfrak{m}(K,d) = q_K$ for all $d \geq 1$ and, for $k = 1, \dots, K-1$ and $d \geq 1$,

$$\mathfrak{m}(k,d) = \mu_{k,d}(0)q_k + \sum_{\ell\geq 1}\mu_{k,d}(\ell)\Big[\mathbf{1}_{\{k \text{ is odd}\}}(\mathfrak{m}(k+1,\ell))^\ell + \mathbf{1}_{\{k \text{ is even}\}}(1-[1-\mathfrak{m}(k+1,\ell)]^\ell)\Big].$$

(ii) $\mathfrak{s}(K,d) = q_K(1-q_K)$ for all $d \geq 1$ and, for $k = 1, \dots, K-1$ and $d \geq 1$,

$$\begin{aligned}\mathfrak{s}(k,d) =& \mu_{k,d}(0)[q_k(1-\mathfrak{m}(k,d))^2 + (1-q_k)(\mathfrak{m}(k,d))^2]\\ &+ \mathbf{1}_{\{k \text{ is odd}\}}\sum_{\ell\geq 1}\mu_{k,d}(\ell)\Big[(\mathfrak{m}(k+1,\ell))^{2\ell-2}\mathfrak{s}(k+1,\ell) + \big((\mathfrak{m}(k+1,\ell))^\ell - \mathfrak{m}(k,d)\big)^2\Big]\\ &+ \mathbf{1}_{\{k \text{ is even}\}}\sum_{\ell\geq 1}\mu_{k,d}(\ell)\Big[(1-\mathfrak{m}(k+1,\ell))^{2\ell-2}\mathfrak{s}(k+1,\ell) + \big((1-\mathfrak{m}(k+1,\ell))^\ell - (1-\mathfrak{m}(k,d))\big)^2\Big].\end{aligned}$$

We could easily express $m(\{r\}, r)$ and $s(\{r\}, r)$, but these values are useless as far as Algorithm 2.11 is concerned.

Proof. First, Assumption 2.5 is satisfied. Indeed, consider $S \in \mathcal{S}_f$ and $x \in L_S$ such that $\Pr(A_S) > 0$. Conditionally on A_S, we can write $\mathcal{T} = S \cup \bigcup_{x\in L_S}\mathcal{T}_x$ and the family $((\mathcal{T}_x, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}, x \in L_S)$ is independent by construction. Furthermore, for any $x \in L_S$, the law $G_{S,x}$ of $(\mathcal{T}_x, (R(y))_{y\in\mathcal{L}\cap\mathcal{T}_x}$ knowing A_S depends only on the depth $|x|$ and of $|C^S_{f(x)}|$ (except if $S = \{r\}$ and $x = r$).

Consequently, there are $(\mathfrak{m}(k,d))_{k=1,\dots,K,\,d\geq 1}$ and $(\mathfrak{s}(k,d))_{k=1,\dots,K,\,d\geq 1}$ such that for $S\in\mathcal{S}_f$ and $x\in L_S$ with $|C^S_{f(x)}|=d$, $\Pr(A_S)>0$, $m(S,x)=\mathfrak{m}(|x|,d)$ and $s(S,x)=\mathfrak{s}(|x|,d)$.

If $|x|=K$, then x is necessarily a leave, so that $\Pr(x\in\mathcal{L}|A_S)=1$, whence, by Lemma 4.1, $\mathfrak{m}(K,d)=m(S,x)=\Pr(x\in\mathcal{L},R(x)=1|A_S)=q_K$ and

$$\mathfrak{s}(K,d)=s(S,x)=\Pr(x\in\mathcal{L},R(x)=1|A_S)(1-m(S,x))^2+\Pr(x\in\mathcal{L},R(x)=0|A_S)(m(S,x))^2,$$

which equals $q_K(1-q_K)^2+(1-q_K)q_K^2=q_K(1-q_K)$ as desired.

Finally, the proof can be completed by using Lemma 4.1 and that if $|x|=k\in\{0,\dots,K-1\}$ (and $|C^S_{f(x)}|=d$) and if for example $t(x)=0$ (*i.e.* k is odd), for any $\ell\geq 1$,

- $\Pr(x\in\mathcal{L},R(x)=1|A_S)=\mu_{k,d}(0)q_k$ and $\Pr(x\in\mathcal{L},R(x)=0|A_S)=\mu_{k,d}(0)(1-q_k)$,
- $\sum_{\mathbf{y}\subset\mathbb{C}_x,|\mathbf{y}|=\ell}\Pr(\mathcal{C}_x=\mathbf{y}|A_S)=\mu_{k,d}(\ell)$,
- $\Theta(S,x,\mathbf{y})$ and $\Gamma(S,x,\mathbf{y},y)$ depend only on k,d and $\ell=|\mathbf{y}|$ and, if *e.g.* $t(x)=0$ (*i.e.* k is odd), $\Theta(S,x,\mathbf{y})=[\mathfrak{m}(k+1,\ell)]^\ell$ and $\Gamma(S,x,\mathbf{y},y)=\mathfrak{s}(k+1,\ell)[\mathfrak{m}(k+1,\ell)]^{2\ell-2}+[(\mathfrak{m}(k+1,\ell))^\ell-\mathfrak{m}(k,d)]^2$.

The last point uses that if $\mathcal{C}_x=\mathbf{y}$ with $|\mathbf{y}|=\ell$, then $|\mathcal{C}_{f(y)}|=|\mathcal{C}_x|=\ell$ for all $y\in\mathbf{y}$. □

6.3. Symmetric minimax values

Here we discuss the formulas introduced in Section 2.14.

We fix some value $a\in(0,1)$. For $S\in\mathcal{S}_f$, we build the family $(m_a(S,x))_{x\in S}$ by induction, starting from the root, setting $m_a(S,r)=a$ and, for all $x\in S\setminus\{r\}$,

$$m_a(S,x)=\mathbf{1}_{\{t(f(x))=0\}}[m_a(S,f(x))]^{1/|C^S_{f(x)}|}+\mathbf{1}_{\{t(f(x))=1\}}(1-[1-m_a(S,f(x))]^{1/|C^S_{f(x)}|}). \tag{6.1}$$

Observe that $m_a(S,x)$ actually depends only on K^S_x, *i.e.* $m_a(S,x)=m_a(K^S_x,x)$

Example 6.4. Consider a possibly random tree $\mathcal{T}\in\mathcal{S}_f$ enjoying the property that for any $S\in\mathcal{S}_f$ with leaves L_S, the family $(\mathcal{T}_x)_{x\in L_S}$ is independent conditionally on $A_S=\{S\subset\mathcal{T},\mathcal{D}_{L_S}=\emptyset\}$ as soon as $\Pr(A_S)>0$. Fix $a\in(0,1)$ and assume that conditionally on $\mathcal{T}$, the family $(R(y))_{y\in\mathcal{L}}$ is independent and $R(y)\sim\text{Bernoulli}(m_a(\mathcal{T},y))$ for all $y\in\mathcal{L}$. Then Assumption 2.5 is fulfilled, and for any $S\in S_f$ such that $\Pr(A_S)>0$ and any $x\in L_S$, we have $m(S,x)=m_a(S,x)$. We are generally not able to compute $s(S,x)$.

Observe that this is a qualitative *symmetry* assumption, saying that knowing $\mathcal{T}$, for any $v\in\mathcal{T}\setminus\mathcal{L}$, the family of the minimax values $(R(x),x\in\mathcal{C}_v)$ is i.i.d. Once this is assumed, the only remaining parameter is the mean minimax rating of the root (which we set to a).

Once the value $a=m_a(\mathcal{T},r)$ is chosen (even if not knowing $\mathcal{T}$), it is easy to make the algorithm compute the necessary values of m_a, as explained in Section 2.14: each time a new node x of $\mathcal{T}$ is created by the algorithm, we can compute $m_a(\mathcal{K}_x,x)$ from $m_a(\mathcal{K}_{f(x)},f(x))$ and $|\mathcal{C}_{f(x)}|$.

Proof. Assumption 2.5 is satisfied because (a) the random tree $\mathcal{T}$ is supposed to satisfy the required independence property and (b) conditionally on $\mathcal{T}$, for any $x\in\mathcal{L}$, $m_a(\mathcal{T},x)$ depends only on $\mathcal{K}_x$.

It remains to verify that $m(S,x)=m_a(S,x)$ for all $S\in\mathcal{S}_f$ of which x is a leave.

We first show by backward induction that $\Pr(R(x)=1|\mathcal{T})=m_a(\mathcal{T},x)$ for all $x\in\mathcal{T}$. First, this is obvious if $x\in\mathcal{L}$ by construction. Next, if this is true for all the children (in $\mathcal{T}$) of $x\in\mathcal{T}\setminus\mathcal{L}$ with *e.g.* $t(x)=1$, then we have $\Pr(R(x)=1|\mathcal{T})=1-\prod_{y\in\mathcal{C}_x}\Pr(R(y)=0|\mathcal{T})=1-\prod_{y\in\mathcal{C}_x}(1-m_a(\mathcal{T},y))$. We first used that the family $(R(y))_{y\in\mathcal{C}_x}$ is independent conditionally on $\mathcal{T}$ and then the induction assumption. Using finally (6.1) (recall that $t(x)=1$ and that $f(y)=x$ for all $y\in\mathcal{C}_x$), we find $\Pr(R(x)=1|\mathcal{T})=1-\prod_{y\in\mathcal{C}_x}(1-[1-(1-m_a(\mathcal{T},x))^{1/|\mathcal{C}_x|}])=m_a(\mathcal{T},x)$.

Fix now $S\in S_f$ such that $\Pr(A_S)>0$, where $A_S=\{S\subset\mathcal{T},\mathcal{D}_{L_S}=\emptyset\}$ and $x\in L_S$. Since $A_S\in\sigma(\mathcal{T})$, we deduce that $m(S,x)=\Pr(R(x)=1|A_S)=\mathbb{E}[\Pr(R(x)=1|\mathcal{T})|A_S]=\mathbb{E}[m_a(\mathcal{T},x)|A_S]$. But we know that

$m_a(\mathcal{T},x)=m_a(\mathcal{K}_x,x)$. Since $\mathcal{K}_x=K_x^S$ on A_S, we conclude that $m(S,x)=m_a(K_x^S,x)=m_a(S,x)$ as desired. □

Let us mention that Pearl's model, which we already interpreted as a particular case of Example 6.1, can also be seen as a particular case of Example 6.4, where we can furthermore compute s.

Remark 6.5. Consider again Pearl's model [18]: $\mathcal{T}$ is the deterministic regular tree with degree $d\geq 2$ and depth $K\geq 1$ and the family $(R(x))_{x\in\mathcal{L}}$ is i.i.d. Bernoulli(p)-distributed. Then we already know that for all $S\in\mathcal{S}_f$ and $x\in L_S$ such that $\Pr(A_S)>0$, we have $m(S,x)=\mathfrak{m}(|x|)$ and $s(S,x)=\mathfrak{s}(|x|)$, with $\mathfrak{m}$ and $\mathfrak{s}$ as in Remark 6.2. One then also has, for all $S\in\mathcal{S}_f$ and $x\in L_S$ such that $\Pr(A_S)>0$, if $x\neq r$, denoting by $v=f(x)$ and $S_v=S\setminus C_v^S$,

$$\begin{aligned}m(S,x)=&\mathbf{1}_{\{t(v)=0\}}[m(S_v,v)]^{1/|C_v^S|}+\mathbf{1}_{\{t(v)=1\}}(1-[1-m(S_v,v)]^{1/|C_v^S|}),\\ s(S,x)=&\Big(\mathbf{1}_{\{t(v)=0\}}m(S_v,v)+\mathbf{1}_{\{t(v)=1\}}[1-m(S_v,v)]\Big)^{2(|C_v^S|-1)}s(S_v,v).\end{aligned}$$

Setting $a=\mathfrak{m}(0)$, which can be computed from p,K,d, we thus have $m(S,x)=m_a(S,x)$ as defined in (6.1), and we can compute $s(S,x)$. Note that it is not necessary to determine precisely $s(\{r\},r)$: we can set $s(\{r\},r)=1$ (or any other positive constant) by Remark 2.12.

Indeed, the above formulas are nothing but a complicated version of the ones in Remark 6.2, since we have $m(S,x)=\mathfrak{m}(|x|)$, $s(S,x)=\mathfrak{s}(|x|)$, $|C_v^S|=d$, $m(S_v,v)=\mathfrak{m}(|v|)$ and $s(S_v,v)=\mathfrak{s}(|v|)$.

There are other cases where we can characterize s, which should thus be numerically computable.

Example 6.6. Assume that $\mathcal{T}$ is a homogeneous Galton–Watson tree with reproduction law μ such that $\sum_{\ell\geq 1}\ell\mu(\ell)\leq 1$ and $\mu(0)>0$, so that $\mathcal{T}$ is a.s. finite. Fix $a\in(0,1)$ and assume that conditionally on $\mathcal{T}$, the family $(R(y))_{y\in\mathcal{L}}$ is independent and that $R(y)\sim\text{Bernoulli}(m_a(\mathcal{T},y))$ for all $y\in\mathcal{L}$. Then for all $S\in S_f$ such that $\Pr(A_S)>0$ and all $x\in L_S$, we have $m(S,x)=m_a(S,x)$ and $s(S,x)=\mathfrak{s}(1-m_a(S,x))\mathbf{1}_{\{t(x)=0\}}+\mathfrak{s}(m_a(S,x))\mathbf{1}_{\{t(x)=1\}}$, where $\mathfrak{s}$ is the unique function from $[0,1]$ into $[0,1]$ such that for all $\alpha\in[0,1]$,

$$\mathfrak{s}(\alpha)=\mu(0)\alpha(1-\alpha)+\sum_{\ell\geq 1}\mu(\ell)(1-\alpha)^{2(\ell-1)/\ell}\mathfrak{s}((1-\alpha)^{1/\ell}). \tag{6.2}$$

Proof. We already know from Example 6.4 that Assumption 2.5 is satisfied and that $m(S,x)=m_a(S,x)$. Next, (6.2) has a unique solution because the map $F:E\mapsto E$, where E is the set of all functions from $[0,1]$ into $[0,1]$, defined by

$$F(\mathfrak{s})(\alpha)=\mu(0)\alpha(1-\alpha)+\sum_{\ell\geq 1}\mu(\ell)(1-\alpha)^{2(\ell-1)/\ell}\mathfrak{s}((1-\alpha)^{1/\ell}),$$

is a contraction. Indeed, $||F(\mathfrak{s}_1)-F(\mathfrak{s}_2)||_\infty\leq\kappa||\mathfrak{s}_1-\mathfrak{s}_2||_\infty$ with $\kappa=\sum_{\ell\geq 1}\mu(\ell)<1$.

Let us denote by $\mathfrak{s}(a)=s(\{r\},r)$, which clearly depends only on a (and μ).

For any $S\in S_f$ such that $\Pr(A_S)>0$ and $x\in L_S$, we have $s(S,x)=\mathfrak{s}(m_a(S,x))$ if $t(x)=1$ and $s(S,x)=\mathfrak{s}(1-m_a(S,x))$ if $t(x)=0$. Indeed, the law of $(\mathcal{T}_x,(R(u))_{u\in\mathcal{L}\cap\mathcal{T}_x})$ conditionally on A_S is the same as that of $(\mathcal{T},(R(u))_{u\in\mathcal{L}})$ (re-rooted at x), replacing a by $m_a(S,x)$: $\mathcal{T}_x$ is a Galton–Watson tree with reproduction law μ and, knowing A_S and $\mathcal{T}_x$, one easily checks that $m_a(\mathcal{T},y)=m_{m_a(S,x)}(\mathcal{T}_x,y)$ for all $y\in\mathcal{L}\cap\mathcal{T}_x$. Hence we have $s(S,x)=\mathfrak{s}(m_a(S,x))$ if $t(x)=1$. If now $t(x)=0$, we see that $s(S,x)=\mathfrak{s}(1-m_a(S,x))$ by exchanging the roles of the two players.

It remains to verify that $\mathfrak{s}$ satisfies (6.2). To this end, it suffices to apply the formula of Lemma 4.1 concerning s with $S=\{r\}$ (whence $A_S=\Omega$) and $x=r$ (with $t(r)=1$) and to observe that

- $s(\{r\},r)=\mathfrak{s}(a)$ and $m(\{r\},r)=a$,

- $\Pr(r \in \mathcal{L}, R(r) = 1) = \mu(0)a$ and $\Pr(r \in \mathcal{L}, R(r) = 0) = \mu(0)(1-a)$,
- $\sum_{\mathbf{y} \subset \mathbb{C}_r, |\mathbf{y}|=\ell} \Pr(\mathcal{C}_r = \mathbf{y}) = \mu(\ell)$,
- for any $\mathbf{y} \subset \mathbb{C}_r$ with $|\mathbf{y}| = \ell \geq 1$, conditionally on $\mathcal{C}_r = \mathbf{y}$, for all $y \in \mathbf{y}$, we have $m_a(\{r\} \cup \mathbf{y}, y) = 1 - (1-a)^{1/\ell}$ and thus, since $t(y) = 1$, $s(\{r\} \cup \mathbf{y}, y) = \mathfrak{s}((1-a)^{1/\ell})$, whence $\Gamma(\{r\}, r, \mathbf{y}, y) = \mathfrak{s}((1-a)^{1/\ell})(1-a)^{2(\ell-1)/\ell} + [(1-a)^{\ell/\ell} - (1-a)]^2 = (1-a)^{2(\ell-1)/\ell}\mathfrak{s}((1-a)^{1/\ell})$. □

It does not seem easy to solve (6.2). However, here is one possibility.

Remark 6.7. Assume that $\mathcal{T}$ is a homogeneous Galton–Watson tree with reproduction law $\mu = (1-p)\delta_0 + p\delta_d$, with $d \geq 2$ and $p \in (0, 1/d]$. Consider the unique solution $a_0 \in (0,1)$ to $a_0 = (1-a_0)^{1/d}$. Assume that conditionally on $\mathcal{T}$, the family $(R(y))_{y \in \mathcal{L}}$ is independent and that $R(y) \sim \text{Bernoulli}(m_{a_0}(\mathcal{T}, y))$ for all $y \in \mathcal{L}$. Then for all $S \in S_f$ such that $\Pr(A_S) > 0$ and all $x \in L_S$, we have $m(S,x) = m_{a_0}(S,x)$ and $s(S,x) = \mathfrak{s}(a_0) = (1-p)a_0^{d+1}/[1 - pa_0^{2d-2}]$ is constant.

Indeed, in such a case (6.2) rewrites as

$$\mathfrak{s}(\alpha) = (1-p)\alpha(1-\alpha) + p(1-\alpha)^{2(d-1)/d}\mathfrak{s}((1-\alpha)^{1/d}),$$

whence $\mathfrak{s}(a_0) = (1-p)a_0^{d+1}/[1-pa_0^{2d-2}]$. Also, one easily checks that for any $S \in \mathcal{S}_f$ such that $\Pr(A_S) > 0$ (so that S is d-regular) and any $x \in L_S$, we have $m_{a_0}(S,x) = a_0$ if $t(x) = 1$ and $m_{a_0}(S,x) = 1 - a_0$ if $t(x) = 0$. This of course uses (6.1) and that $1 - (1-a_0)^{1/d} = 1 - a_0$ and $(1-a_0)^{1/d} = a_0$. Consequently, $s(S,x)$ always equals $\mathfrak{s}(a_0)$: $s(S,x) = \mathfrak{s}(m_{a_0}(S,x)) = \mathfrak{s}(a_0)$ if $t(x) = 1$ and $s(S,x) = \mathfrak{s}(1 - m_{a_0}(S,x)) = \mathfrak{s}(a_0)$ if $t(x) = 0$.

7. Global optimality fails

Proof of Remark 2.15. We assume here that $\mathcal{T}$ is the binary tree with depth 3. We thus have the eight leaves 111, 112, 121, 122, 211, 212, 221, 222 (recall Sect. 2.1). We also assume that the family $(R(x))_{x \in \mathcal{L}}$ is i.i.d. with common law Bernoulli(1/2).

Observe that $\mathcal{T}$ can be seen as an inhomogeneous Galton–Watson tree with reproduction laws $\mu_0 = \mu_1 = \mu_2 = \delta_2$ and $\mu_3 = \delta_0$. Applying Example 6.1 (with $q_3 = 1/2$, the values of q_0, q_1, q_2 being irrelevant), we can compute the functions m and s: for any $S \in \mathcal{S}_f$ such that $\Pr(A_S) > 0$ and $x \in L_S$, we have $m(S,x) = \mathfrak{m}(|x|)$ and $s(S,x) = \mathfrak{s}(|x|)$, where

$$\mathfrak{m}(1) = \frac{9}{16}, \quad \mathfrak{m}(2) = \frac{3}{4}, \quad \mathfrak{m}(3) = \frac{1}{2}, \quad \mathfrak{s}(1) = \frac{9}{256}, \quad \mathfrak{s}(2) = \frac{1}{16}, \quad \mathfrak{s}(3) = \frac{1}{4}.$$

The value of $\mathfrak{m}(0)$ and $\mathfrak{s}(0)$ are not useful to the algorithm. Let us however notice that $\mathbb{E}[R(r)] = \mathfrak{m}(0) = 207/256$.

By symmetry, we can replace the uniformly random matches (starting from some z) used in any admissible algorithm, see Definition 2.3, by the visit of any deterministic leave (under z), without changing (at all) the performance of the algorithm. With this slight modification, some tedious computations show that Algorithm 2.11, using the above function m and s, leads to the following strategy (and results) for the three first steps.

Visit the leave $x_1 = 111$.
If $R(x_1) = 1$ (whence $R_1(r) = 228/256$), then
  { visit the leave $x_2 = 121$.
    If $R(x_2) = 1$ (whence $R_2(r) = 1$), then
      { stop here (or visit any other leave). We have $R_3(r) = 1$. }
    If $R(x_2) = 0$ (whence $R_2(r) = 200/256$), then
      { visit the leave $x_3 = 122$. We have $R_3(r) = 1$ if $R(x_3) = 1$ and $R_3(r) = 144/256$
        if $R(x_3) = 0$. } }
If $R(x_1) = 0$ (whence $R_1(r) = 186/256$), then
  { visit the leave $x_2 = 112$.

```
If R(x_2) = 1 (whence R_2(r) = 228/256), then
  { visit the leave 121. We have R_3(r) = 1 if R(x_3) = 1 and R_3(r) = 200/256
    if R(x_3) = 0. }
If R(x_2) = 0 (whence R_2(r) = 144/256), then
  { visit the leave x_3 = 211. We have R_3(r) = 192/256 if R(x_3) = 1
    and R_3(r) = 96/256 if R(x_3) = 0. } }
```

Noting that $\mathbb{E}[(R_n(r)-R(r))^2] = \mathbb{E}[(R(r))^2] - \mathbb{E}[(R_n(r))^2] = 207/256 - \mathbb{E}[(R_n(r))^2]$ because $\mathbb{E}[R(r)R_n(r)] = \mathbb{E}[(R_n(r))^2]$ (since $R_n(r) = \mathbb{E}[R(r)|\mathcal{F}_n]$) and since $\mathbb{E}[(R(r))^2] = \mathbb{E}[R(r)] = \mathfrak{m}(0) = 207/256$, we conclude that

$$\mathbb{E}[(R_1(r)-R(r))^2] = \frac{207}{256} - \frac{1}{2}\left[\left(\frac{228}{256}\right)^2 + \left(\frac{186}{256}\right)^2\right] = \frac{4851}{32768},$$

$$\mathbb{E}[(R_2(r)-R(r))^2] = \frac{207}{256} - \frac{1}{4}\left[1 + \left(\frac{200}{256}\right)^2 + \left(\frac{228}{256}\right)^2 + \left(\frac{144}{256}\right)^2\right] = \frac{2107}{16384},$$

$$\mathbb{E}[(R_3(r)-R(r))^2] = \frac{207}{256} - \frac{1}{8}\left[1+1+1+\left(\frac{144}{256}\right)^2 + 1 + \left(\frac{200}{256}\right)^2 + \left(\frac{192}{256}\right)^2 + \left(\frac{96}{256}\right)^2\right] = \frac{859}{8192}.$$

The following strategy, that we found with the help of a computer, is less efficient in two steps but more efficient in three steps. We use the notation $\tilde{R}_n(r)$ as in the statement.

```
Visit the leave x̃_1 = 111.
If R(x̃_1) = 1 (whence R̃_1(r) = 228/256), then
  { visit the leave x̃_2 = 121.
    If R(x̃_2) = 1 (whence R̃_2(r) = 1), then
      { stop here (or visit any other leave). We have R̃_3(r) = 1. }
    If R(x̃_2) = 0 (whence R̃_2(r) = 200/256), then
      { visit the leave x̃_3 = 122. We have R̃_3(r) = 1 if R(x̃_3) = 1 and R̃_3(r) = 144/256
        if R(x̃_3) = 0. } }
If R(x̃_1) = 0 (whence R̃_1(r) = 186/256), then
  { visit the leave x̃_2 = 211.
    If R(x̃_2) = 1 (whence R̃_2(r) = 216/256), then
      { visit the leave 221. We have R̃_3(r) = 1 if R(x̃_3) = 1 and R̃_3(r) = 176/256
        if R(x̃_3) = 0. }
    If R̃(x̃_2) = 0 (whence R̃_2(r) = 156/256), then
      { visit the leave x̃_3 = 112. We have R̃_3(r) = 216/256 if R(x̃_3) = 1
        and R̃_3(r) = 96/256 if R(x̃_3) = 0. } }
```

We conclude, using the same argument as previously, that

$$\mathbb{E}[(\tilde{R}_1(r)-R(r))^2] = \frac{207}{256} - \frac{1}{2}\left[\left(\frac{228}{256}\right)^2 + \left(\frac{186}{256}\right)^2\right] = \frac{4851}{32768},$$

$$\mathbb{E}[(\tilde{R}_2(r)-R(r))^2] = \frac{207}{256} - \frac{1}{4}\left[1 + \left(\frac{200}{256}\right)^2 + \left(\frac{216}{256}\right)^2 + \left(\frac{156}{256}\right)^2\right] = \frac{2215}{16384},$$

$$\mathbb{E}[(\tilde{R}_3(r)-R(r))^2] = \frac{207}{256} - \frac{1}{8}\left[1+1+1+\left(\frac{144}{256}\right)^2 + 1 + \left(\frac{176}{256}\right)^2 + \left(\frac{216}{256}\right)^2 + \left(\frac{96}{256}\right)^2\right] = \frac{847}{8192}.$$

The proof is complete. □

8. Numerical results

8.1. Numerical problems

Algorithm 2.11 is subjected to numerical problems due to the fact that it proceeds to a high number of multiplications of reals in $[0,1]$. For example, the algorithm continuously computes products of the form $\prod_{k=1}^{d} r_k$ and $1-\prod_{k=1}^{d}(1-r_k)$, with $r_1,\dots,r_d\in[0,1]$. If coded naively, it immediately finds 0 or 1 and does not work at all. We overcame such problems with the change of variables $\phi(r)=\log[r/(1-r)]$. Everywhere, we used $\phi(R)$, $\phi(U)$ and $\phi(Z)$ instead of R, U and Z (we mean, concerning the values $R_n(x)$, $U_n(x)$ and the $Z_n(x)$). Actually, for large games, some numerical problems persist: at some steps, we have numerically $(R_{n+1}(r),U_{n+1}(r),Z_{n+1}(r))=(R_n(r),U_n(r),Z_n(r))$ (even after the change of variables), which should never be the case. However, the above trick eliminates most of them. Instead of using ϕ, one could manipulate simultaneously $\log r$ and $\log(1-r)$. This would be more or less equivalent, the use of ϕ is just slightly more concise.

We used the following expressions. We carefully separated different cases, because *e.g.* for u very large (say, $u\ge 750$), the computer answers $\log(1+\exp(u))=+\infty$ but $u+\log(1+\exp(-u))=u$, these two quantities being theoretically equal. Consider $r,s\in[0,1]$ and $u=\phi(r)$, $v=\phi(s)$. We *e.g.* assume that $0\le s\le r\le 1$ (whence $-\infty\le v\le u\le+\infty$) and we naturally allow $\phi(0)=-\infty$, $\phi(1)=+\infty$, $\exp(-\infty)=0$, etc. Observe that $r+s\le 1$ if and only if $u+v\le 0$.

(a) $\phi(1-r)=-\phi(r)$,

(b) $$\phi(rs)=\begin{cases} u+v-\log(1+e^u+e^v) & \text{if } u<0,\\ +\infty & \text{if } v=+\infty,\\ v-\log(1+e^{-u}+e^{v-u}) & \text{if } v<+\infty \text{ and } u\ge 0.\end{cases}$$

(c) $$\phi(r+s)=\begin{cases} -\infty & \text{if } u=-\infty,\\ +\infty & \text{if } v=-\infty \text{ and } u=+\infty,\\ u+\log(1+e^{v-u}+2e^v)-\log(1-e^{u+v}) & \text{if } u>-\infty \text{ and } u+v\le 0.\end{cases}$$

(d) $$\phi(s/r)=\begin{cases} +\infty & \text{if } u=v,\\ v & \text{if } v<u=+\infty,\\ \log(1+e^u)-u+v-\log(1-e^{v-u}) & \text{if } v<u<0,\\ \log(1+e^{-u})+v-\log(1-e^{v-u}) & \text{if } 0\le u<+\infty \text{ and } v<u.\end{cases}$$

(e) $$\log r=\begin{cases} u-\log(1+e^u) & \text{if } u<0,\\ -\log(1+e^{-u}) & \text{if } u\ge 0.\end{cases}$$

We can compute $\phi(r-s)=\phi(r(1-s/r))$ from u,v using (a), (b) and (d). For $r_1,\dots,r_d\in[0,1]$, we can compute $\phi(\prod_1^d r_k)$ from the values of $u_k=\phi(r_k)$ recursively using (b), and we can get $\phi(1-\prod_1^d(1-r_k))=-\phi(\prod_1^d(1-r_k))$ using furthermore (a). Etc.

8.2. The algorithms

We will make play some versions of our algorithm against the two versions of MCTS recalled in Appendix A on a few real games (variations of Connect Four) and on Pearl's model.

We call MCTS(a,b) Algorithm A.2 and MCTS$'(a,b)$ Algorithm A.1 with the parameters $a,b>0$.

We call GW (resp. GW2) Algorithm 2.11 with the functions m and s of Example 6.1 (resp. Example 6.3), assuming that the game is the realization of an inhomogeneous Galton–Watson tree (resp. inhomogeneous Galton–Watson tree of order 2). The functions $\mathfrak{m}$ and $\mathfrak{s}$ are computed as follows, for example in the case of GW. We handle a large number (10^8) of uniformly random matches starting from the initial configuration of the game, this allows us to estimate $\mu_0,\dots,\mu_K$. The values $q_0,\dots,q_K$ are trivial for Connect Four, since we simply have $q_k=\mathbf{1}_{\{k \text{ is odd}\}}$. We then use the formulas stated in Example 6.1. We obtain rather stable results. Of course, this is done once for all for each game.

We call Sym Algorithm 2.11 with the function $m = m_a$ defined by (6.1), with the choice $a = 1/2$, and with $s \equiv 1$. Recall that $a \in (0,1)$ is the expected minimax rating of the root. This is the most simple and universal algorithm, although not fully theoretically justified (see however Rem. (6.7) in Sect. 6.3).

We finally call SymP Algorithm 2.11 with the functions m and s defined in Remark 6.5, here again with $a = 1/2$. This is the theoretical algorithm furnished by our study in the case of Pearl's game (if $a = 1/2$) and it is precisely the same as GW in this case, see Remarks 6.2 and 6.5.

Let us now give a few precisions.

(i) In all the experiments below, each algorithm keeps the information provided by its own simulations handled to decide its previous moves. In practice, this at most doubles the quantity of information (when compared to the case where we would delete everything at each new move), because most of the previous simulations led to other positions.

(ii) Concerning Sym and SymP, we actually use $a = 1/2$ as expected minimax value of the *true* root of the game, that is the true initial position. When in another configuration x, we use $m_a(\mathcal{T}, x)$ (with $a = 1/2$, here $\mathcal{T}$ is the tree representing the whole game) as expected minimax value of the *current* root x (*i.e.* the current position of the game). Such a value is automatically computed when playing the game.

8.3. The numerical experiments

First, let us mention that we handled many trials using GW and GW2. They almost always worked less well than Sym concerning Connect Four, so we decided not to present those results. Also, concerning Sym and SymP, we tried other values for the expected minimax value $a \in (0,1)$ of the root without observing significantly better results, so we always use $a = 1/2$. Similarly, we experimented other values for $b \in \mathbb{R}$ (see Sect. 2.14, Sym corresponds to the case $b = 0$ and SymP to the case where $b = 2$), here again without clear success.

In each subsection below (except Sect. 8.11), which concerns one given game, we proceed as follows.

For each given amount of time per move, we first fit the parameters of MCTS. To this aim, we perform a championship involving MCTS(a,b), for all $a = k/2$, $b = \ell/2$ with $1 \le k \le \ell \le 10$ (we thus have 55 players). Each player competes 40 times against all the other ones (20 times as first player, 20 times as second one), and we select the player with the highest number of victories. Observe that each player participates to 2160 matches. The resulting best player is rather unstable, but we believe this is due to the fact that the competition is very tight among a few players. Hence even if we do not select the true best player, we clearly select a very good one. Note that we impose $a \le b$ because, after many trials allowing $a > b$, the best player was always of this shape.

Of course, we do exactly the same thing to fit the parameters of MCTS′.

Then, we make our algorithm (Sym or SymP) compete against the best MCTS and the best MCTS′, 10000 times as first player and 10000 times as second one.

Also, we indicate the (rounded) mean number of iterations made by each algorithm *at the first move*. This sometimes looks ridiculous: when *e.g.* playing a version of Connect Four with large degree with 1 millisecond per move, this mean number of iterations is 8 for Sym and 11 for MCTS. However, after sufficiently many moves, this mean number of iterations becomes much higher. In other words, the algorithms more or less play at random at the beginning, but become more and more *clever* as the game progresses. So in some sense, the algorithm that wins is the one becoming clever before the other.

Finally, let us explain how to read the tables below, which are all of the same shape. For example, the first table, when playing a large Pearl game, is as follows.

$t = 1$ ms. MCTS(3.5,5): 498, MCTS′(2,3): 442, SymP: 163	
SymP *vs.* MCTS(3.5,5)	SymP *vs.* MCTS′(2,3)
5882/4118(0), 4340/5660(0)	**4992/5008(0), 5221/4779(0)**

Each player had 1 millisecond to decide each of its moves. The championships were won by MCTS(3.5,5) and MCTS′(2,3). When playing its first move, MCTS(3.5,5) (resp. MCTS′(2,3), resp. SymP) proceeded in mean to 498 (resp. 442, resp. 163) iterations.

When playing first, SymP won 5882 times and lose 4118 times against MCTS(3.5,5), and there has been 0 draw. When playing first, MCTS(3.5,5) won 4340 times and lose 5660 times against SymP, and there has been 0 draw.

When playing first, SymP won 4992 times and lose 5008 times against MCTS′(2,3), and there has been 0 draw. When playing first, MCTS′(2,3) won 5221 times and lose 4779 times against SymP, and there has been 0 draw.

Finally, the results **in bold** mean that our algorithm (Sym or SymP) is beaten.

8.4. Large Pearl game

We first consider Pearl's game, that is the game of Example 6.2 with the regular tree with degree $d = 2$ and depth $K = 32$, with i.i.d. Bernoulli(p) random variables on the leaves and with p such that $\mathbb{E}[R(r)] = 1/2$ (p is easily computed numerically using the formula of Remark 6.2 and that $p = \mathfrak{m}(K)$ and $1/2 = \mathfrak{m}(0)$).

Here and in the next subsection, to be as fair as possible, in each cell of the tables below, each match is played in both senses on a given realization of the model, so that if the two opponents were playing perfectly, one would find twice the same results in each cell.

$t = 1$ ms. MCTS(3.5,5): 498, MCTS′(2,3): 442, SymP: 163	
SymP *vs.* MCTS(3.5,5)	SymP *vs.* MCTS′(2,3)
5882/4118(0), 4340/5660(0)	**4992/5008(0), 5221/4779(0)**

$t = 2$ ms. MCTS(3.5,5): 717, MCTS′(3,4.5): 579, SymP: 238	
SymP *vs.* MCTS(3.5,5)	SymP *vs.* MCTS′(3,4.5)
5751/4249(0), 4267/5733(0)	**4727/5273(0), 5372/4628(0)**

$t = 4$ ms. MCTS(4,4.5): 1137, MCTS′(3.5,5): 848, SymP: 395	
SymP *vs.* MCTS(4,4.5)	SymP *vs.* MCTS′(3.5,5)
5921/4079(0), 4044/5956(0)	**4676/5324(0), 5306/4694(0)**

$t = 8$ ms. MCTS(3,3.5): 2210, MCTS′(3.5,5): 1561, SymP: 693	
SymP *vs.* MCTS(3,3.5)	SymP *vs.* MCTS′(3.5,5)
5926/4074(0), 4080/5920(0)	**4641/6359(0), 5438/4562(0)**

$t = 16$ ms. MCTS(4,5): 4229, MCTS′(4,4.5): 2893, SymP: 1307	
SymP *vs.* MCTS(4,5)	SymP *vs.* MCTS′(4,4.5)
5998/4002(0), 4012/5988(0)	**5000/5000(0), 5112/4888(0)**

$t = 32$ ms. MCTS(4,4.5): 8277, MCTS′(4.5,5): 5922, SymP: 2473	
SymP *vs.* MCTS(4,4.5)	SymP *vs.* MCTS′(4.5,5)
6154/3846(0), 3781/6219(0)	5059/4941(0), 4923/5077(0)

$t = 64$ ms. MCTS(4.5,4.5): 15983, MCTS′(3.5,4.5): 14399, SymP: 4766	
SymP *vs.* MCTS(4.5,4.5)	SymP *vs.* MCTS′(3.5,4.5)
6380/3620(0), 3628/6372(0)	5244/4756(0), 4800/5200(0)

We observe that SymP is better than MCTS but beats MCTS′ only when the amount of time per play is high enough.

For this game, SymP performs around 4 times fewer iterations per unit of time than MCTS.

8.5. Small Pearl game

We next consider a much smaller Pearl game: as previously, $d = 2$ and p is chosen is such that $\mathbb{E}[R(r)] = 1/2$, but the depth of the tree is $K = 16$ (instead of $K = 32$).

Here, and only here, due to the smallness of the game, we needed to modify the way we fit the parameters of MCTS and MCTS′. Namely, we perform a championship involving MCTS(a, b), for all $a = k/2$, $b = \ell/2$ with $1 \leq k \leq \ell \leq 20$ (we thus have 210 players). Each player competes 20 times against all the other ones (10 times as first player, 10 times as second one), and we select the player with the highest number of victories. Each player participates to 4180 matches.

Proceeding as in the previous subsection, we found the following results.

$t = 1$ ms. MCTS(5,8.5): 1100, MCTS′(5,7.5): 954, SymP: 301	
SymP *vs.* MCTS(5,8.5)	SymP *vs.* MCTS′(5,7.5)
5242/4758(0), 3966/6034(0)	**4334/5666(0), 4868/5132(0)**

$t = 2$ ms. MCTS(6,10): 1637, MCTS′(5.5,9.5): 1584, SymP: 444	
SymP *vs.* MCTS(6,10)	SymP *vs.* MCTS′(5.5,9.5)
5104/4896(0), 4027/5973(0)	**4346/5654(0), 4899/5101(0)**

$t = 4$ ms. MCTS(5.5,9): 2829, MCTS′(5.5,9): 2776, SymP: 725	
SymP *vs.* MCTS(5.5,9)	SymP *vs.* MCTS′(5.5,9)
5307/4693(0), 3953/6047(0)	**4296/5704(0), 4797/5203(0)**

$t = 8$ ms. MCTS(7,9.5): 4876, MCTS′(5.5,9): 5745, SymP: 1128	
SymP *vs.* MCTS(7,9.5)	SymP *vs.* MCTS′(5.5,9)
4968/5032(0), 3817/6183(0)	**4434/5566(0), 4826/5174(0)**

$t = 16$ ms. MCTS(6.5,7.5): 10303, MCTS′(4.5,6): 12729, SymP: 1692	
SymP *vs.* MCTS(6.5,7.5)	SymP *vs.* MCTS′(4.5,6)
5132/4868(0), 4519/5481(0)	4950/5050(0), 4943/5057(0)

$t = 32$ ms. MCTS(6.5,8.5): 25929, MCTS′(7.5,9): 27585, SymP: 1840	
SymP *vs.* MCTS(6.8,8.5)	SymP *vs.* MCTS′(7.5,9)
5034/4966(0), 5010/4990(0)	5083/4917(0), 5083/4917(0)

Here it seems that SymP almost always finds the winning strategy at the first move with 16 ms, this explains why the number of iterations is so small (at 16 and 32 ms): SymP stops before 16 ms are elapsed. We thus believe it always finds it with 32 ms.

At 32 ms, it seems that MCTS′ also always found the winning strategy among 5083 times it started the game with a possible winning strategy. MCTS missed 24 times the winning strategy among 5034 times it started the game with a possible winning strategy.

Observe that SymP is better than MCTS but, here again, beats MCTS′ only when the amount of time per play is high enough.

Also, even if the game is theoretically fair (because $\mathbb{E}[R(r)] = 1/2$), it seems easier, for all the algorithms, to find the winning strategy when being the second player (this can be observed until 8 ms). This might be explained by the fact that the second player is the one playing the last move.

Here also, SymP performs around 4 times fewer iterations per unit of time than MCTS.

8.6. Standard Connect Four

We now play Connect Four in its usual version: we have 7 columns, 6 lines, and the goal is to connect (horizontally, vertically, or diagonally) 4 discs.

$t = 1$ ms. MCTS(1.5,1.5): 53, MCTS′(1,1): 50, Sym: 33	
Sym *vs.* MCTS(1.5,1.5)	Sym *vs.* MCTS′(1,1)
4174/5809(17), 6806/3146(48)	**4066/5904(30), 6983/2966(51)**

$t = 2$ ms. MCTS(1,1): 82, MCTS′(2,2.5): 76, Sym: 48	
Sym *vs.* MCTS(1,1)	Sym *vs.* MCTS′(2,2.5)
3099/6873(28), 7558/2392(50)	**3184/6799(17), 7632/2317(51)**

$t = 4$ ms. MCTS(2,2): 137, MCTS′(1.5,1.5): 124, Sym: 81	
Sym *vs.* MCTS(2,2)	Sym *vs.* MCTS′(1.5,1.5)
2460/7527(13), 7989/1966(45)	**2485/7500(15), 7908/2043(49)**

$t = 8$ ms. MCTS(3,3.5): 254, MCTS′(3.5,4): 225, Sym: 146	
Sym *vs.* MCTS(3,3.5)	Sym *vs.* MCTS′(3.5,4)
1923/8074(3), 8163/1791(46)	**2004/7991(5), 8211/1751(38)**

$t = 16$ ms. MCTS(4,4.5): 491, MCTS′(3.5,4): 441, Sym: 280	
Sym *vs.* MCTS(4,4.5)	Sym *vs.* MCTS′(3.5,4)
1353/8644(3), 8341/1599(60)	**1520/8477(3), 8312/1647(41)**

We are largely beaten, and this is worse and worse as the given amount of time increases. Observe however the high number of draws, which indicates that even if MCTS almost always wins at the end, the competition is tight.

Let us mention that with 1024 ms per move, we found, for Sym *vs.* MCTS(10,10): 14/986(0), 925/74(1) and, for Sym *vs.* MCTS′(10,10): 14/986(0), 924/74(2): we are destroyed.

Sym performs here around twice fewer iterations per unit of time than MCTS.

8.7. A version of Connect Four with small degree

We next consider the variation of Connect Four with 4 columns, 10 lines, and where the goal is to connect (horizontally, vertically, or diagonally) 3 discs.

$t = 1$ ms. MCTS(4,5): 223, MCTS′(2,2.5): 257, Sym: 107	
Sym *vs.* MCTS(4,5)	Sym *vs.* MCTS′(2,2.5)
7591/2409(0), 8685/1315(0)	**7082/2918(0), 8944/1056(0)**

$t = 2$ ms. MCTS(1.5,2): 441, MCTS′(2.5,3): 432, Sym: 167	
Sym *vs.* MCTS(1.5,2)	Sym *vs.* MCTS′(2.5,3)
7941/2059(0), 8805/1195(0)	**7585/2415(0), 9286/714(0)**

$t = 4$ ms. MCTS(2,2): 722, MCTS′(3.5,4): 921, Sym: 294	
Sym *vs.* MCTS(2,2)	Sym *vs.* MCTS′(3.5,4)
8647/1353(0), 9181/819(0)	**8445/1555(0), 9553/447(0)**

$t = 8$ ms. MCTS(4.5,5): 2215, MCTS′(4.5,5): 2966, Sym: 530	
Sym *vs.* MCTS(4.5,5)	Sym *vs.* MCTS′(4.5,5)
9920/80(0), 9694/306(0)	9908/92(0), 9879/121(0)

$t = 16$ ms. MCTS(3.5,4): 7409, MCTS′(4,4): 6705, Sym: 982	
Sym *vs.* MCTS(3.5,4)	Sym *vs.* MCTS′(4,4)
10000/0(0), 9952/48(0)	10000/0(0), 9983/17(0)

Here we observe that MCTS and MCTS′ win when having a lot time (for such a small game, 1 millisecond is much), but when the amount of time is so high that we are close to finding the winning strategy at the first move, Sym is better. This might be due to the fact that Sym automatically does some pruning.

Here Sym performs around 3 or 4 times fewer iterations per unit of time than MCTS. This is not the case with 16 ms because it finds the right move, and thus stops, before the 16 ms are elapsed.

8.8. A first version of connect 4 with large degree

Here we consider a very simple version of Connect Four, with 15 columns, 15 lines, and where the goal is to connect (horizontally, vertically, or diagonally) 3 discs. A human player immediately finds the winning strategy when playing first. However, for a computer, the situation is not so easy, because there are many possibilities (if not taking advantage of the symmetries of the game).

$t = 1$ ms. MCTS(2.5,4): 21, MCTS′(1.5,2.5): 20, Sym: 16	
Sym *vs.* MCTS(2.5,4)	Sym *vs.* MCTS′(1.5,2.5)
8396/1604(0), 5236/4764(0)	8453/1547(0), 5292/4708(0)

$t = 2$ ms. MCTS(2,3.5): 32, MCTS′(2,3.5): 31, Sym: 24	
Sym *vs.* MCTS(2,3.5)	Sym *vs.* MCTS′(2,3.5)
8368/1632(0), 5652/4348(0)	8355/1645(0), 5812/4188(0)

$t = 4$ ms. MCTS(1.5,2.5): 54, MCTS′(3,5): 52, Sym: 40	
Sym *vs.* MCTS(1.5,2.5)	Sym *vs.* MCTS′(3,5)
8792/1208(0), 5898/4102(0)	8749/1251(0), 5893/4107(0)

$t = 8$ ms. MCTS(2.5,4): 100, MCTS′(3,5): 96, Sym: 72	
Sym *vs.* MCTS(2.5,4)	Sym *vs.* MCTS′(3,5)
9207/793(0), 5894/4106(0)	9219/781(0), 6032/3968(0)

$t = 16$ ms. MCTS(1.5,2): 194, MCTS′(3.5,5): 182, Sym: 134	
Sym *vs.* MCTS(1.5,2)	Sym *vs.* MCTS′(3.5,5)
9388/612(0), 7048/2952(0)	9417/583(0), 7153/2847(0)

$t = 32$ ms. MCTS(3,3.5): 379, MCTS′(3,4): 368, Sym: 260	
Sym *vs.* MCTS(3,3.5)	Sym *vs.* MCTS′(3,4)
9656/344(0), 8430/1570(0)	9626/374(0), 8512/1488(0)

$t = 64$ ms. MCTS(4,5): 781, MCTS′(2,2.5): 785, Sym: 523	
Sym *vs.* MCTS(4,5)	Sym *vs.* MCTS′(2,2.5)
9851/149(0), 8953/1047(0)	9830/170(0), 8784/1216(0)

$t = 128$ ms. MCTS(2.5,3): 1735, MCTS′(2,2.5): 1704, Sym: 1032	
Sym *vs.* MCTS(2.5,3)	Sym *vs.* MCTS′(2,2.5)
9919/81(0), 9221/779(0)	9928/72(0), 9117/883(0)

Here we are really better than MCTS. We believe this is due to the fact that the degree of the game is very large.

For this game, Sym performs around 2 times fewer iterations per unit of time than MCTS.

8.9. A second version of Connect Four with large degree

We consider here the version of Connect Four with 15 columns, 6 lines, and where the goal is to connect (horizontally, vertically, or diagonally) 5 discs. Here the situation is intractable for a normal human player.

$t = 1$ ms. MCTS(2.5,4): 11, MCTS′(3,5): 10, Sym: 8	
Sym *vs.* MCTS(2.5,4)	Sym *vs.* MCTS′(3,5)
6368/3631(1), 4892/5106(2)	6521/3477(2), 4892/5104(4)

$t = 2$ ms. MCTS(0.5,0.5): 16, MCTS′(1,1.5): 15, Sym: 12	
Sym *vs.* MCTS(0.5,0.5)	Sym *vs.* MCTS′(1,1.5)
6730/3270(0), 4412/5586(2)	6793/3205(2), 4537/5458(5)

$t = 4$ ms. MCTS(0.5,0.5): 27, MCTS′(0.5,0.5): 25, Sym: 19	
Sym vs MCTS(0.5,0.5)	Sym *vs.* MCTS′(0.5,0.5)
6305/3692(3), 5056/4936(8)	6479/3515(6), 5038/4959(3)

$t = 8$ ms. MCTS(0.5,0.5): 50, MCTS′(0.5,0.5): 45, Sym: 33	
Sym *vs.* MCTS(0.5,0.5)	Sym *vs.* MCTS′(0.5,0.5)
6160/3821(19), 4925/5044(31)	5966/4022(12), 5159/4808(33)

$t = 16$ ms. MCTS(1.5,2): 95, MCTS′(1.5,2): 88, Sym: 61	
Sym *vs.* MCTS(1.5,2)	Sym *vs.* MCTS′(1.5,2)
4377/5598(25), 6499/3427(74)	**4286/5692(22), 6526/3388(86)**

The number of iterations seems very small here, but let us recall that this concerns only the beginning of the game. As already mentioned, this number of iterations actually increases considerably when approaching the end of the game.

We observe that Sym is really better than MCTS and MCTS′ when the amount of time per move is small. When this amount of time increases, we are beaten. We have two main explanations for this, see Section 8.12.

Here Sym performs around twice fewer iterations per unit of time than MCTS.

8.10. Inverse and large Connect Four

We next consider the inverse version of Connect Four with 15 columns, 6 lines, and where the first player that connects (horizontally, vertically, or diagonally) 5 discs looses. Although this modification is coded very easily (only one line has to be modified so that *win* and *loss* are exchanged), this considerably modifies the game. The algorithms (MCTS, MCTS′ and Sym) all take this into account immediately: at the beginning of a match, one usually sees the algorithms play in the middle of the board, while they play near the extremities in the *inverse* case.

$t = 1$ ms. MCTS(1.5,3): 10, MCTS′(2,4): 9, Sym: 8	
Sym *vs.* MCTS(1.5,3)	Sym *vs.* MCTS′(2,4)
5094/4889(17), 3793/6182(25)	4954/5037(9), 3792/6185(23)

$t = 2$ ms. MCTS(1.5,3): 15, MCTS′(1,2): 14, Sym: 12	
Sym *vs.* MCTS(1.5,3)	Sym *vs.* MCTS′(1,2)
4478/5507(15), 4338/5628(34)	**4305/5681(14), 4444/5526(30)**

$t = 4$ ms. MCTS(0.5,0.5): 25, MCTS′(2,4): 23, Sym: 19	
Sym *vs.* MCTS(0.5,0.5)	Sym *vs.* MCTS′(2,4)
4058/5931(11), 5084/4887(29)	**4236/5757(7), 4607/5356(37)**

Here again, we observe that Sym wins when the amount of time per move is very small.

8.11. Back to Pearl's game: rate of convergence

Here we consider Pearl's game with $d = 2$, K even and with the value $p = (\sqrt{5} - 1)/2 \simeq 0.618$. With this particular value of p, Pearl [18] showed that $\mathbb{E}[R(r)] = p$ and that the expected required number of visited

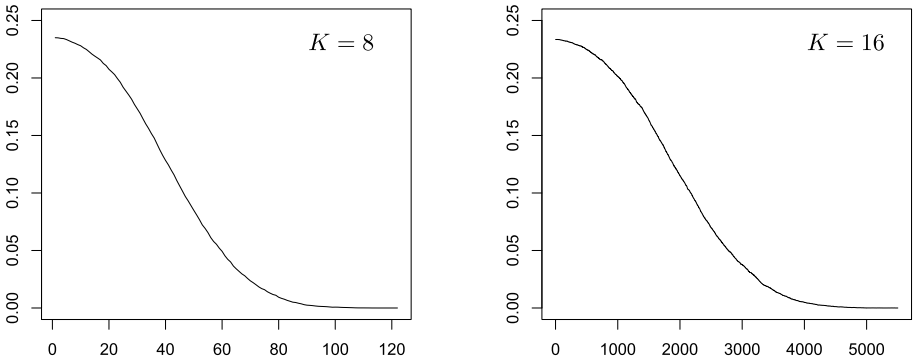

FIGURE 4. $\mathbb{E}[(R_n(r) - R(r))^2]$ as a function of the number n of iterations.

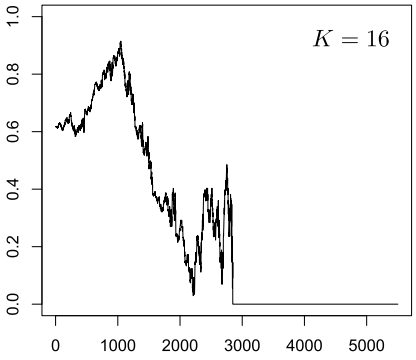

FIGURE 5. One trajectory of $n \mapsto R_n(r)$.

leaves for AlphaBeta to determine $R(r)$ equals $(2/(\sqrt{5}-1))^K$. Here r is the true root of the game. In the whole subsection, we use SymP with the correct value of a, *i.e.* $a = p$.

First, we call τ_K the number of leaves that SymP needs to determine $R(r)$. Denoting by $\bar{\tau}_K$ the average value over 10000 trials, we found

K	4	8	12	16
$\bar{\tau}_K$	6.84	47.14	323.51	2207.89
$(2/(\sqrt{5}-1))^K$	6.8541	46.9787	321.9969	2206.9995

I thus seems highly plausible that our algorithm visits the leaves in the same order as AlphaBeta (for a Pearl game), up to some random permutation. This is rather satisfying, since Tarsi [21] showed that for a Pearl game, AlphaBeta is optimal in the sense of the expected number of leaves necessary to determine $R(r)$.

Finally, we plot a Monte-Carlo approximation (with 10000 trials) of $\mathbb{E}[(R_n(r) - R(r))^2]$ as a function of the number n of iterations, when $K = 8$ and $K = 16$, as well as one trajectory of $n \mapsto R_n(r)$ when $K = 16$ (Figs. 4 and 5).

8.12. Conclusion

When playing Pearl's game, SymP beats MCTS and seems competitive against MCTS′. This is reassuring, since our algorithms are typically designed for such games.

On true games, MCTS and MCTS′ seem globally much better than Sym. However, we found two situations where Sym may win.

The first and most interesting situation is the one where the game is so large (or the amount of time so small) that very few iterations can be performed by the challengers. This is quite natural, and there are two possible reasons for that.

- Our algorithm is only optimal *step by step*. So, it is absolutely not clear that it works well when we have enough time to handle many iterations.
- Assumption 2.5 imposes some independence properties. While this is reasonable, on any game, in some sense to be precised, after a small number of iterations (when playing a small number of uniformly random matches, it is rather clear that the issues will be almost independent), this is clearly not the case when performing a large number of well-chosen matches.

The second situation is that where the game is so small that we can hope to find the winning strategy at the first move, and where Sym may find it before MCTS and MCTS′. As already mentioned, we believe this is due to the fact that Sym does some pruning.

From a theoretical point of view, it would be very interesting to study more relevant models such as the one proposed by Devroye-Kamoun [10]. Clearly, this falls completely out of our scope. On the contrary, it does not seem completely desesperate to find empirically variants of our algorithm that work much better in practise. For example, it may be relevant to use other choices of functions m and s, to try a clever default policy, etc.

Appendix A. Monte Carlo Tree Search algorithms

In this subsection, we write down precisely the versions of the MCTS algorithm we used to test our algorithm. We start with a modified version, more close to our study, where we do not throw down any information.

Algorithm A.1 (Modified MCTS)**.** Consider $\phi : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}$, *e.g.* $\phi(w,c) = \frac{w+a}{c+b}$ for some $a, b > 0$.

Step 1. Simulate a uniformly random match from r, call x_1 the resulting leave and put $\mathbf{x}_1 = \{x_1\}$.

During this random match, keep track of $R(x_1)$, of $B_{\mathbf{x}_1} = B_{rx_1}$ and of $\mathcal{D}_{\mathbf{x}_1} = \cup_{y \in B_{rx_1}} \mathcal{H}_y$ and set $C_1(x) = W_1(x) = 0$ for all $x \in \mathcal{D}_{\mathbf{x}_1}$.

For all $x \in B_{x_1}$, set $C_1(x) = 1$, $W_1(x) = R(x_1)$.

Step n+1. Put $z = r$ and do $z = \text{argmax}\{\phi(V_n(y), C_n(y)) : y \in \mathcal{C}_z\}$ until $z \in \mathbf{x}_n \cup \mathcal{D}_{\mathbf{x}_n}$, where

$$V_n(y) = \mathbf{1}_{\{t(f(y))=1\}} W_n(y) + \mathbf{1}_{\{t(f(y))=0\}} (C_n(y) - W_n(y)).$$

Set $z_n = z$.

(i) If $z_n \in \mathbf{x}_n$ (this will almost never occur if n is reasonable for a large game), set $\mathbf{x}_{n+1} = \mathbf{x}_n$, $B_{\mathbf{x}_{n+1}} = B_{\mathbf{x}_n}$ and $\mathcal{D}_{\mathbf{x}_{n+1}} = \mathcal{D}_{\mathbf{x}_n}$
For all $x \in B_{rz_n}$, set $C_{n+1}(x) = C_n(x) + 1$, $W_{n+1}(x) = W_n(x) + R(z_n)$.
For all $x \in (B_{\mathbf{x}_{n+1}} \cup \mathcal{D}_{\mathbf{x}_{n+1}}) \setminus B_{rz_n}$, set $C_{n+1}(x) = C_n(x)$, $W_{n+1}(x) = W_n(x)$.

(ii) Else (then $z_n \in \mathcal{D}_{\mathbf{x}_n}$), simulate a uniformly random match from z_n, call x_{n+1} the resulting leave, set $\mathbf{x}_{n+1} = \mathbf{x}_n \cup \{x_{n+1}\}$.

During this random match, keep track of $R(x_{n+1})$, of $B_{\mathbf{x}_{n+1}} = B_{\mathbf{x}_n} \cup B_{rx_{n+1}}$ and of $\mathcal{D}_{\mathbf{x}_{n+1}} = (\mathcal{D}_{\mathbf{x}_n} \setminus \{z_n\}) \cup \bigcup_{y \in B_{z_n x_{n+1}} \setminus \{z_n\}} \mathcal{H}_y$ and set $R_{n+1}(x) = W_{n+1}(x) = 0$ for all $x \in \bigcup_{y \in B_{z_n x_{n+1}} \setminus \{z_n\}} \mathcal{H}_y$.

For all $x \in B_{rx_{n+1}}$, set $C_{n+1}(x) = C_n(x) + 1$, $W_{n+1}(y) = W_n(x) + R(x_{n+1})$.

Finally, set $C_{n+1}(x) = C_n(x)$, $W_{n+1}(x) = W_n(x)$ for all $x \in (B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}) \setminus B_{rx_{n+1}}$.

Conclusion. Stop after a given number of iterations n_0 (or after a given amount of time). As *best child* of r, choose $x_* = \text{argmax}\{\phi(V_{n_0}(x), C_{n_0}(x)) : x \text{ child of } r\}$.

Algorithm A.1 updates the information on the whole visited branch at each new rollout. Here is a more standard version: it only creates, after each new simulation, one new node (together with its brothers) and updates the information only on the branch from the root to this new node. It seems clear that Algorithm A.1

should be better, but it may lead to memory problems if the game is very large. We do not discuss such memory problems in the present paper.

Algorithm A.2 (MCTS)**.** Consider $\phi : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}$, *e.g.* $\phi(w,c) = \frac{w+a}{c+b}$ for some $a, b > 0$.

Step 1. Simulate a uniformly random match from r, call u the resulting leave.
During this random match, keep track of $R(u)$ and of $T_1 = \{r\} \cup \mathcal{C}_r$.
Set $C_1(x) = W_1(x) = 0$ for all $x \in T_1 \setminus B_{ru}$.
Set $C_1(x) = 1$ and $W_1(x) = R(u)$ for all $x \in T_1 \cap B_{ru}$.
Step n+1. Put $z = r$ and do $z = \text{argmax}\{\phi(V_n(y), C_n(y)) : y \in \mathcal{C}_z\}$ until $z \in L_{T_n}$, where

$$V_n(y) = \mathbf{1}_{\{t(f(y))=1\}} W_n(y) + \mathbf{1}_{\{t(f(y))=0\}} (C_n(y) - W_n(y)).$$

Set $z_n = z$.

(i) If $z_n \in \mathcal{L}$ (this will never occur if n is reasonable for a large game), set $T_{n+1} = T_n$.
For all $x \in B_{rz_n}$, set $C_{n+1}(x) = C_n(x) + 1$, $W_{n+1}(x) = W_n(x) + R(z_n)$.
For all $x \in T_{n+1} \setminus B_{rz_n}$, set $C_{n+1}(x) = C_n(x)$, $W_{n+1}(x) = W_n(x)$.
(ii) If $z_n \notin \mathcal{L}$, simulate a uniformly random match from z_n, call u the resulting leave, define y as the child of z_n belonging to $B_{z_n u}$ and set $T_{n+1} = T_n \cup \mathcal{C}_{z_n}$.

For all $x \in B_{rz_n}$, set $C_{n+1}(x) = C_n(x) + 1$, $W_{n+1}(y) = W_n(x) + R(u)$.
Set $C_{n+1}(y) = 1$, $W_{n+1}(y) = \mathbf{1}_{\{R(u)=1\}}$ and, for all $x \in \mathcal{H}_y$, set $C_{n+1}(x) = W_{n+1}(x) = 0$.
For all $x \in T_n \setminus B_{rz_n}$, set $C_{n+1}(x) = C_n(x)$ and $W_{n+1}(x) = W_n(x)$.

Conclusion. Stop after a given number of iterations n_0 (or after a given amount of time). As *best child* of r, choose $x_* = \text{argmax}\{\phi(V_{n_0}(x), C_{n_0}(x)) : x \text{ child of } r\}$.

Of course, in both algorithms, the choice of the function ϕ is debatable. The choice $\phi(w,c) = (w+a)/(c+b)$, with $a > 0$ and $b > 0$ chosen empirically, seems to be a very good choice and was proposed by Lee *et al.* [16] Section II-B-1.

Algorithm A.1 is not admissible in the sense of Definition 2.3 because it may take different decisions with the same information. Indeed, it might visit twice the same leave consecutively: this does not modify the information but changes the values of W_n and C_n. However, it is *almost* admissible: it would suffice to forbid two visits at the same leave (or alternatively to set $C_{n+1}(x) = C_n(x)$ and $W_{n+1}(x) = W_n(x)$ for all $x \in B_{\mathbf{x}_n} \cup \mathcal{D}_{\mathbf{x}_n}$ in the case where $x_{n+1} \in \mathbf{x}_n$) to make it admissible. Since such a double visit almost never happens in practice, we decided not to complicate the definition of admissible algorithms nor to modify Algorithm A.1.

Algorithm A.2 is not admissible, because it has not the good structure (it does not keep track of the whole observed information), but we see it as an truncated version of Algorithm A.1, which is itself almost admissible.

Observe that in Algorithm A.1, $C_n(x)$ is the number of times (iterations) where the node x has been crossed and $W_n(x)$ is the number of times where x has been crossed and where this has led to a victory of J_1, all this after n matches. The $(n+1)$th match is as follows: we start from the root and make J_1 play the most promising move (for itself, *i.e.* the child with the highest $\phi(W_n, C_n)$) and J_0 play the most promising move (for itself, *i.e.* the child with the highest $\phi(C_n - W_n, C_n)$) until we reach an uncrossed position $z_n \in \mathcal{D}_{\mathbf{x}_n}$. From there, we end the match at uniform random, until we arrive at some leave u. We finally update the explored tree as well as its boundary and the values of the numbers of crosses and of victories of each node of the branch from r to u.

References

[1] B. Abramson, Expected-outcome: a general model of static evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.* **12** (1990) 182–193.
[2] P. Auer, N. Cesa-Bianchi and P. Fischer, Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.* **47** (2002) 235–256.
[3] E.B. Baum and W.D. Smith, A Bayesian approach to relevance in game playing. *Artif. Intell.* **97** (1997) 195–242.

[4] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton, A survey monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games* **4** (2012) 1–43.
[5] S. Bubeck and N. Cesa-Bianchi, Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Found. Trends Mach. Learn.* **5** (2012) 1–122.
[6] L. Buşoniu, R. Munos and E. Páll, An analysis of optimistic, best-first search for minimax sequential decision making. *IEEE Int. Symp. Approx. Dyn. Program. Reinf. Learn.* (2014).
[7] G.M.J.B. Chaslot, M.H.M. Winands, H.J. van den Herik, J.W.H.M. Uiterwijk and B. Bouzy, Progressive strategies for Monte-Carlo tree search. *New Math. Nat. Comput.* **4** (2008) 343–357.
[8] P.A. Coquelin and R. Munos, *Bandit Algorithms for Tree Search*. Technical report, INRIA RR-6141 (2007).
[9] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, in *Proc. of 5th Int. Conf. Comput. and Games, Turin, Italy* (2006) 72–83.
[10] L. Devroye and O. Kamoun, Random minimax game trees, in Random Discrete Structures (Minneapolis, MN, 1993) Vol. 76 of *The IMA Volumes in Mathematics and its Applications*. Springer, New York (1996) 55–80.
[11] A. Garivier, E. Kaufmann and W.M. Koolen, Maximin action identification: a new bandit framework for games, in *JMLR: Workshop and Conference Proceedings*, Vol. 49 (2016) 1–23.
[12] S. Gelly, Y. Wang, R. Munos and O. Teytaud, *Modification of UCT With Patterns in Monte-Carlo Go*. Tech. Rep. Inst. Nat. Rech. Inform. Auto. (INRIA), Paris (2006).
[13] M.L. Ginsberg, GIB: imperfect information in a computationally challenging game. *J. Artif. Intell. Res.* **14** (2001) 303–358.
[14] D. Golovin and A. Kraus, Adaptive submodularity: theory and applications in active learning and stochastic optimization. *J. Artif. Intell. Res.* **42** (2011) 427–486.
[15] L. Kocsis and C. Szepesvári, Bandit based Monte-Carlo planning, in Machine Learning: ECML, Vol. 4212 of *Lecture Notes in Comput. Sci.* Springer, Berlin (2006) 282–293.
[16] C.S. Lee, M.H. Wang, G.M.J.B. Chaslot, J.B. Hoock, A. Rimmel, O. Teytaud *et al.,* The computational intelligence of MoGo revealed in taiwans computer go tournaments. *IEEE Trans. Comput. Intell. AI Games* **1** (2009) 73–89.
[17] R. Munos, From bandits to Monte-Carlo tree search: the optimistic principle applied to optimization and planning, in Foundations and Trends in Machine Learning (Book 21). Now Publishers Inc. (2014) 146.
[18] J. Pearl, Asymptotic properties of minimax trees and game-searching procedures. *Artif. Intell.* **14** (1980) 113–138.
[19] B. Sheppard, World-championship-caliber srabble. *Artif. Intell.* **134** (2002) 241–275.
[20] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search. *Nature* **529** (2016) 484–489.
[21] M. Tarsi, Optimal search on some game trees. *J. Assoc. Comput. Mach.* **30** (1983) 389–396.
[22] G. Tesauro, V.T. Rajan and R. Segal, Bayesian inference in Monte-Carlo tree search, in *UAI'10 Proc. of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence* (2010) 580–588.