Single Player Monte-Carlo Tree Search Based on the Plackett-Luce Model

Felix Mohr[1], **Viktor Bengs**[2], **Eyke Hüllermeier**[2]
[1] Universidad de La Sabana, Campus del Puente del Común, Km. 7, Autopista Norte de Bogotá, Chía, Colombia
[2] Paderborn University, Warburgerstraße 100, Paderborn, Germany
felix.mohr@unisabana.edu.co, viktor.bengs@uni-paderborn.de, eyke@uni-paderborn.de

Abstract

The problem of minimal cost path search is especially difficult when no useful heuristics are available. A common solution is roll-out-based search like Monte Carlo Tree Search (MCTS). However, MCTS is mostly used in stochastic or adversarial environments, with the goal to identify an agent's best next move. For this reason, even though single player versions of MCTS exist, most algorithms, including UCT, are not directly tailored to classical minimal cost path search. We present Plackett-Luce MCTS (PL-MCTS), a path search algorithm based on a probabilistic model over the qualities of successor nodes. We empirically show that PL-MCTS is competitive and often superior to the state of the art.

Introduction

In the context of minimal cost path search, we consider the problem of finding an optimal path in a (possibly infinite) tree, i.e., a leaf $l^* \in \operatorname{argmin}_{l \in L}\ \phi(l)$, where $\phi : L \to \mathbb{R}^+$ is a scoring function on the set L of leaf nodes. As an important difference to classical planning problems, we assume that there is no informative heuristic to reliably approximate $\phi(l_n^*)$, where l_n^* is the best scored leaf underneath n.

Examples include offline-planning problems, in which performances can only be assessed by simulation, e.g., traffic light placement [Corti, Manzoni, and Savaresi 2012] or economic well-fare evolvement [Borghesi et al. 2013]. Another area that deals with such problems is software and algorithm configuration, e.g., software product lines [Henard et al. 2015], automated machine learning [Mohr, Wever, and Hüllermeier 2018] or automated SAT solver selection and configuration [Bischl et al. 2016].

In the absence of an informative heuristic, one approach to solve this problem is to use Monte Carlo Tree Search (MCTS) [Browne et al. 2012]. MCTS is a tree search algorithm scheme that was originally designed to learn optimal policies for Markov Decision Processes (MDPs). While our tree could be considered as a special type of MDP and tackled with classical MCTS algorithms such as UCT [Kocsis, Szepesvári, and Willemson 2006], we know from the realm of single player games that variants of MCTS more specialized to deterministic conditions are superior to that approach [De Mesmay et al. 2009; Schadd et al. 2008; Bjornsson and Finnsson 2009].

The contribution of this paper is to complement the group of MCTS algorithms for single player problems by Plackett-Luce MCTS (PL-MCTS), a version of MCTS based on *preferences*. Different to previous algorithms that directly use the quantitative observations to control the search process, PL-MCTS associates the successors of a node with a *probability* that following it will reveal the *best improvement* over the currently best solution. An important advantage of this approach is its ability to work upon rankings over child nodes. Such rankings are a natural choice to capture the only vague belief of node promisingness one usually has in MCTS due to the unavoidable distribution drifts. In three heterogeneous problem domains, we show that PL-MCTS is highly competitive and often superior to all other state-of-the-art algorithms when run with a standard configuration and that its flexibility to customize preference schemes can improve the performance of PL-MCTS even more.

The code of PL-MCTS is published in the Java library AILibs[1] and can be directly used through a Maven artifact[2]. This paper is accompanied by a code repository to reproduce the experiments [3], which also contains the *technical supplementary material.*

Problem Formulation

Since we tackle the optimal path problem via the MCTS algorithm scheme, we reformulate the problem in this context. Algorithm 1 depicts the MCTS scheme for this case. At its core, MCTS is a loop in which each iteration consists of the generation of a random path from the root to a leaf (roll-out), the evaluation of that path, and the back-propagation of the observation along the path. The roll-out is computed using two types of policies, which determine, for a given inner node, a successor to complement the current path. Simply speaking, the *tree policy* (TP) is used to determine successors of nodes that have already been considered in the past and uses past observations to make its choices, and the *default policy* (DP) is used for all other nodes. In this paper, we assume the absence of prior knowledge, so that the DP shall

[1]https://github.com/starlibs/AILibs
[2]https://mvnrepository.com/artifact/ai.libs/jaicore-search/0.2.4
[3]https://github.com/fmohr/aaai2021

Algorithm 1: MonteCarloTreePathSearch

```
create root node $n_0$, $\overline{L} = \emptyset$
while within computational budget $T$ do
    /* iteratively call $\pi(s, n_i, T)$ */
    $n \leftarrow$ TreePolicy.getOpenNode($n_0$)
    $l \leftarrow$ DefaultPolicy($n$)
    $\overline{L} = \overline{L} \cup \{l\}$
    /* update state $s$ used for $\pi$ */
    TreePolicy.Update($\langle n_0, .., n\rangle, \phi(l)$)
end
return $\langle n_0, .., l^*\rangle$ with $l^* \in \mathrm{argmin}_{l \in \overline{L}}\, \phi(l)$
```

be a uniform sampler. The below algorithm summarizes the successive path generation in two commands that first produces a node n up to which the TP could be used and from whereon the DP is used. We refer the reader to [Browne et al. 2012] for details on the MCTS scheme.

There is an important though subtle difference to MCTS applied to MDPs. General MCTS schemes for MDPs aim at returning an *action* whose *application* in the current state of an MDP appears best *on average*. In contrast, in the deterministic case we will simply return the best *path* seen *during* search. In particular, there is no notion of "applying" the solution path to the MDP after search.

This being said and observing that the TP is the only component of the algorithm we can influence (after fixing the DP to uniform sampling), our goal is to design a TP that has the *highest probability* of finding a good path *during* search. Formally, for a space N of tree nodes and a space S of MCTS search process states (e.g. elapsed time, past observations, the inner tree to which the TP can be applied, etc.), a tree policy π in the deterministic setup is a mapping $\pi : S \times N \times \mathbb{N} \rightarrow N$ that chooses for an inner node $n \in N$ a successor, taking into account the current state $s \in S$ of the MCTS search *process* and the total time budget $T \in \mathbb{N}$ of the *search process*. When running MCTS with a TP π with budget T (number of roll-outs), the observations form a finite sequence $(O_1^\pi, \ldots, O_T^\pi)$ of real-valued random variables obtained by applying the evaluation ϕ to the leaf nodes of the roll-outs. The observations are random variables, because the deterministic policy π is only used in the already explored part of the tree, while the random DP is used everywhere else. Assuming a uniformly sampling DP, the distributions of those variables only depend on the score landscape and the TP; hence the super-script π. Being interested in making the best observation possible (the smallest score), we let $X_t^\pi = \min\{O_1^\pi, \ldots, O_t^\pi\}$ be the random variable reflecting the smallest score seen up to time t when using TP π, and then seek to find

$$\pi^* \in \underset{\pi}{\mathrm{argmax}}\, \mathbb{P}\left(X_T^\pi = \min_{\pi'}(X_T^{\pi'}) \right), \tag{1}$$

where π and π' are chosen among all possible TPs. In words, we seek the TP that has the highest probability to exhibit the best score within the time budget T.

This goal formulation substantially differs from others in the context of MCTS, which are typically based on the notion of regret, i.e. the deviation of the chosen policy from some baseline. First note that the policy π we assess here is the *tree policy* used within MCTS and *not* a policy *applied* in the underlying MDP *after* search. This is a substantial difference to *simple regret* [Feldman and Domshlak 2014], which characterizes the performance of an MDP policy, namely the produced policy *applied* in the MDP (*after* finishing the search process). Simple regret hence qualifies a TP only indirectly by the performance of the output of MCTS produced with it but not its *probability* of finding good solutions. In this sense, the *cumulative* regret as optimized in UCT [Kocsis, Szepesvári, and Willemson 2006] is closer related to our objective since it considers the performance *during* search. However, the cumulative regret considers the *average* performance, while we are interested in the *extreme* performance. The underlying notion of extreme regret has been analyzed on the level of bandits [Streeter and Smith 2006; Carpentier and Valko 2014] but has been lifted into the context of MCTS only in an ad-hoc manner [De Mesmay et al. 2009]. From [Carpentier and Valko 2014], we also know that the extreme regret cannot be guaranteed to be bound without having knowledge about the underlying distributions. Since we seek for an algorithm that does not require (generally unavailable) distribution knowledge as a parameter, regret bounds are out of reach. As a consequence, the goal definition (1) abstains from a notion of regret altogether.

Plackett-Luce MCTS

We now propose a tree policy based on the Plackett-Luce (PL) model. Throughout this section, we denote by $n \in N \backslash L$ any inner node and by $n_1, \ldots, n_k$ its child nodes.

Past vs Future Observations

The motivating assumption of PL-MCTS is that we can maintain for each inner node a *belief model* that specifies for each of its successor nodes a probability of revealing the best improvement (compared to its siblings) when choosing it the next time. For a concrete decision situation characterized by an MCTS state s_t reached after t iterations and an inner node n with children n_i, we denote as $Y_{s_t, T-t}^{(n_i, \pi)}$ the (unknown) improvement over the currently best solution X_t^π we will see if choosing n_i and adopt π in all other upcoming decisions. It then seems natural to select the child by the means of

$$\pi(s_t, n, T) = \underset{n_i : 1 \leq i \leq k}{\mathrm{argmax}}\, \mathbb{P}\left(Y_{s_t, T-t}^{(n_i, \pi)} = \max_{1 \leq j \leq k} \left\{ Y_{s_t, T-t}^{(n_j, \pi)} \right\} \right), \tag{2}$$

and in the supplement we prove that this is in fact optimal in the sense that a tree policy satisfies Eq. (1) if and only if it selects nodes according to Eq. (2).

Our main line of argumentation is that due to the *distribution drift* in MCTS, is can be better to refrain from using the concrete observations for decision making altogether and only use the observations *indirectly* to shape a belief model over which nodes are best. The distribution drift refers to the fact that the successors of an inner node are not associated with a fixed distribution but with a distribution that changes over time, because the tree policy will be used more and more below that node in each new iteration. This model has typically a self-reinforcing effect, because the more the TP is used, the better the observations become, making it even more likely to choose the same node again in the upcoming iterations. Needless to say, due to the distribution drift in MCTS already recognized in UCT [Kocsis, Szepesvári, and Willemson 2006] caused by the increasing use of the TP over time, the samples observed in the past come from distributions that (usually substantially) differ from those of $Y^{(n_i,\pi)}_{s_t,T-t}$, so that the distribution of those variables cannot be reliably estimated. However, since we only need to decide where to go next, the concrete distributions of those variables are not even relevant as long as we can estimate which of the children will be the best in (2). In other words, it would be enough to have a *ranking* of child nodes in which the top-ranked element is the one that will reveal the best improvement in the remaining time.

This view differs from other approaches, which are centered around models for *past* observations. The typical approach has been to build models around the distributions that have generated the *past* observations and to prepare for drift through confidence bounds [Kocsis, Szepesvári, and Willemson 2006; Schadd et al. 2008; De Mesmay et al. 2009] or model variance [Bai, Wu, and Chen 2013]. We propose PL-MCTS, which seeks to differ from this approach in being more explicit in the attempt to use the past observations to fit a model of beliefs for *future* observations. In other words, we do not seek to fit a model that best matches the past observations but a model that best matches what we *expect* to see *based* on the past observations. Of course, without perfect information, this can only be achieved to a certain extent. However, the idea is to provide a framework that is more amenable to this view than models that are by definition fixed to the past.

In order to establish rankings among child nodes with respect to their promisingness for improvement, we rely on the so-called Plackett-Luce (PL) model, a well established statistical model to learn probabilities for rankings among a finite set of objects. To this end, we exploit concepts from the statistical branch of extreme value theory. Sections and theoretically introduce the PL-model and explain why it *can* be used to determine the best child according to (2) *if* the available information really was from the distributions of the $Y^{(n_i,\pi)}_{s_t,T-t}$. Given the absence of such samples, Section argues that the PL model seems *preferable* over quantiative statistical models for distribution parameter estimation, because it allows to work only on *rankings* among child nodes, which are an arguably natural and easy way to express node promisingness in the face of distribution drift. Finally, Section describes a tree policy built upon the PL model.

Gumbel-Distributed Future Observations

When seeking for the optimal child node in (2), we need to understand the probabilities that the child will reveal the best score, i.e., the distributions of $Y^{(n_j,\pi)}_{s_t,T-t}$. Distributions of that kind, i.e., maximum statistics, have been exhaustively studied in the realm of *extreme value theory* [De Haan and Ferreira 2007]. A core result, the Fisher-Tippet-Gnedenko theorem, states that the limit distribution of $M^P_s := \max\{Z_1,\dots,Z_s\}$, where $Z_1,\dots,Z_s$ are iid with distribution P, converges[4] for growing s towards one of the three non-degenerated parametric probability distribution families *Gumbel*, *Fréchet*, or *Weibull*.

Many practically important distributions of Z_i, including the normal distribution, have the Gumbel distribution as their asymptotic limit (see Table 1 in [Charras-Garrido and Lezaud 2013] for an overview), so that we focus in the following on the latter. A random variable G is Gumbel distributed with location parameter $\mu \in \mathbb{R}$ and scale parameter $\sigma > 0$ $\big($denoted by $G \sim \mathcal{G}(\mu,\sigma)\big)$ if its cumulative distribution function is given by

$$\mathbb{P}(G \le z) = \exp\Big(-\exp\Big(-\frac{z-\mu}{\sigma}\Big)\Big), \quad \forall z \in \mathbb{R}.$$

In the following, we will assume $Y^{(n_j,\pi)}_{s_t,T-t}$ to be distributed Gumbel-like. Since the time horizon $T-t$ is only finite and may even be quite small, this is an idealized assumption but comparable to using the central limit theorem to justify a normal-like distribution of the statistical mean of a rather small sample. Given the Gumbel distribution and the following properties well known in statistics literature [Train 2009], we will derive a closed form for condition (2).

Theorem 1. *(i) Let $G \sim \mathcal{G}(\mu,\sigma)$. Then, for $a > 0$ and $b \in \mathbb{R}$, it holds that $aG + b \sim \mathcal{G}(a\mu + b, a\sigma)$.*
(ii) If $(G_i)_{i=1,\dots,k}$ are independent and $G_i \sim \mathcal{G}(\mu_i,\sigma)$, where $\sigma > 0$ and $\mu_i \in \mathbb{R}$ for $i = 1,\dots,n$, then

$$\mathbb{P}\Big(G_j = \max_{1\le i\le k} G_i\Big) = \frac{\exp(\mu_j/\sigma)}{\sum_{i=1}^k \exp(\mu_i/\sigma)}.$$

The distribution P is said to be in the domain of attraction of $\mathcal{G}(\mu,\sigma)$ if and only if there exist sequences $(a_s)_{s\ge1} \subset \mathbb{R}_+$ and $(b_s)_{s\ge1} \subset \mathbb{R}$ such that, for any $z \in \mathbb{R}$,

$$\lim_{s\to\infty} \mathbb{P}\big({}^{(M^P_s - b_s)}/_{a_s} \le z\big) = \exp\big(-\exp\big(-{}^{(z-\mu)}/_{\sigma}\big)\big).$$

In particular, for a distribution P in the domain of attraction of $\mathcal{G}(\mu,\sigma)$ property (i) of Theorem 1 implies that, for sufficiently large s,

$$\mathbb{P}(\,M^P_s \le z) \approx \exp\big(-\exp\big(-{}^{(z-\mu_s)}/_{\sigma_s}\big)\big), \qquad (3)$$

where $\mu_s = a_s\mu + b_s$ and $\sigma_s = \sigma a_s$. In other words, M^P_s is approximately distributed as $\mathcal{G}(\mu_s,\sigma_s)$.

Assuming that the $Y^{(n_j,\pi)}_{s_t,T-t}$ are independent as well as Gumbel distributed with different locations but same scales, which is justified for $T-t$ large enough, the probability in (2) can be expressed by means of (ii) of Theorem 1 in closed form as

$$\mathbb{P}\big(Y^{(n_i,\pi)}_{s_t,T-t} = \max_{1\le j\le k} Y^{(n_j,\pi)}_{s_t,T-t}\big) \approx \frac{\exp(\mu^{(n_i)}/\sigma)}{\sum_{j=1}^k \exp(\mu^{(n_j)}/\sigma)}\,. \quad (4)$$

Certainly, the parameters $\mu^{(n_i)}$ and σ, which determine the distribution, are not known. However, given that the distribution is Gumbel, there is a solid statistical model, the Plackett-Luce model, to *estimate* the ratio of the parameters based on observations in the form of rankings, which is sufficient for our purposes.

[4]if properly localized and scaled

The Plackett-Luce Model

The Plackett-Luce (PL) model is a parametric probability model for rankings over a finite set of objects. The assumption underlying the PL model is that every object is associated with a so-called *skill parameter* reflecting its utility. One of the main properties of the PL model is that the normalized skill parameter corresponds to the probability that the object turns up on the top position in a drawn ranking.

In our context, we consider one such model for each inner node, and the ranked objects are the respective successor nodes. Formally, all child nodes $n_1, \ldots, n_k$ have a (unknown but fixed) skill $\theta_i \in \mathbb{R}$, $i = 1, \ldots, k$. The intended semantic of these skill parameters (by mapping them into $[0, 1]$) is the (estimated) probability of revealing the optimal score underneath the corresponding node. The goal will be to *learn* the skill parameters of the child nodes based on rankings over them.

Suppose that we obtain a ranking ρ (we explain in the next section how to acquire observations of such rankings) over the child nodes, which is a (random) observation of sorting noisy estimates of the skill parameters by virtue of

$$y_i = \theta_i + \varepsilon_i, \qquad i = 1, \ldots, k, \tag{5}$$

where $(\varepsilon_i)_{i=1,\ldots,k}$ are *random* error terms. Then, the likelihood function for any skill vector $(\theta_1, \ldots, \theta_k)$ given the ranking ρ is given by

$$\mathbb{P}_\theta(y_{\rho_1} > y_{\rho_2} > \ldots > y_{\rho_k}), \tag{6}$$

where ρ_j, for $1 \leq j \leq k$ denotes the child node on the jth position with respect to ρ.

The probability in (6) depends on the concrete values of θ and the distribution of the noise terms ε_i. Different parametric probability models are used for the random noise in order to assess the probability in (6), whereas it is common to assume that the noise terms in (5) are independent. Using $\mathcal{G}(0, 1)$ as the distribution for each ε_i leads to the *Plackett-Luce* (PL) model [Alvo and Philip 2014], which allows a closed-form expression of the likelihood in (6) by using property (ii) in Theorem 1:

$$\mathbb{P}_\theta(y_{\rho_1} > y_{\rho_2} > \ldots > y_{\rho_k}) = \prod_{j=1}^{k} \frac{\exp(\theta_{\rho_j})}{\sum_{l=j}^{k} \exp(\theta_{\rho_l})}.$$

Even better for our problem at hand, there is a closed-form expression available for the probability that a particular child node $n_i \in \{n_1, \ldots, n_k\}$ is on top in the preference relation:

$$\mathbb{P}_\theta(y_i = \max_{1 \leq j \leq k} y_j) = \frac{\exp(\theta_i)}{\sum_{j=1}^{k} \exp(\theta_j)}. \tag{7}$$

Comparing the probability the PL model assigns to its top-positional item in (7) to the Gumbel-approximation in (4), they coincide if $\theta_j \equiv \mu^{(n_j)}/\sigma$ holds for all $j = 1, \ldots, k$, which yields an approximation of criterion (2).

In practice, this model can be solved efficiently. The skill parameters are identifiable up to a multiplicative constant, and the likelihood function of the model has a simple analytical solution, which makes methods based on Maximum Likelihood Estimation, such as Expectation-Maximization (EM) inference and Bayesian approaches, tractable [Hunter 2004; Guiver and Snelson 2009; Caron and Doucet 2012].

Appropriateness of the PL Model in MCTS

The above model is based on the assumption that the rankings used to build the PL model are based on the true (still unknown) distributions of $Y^{(n_i,\pi)}_{s_t,T-t}$. This is exactly the same dilemma all MCTS approaches have: They try to assess the quality of one random variable (future performance) based on samples generated by (a sequence of) other random variables with deviating distributions.

While no model can generally close this gap without being omniscient, we consider the application of PL in this context superior to quantitative models. Like in the PL case, a quantitative model would require to assume a particular distribution class for the $Y^{(n_i,\pi)}_{s_t,T-t}$, e.g., all bounded distributions on $[0, 1]$, or a specific parametric family of probability distributions such as Gaussian or Gumbel. However, it is generally unclear how the past observations should be used to learn such distributions, because the relation between those observations and the distribution we want to estimate is difficult to grasp. A reasonable way to solve this problem are *rankings*: Based on the previous observations, we can produce rankings among child nodes that order nodes based on the current belief that they will reveal the best improvement; we compile the concrete values away into qualitative beliefs. The PL model is then a canonical solution to work upon such rankings. An additional advantage of this approach compared to Single Player UCT is that we neither need to map scores into the unit interval nor does it require a scaling parameter as adopted in [Schadd et al. 2008]. Since the notion of rankings disregards the absolute values, there is no need to adapt the algorithm to domain specific score ranges. Even more, the approach could even work with *ordinal* scores in the leaf nodes.

In the following, we describe strategies to compile rankings and to use the PL-model to make recommendations.

A Plackett-Luce Tree Policy

The PL Tree Policy works as follows. For each visited node n, the policy memorizes the observations made under it in a set $o^{(n)}$ containing one value for each roll-out that went over n. To recommend a concrete child of some node n, it generates a set of rankings from the observations of the children of n, fits a PL model, and recommends each node with a probability that is based on its skill parameter. To fit the PL model, we adopt the MM method described in [Hunter 2004], which is a maximum likelihood estimator for the skill parameters of the children. In the back-propagation phase, we simply include the new observation into the set of observations of all nodes on the path who are subject to tree policy selection themselves or their parent; the latter simply avoids that we waste memory to store observations that will never be relevant for decision making.

Three natural questions emerge from this description:

1. How are the rankings created from the observations?
2. Why does the policy return a *random* child node instead of simply returning the best one?
3. The policy seems to be memory and computation intense. Is this a problem and if yes, can we improve this?

We now discuss these questions in turn.

Rankings via Bootstrapping Preference Kernels We call the mechanism to create rankings a *preference kernel*. A preference kernel is a function κ that maps a vector of sets of observations (one set for each child node) to a multi-set of rankings. Formally, $\kappa(\langle o_1^{(n_1)}, .., o_{m_{n_1}}^{(n_1)}\rangle, .., \langle o_1^{(n_k)}, .., o_{m_{n_k}}^{(n_k)}\rangle)$ is a set that may contain any ranking of nodes $n_1, .., n_k$ an arbitrary number of times (hence it is a multi-set).

In this paper, we propose to build preference kernels upon *bootstrapping*, a well-known technique in statistics to reduce sample noise [Diaconis and Efron 1983]. Intuitively, we might want to create rankings based on the best value observed among all the roll-outs that included each of the children. That is, for an inner node n with children $n_1, .., n_k$, we may want to rank the k children based on $\min(o^{(n_i)})$. However, this would yield only one ranking and also may be too biased towards lucky roll-outs. Instead, we can draw a series of *sub-samples* of $o^{(n_i)}$ for every n_i of which we then compute the statistic that we believe to best *indicate* that the best solution can be observed under it; this yields a bootstrapping procedure. In order to obtain l independent rankings, we can l times draw a sub-sample for each child, and order the children by the mean of the desired statistics.

Every statistic over the bootstrap samples then constitutes a distinct preference kernel. For a principled procedure that follows the idea of condition (2), we should choose a statistic that leads to rankings that favor the child n_i that has the best probability to exhibit the best improvement when selecting it. While using the sample mean is similar to other MCTS approaches and is a robust default configuration, our evaluation shows that other preference kernels like mean minus standard deviation as in [Schadd et al. 2008] or the minimum can yield better (or worse) solutions in some scenarios. Of course, it is hard to know in advance which kernel will perform well, and the attempt to learn the suitability of kernels for specific problem classes plants interesting future work.

Probabilistic Recommendations One issue with adopting the PL approach above is that it is not directly clear *how* and *when* to use the estimates of the PL model to guide the search. With respect to the first point (how), it seems natural to recommend the node with the best estimated skill parameter. But doing this would effectively disable further exploration at this node as long as roll-outs under it maintain or improve upon past qualities, which is what we expect to happen due to the distribution drift. To enable exploration that is in accordance with the belief of optimality, we use the re-scaled skill parameter estimates of the PL model to derive probabilities to recommend the different nodes; this is analogous to how one proceeds in Thompson sampling. Regarding the second aspect (when), we must be aware that the skill parameters estimated by the PL model take very extreme values if the underlying sample basis is thin. If we would adopt the model with already one sample, the best node would have a skill parameter of 1 and the others a value of 0, again disabling exploration. It is hence indispensable to accumulate a certain basis of observations in order to enable a sane exploration behavior.

Since we are not aware of any results on reliability conditions of the PL model that would motivate a particular number of observations for the time when using it, we adopt a resource-aware probabilistic model to control the influence of the PL model onto the decisions of the tree policy. To this end, we introduce a non-decreasing influence function $\gamma : N \times \mathbb{N} \times \mathbb{N} \to \mathbb{R}_+$, which is used as an exponent for the determined skill parameters to obtain the probabilities of the child nodes. The PL tree policy chooses for each inner node n the successor n_j among $n_1, \dots, n_k$ by virtue of

$$\mathbb{P}\left(\pi^{PL}(n) = n_j\right) = v_j^{\gamma(n,t(n),T))}\Big(\sum_{i=1}^{k} v_i^{\gamma(n,t(n),T)}\Big)^{-1}$$

where v_j are the estimates of the exponentiated skill parameters, i.e., $\exp(\theta_i)$, of the underlying PL-model, $t(n)$ are the visits of node n up to time t, and T is the roll-out budget.

In spite of this conceptually probabilistic view, the PL tree policy can be still considered deterministic from the *functional* viewpoint. Internally, the policy draws a random successor, but this randomness can be seeded based on the current state and hence be made deterministic on the functional level such that the policy always returns the same successor for a given node in the light of a particular time budget and MCTS search state.

According to the above discussion, the values of γ for any node should traverse three phases depending on t and the node depth. For small t, γ should be 0 to guarantee unconditional exploration in the beginning. It should then increase and approach 1 for t in which exploration and following the PL model should be balanced, and eventually may be > 1 for t when the search should be fully steered by the best PL option. The question is where to locate the thresholds, say τ_1 and τ_2, between the three phases.

Our proposal for a realization of γ is to set the thresholds between the phases based on the time budget in a linear descending manner. The motivation is that exploration makes generally more sense in upper parts of the trees as already recognized in the context of UCT [Kocsis, Szepesvári, and Willemson 2006]. Without further knowledge, a robust choice is to let the thresholds between the phases decrease linearly with increasing node depth and hence make the algorithm commit more quickly to the seemingly best candidate in lower parts of the tree and enforce more exploration in upper parts. To achieve this balance, we propose to set the second threshold τ_2, i.e. for which $\gamma = 1$, to $\tau_2(d, T) := \tau_2(0, T)(1 - d/d_{max})$ where $d > 0$ is the node depth, d_{max} is the depth of the tree (average, sensed during the roll-outs), and $\tau_2(0, T) := T/(\sum_{d=0}^{d_{max}} \sum_{t=0}^{d}(1 - \frac{d-t}{d_{max}})(-b)^{-t})$ is the threshold for the root node in which b plays the role of the average branching factor sensed during search. In the supplement, we show that this choice for the threshold $\tau_2(0, T)$ in the root will yield an overall distribution of exploration such that the time budget is exhausted when the tree policy hits a leaf node for the first time. The first threshold τ_1, for the infimum of the support of γ, can be set to a value with moderate distance to τ_2, e.g. $\tau_1(d, T) := \max\{0.5\tau_2(d, T), \tau_2(d, T) - 100\}$. Using a sigmoid-shaped function between τ_1 and τ_2, e.g. a cosine function, we enable a relatively long unbiased exploration phase by a high τ_1, followed by smooth transition to model exploitation defined by τ_2. Needless to say, these parameters can be subject to optimization themselves in future work.

Computational Efficiency The PL tree policy has two potential resource bottlenecks. First, it is memory intense since it needs to memorize the observations in each node. Second, in the naive variant, it fits a PL model in each node in each roll-out. Even though each of them only takes a few milliseconds, the accumulation of those operations might lead to substantially larger decision times compared to, say, UCB1.

Whether or not these bottlenecks become effective depends on the application domain. In many domains that include simulation, computing the *score* of a path is already so costly that the computation of the *roll-out* is negligible. For example, in AutoML [Mohr, Wever, and Hüllermeier 2018], evaluating a candidate can easily take several minutes, making discussions about more effective roll-out computation obsolete.

If efficiency in the path generation matters, there are several options to reduce this computational burden without significantly affecting the inference mechanism. First, we can simply *update* the previous PL model by starting the MM algorithm from the optimal point of the last iteration; this often leads to one-step optimization runs. Second, we can safely commit to a particular child if the probability of other children being selected becomes negligible, e.g. for a probability of 99%. We can then not only free memory by removing the observations but also save the time of fitting a model for such a node. For the latter, we can decide more generally to only infer a new PL model with probability $k(1 - \max p_j)/(k-1)$, which enforces rebuilds with probability 1 for highly undecided nodes and 0 for completely decided nodes; here, p_j is the probability of child j being selected in the *previous* round. Other possibilities include the check whether an observation will significantly change the probability model, or conducting updates only sparsely as in the BRUE algorithm [Feldman and Domshlak 2014].

Evaluation

Experimental Setup

Empirical evaluations on the addressed problem are hard to conduct, especially if they require the (very costly) development of a simulator to compute the score function ϕ. While most previous approaches have only been evaluated on a single domain, we use three benchmark problems with different variations to compare the algorithms. Also the set of baselines has been usually quite limited in that the only baseline has been UCT. In contrast, we compare PL-MCTS against seven algorithms to enable more profound insights.

The technical setup is as follows. To maximize comparability, all algorithms are implemented in Java, and the MCTS algorithms only deviate in the implementation of the tree policy. Implementations are available at <*anonymized*> and seedable such that the results are reproducible. All computations were run on one core of an Intel Xeon Gold "Skylake" 6148, 2.4 GHz with 16GB memory. In order to guarantee a fair comparison, each algorithm was run until a predefined overall time limit was met, which is the same for all algorithms but varies among the problems as specified below.

Benchmark Problems We consider three problems to evaluate our approach, which we now describe in detail. To satisfy the needs of UCT, all scores are mapped into the unit interval by exploiting lower and upper bounds respectively; the score in the maximization problem is subtracted from 1 in order to obtain a minimization problem.

The *Timed TSP with Breaks* (TTSPB) extends the classical TSP in two ways. First, the cost to travel from one location to another now depends on the time (hour of day) when the edge is used (to model traffic jams) as suggested in [Picard and Queyranne 1978]. Second, truck drivers must take a break after a time as considered recently in [van der Tuin, de Weerdt, and Batz 2018]. A driver must make a small break of 15 minutes every 4 hours, and a long break (of 8 hours) every 8 hours. We are not aware of any heuristic for this highly state-dependent action cost setup. We consider problems with 20, 30, and 50 locations with timeouts of 1, 4, and 10 minutes.

The *Job Scheduling With Variable Release Dates* (VRDJS) is a scheduling problem of minimizing total weighted flowtime and in which the arrival time of jobs can be chosen but must not be later than a common deadline d. This recently studied problem has shown to be not only NP-hard but also that no meaningful heuristics can be generated by currently known techniques [Mohr, Mejía, and Yuraszeck 2021]. We evaluate the case of 100 jobs on 2, 5, and 9 machines with job weights and processing times sampled uniformly from $\{1, .., 100\}$ and d corresponding to 40% of the total processing time, which constitutes difficult instances. The timeout is 5 minutes in all setups.

SameGame is a single-player board game, in which the player has to clear a vertical $l \times l$ board initially filled randomly with pieces of 5 different colors. SameGame has been subject of evaluation in single-player MCTS [Schadd et al. 2008]. We refer to this paper for details, in which the authors also discuss why SameGame cannot be addressed efficiently with classical A* or IDA*. Timeouts are 5, 15, and 90 minutes for board sizes $l = 10, 15,$ and 20 respectively.

Algorithm Setup and Baselines PL-MCTS can be parametrized in the preference kernel κ and the influence function γ. Using a bootstrapping based kernel for κ, one can configure the number, size, and aggregation statistic. The only reason to set the first two parameters low are resource limitations, and, using generous values of 1000 bootstraps (rankings) based on 100 samples per child each, we will see that we do not observe performance issues. As an aggregation function, we use the mean value of each bootstrap sample to derive rankings as discussed above as a default. In addition, we show results that would result from choosing the mean - standard deviation or the minimum as an aggregation function. Regarding the influence function γ, the above resource-aware proposal constitutes a reasonable default setup. An estimate of the number of roll-outs T is updated in each iteration based on the empirical roll-out times and the time budget (in terms of wall-time), and thresholds for τ_1 and τ_2 are updated based on this estimate (and hence vary over time for each node).

In the following, we describe the algorithms against which we compare PL-MCTS in a *default parameter* evaluation. Comparing MCTS algorithms is a tricky undertaking, because these algorithms have parameters, and the success of an algorithm can substantially vary with different parameter values. The two canonical possibilities for evaluation are to either fix one default value for each of the algorithms that appears "reasonable" or to *optimize* over the parameters and search the "best" parameters for each algorithm in each setup. The problem of the latter is that it is methodologically difficult to decide which optimization is fair and which is not. For example, if one algorithm has five parameters and another algorithm has only one, the optimization process for the second algorithm is much faster. How much time is allowed for this optimization? In fact, doing this kind of optimization in fact already *is* a kind of search on a meta-level, and the used resources should count into the overall time budget of each considered algorithm. While this approach is not uninteresting, we are not aware that it has been done in a methodologically clean way up to date. Instead, we base our evaluation on a *default* parametrization per algorithm in which all algorithms are equipped with a reasonable paramatrization that should work well "on average". The exact parametrizations can be found in the supplement.

The variants of MCTS we consider are as follows. First, we consider with UCT [Kocsis, Szepesvári, and Willemson 2006], DNG [Bai, Wu, and Chen 2013], and BRUE [Feldman and Domshlak 2014] three state-of-the-art MCTS approaches to learn policies for MDPs. The parametrizations are fixed to reasonable default values; details per algorithm are contained in the supplement. Second, we consider the single-player versions of UCT as proposed in [Schadd et al. 2008], which applies a slight modification of UCB1 considering not only the mean but also the standard deviation of the samples; this is denoted as SP-UCT. Third, we consider the TAG algorithm [De Mesmay et al. 2009], which is an MCTS approach specifically taylored for single-player games based on extreme bandits.

We also include two algorithms specifically dedicated to single player games even though not directly fitting into the MCTS scheme. The first is Nested Monte Carlo Tree Search (NMCS) [Cazenave 2009]. NMCS is simply a random search that, instead of starting a single random search at the root, starts one random search under *each* node in a predefined depth (level). The algorithm then simply memorizes the best solution seen so far, and this process is repeated until the time budget is exhausted. There is no notion of using observations to *guide* the search; in particular there is nothing like a tree policy. The Nested Rollout Policy Adaptation for Monte Carlo Tree Search (NRPA) is a kind of extension of this algorithm [Rosin 2011]. The idea to initiate different searches in a pre-defined depth is the same, but NRPA adopts a learning search process under those nodes instead of drawing only one single sample. While it also does not make use of the explicit concept of a tree policy or default policy, it *does* incorporate observations into the decision making process by modifying the *probability* of the children, more precisely by increasing the probabilities of better children to be drawn the next time. It therefore *evolves* the policy over time. In a way, it can be seen as an algorithm that initiates for each node in depth l a learning search procedure that is independent of the other nodes under which such a search is started. Like NMCS, it returns the best result observed during search.

In addition, we add two algorithms to the baseline portfolio, which are extreme in the exploration-exploitation spectrum. On the exploration side, we add a simple random search (RANDOM), which does not take into account the observations made during search at all. On the exploitation side, we add a classical best-first (BF) search that gains its node evaluations from a fixed number of 3 random samples (taking the minimum value of them).

Experimental Results

The experimental results are summarized in Table 1. Due to the different ranges of possible scores in the different instances, we report the *relative* scores in that the score of each algorithm is $(score - best)/best$ on instances of the minimization problems and $score/best$ on instances of SameGame, where $best$ is the best observed score of any algorithm in the respective instance. We report the empirical average of those values over 100 different instances and their empirical standard deviation. Best average scores are in bold, while the results whose distribution might be identical to the best (according to a Wilcoxon signed rank test with 95% significance level) are underlined.

We do not want to give the impression to cherry-pick among the different kernels for PL-MCTS. Therefore, PL-MCTS is primarily represented by the mean kernel as a choice to use PL-MCTS out of the box. However, to increase insights, the results for the other two kernels are still provided in the separated last two rows. They are not considered in the computation of the best algorithm performances (hence never marked in bold even if best). Instead, we use the • symbol to indicate that the respective kernel *would* have improved upon the best other solution in that scenario.

The experimental evaluation gives clear evidence that PL-MCTS is a highly competitive algorithm for the addressed problem. On almost all problems, PL-MCTS achieves best results and in several cases with substantial gaps to the runner-up, e.g., in all TTSPB problems, and in the 2 and 5 machine problems of the VRDJS. PL-MCTS is in *no* problem substantially worse than any other tree policy; the only algorithm that achieves to outperform PL-MCTS in one case is the BF algorithm on the 15x15 SameGame problem.

The mediocre performance of theoretically sustained UCT and BRUE should not surprise. First, the bounds are for cumulative and simple regret, which we discussed above to be not particularly relevant for our problem. Second, the regret bounds are fairly loose and hence, admit slow convergence.

Another remarkable observation is that SP-UCT and the TAG algorithm, both being specifically tailored for this problem class, largely fail to produce competitive results. The TAG algorithm is extremely greedy and similar to the minimum preference kernel. The important difference in this case is that PL-MCTS is not so quick in the *application* of

	TTSPB (MIN) - per num of locations			VRDJS (MIN) - per num of machines			SameGame (MAX) - per board size		
	20	30	50	2	5	9	10	15	20
RANDOM	0.48±0.19	0.76±0.17	0.76±0.25	0.25±0.33	0.23±0.35	0.27±0.38	0.84±0.24	0.57±0.24	0.68±0.17
BF	0.38±0.24	0.58±0.21	0.59±0.28	0.24±0.34	$\underline{0.23\pm0.4}$	$\underline{0.2\pm0.38}$	0.5±0.37	**0.7±0.26**	$\underline{0.72\pm0.19}$
UCT	0.48±0.17	0.75±0.17	0.75±0.26	0.18±0.26	0.23±0.36	0.28±0.39	**0.85±0.22**	0.54±0.23	$\underline{0.72\pm0.16}$
DNG	0.21±0.26	0.27±0.26	0.25±0.31	0.27±0.28	0.25±0.33	0.24±0.33	0.59±0.36	0.53±0.24	$\underline{0.72\pm0.17}$
BRUE	0.54±0.18	0.77±0.17	0.78±0.26	0.22±0.32	0.2±0.34	0.23±0.36	0.81±0.27	0.6±0.25	**0.74±0.17**
SP-UCT	0.54±0.18	0.79±0.16	0.81±0.22	0.21±0.29	0.19±0.33	0.27±0.38	0.8±0.27	0.63±0.26	**0.74±0.17**
TAG	0.82±0.19	0.94±0.12	0.84±0.25	0.27±0.08	0.19±0.06	0.2±0.06	0.14±0.24	0.46±0.23	0.61±0.17
NMCS-3	0.48±0.03	0.61± 0.02	0.74±0.03	0.19±0.08	0.20±0.06	0.23±0.06	0.69±0.24	0.52±0.23	0.68±0.17
NRPA-3	0.45±0.03	0.59± 0.02	0.68±0.03	0.18±0.05	0.18±0.03	0.21±0.04	0.7±0.12	0.52±0.23	0.68±0.17
PL-MCTS-MEAN	**0.16±0.26**	**0.15±0.25**	**0.15±0.28**	**0.14±0.32**	**0.13±0.28**	**0.19±0.35**	**0.85±0.27**	0.62±0.26	**0.74±0.18**
PL-MCTS-MEAN-STD	$\underline{0.14\pm0.25}$●	$\underline{0.13\pm0.25}$●	$\underline{0.18\pm0.26}$	0.19±0.35	$\underline{0.19\pm0.34}$	$\underline{0.22\pm0.38}$	$\underline{0.88\pm0.24}$●	0.6±0.27	$\underline{0.75\pm0.18}$●
PL-MCTS-MIN	0.42±0.2	0.71±0.21	0.73±0.26	0.17±0.34	$\underline{0.22\pm0.39}$	$\underline{0.25\pm0.4}$	0.8±0.29	$\underline{0.77\pm0.25}$●	$\underline{0.76\pm0.17}$●

Table 1: Relative results for the respective wall-times. Best results in bold, not significantly worse underlined.

the policy and hence allows for more exploration prior to committing to the node recommended by the policy.

The 15x15 SameGame scenario is interesting as it constitutes a scenario in which following the mean is a bad advisor. In fact, all mean-oriented techniques like UCT, BRUE, DNG, PL-MCTS-MEAN are not so stable in reaching the high-quality regions. The fact that TAG also fails on this problem shows that here it is a good idea to follow the minimum but not all too greedy (PL-MCTS-MIN has the advantage of not committing too quickly and to regularize the observations by bootstrapping).

It is also noteworthy that the potential performance bottlenecks discussed above were successfully eliminated by the proposed remedies. In the supplement we provide a detailed overview of the achieved iterations of each algorithm from which it becomes evident that PL-MCTS is not significantly slower than the competitors and even often faster than DNG.

Our conclusion from this evaluation is that PL-MCTS is a very promising approach for finding good solutions to optimal path search problems for which no meaningful heuristic is available. While providing a solid fall-back with the mean kernel, its flexibility in the control strategy via the preference kernels points to interesting future work in the question to *learn* which kernel to use in a particular domain or even to detect *online* which one to use.

Conclusion

We proposed PL-MCTS, a single-player MCTS approach to address optimal path search problems for which no classical heuristics are available. Combining preference kernels with the ranking-based Plackett-Luce model provides a variant of MCTS that conceptually better closes the gap between past observations and decision making for future observations than other approaches. Fortunately, the robust mean kernel makes the customization of PL-MCTS *optional* and allows its use out of the box. In future work, we plan to *learn* the score landscape and decision heights during search, and then to adjust the kernels and the influence parameter *online*. Moreover, it would be interesting to analyze how the time at which observations were made can be considered to improve the ranking construction. Besides, exploring options to apply PL-MCTS for MDPs poses an interesting challenge.

References

Alvo, M.; and Philip, L. 2014. *Statistical methods for ranking data*. Springer.

Bai, A.; Wu, F.; and Chen, X. 2013. Bayesian mixture modelling and inference based Thompson sampling in Monte-Carlo tree search. In *Advances in neural information processing systems*, 1646–1654.

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; et al. 2016. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence* 237: 41–58.

Bjornsson, Y.; and Finnsson, H. 2009. Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games* 1(1): 4–15.

Borghesi, A.; Milano, M.; Gavanelli, M.; and Woods, T. 2013. Simulation Of Incentive Mechanisms For Renewable Energy Policies. In *ECMS*, 32–38.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1): 1–43.

Caron, F.; and Doucet, A. 2012. Efficient Bayesian inference for generalized Bradley–Terry models. *Journal of Computational and Graphical Statistics* 21(1): 174–196.

Carpentier, A.; and Valko, M. 2014. Extreme bandits. In *Advances in Neural Information Processing Systems*, 1089–1097.

Cazenave, T. 2009. Nested monte-carlo search. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Charras-Garrido, M.; and Lezaud, P. 2013. Extreme value analysis: an introduction. *Journal de la Société Française de Statistique* 154(2): pp–66.

Corti, A.; Manzoni, V.; and Savaresi, S. M. 2012. Simulation of the impact of traffic lights placement on vehicle's energy consumption and CO 2 emissions. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, 620–625. IEEE.

De Haan, L.; and Ferreira, A. 2007. *Extreme value theory: an introduction*. Springer Science & Business Media.

De Mesmay, F.; Rimmel, A.; Voronenko, Y.; and Püschel, M. 2009. Bandit-based optimization on graphs with application to library performance tuning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 729–736. ACM.

Diaconis, P.; and Efron, B. 1983. Computer-intensive methods in statistics. *Scientific American* 248(5): 116–131.

Feldman, Z.; and Domshlak, C. 2014. Simple regret optimization in online planning for Markov decision processes. *Journal of Artificial Intelligence Research* 51: 165–205.

Guiver, J.; and Snelson, E. 2009. Bayesian inference for Plackett-Luce ranking models. In *proceedings of the 26th annual international conference on machine learning*, 377–384. ACM.

Henard, C.; Papadakis, M.; Harman, M.; and Le Traon, Y. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, 517–528. IEEE Press.

Hunter, D. R. 2004. MM algorithms for generalized Bradley-Terry models. *The annals of statistics* 32(1): 384–406.

Kocsis, L.; Szepesvári, C.; and Willemson, J. 2006. Improved monte-carlo search. *Univ. Tartu, Estonia, Tech. Rep* 1.

Mohr, F.; Mejía, G.; and Yuraszeck, F. 2021. Single and Parallel Machine Scheduling with Variable Release Dates.

Mohr, F.; Wever, M.; and Hüllermeier, E. 2018. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning* 107(8-10): 1495–1515.

Picard, J.-C.; and Queyranne, M. 1978. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations research* 26(1): 86–110.

Rosin, C. D. 2011. Nested rollout policy adaptation for Monte Carlo tree search. In *Ijcai*, 649–654.

Schadd, M. P.; Winands, M. H.; Van Den Herik, H. J.; Chaslot, G. M.-B.; and Uiterwijk, J. W. 2008. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, 1–12. Springer.

Streeter, M. J.; and Smith, S. F. 2006. A simple distribution-free approach to the max k-armed bandit problem. In *International Conference on Principles and Practice of Constraint Programming*, 560–574. Springer.

Train, K. E. 2009. *Discrete choice methods with simulation*. Cambridge university press.

van der Tuin, M. S.; de Weerdt, M.; and Batz, G. V. 2018. Route Planning with Breaks and Truck Driving Bans Using Time-Dependent Contraction Hierarchies. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.