PARAMETER TUNING BY SIMPLE REGRET ALGORITHMS AND MULTIPLE SIMULTANEOUS HYPOTHESIS TESTING

Amine Bourki**, Matthieu Coulm**, Philippe Rolet*,
**TAO, Inria, Umr CNRS 8623, Univ. Paris-Sud, 91405 Orsay, France*
amine.bourki@gmail.com, mcoulm@gmail.com, philippe.rolet@lri.fr

Olivier Teytaud*, Paul Vayssière**
***EPITA, 16 rue Voltaire, 94270 Le Kremlin-Bicêtre, France*
olivier.teytaud@inria.fr, paul.vayssiere@gmail.com

Keywords: Simple Regret; Automatic Parameter Tuning; Monte-Carlo Tree Search.

Abstract: "Simple regret" algorithms are designed for noisy optimization in unstructured domains. In particular, this literature has shown that the uniform algorithm is indeed optimal asymptotically and suboptimal non-asymptotically. We investigate theoretically and experimentally the application of these algorithms, for automatic parameter tuning, in particular from the point of view of the number of samples required for "uniform" to be relevant and from the point of view of statistical guarantees. We see that for moderate numbers of arms, the possible improvement in terms of computational power required for statistical validation can't be more than linear as a function of the number of arms and provide a simple rule to check if the simple uniform algorithm (trivially parallel) is relevant. Our experiments are performed on the tuning of a Monte-Carlo Tree Search algorithm, a great recent tool for high-dimensional planning with particularly impressive results for difficult games and in particular the game of Go.

1 INTRODUCTION

We consider the automatic tuning of new modules. It is quite usual, in artificial intelligence, to design a module, for which there are several free parameters. This is natural in supervised learning, optimization (Nannen and Eiben, 2007b; Nannen and Eiben, 2007a), control (Lee et al., 2009; Chaslot et al., 2009). We will here consider the particular case of Monte-Carlo Tree Search (Chaslot et al., 2006; Coulom, 2006a; Kocsis and Szepesvari, 2006; Lee et al., 2009).

Consider a program, in which a new module with parameter $\theta \in \{1,\dots,K\}$ has been added. In the bandit literature, $\{1,\dots,K\}$ is referred to as the set of arms. Then, we're looking for the best parameter $\theta \in \{1,\dots,K\}$ for some performance criterion; the performance criterion $L(\theta)$ is stochastic. We have a finite time budget T (also termed horizon), we can have access to T realizations of $L(\theta_1), L_(\theta_2), \dots, L(\theta_T)$ and we then choose some $\hat{\theta}$. The game is as follows:

- The algorithm chooses $\theta_1 \in \{1,\dots,K\}$.
- The algorithm gets a realization r_1 distributed as $L(\theta_1)$.
- The algorithm chooses $\theta_2 \in \{1,\dots,K\}$.
- The algorithm gets a realization r_2 distributed as $L(\theta_2)$.
- ...
- The algorithm chooses $\theta_T \in \{1,\dots,K\}$.
- The algorithm gets a realization r_T distributed as $L(\theta_T)$.
- The algorithm chooses $\hat{\theta}$.
- The loss is $r_T = \max_\theta \mathbb{E}L(\theta) - \mathbb{E}L(\hat{\theta})$.

The performance measure is the simple regret(Bubeck et al., 2009), i.e. $r_T = \max_\theta \mathbb{E}L(\theta) - \mathbb{E}L(\hat{\theta})$, and we want to minimize it. Then main difference with noisy nonlinear optimization is that we don't use any structure on the domain.

We point out the link with No Free Lunch theorems (NFL (Wolpert and Macready, 1997)), which claim that all algorithms are equivalent when no prior knowledge can be explored. Yet, there are some differences in the framework: NFL considers deterministic optimization, in which testing several times the same point is meaningless. We here consider noisy optimization, with a small search space: all the difficulty is in the statistical validation, for choosing which points in the search space should be tested more intensively.

Useful notations:

- $\#E$ is the cardinal of the set E;
- $N_t(i)$ is the number of times the parameter i has been tested at iteration t, i.e.

  $$N_t(i) = \#\{j \leq t; \theta_j = i\}.$$

- $\hat{L}_t(i)$ is the average reward for parameter i at iteration t, i.e.

  $$\hat{L}_t(i)\frac{1}{N_t(i)}\sum_{j\leq t;\theta_j=i} r_j.$$

  (well defined if $N_t(i) > 0$)

Section 2 recalls the terminology of simple regret and discusses the relevance for Automatic Parameter Tuning (APT). Section 3 mathematically considers the statistical validation, which was not yet, to the best of our knowledge, considered for simple regret algorithms; we will in particular show that the dependency of the computational cost as a function of the number of tested parameter values is at best linear, and therefore it is not possible to do better than this linear improvement in terms of statistical validation - we will then switch to experimental analysis, and we'll show that the improvement is indeed improved by far less than a linear factor in our real world setting (section 4).

2 SIMPLE REGRET: STATE OF THE ART AND RELEVANCE FOR AUTOMATIC PARAMETER TUNING

We consider the case in which $L(\theta)$ is, for all θ, a Bernoulli distribution. (Bubeck et al., 2009) states that (i) the naive algorithm distributing θ_i uniformly among the possible parameters, i.e. $\theta_i = \mod(i,K)+1$ with $\mod$ the modulo operator, with $\hat{\theta} = \arg\max_i \hat{L}(i)$, has simple regret

$$\mathbb{E}r_T = O(\exp(-c\cdot T)) \tag{1}$$

for some constant c depending on the Bernoulli parameters (more precisely, on the difference between the parameters of the best arm and of the other arms). This is for $\hat{\theta}$ maximizing the empirical reward, *i.e.*

$$\hat{\theta} \in \arg\min_{\theta} \hat{L}_T(\theta)$$

and this is proved optimal.

If we consider distribution-free bounds (i.e. for a fixed T, we consider the supremum of $\mathbb{E}r_T$ for all Bernoulli parameters), then (Bubeck et al., 2009) shows that, with the same algorithm,

$$\sup_{distribution} \mathbb{E}r_T = O(\sqrt{K\log K/T}), \tag{2}$$

where the constant in the $O(.)$ is a universal constant; Eq. 2 is tight within logarithmic factors of K; there's a lower bound for all algorithms of the form.

$$\sup_{distribution} \mathbb{E}r_T = \Omega(\sqrt{K/T}).$$

Importantly, the best known upper bounds for variants of UCB(Auer et al., 2002) are significantly worse than Eq. 1 (the simple regret is then only polynomially decreasing) and significantly worse than Eq. 2 (by a logarithmic factor of T) - see (Bubeck et al., 2009) for more on this.

However, it is clearly shown also in (Bubeck et al., 2009) that for small values of T, using a variant of UCB for choosing the θ_i and $\hat{\theta}$ is indeed much better than uniform sampling. The variant of UCB is as follows, for some parameter $\alpha > 1$:

$$\hat{\Theta}_t = \arg\max_i N_t(i).$$

$$\Theta_i = mod(i,K)+1 \text{ if } i \leq K$$

$$\Theta_i = \arg\max_i \hat{L}_t(i) + \sqrt{\alpha\log(t-1)/N_{t-1}(i)} \text{ otherwise.}$$

Simple regret is a natural criterion when working on automatic parameter tuning. However, the theoretical investigations on simple regret did not answer the following question: how can we validate an arm selected by a simple regret algorithm when a baseline is present ? In usual cases, for the application to parameter tuning, we know the score before a modification, and then we tune the parameters of the optimization: we don't only tune, we validate the tuned modification; this question is nonetheless central in many applications in particular when modifications are included automatically by the tuning algorithm (Nannen and Eiben, 2007b; Nannen and Eiben, 2007a; Hoock and Teytaud, 2010). We'll see in next sections that the naive solution, consisting in testing separately each arm, is not so far from being optimal.

3 MULTIPLE SIMULTANEOUS HYPOTHESIS TESTING IN AUTOMATIC PARAMETER TUNING

As pointed out above, a goal different from minimizing the simple regret consists in finding a good arm could be (i) finding a good arm if any (ii) avoiding selecting a bad arm if there's no good arm (no arm which outperforms the baseline). We'll briefly show how to apply Multiple Simultaneous Hypothesis Testing (MSHT), and in particular its simplest and most well known variant termed the Bonferroni correction, to Automatic Parameter Tuning. MSHT(Holm, 1979; Hsu, 1996) is very classical in neuro-imagery(Pantazis et al., 2005), bioinformatics, tuning of optimizers(Nannen and Eiben, 2007b; Nannen and Eiben, 2007a).

MSHT consists in statistically testing several hypothesis in same time: for example, when 100 sets of parameters are tested simultaneously, then, whenever each set is tested with confidence 95%, and whenever all sets of parameters have no impact on the result, then with probability $1-(1-0.05^100) \simeq 99.4\%$ at least one set of parameters will be validated. MSHT is aimed at correcting this effect, so that taking into account the multiplicity of tests we can have modified tests so that the overall risk remains lower than 5%.

Assume that we expect arms with standard deviation σ (we'll see that for our applications, σ is usually nearly known in advance; it can also be estimated dynamically during the process). Then, the standard Gaussian approximation says that with probability 90%[1], the difference between $\hat{L}_t(\theta)$ and $L_t(\theta)$ for arm θ is lower than $1.645\sigma/\sqrt{N_t(i)}$:
with probability 90%,

$$|\hat{L}_t(\theta)-L_t(\theta)| \leq 1.645\sigma/\sqrt{N_t(i)}. \qquad (3)$$

The constant 1.645 directly corresponds to the Gaussian probability distribution (the precise value is $\Phi^{-1}((1+0.9)/2)=1.645$); a Gaussian standard distribution is ≤ 1.645 in absolute value with probability 90%. If we consider several tests simultaneously, i.e. we consider K arms, then Eq. 3 becomes Eq. 4:
with probability 90%,

$$\forall\theta\in\{1,2,\dots K\}|\hat{L}_t(\theta)-L_t(\theta)| \leq t_K\sigma/\sqrt{N_t(i)} \quad (4)$$

where, with the so-called Bonferroni correction, $t_K = -\Phi^{-1}(0.05/K)$ where Φ is the normal cumulative distribution function[2]. This is usually estimated with

$$\frac{\exp(-t_K^2)}{t_K\sqrt{2\pi}} = 0.05/K \qquad (5)$$

and therefore if we expect improvements of size δ, we can only validate a modification with confidence 90% with n experiments per arm if t_K solving Eq. 5 verifies $t_K\sigma/\sqrt{n}\leq\delta$; a succinct equation for this is

$$s = \delta\sqrt{n}/\sigma \qquad (6)$$

$$\frac{\exp(-s^2)}{s\sqrt{2\pi}} \leq 0.05/K \qquad (7)$$

This shows that for other quantities fixed, n has a logarithmic dependency as a function of K.

A numerical application for $\delta=0.02$, $K=49$ and $\sigma=\frac{1}{2}$ is

$$s = 0.04\sqrt{n},$$

$$\frac{\exp(-s^2)}{s\sqrt{2\pi}} = 0.05/49.$$

which implies $n\geq 3219$; this implies that for our confidence interval, we require 3219 runs per arm (i.e. $\inf_\theta N_T(\theta)\geq 3219$). We'll see that this number is consistent with our numerical experiments later. Interestingly, with only one arm, *i.e.* $K=1$, we get $n\geq 1117$; this is not so much better, and suggests that whatever we do, it will be difficult to get significant results with subtle techniques for pruning the set of arms: if there is only one arm, we can only divide the computational cost for this arm by $O(\log(K))$. In case of perfect pruning, n is also naturally multiplied by K (as all the computational power is spent on only one arm instead of K arms); this provides an additional linear factor, leading to a roughly linear improvement in terms of computational power as a function of the number of arms, in case of perfect pruning.

Bernstein races

This paper is devoted to the use of simple regret algorithms to APT, compared to the most simple APT algorithm, namely uniform sampling (which is known asymptotically optimal for simple regret); Bernstein races are therefore beyond the scope of this paper. Nonetheless, as our results emphasize the success of uniform sampling (at least in some cases), we briefly discuss Bernstein races. In (Mnih et al., 2008; Hoock and Teytaud, 2010), Bernstein races were considered as tools for discarding statistically bad arms: this is equivalent to $Uniform$, except that tests as above are applied periodically, and statistically bad arms are discarded. This discards arms earlier than the uniform algorithm above which just checks the result at the end, but increases the quantity K involved in tests (as in Eqs. 6 and 7), *even if no arm can be rejected*. The fact that testing arms for discarding on the fly has a cost, whenever no arm is discarded, might be surprising at first view - it is a known effect that when multiple tests are performed, then the number of samples required for a same confidence

[1] The constant 90% is arbitrary; it means that we decide that results are guaranteed within risk 10%.

[2] Note that a tighter formula is $t_K = -\Phi^{-1}(1-(1-0.05)^K)$; this holds thanks to independence of the different arms.

rate on the result is much higher. This approach can therefore at most divide the computational power by $K\log(K)$ before an arm is validated, and the computational power is indeed increases when no early discarding is possible. Nonetheless, this sound approach is probably the best candidate when the visualization is not crucial - $Uniform$ can provide nice graphs as Fig. 1 or 2, whereas Bernstein races can almost ignore some arms, but the statistical validation is nonetheless made.

4 EXPERIMENTAL VALIDATION: THE TUNING OF MOGO

Monte-Carlo Tree Search (MCTS (Chaslot et al., 2006; Coulom, 2006a; Kocsis and Szepesvari, 2006)) is now an important area of artificial intelligence. Interestingly, it is efficient both for games and for planning problems, and it has particularly good performance when the size of the state space is huge and supervised learning on the domain is difficult. MoGo(Lee et al., 2009) is one of the main MCTS programs[3], dedicated to the most classical testbed: the game of Go. MoGo relies on a large number of modules, with complicated tuning (Chaslot et al., 2009). A puzzling feature of MoGo, and of Monte-Carlo Tree Search algorithms, is that its tuning is highly dependent of the conditions: the best parameters are not the same for different experimental conditions. Without loss of generality, we will here consider the tuning of a neural network for biasing the patterns (see (Lee et al., 2009) for more on this), in two experimental conditions:

- tuning of MoGo for blitz games;
- tuning of MoGo for standard time settings.

Both settings are important. In particular, in sudden death games, the final part of the games is sometimes played very quickly by one or two of the players. The neural network was handcrafted (see section 4.1), with only two parameters left, and these two parameters were then discretized. The modification is described in the next section, and we then present the tuning of this modification in two different frameworks (namely blitz and non-blitz).

[3]The authors of MoGo make its CVS available upon request.

4.1 Automatic Parameter Tuning for Monte-Carlo Tree Search

Biases have been introduced in Monte-Carlo Tree Search since (Coulom, 2006b; Chaslot et al., 2006). It has been shown in (Lee et al., 2009) that it provides a very big improvement in MoGo. The bias introduced by a pattern in MoGo is linear as a function of a weight w, which is computed usually as follows:

$$w_{pattern} = \gamma \times p_{pattern}$$

where γ is a constant and $p_{pattern}$ is the probability that a move is played, according to a database of professional games, when it matches $pattern$. Other informations around a pattern include

- $k_{pattern}$, representing the size of the pattern (in terms of the number of locations that should match for this pattern)
- $v_{pattern}$, termed the significance of the pattern, equal to $p_{pattern} \times \sqrt{n_{pattern}}$, where $n_{pattern}$ is the support of the pattern, i.e. the number of times the pattern appears in the database.

Several modifications for the formula specifying the bias as a function of $w_{pattern}$ (the bias depends on several quantities) have been shown efficient in (Lee et al., 2009); several other works around MCTS have shown that the detailed optimization of the formula could provide significant improvements (Rolet et al., 2009; De Mesmay et al., 2009; Auger and Teytaud, pted). We here consider the following modification as a typical example of automatic parameter tuning:

$$w_{pattern} = \gamma p_{pattern} \times (1 + \tanh(\alpha_1 k_{pattern} + \alpha_2 v_{pattern}))\,. \tag{8}$$

This can be considered as the inclusion of a single neuron into the bandit formula. It has been shown in (Lee et al., 2009) that a good parameter tuning for one experimental condition is not necessarily a good parameter tuning for another experimental condition. Therefore, we look for an automatic parameter tuning which has the following properties:

- It should be reliable and automatic, so that the program can be automatically adapted to other experimental conditions.
- As testing this kind of modifications is usually very expensive, the algorithm should be parallel.

Under these conditions, such an automatic parameter tuning can be included in the non-regression testing sequence of a program, and be applied automatically for new experimental conditions.

4.2 Tuning of MoGo for Blitz Games

This section considers the tuning of MoGo for 2000 simulations per move in 19x19; this is half a second per move on one core. We tested, each parameter (α_1, α_2) in $\{-0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3\}^2$.

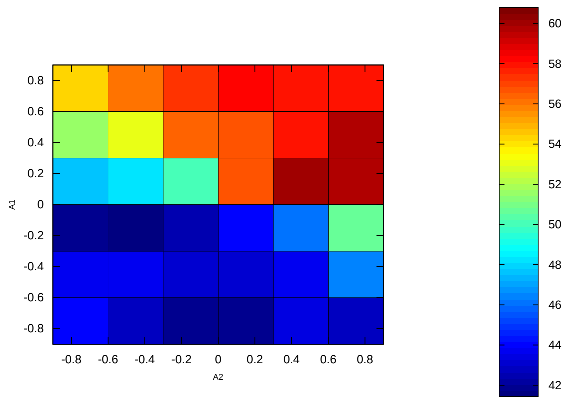

Figure 1: Success rate of MoGo as black against the baseline. The baseline reaches 54% (for this time setting, the game is easier for black than for white). We see a clear gradient, and visually the user has the feeling that the optimal parameter is nearly found. This is performed by the uniform sampling method advocated in section 3 for large number of time steps; we are in the case of a large number of time steps (around $T = 300000$), because MoGo is used in blitz and is therefore quite fast (we exceed the number of evaluations required for $Uniform$ according to Eq. 7). The experiments were performed on Grid5000.

The success rate, as a function of the two-dimensional parameter, is presented in Fig. 1. Each parameter was tested between 3500 and 11000 times (most of them nearly 6000 times); this discrepancy is due to the submission of jobs in preemptable mode on the grid. We satisfy Eq. 7, and consistently with the theory of MSHT we have significant results (see section 3; the numerical application is consistent with the experiments in this figure).

4.3 Tuning of MoGo for standard time settings

This section considers the tuning of MoGo for 10 seconds per move in 19x19. We tested each parameter (α_1, α_2) in $\{-0.09, -0.06, -0.03, 0, 0.03, 0.06, 0.09\}^2$, nearly 1800 times each. This is a computational cost already much bigger than in the experiment above. The unclear results are presented in Fig. 2: the empirically best arm (0.09,0.03).

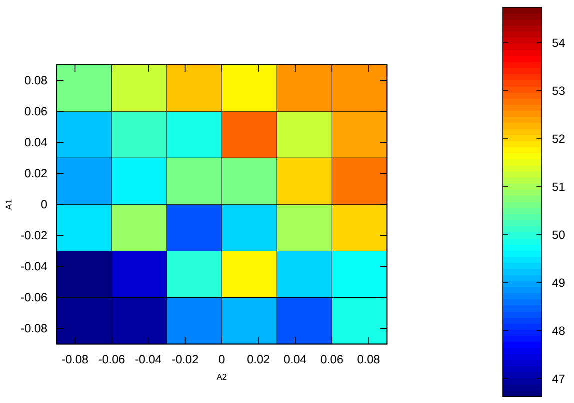

Figure 2: Success rate of MoGo as black against the baseline with 10s/move, as a function of the two-dimensional parameter. Here $T \simeq 88200$ (i.e. after division by $K = 49$ we see that we are below the budget required for $Uniform$ to work according to Eq. 7 in section 3), the sampling is nearly uniform, and the computational cost for this is already around 6 times bigger than in Fig. 1 which was in the much easier case of blitz game.

We now reproduce the same experiment, but with UCB(2) (see Alg. 1) instead of uniformly sampling all the values of the parameters. $\hat{\theta}$ is then the arm which is most explored by UCB(2). With the same horizon, the most played arm was (0.00,0.09).

We then compared the two approaches, from the point of view of $\mathbb{E}L(\hat{\theta})$, as shown in Table 1.

The results are significant in the sense that all algorithms provided an arm which significantly outperformed the baseline (except "uniform" for which the significance is questionable), but it's difficult to compare these results which are equal within 2 standard deviations and which correspond each to one run of the algorithm. We therefore switch to carefully designed artificial experiments.

4.4 The tuning of MoGo - extended experiments

The purpose of this experimental section is to empirically support multiple claims regarding multi-armed bandit problems optimizing simple regret instead of cumulative regret.

Let us bear in mind that the goal is to get the minimal expected simple regret for a given number of pulls. (Bubeck et al., 2009) has shown that uniformly pulling arms is asymptotically optimal in this case. Nevertheless, it has been suggested in the same paper that in practical applications, using strategies designed for cumulative regret on simple regret prob-

Algorithm	Horizon	Selected arm	Generalization performance
Baseline			51.37 % ± 0.3%
UCB(2)+most played arm	88 000	(0.00,0.09)	52.65% ± 0.3%
UCB(2)+most played arm	150 000	(0.03,0.09)	52.42% ± 0.3%
Uniform+empirically best arm	88 000	(0.09,0.03)	52.00% ± 0.3%

Table 1: Comparison between the different algorithms in the non-blitz settings. In order to have more significant experiments, we will "randomize" and rerun this dataset in section 4.4.

lems often yields much better results, although those strategies are suboptimal asymptotically.

Two arm selection strategies are investigated in this section:

$Uniform$ The arm selected for the n^{th} arm pull is $n\%K$, where K is the number of arms. Thus, all the arms are pulled the same number of times if K is a divisor of N;

$UCB(p)$ The n^{th} arm is selected according to the Upper Confidence Bound formula(Auer et al., 2002), with p being a parameter weighing the exploration term (see algorithm 1).

Algorithm 1 The UCB(p) algorithm. It plays at each round the arm with the highest upper confidence bound. $X_{i,s}$ is the random reward of arm i after s pulls. $T_i(t)$ is the number of times i has been pulled after a total number t of pulls. For more details see(Bubeck et al., 2009; Auer, 2003).

```
Algorithm $UCB(p)$
Input: K arms
Parameter: $p$ accounting for the weight of exploration in the strategy
t=0 // current total number of arms pulled
while true do
  t++
  for $i \in \{1,\dots,K\}$ do
    if $T_i(t-1) == 0$ then
      $B_{i,t} = +\infty$
    else
      $B_{i,t} = \hat{\mu_{i,t-1}} + \sqrt{\frac{\alpha \ln t}{T_i(t-1)}}$
        where $\mu_{i,t-1} = \frac{1}{T_i(t-1)} \sum_{s=1}^{T_i(t-1)} X_{i,s}$
    end if
  end for
  Pull any arm $i \in \arg\max_i B_{i,t}$
end while
```

Note that after having exhausted the allowed number of pulls to choose an arm, there are various rules to decide which arm to keep (these rules are termed recommendation rules):

Empirical best arm (EBA): the arm kept is the one with the best empirical mean;

Most played arm (MPA): the arm is the one which has been played most;

Empirical distribution of plays (EDP): the arm is selected by random draw among the K arms, where each arm has a probability of being chosen proportionnal to the number of times it has been played.

Here are a few remarks regarding these decision rules:

- For the $Uniform$ selection strategy, MPA makes no sense, since the first $N\%K$ arms have been played exactly once more than the remaining ones. Furthermore, EDP degenerates to randomly choosing an arm.
- For UCB-based selection strategies, or other confidence bound strategies, EBA is the most aggressive choice, since an arm that has been played less, but that has a better average, will be preferred. On the other hand, MPA will select arms in which the uncertainty on the mean is low—both are sound, however, and converge to the same result for confidence-bound based selection strategies.

The following experiments investigate these arm selection procedures on three "almost real-world" applications, designed as follows:

- We applied the $Uniform$ strategy during a very long time.
- We recorded the results.
- Then, we could simulate various strategies "offline"; for each simulation, we permuted the random draws, so that there's no systematic bias.

Each problem consists in a few hundreds of arms whose rewards are bernouilli distributions with parameter p close to $1/2$. Experiments are performed by confronting strategies $Uniform$, $UCB(0.1)$, $UCB(1)$ and $UCB(10)$ for each recommendation rule (EBA, MPA and EDP). For each of the 3 experiments, each selection strategy and each decision rule, experiments were performed for various horizons (number of allowed pulls), going from 100 to 819200. Results are presented on Fig. 4.4 and show that (i) UCB performs better than $Uniform$ (ii) the improvement is moderate, in particular when the horizon exceeds the threshold proposed in Eq. 7. A strength of $Uniform$ is that the performance is naturally evaluated on the fly for all arms; therefore both the statistical validation and the visualization are straightforward and for free.

5 DISCUSSION

We have surveyed simple regret algorithms. They are noisy optimization algorithms, and they don't assume any structure on the domain. We compared $Uniform$ (known as optimal for sufficiently large horizon, i.e. sufficiently large time budget) and UCB for automatic parameter tuning. Our results are as follows:

- **MSHT (even the simple Bonferroni correction) is relevant for Automatic Parameter Tuning.** It predicts how many computational power is required for $Uniform$; when the number K of tested sets of parameters depends on a discretization, MSHT can be applied for choosing the grain of the discretization. The $Uniform$ approach combined with MSHT by Bonferroni correction might be the best approach when the computational power is large in front of K, thanks to its statistical guarantees, the easy visualization, the optimality in terms of simple regret. However, non-asymptotically, it is not optimal and the rule below is here for deciding the relevance of $Uniform$ when K and T are known.
- **Choosing between the naive solution ($Uniform$ sampling) and sophisticated algorithms.** The naive $Uniform$ algorithm is provably optimal for large values of the horizon. We propose the following simple rule for choosing if it is worth using something else than the simple uniform sampling:
  - Compute
  $$s = \delta\sqrt{n}/\sigma.$$
  where
    - δ is the amplitude of the expected change in reward;
    - σ is the expected standard deviation;
    - n is the number of experiments you can perform for each arm with your computational power.
  - Test if
  $$\frac{\exp(-s^2)}{s\sqrt{2\pi}} \leq 0.05/K \text{ where } K \text{ is the number of arms.}$$
  - If yes, then uniform sampling is ok. Otherwise, you can try UCB-like algorithms (but, in that case, there's no statistical guarantee), or Bernstein races. At first view, our choice would be Bernstein races for an implementation aimed at automatically tuning and validating several modifications (as in (Hoock and Teytaud, 2010)) as soon as conditions above are not met by the computational power available; if the computational power available is strong enough, $Uniform$ has nice visualization properties.
  - **What if uniform algorithms can't do it ?** If K is not large, nothing can be much better than uniform; at most the required horizon can be divided by $K\log(K)$. What if K is large ? UCB is probably much better when K is large. A drawback is that it does not include any statistical validation, and is not trivially parallel; therefore, classical algorithms derived far from the field of simple regret, like Bernstein races(Bernstein, 1924), might be more relevant. Bernstein races are close to the $Uniform$ algorithm, except that they discard arms as early as possible (Mnih et al., 2008; Hoock and Teytaud, 2010) by performing statistical tests on the fly. A drawback is that Bernstein races do not provide a complete picture of the search space and of the fitness landscape as $Uniform$; also, if no arm can be discarded early, the horizon required for statistical validation is bigger than for $Uniform$ as tests are performed during the run. Yet, Bernstein races might be the most elegant tool for doing better than $Uniform$ as they adapt to various frameworks(Hoock and Teytaud, 2010): when many arms can be discarded easily, they will save up a lot of computational power.
- **Results on our application to MCTS.** For the specific application, the results were significant but moderate; however, it can be pointed out that many handcrafted modifications around Monte-Carlo Tree Search provide such small improvements of a few percents each. Moreover, as shown in (Hoock and Teytaud, 2010), improvements performed automatically by bandits can be applied incrementally, leading to huge improvements once they are cumulated.
- **Comparing recommendation techniques: most played arm is better.** The empirically best arm and the most played arm in UCB are usually the same (this is not the case for various other bandit algorithms), and are much better than the “empirical distribution of play” technique. The most played arm and the empirical distribution of play obviously do not make sense for $Uniform$. Please

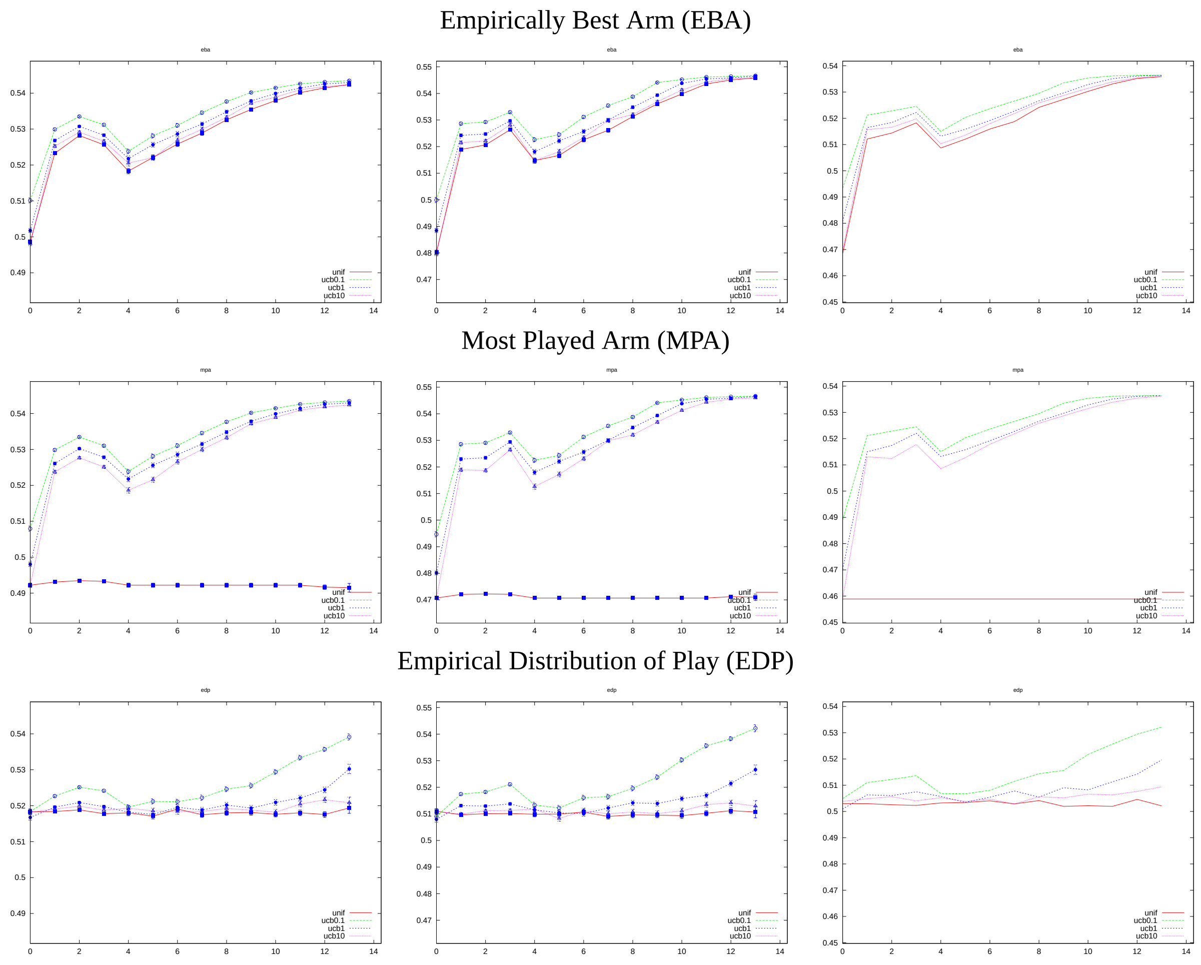

Figure 3: Simple regret strategies compared on the blitz case as black (left), blitz case as white (middle) and standard case (right) for various horizons. The Y-axis is the true mean of the arm chosen by a simple regret strategy being allowed H pulls on arms on the whole. On the X-axis lies $\log(H/100)$. Each point of each curve is the result of an average over more than 1000 experiments; this explains the almost unoticeable 95% error bars. Consequently, even small gaps between curves are statistically significant. The order of the curves is as follows: in all cases, with $\geq$ for "performed better", $UCB(0.1) \geq UCB(1) \geq UCB(10) \geq Uniform$, with almost equality for large horizon. We can see that $EBA \simeq MPA \geq EDP$.

note that it is known in other settings (see (Wang and Gelly, 2007)) that the most played arm is better(Wang and Gelly, 2007). MPA is seemingly a reliable tool in many settings.

A next experimental step is the automatic use of the algorithm for more parameters, or e.g. by extending automatically the neural network used in the Monte-Carlo Tree Search so that it takes into account more inputs: instead of performing one big modification, apply several modifications the one after the other, and tune them sequentially so that all the modifications can be visualized and checked independently. The fact that the small constant 0.1 was better in UCB is consistant with the known fact that tuned version of UCB (with p related to the variance) provides better results; using tuned-UCB might provide further improvements(Audibert et al., 2006).

ACKNOWLEDGEMENTS

This work has been supported by French National Research Agency (ANR) through COSINUS program (project EXPLO-RA No ANR-08-COSI-004), and grant No. ANR-08-COSI-007-12 (OMD project). It benefited from the help of Grid5000 for parallel experiments.

REFERENCES

Audibert, J.-Y., Munos, R., and Szepesvari, C. (2006). Use of variance estimation in the multi-armed bandit problem. In *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*.

Auer, P. (2003). Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3):235–256.

Auger, A. and Teytaud, O. (Accepted). Continuous lunches are free plus the design of optimal optimization algorithms. *Algorithmica*.

Bernstein, S. (1924). On a modification of chebyshev's inequality and of the error formula of laplace. *Original publication: Ann. Sci. Inst. Sav. Ukraine, Sect. Math. 1*, 3(1):38–49.

Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration in multi-armed bandits problems. In *ALT*, pages 23–37.

Chaslot, G., Hoock, J.-B., Teytaud, F., and Teytaud, O. (2009). On the huge benefit of quasi-random mutations for multimodal optimization with application to grid-based tuning of neurocontrollers. In *ESANN*, Bruges Belgium.

Chaslot, G., Saito, J.-T., Bouzy, B., Uiterwijk, J. W. H. M., and van den Herik, H. J. (2006). Monte-Carlo Strategies for Computer Go. In Schobbens, P.-Y., Vanhoof, W., and Schwanen, G., editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91.

Coulom, R. (2006a). Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*.

Coulom, R. (2006b). Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*.

De Mesmay, F., Rimmel, A., Voronenko, Y., and Püschel, M. (2009). Bandit-Based Optimization on Graphs with Application to Library Performance Tuning. In *ICML*, Montréal Canada.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. scand. j. statistic., 6:65-70.

Hoock, J.-B. and Teytaud, O. (2010). Bandit-based genetic programming. In *Accepted in EuroGP 2010*, LLNCS. Springer.

Hsu, J. (1996). Multiple comparisons, theory and methods, chapman & hall/crc.

Kocsis, L. and Szepesvari, C. (2006). Bandit based monte-carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293.

Lee, C.-S., Wang, M.-H., Chaslot, G., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., and Hong, T.-P. (2009). The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*.

Mnih, V., Szepesvári, C., and Audibert, J.-Y. (2008). Empirical Bernstein stopping. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA. ACM.

Nannen, V. and Eiben, A. E. (2007a). Relevance estimation and value calibration of evolutionary algorithm par ameters. In *International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 975–980.

Nannen, V. and Eiben, A. E. (2007b). Variance reduction in meta-eda. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 627–627, New York, NY, USA. ACM.

Pantazis, D., Nichols, T. E., Baillet, S., and Leahy, R. (2005). A comparison of random field theory and permutation methods for the statistical analysis of MEG data. *Neuroimage*, 25:355–368.

Rolet, P., Sebag, M., and Teytaud, O. (2009). Optimal active learning through billiards and upper confidence trees in continous domains. In *Proceedings of the ECML conference*.

Wang, Y. and Gelly, S. (2007). Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In

IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii, pages 175–182.

Wolpert, D. and Macready, W. (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.