Automatic discovery of ranking formulas for playing with multi-armed bandits

Francis Maes, Louis Wehenkel, and Damien Ernst

University of Liège
Dept. of Electrical Engineering and Computer Science
Institut Montefiore, B28, B-4000, Liège - Belgium

Abstract. We propose an approach for discovering in an automatic way formulas for ranking arms while playing with multi-armed bandits.
The approach works by defining a grammar made of basic elements such as for example addition, subtraction, the max operator, the average values of rewards collected by an arm, their standard deviation etc., and by exploiting this grammar to generate and test a large number of formulas. The systematic search for good candidate formulas is carried out by a built-on-purpose optimization algorithm used to navigate inside this large set of candidate formulas towards those that give high performances when using them on some multi-armed bandit problems.
We have applied this approach on a set of bandit problems made of Bernoulli, Gaussian and truncated Gaussian distributions and have identified a few simple ranking formulas that provide interesting results on every problem of this set. In particular, they clearly outperform several reference policies previously introduced in the literature. We argue that these newly found formulas as well as the procedure for generating them may suggest new directions for studying bandit problems.

Keywords: Multi-armed Bandits, Exploration vs. exploitation, Automatic formula discovery

1 Introduction

In the recent years, there has been a revival of interest in multi-armed bandit problems, probably due to the fact that many applications might benefit from progress in this context [5, 3].

A very popular line of research that has emerged for solving these problems focuses on the design of simple policies that compute in an incremental way an index for each arm from the information collected on the arm and then play the arm with the highest ranking index. One of the most well-known such "index-based" policies is UCB1 proposed in [2]. It associates at every play opportunity t the value $\overline{r}_k + \sqrt{(C \ln t)/t_k}$ to every arm k where $\overline{r}_k$ is the average value of the rewards collected so far by playing this arm, t_k the number of times it has been played, and C is typically set to 2.

Simple index-based policies as UCB1 have two main advantages. First they can be implemented on-line with bounded memory and computing resources,

which is important in some applications. Second, due to their simplicity, they are interpretable and hence may lend themselves to a theoretical analysis that may provide insights about performance guarantees.

For designing such simple policies, researchers often use their own intuition to propose a new arm-ranking formula that they test afterwards on some problems to confirm or refute it. We believe that by proceeding like this, there may be interesting ranking formulas that have little chances to be considered. Therefore we propose a systematic approach for discovering such ranking formulas in the context of multi-armed bandit problems.

Our approach seeks to determine automatically such formulas and it works by considering a rich set of candidate formulas in order to identify simple ones that perform well. This is achieved based on the observation that a formula can be grown using a sequence of operations. For example the simple formula $\sqrt{\overline{r}_k}$ could be seen as the result of two operations: select $\overline{r}_k$ then apply $\sqrt{\cdot}$ to $\overline{r}_k$. From there, the search for a well-performing combination of operations of a given complexity is determined by using a specific search procedure where the constants that may appear in a formula are separately optimized using an estimation of distribution algorithm. Using an optimization procedure to identify in a large set of candidates policies well-performing ones has already been tested with success in a previous paper where we were focusing on the identification of linear index formulas over a high-dimensional feature space, with the sole goal of minimizing the regret. Here we focus on the discovery of simple interpretable formulas within a set of non-linear formulas generated by a richer grammar, with the goal of providing further insight into the nature of the multi-armed bandit problem [7].

The rest of this paper is organized as follows. In the next section, we remind the basics about multi-armed bandit problems. In Section 3, we present our procedure for identifying simple formulas for index-based policies performing well on a set of bandit problems. Section 4 reports simulation results and discusses seven such formulas that have been discovered by our procedure. Finally, Section 5 concludes and presents some future research directions.

2 Multi-armed bandit problem and policies

We now describe the (discrete) multi-armed bandit problem and briefly introduce some index-based policies that have already been proposed to solve it.

2.1 The K-armed bandit problem

We denote by $i \in \{1, 2, \ldots, K\}$ the ($K \geq 2$) arms of the bandit problem, by ν_i the reward distribution of arm i, and by μ_i its expected value; b_t is the arm played at round t, and $r_t \sim \nu_{b_t}$ is the obtained reward. $H_t = [b_1, r_1, b_2, r_2, \ldots, b_t, r_t]$ is a vector that gathers the history over the first t plays, and we denote by $\mathcal{H}$ the set of all possible histories of any length t.

A policy $\pi : \mathcal{H} \rightarrow \{1, 2, \ldots, K\}$ is an algorithm that processes at play t the vector H_{t-1} to select the arm $b_t \in \{1, 2, \ldots, K\}$:

$$b_t = \pi(H_{t-1}). \tag{1}$$

The regret of the policy π after a horizon of T plays is defined by:

$$R_T^\pi = T\mu^* - \sum_{t=1}^{T} r_t, \tag{2}$$

where $\mu^* = \max_k \mu_k$ denotes the expected reward of the best arm. The expected regret is therefore given by

$$E[R_T^\pi] = \sum_{k=1}^{K} E[t_k(T)](\mu^* - \mu_k), \tag{3}$$

where $t_k(T)$ is the number of times arm k is used in the first T rounds.

The K-armed bandit problem aims at finding a policy π^* that for a given K minimizes the expected regret (or, in other words, maximizes the expected reward), ideally for any horizon T and any $\{\nu_i\}_{i=1}^K$.

2.2 Index-based bandit policies

Index-based bandit policies are based on a ranking *index* that computes for each arm k a numerical value based on the sub-history of responses H_{t-1}^k of that arm gathered at time t. These policies are sketched in Algorithm 1 and work as follows. During the first K plays, they play sequentially the machines $1, 2, \ldots, K$ to perform initialization. In all subsequent plays, these policies compute for every machine k the score $index(H_{t-1}^k, t) \in \mathbb{R}$ that depends on the observed sub-history H_{t-1}^k of arm k and possibly on t. At each step t, the arm with the largest score is selected (ties are broken at random).

Here are some examples of popular index functions:

$$index^{\text{UCB1}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{\frac{C \ln t}{t_k}} \tag{4}$$

$$index^{\text{UCB1-Bernoulli}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{\frac{\ln t}{t_k} \min\left(1/4, \overline{\sigma}_k + \sqrt{\frac{2 \ln t}{t_k}}\right)} \tag{5}$$

$$index^{\text{UCB1-Normal}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{16 \frac{t_k \overline{\sigma}_k^2}{t_k - 1} \frac{\ln(t-1)}{t_k}} \tag{6}$$

$$index^{\text{UCB-V}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{\frac{2\overline{\sigma}_k^2 \zeta \ln t}{t_k}} + c\frac{3\zeta \ln t}{t_k} \tag{7}$$

where $\overline{r}_k$ and $\overline{\sigma}_k$ are the mean and standard deviation of the rewards so far obtained from arm k and t_k is the number of times it has been played.

Policies UCB1, UCB1-Bernoulli[1] and UCB1-Normal[2] have been proposed by [2]. UCB1 has one parameter $C > 0$ whose typical value is 2. Policy

[1] The original name of this policy is UCB-Tuned. Since this paper mostly deals with policies having parameters, we changed the name to UCB1-Bernoulli to make clear that no parameter tuning has to be done with this policy.

[2] Note that this index-based policy does not strictly fit inside Algorithm 1 as it uses an additional condition to play bandits that where not played since a long time.

Algorithm 1 Generic index-based discrete bandit policy

Given scoring function $index : \mathcal{H} \times \{1, 2, \ldots, K\} \rightarrow \mathbb{R}$,
for $t = 1$ to K **do**
    Play bandit $b_t = t$ ▷ Initialization: play each bandit once
    Observe reward r_t
end for
for $t = K$ to T **do**
    Play bandit $b_t = \text{argmax}_{k \in \{1,2,\ldots,K\}} \, index(H^k_{t-1}, t)$
    Observe reward r_t
end for

UCB-V has been proposed by [1] and has two parameters $\zeta > 0$ and $c > 0$. We refer the reader to [2, 1] for detailed explanations of these parameters.

3 Systematic search for good ranking formulas

The ranking indexes from the literature introduced in the previous section depend on t and on three statistics extracted from the sub-history H^k_{t-1} : $\overline{r}_k$, $\overline{\sigma}_k$ and t_k. This section describes an approach to systematically explore the set of ranking formulas that can be build by combining these four variables, with the aim to discover the best-performing one for a given bandit problem.

We first formally define the (infinite) space of candidate formulas $\mathcal{F}$ in Section 3.1. Given a bandit problem P, a time horizon T, and $\mathcal{F}$, the aim is to solve:

$$f^* = \underset{f \in \mathcal{F}}{\text{argmin}} \, \Delta_{P,T}(f) \tag{8}$$

where $\Delta_{P,T}(f) \in \mathbb{R}$ is a loss function that reflects the regret of the index-based policy defined by formula f, that is computed by averaging the observed regret – on problem P, with time horizon T – over a large number of runs (we use 1000 runs in our experiments). Since we consider formulas that may contain constant parameters, solving Equation 8 is a mixed structure/parameter optimization problem. Section 3.2 describes the procedure to search for formula structures and Section 3.3 focuses on parameter optimization. Note that in order to avoid overfitting, some form of regularization or complexity penalty can be added to the objective function $\Delta(\cdot)$. This is not necessary in our study since we restrict our search algorithm to rather small formulas.

3.1 A grammar for generating index functions

Sets of mathematical formulas may be formalized by using grammars. In fields such as compiler design or genetic programming [6], these formulas are then described by parse trees which are labeled ordered trees. We use this approach for our index formulas, by using the grammar given in Figure 1: an *index* formula F is either a binary expression $B(F, F)$, a unary expression $U(F)$, an atomic variable V or a constant *cst*. The set of binary operators B includes the four

$F ::= B(F,F) \mid U(F) \mid V \mid cst$
$B ::= + \mid - \mid \times \mid \div \mid min \mid max$
$U ::= log \mid sqrt \mid inverse$
$V ::= \overline{r}_k \mid \overline{\sigma}_k \mid t_k \mid t$

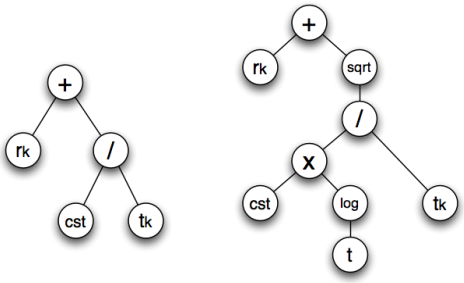

Fig. 1. The grammar used for generating candidate index functions and two example formula parse trees corresponding to $\overline{r}_k + C/t_k$ and $\overline{r}_k + \sqrt{C \log t / t_k}$.

elementary mathematic operations $(+, -, \times, \div)$ and the *min* and *max* operators. The three unary operators U are the logarithm *log*, the square root *sqrt* and the inverse function *inverse*. As mentioned previously, we consider four elementary variables V, which are the empirical reward mean $\overline{r}_k$ and standard deviation $\overline{\sigma}_k$ of arm k, the number t_k of times it has been played so far and the current time step t. For example, two formulas in $\mathcal{F}$ are given in the from of parse trees in the right part of Figure 1.

In the following, the formula *structure* refers to its parse tree with constants labeled as *cst*. Each constant is a *parameter* with values $\in \mathbb{R}$. We denote formula structures by f, formula parameters by $\theta \in \mathbb{R}^C$ where C is the number of constants and parameterized formulas by $f_\theta \in \mathcal{F}$.

3.2 Generation of candidate formula structures

We now describe the iterative depth-limited approach that we adopt to generate candidate formula structures f. We rely on a search space $\mathcal{S}$ the elements s of which are ordered lists of "current" formulas: $s = (f^1, \ldots, f^S)$. The elementary search step consists in expanding a state by appending to it a new formula f^{S+1} constructed from its current formulas $f^1, \ldots, f^S$, by using one of the following operations:

- Create a constant: $f^{S+1} = cst$
- Create a variable: $f^{S+1} = \overline{r}_k, \overline{\sigma}_k, t_k$ or t
- Create an unary operation: $f^{S+1} = U(f^i)$ where $i \in [1, S]$
- Create a binary operation: $f^{S+1} = B(f^i, f^j)$ where $i \in [1, S], j \in [i, S]$

In this search space, starting from the empty initial state $s_0 = ()$, the formula $\overline{r}_k + cst/t_k$ can for example be built with the following operation sequence: $(f^1 = \overline{r}_k) \rightarrow (f^2 = cst) \rightarrow (f^3 = t_k) \rightarrow (f^4 = f^2/f^3) \rightarrow (f^5 = f^1 + f^4)$.

We use the function $\Delta^{structure}(s) \in \mathbb{R}$ to evaluate the loss associated to a state $s \in \mathcal{S}$. This loss only depends on the last formula f^S of state s. If f^S is a parameter-free formula (i.e., without constants), $\Delta^{structure}(s)$ is equal to the estimated regret $\Delta_{P,H}(f^S)$. Otherwise, $\Delta^{structure}(s)$ is equal to the best

Algorithm 2 Formula structure search algorithm

Given loss function $\Delta^{structure} : \mathcal{S} \rightarrow \mathbb{R}$,
Given max depth $d_{max} \geq 1$,

$s_0 \leftarrow ()$
for $\ell \in [0, \infty[$ **do**
  $s^*_\ell \leftarrow$ DepthLimitedSearch $(s_\ell, \Delta^{structure}, d_{max})$
  if $\ell \geq 1$ and $\Delta^{structure}(s^*_\ell) \geq \Delta^{structure}(s^*_{\ell-1})$ **then**
    return $s^*_{\ell-1}$
  end if
  $s_{\ell+1} \leftarrow$ firstSuccessorState(s_ℓ, s^*_ℓ)
  $t \leftarrow \ell + 1$
end for

achieved regret $\Delta_{P,H}(f^S_{\theta^*})$ during the optimization of parameters $\theta \in \mathbb{R}^C$ with the procedure described in the next section.

Our iterative depth-limited search approach is specified by Algorithm 2 and works in the following way. We start with the initial state $s_0 = ()$. Each search iteration uses a depth-limited search to select the next formula that will be added to the current state. At iteration ℓ, depth-limited search works by exploring exhaustively the search space, starting from s_ℓ with a maximum exploration depth of d_{max}. During this exploration, the loss function $\Delta^{structure}$ is computed for every visited state, and the state that minimizes this function is returned[3]. s^*_ℓ is thus the best state that can be reached with the depth-limited search procedure when starting from s_ℓ. Once $s^*_\ell = (f_1, \ldots, f_\ell, f^*_{\ell+1}, \ldots, f^*_{\ell+d_{max}})$ is identified, the formula $f^*_{\ell+1}$ is added to the current state s_ℓ to form the new state $s_{\ell+1} = (f_1, \ldots, f_\ell, f^*_{\ell+1})$, we take one step ahead the best solution found so far. The search procedure stops as soon as an iteration ℓ does not lead to an improved value of $\Delta^{structure}(s^*_\ell)$ compared to its previous iteration.

Our search procedure may evaluate the same formula structures multiple times. To avoid this, we implement $\Delta^{structure}(\cdot)$ with a cache that stores all the values $\Delta^{structure}(\cdot)$ that have already been computed since the beginning of search. To further reduce the number of calls to $\Delta^{structure}(\cdot)$, this cache relies on a normal version of the formula, which is computed by iteratively applying the following rules until the formula remains unchanged:

- Commutativity: fix the order of the operands of commutative operators: $+, \times, max$ and min
- Multiplication $\forall x \in \mathcal{F}, 1 \times x \rightarrow x$
- Division: $\forall x \in \mathcal{F}, x \div x \rightarrow 1$, $x \div 1 \rightarrow x$,
- Subtraction: $\forall x \in \mathcal{F}, x - x \rightarrow 0$
- Inverse function: $\forall x \in \mathcal{F}, y \in \mathcal{F}, x \times inverse(y) \rightarrow x \div y, x \div inverse(y) \rightarrow x \times y, 1 \div x \rightarrow inverse(x), inverse(inverse(x)) \rightarrow x$

Note that our search procedure in rather naive and could be improved in several ways. However, it proves to be efficient enough to already discover several interesting new bandit policies, as shown in next section. A simple way to

[3] Ties are broken by giving the preference to smaller formulas.

improve the efficiency of our approach would be to extend the set of rewriting rules with advanced mathematical rules such as factorization and division simplification.

3.3 Optimization of constants

Each time a formula structure f with a number $C \geq 1$ of parameters is evaluated, we run a derivative-free global optimization algorithm to determine the best values $\theta^* \in \mathbb{R}^C$ for these parameters. In this work, we rely on a powerful, yet simple, class of metaheuristics known as *Estimation of Distribution Algorithms* (EDA) [4]. EDAs rely on a probabilistic model to describe promising regions of the search space and to sample good candidate solutions. This is performed by repeating iterations that first *sample* a population of N candidate values of $\theta \in \mathbb{R}^C$ using the *current* probabilistic model and then *fit* a *new* probabilistic model given the $b < N$ best candidates. Many different kinds of probabilistic models may be used inside an EDA. The most basic form of EDAs uses one marginal distribution per variable to optimize and is known as the *univariate marginal distribution algorithm* [8]. We have adopted this approach that, while simple, proved to be effective to solve our general parameter optimization problem.

Our EDA algorithm thus proceeds as follows. There is one Normal distribution per parameter to optimize $c \in \{1, 2, \ldots, C\}$. At the first iteration, these distributions are initialized as standard Normal distributions. At each subsequent iteration, N candidates $\theta \in \mathbb{R}^C$ are sampled using the current distributions and evaluated; then the p distributions are re-estimated using the $b < N$ best candidates of the current iteration. Optimization stops when the best regret $\Delta_{P,H}(f_\theta)$ has not been outperformed in the last i_{stop} iterations, and then returns the parameters θ^* for which we observed the minimum regret $\Delta_{P,H}(f_{\theta^*})$.

4 Numerical experiments

We now apply our formula discovery approach on a set of twelve discrete bandit problems with different kinds of distributions and different numbers of arms. We will focus on the analysis of seven interesting formulas that were discovered.

4.1 Experimental setup

We considered the following 12 discrete bandit problems:

- **2-Bernoulli arms**: Problem-1, Problem-2 and Problem-3 are two-arms bandit problems with Bernoulli distributions whose expectations (μ_1, μ_2) are respectively $(0.9, 0.6)$, $(0.9, 0.8)$ and $(0.55, 0.45)$.
- **10-Bernoulli arms**: Problem-4 – Problem-6 correspond to the three first problems, in which the arm with lowest reward expectation is duplicated 9 times. For example, the reward expectations of Problem-4 are $(0.9, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6)$.

Problem	$d_{max} = 1$	$d_{max} = 2$	$d_{max} = 3$	
			2 steps	3 steps
PROBLEM-1	$\overline{r}_k$	$\mathbf{t_k}(\overline{\sigma}_\mathbf{k} - \mathbf{C_1})(\overline{\sigma}_\mathbf{k} - \mathbf{C_2})$	$max(\overline{r}_k, C)$	-
PROBLEM-2	$\overline{r}_k$	$(\overline{\sigma}_k - \overline{r}_k - 1)\overline{\sigma}_k$	$max(\overline{r}_k, C)$	$\mathbf{t_k}(\overline{\mathbf{r}}_\mathbf{k} - \mathbf{C})$
PROBLEM-3	$\overline{r}_k$	$\sqrt{2\overline{r}_k + \log \overline{r}_k} - \log \overline{r}_k$	$\overline{r}_k(\overline{r}_k - C)$	$\mathbf{t_k}(\overline{\mathbf{r}}_\mathbf{k} - \mathbf{C})$
PROBLEM-4	$\overline{r}_k$	$\log(\overline{r}_k - \overline{\sigma}_k)$	$\mathbf{max}(\overline{\mathbf{r}}_\mathbf{k}, \mathbf{C})$	-
PROBLEM-5	$\overline{r}_k$	$\frac{1}{\overline{\sigma}_k}$	$\overline{\sigma}_\mathbf{k}(\overline{\sigma}_\mathbf{k} - \mathbf{C})$	-
PROBLEM-6	$\overline{r}_k$	$\overline{r}_k - \overline{\sigma}_k$	$\overline{\mathbf{r}}_\mathbf{k}(\overline{\mathbf{r}}_\mathbf{k} - \mathbf{C})$	-
PROBLEM-7	$\overline{r}_k$	$t_k(\overline{r}_k^2 - \overline{\sigma}_k)$	$\mathbf{max}(\overline{\mathbf{r}}_\mathbf{k}, \mathbf{C})$	-
PROBLEM-8	$\overline{r}_k$	$2\overline{r}_k - \overline{\sigma}_k/\overline{r}_k$	$max(\overline{r}_k, C)$	$\mathbf{max}(\overline{\mathbf{r}}_\mathbf{k}, \frac{\mathbf{C}}{\overline{\mathbf{r}}_\mathbf{k}})$
PROBLEM-9	$\overline{r}_k$	$t_k(\overline{r}_k - \overline{\sigma}_k) + \overline{\sigma}_k$	$max(\overline{r}_k, C)$	$\mathbf{t_k}(\overline{\mathbf{r}}_\mathbf{k} - \mathbf{C})$
PROBLEM-10	$\overline{r}_k$	$\overline{r}_k^2 - \overline{r}_k$	$\mathbf{max}(\overline{\mathbf{r}}_\mathbf{k}, \mathbf{C})$	-
PROBLEM-11	$\overline{r}_k$	$\overline{r}_k\overline{\sigma}_k + \log \overline{r}_k$	$\mathbf{max}(\overline{\mathbf{r}}_\mathbf{k}, \mathbf{C})$	$\mathbf{max}(\overline{\mathbf{r}}_\mathbf{k}, \mathbf{C})$
PROBLEM-12	$\overline{r}_k$	$\overline{r}_k(\sqrt{\overline{r}_k} - 1)$	$\overline{r}_k + 1/t_k$	$\mathbf{t_k}(\overline{\mathbf{r}}_\mathbf{k} - \mathbf{C})$

Table 1. Discovered index formulas for horizon $T = 100$ on twelve bandit problems. For each problem, the best performing index formula is shown in bold. - denotes results that we could not compute within 10 days.

- **2-Gaussian arms**: PROBLEM-7 – PROBLEM-9 are two-arms bandit problems with Normal distributions. As previously the expectations μ_1 and μ_2 of these distributions are respectively $(0.9, 0.6)$, $(0.9, 0.8)$ and $(0.55, 0.45)$. We use the same standard deviation $\sigma = 0.5$ for all these Normal distributions.
- **2-Truncated Gaussian arms**: PROBLEM-10 – PROBLEM-12 correspond to the three previous problems with Gaussian distributions truncated to the interval $[0, 1]$ (that means that if you draw a reward which is outside of this interval, you throw it away and draw a new one).

We have applied our index formula discovery algorithm to PROBLEM-1 – PROBLEM-12 to optimize the regret $\Delta_{P,T}$ with a time horizon $T = 100$ and parameter $d_{max} \in \{1, 2, 3\}$. The population size of the EDA optimizer is $N = 12C$ and $b = 3C$ is used as the number of best solutions used to fit new distributions, where C is the number of parameters. EDA optimization iterations are stopped after $i_{stop} = 5$ iterations without improvement.

In this setting, with our C++ based implementation and a 1.9 Ghz processor, performing the search with $d_{max} = 1$ takes $\simeq 10$ seconds, with $d_{max} = 2$ it takes a few minutes, and with $d_{max} = 3$ it takes some weeks of CPU time.

4.2 Discovered policies

Table 1 reports the best formulas that were discovered for each problem and each value d_{max}. Since we could not always complete the search with d_{max}, we only report the best formulas found after 2 search iterations and 3 search iterations.

When $d_{max} = 1$, Algorithm 2 aims at performing purely greedy search. As could be expected the best formula that can be found in a greedy way, when starting from an empty formula, is always $\text{GREEDY}(H_{t-1}^k, t) = \overline{r}_k$. To discover more interesting formulas, the search algorithm has to perform a deeper exploration. With $d_{max} = 2$, the algorithm discovers medium-performing index

functions, without any recurrent pattern between the found solutions. The results with $d_{max} = 3$ are more interesting. In this last setting, the automatic discovery process most of the time converges to one of the following formulas:

$$\textsc{Formula}\text{-}1(H^k_{t-1}, t) = max(\overline{r}_k, C) \qquad \textsc{Formula}\text{-}2(H^k_{t-1}, t) = t_k(\overline{r}_k - C)$$
$$\textsc{Formula}\text{-}3(H^k_{t-1}, t) = \overline{r}_k(\overline{r}_k - C) \qquad \textsc{Formula}\text{-}4(H^k_{t-1}, t) = \overline{\sigma}_k(\overline{\sigma}_k - C)$$

In addition to these four formulas, we also consider below formulas that we obtained by manually modifying one of the discovered formulas:

$$\textsc{Formula}\text{-}5(H^k_{t-1}, t) = \overline{r}_k + \frac{C}{t_k} \qquad \textsc{Formula}\text{-}6(H^k_{t-1}, t) = t_k(\overline{r}_k - C\overline{\sigma}_k)$$
$$\textsc{Formula}\text{-}7(H^k_{t-1}, t) = t_k(\overline{r}^2_k - C\overline{\sigma}_k)$$

Formula-5 is the generalization of $\overline{r}_k + 1/t_k$ found on Problem-12. Formula-6 and Formula-7 are respectively derived from $d_{max} = 2$ solution to Problem-9 and Problem-7.

4.3 Evaluation of the discovered ranking formulas

Policies used for comparison We have compared our discovered policies against various policies that have been proposed in the literature. These are the ϵ_n-Greedy policy [9] as described in [2], the policies introduced by [2]: UCB1, UCB1-Bernoulli, UCB1-Normal and UCB2, and the policy UCB-V proposed by [1]. UCB1-Bernoulli and UCB1-Normal are parameter-free policies respectively designed for bandit problems with Bernoulli distributions and for problems with Normal distributions.

Results with optimized constants. Table 2 reports the regrets achieved by our discovered policies when their constants are optimized as well as the regrets of the set of policies used for comparison. The parameters of the latter policies have also been optimized for generating these results, except for UCB2 which is used with a value for its parameter α equal to 0.001, as suggested in [2]. As we can see, five out of the seven formulas discovered outperform all the policies used for comparison. Formula-2 seems to be the best since it outperforms all the other policies on 9 among the 16 test problems. Another formula that behaves very well on all the problems is Formula-7. Though Formula-3 and Formula-4 may work well on very specific problems with the horizon $T = 100$ used for search, these formulas gives poor regret values on all problems with horizon $T = 1000$ and are thus no further investigated.

Influence of the parameters. For a parameter-dependant formula to be good, it is important to be able to identify default values for its parameters that will lead to good results on the bandit problem to be played, given that no or very little a priori knowledge (e.g., only the type of distribution and the number of arms is known) on this problem is available. Figure 2 illustrates the performances of the formulas that appeared in Table 2 to be the most promising, with respect to the evolution of the value of their parameter. We see that Formula-2 always

Policy	2-Bernoulli			10-Bernoulli			2-Gaussian			2-Tr. Gaussian		
	1	2	3	4	5	6	7	8	9	10	11	12
Baseline policies												
UCB1	4.3	7.5	9.8	25.0	34.7	49.2	6.2	10.9	10.9	17.6	23.3	22.8
UCB1-Bernoulli	4.4	9.1	9.8	39.1	57.0	60.9	6.2	10.9	10.9	34.9	31.2	30.8
UCB1-Normal	47.7	35.4	30.6	248.8	90.6	85.6	36.0	30.7	30.7	73.3	40.5	39.2
UCB2(0.001)	6.9	10.1	11.4	61.6	67.0	67.8	7.4	11.4	11.4	39.3	31.5	30.7
UCB-V	4.6	5.8	9.5	26.3	34.1	50.8	5.3	10.0	10.0	14.4	21.4	22.0
ϵ_n-Greedy	5.9	7.5	12.8	32.9	37.5	54.7	8.2	11.2	11.2	20.0	23.6	21.5
Discovered policies												
Greedy	31.9	23.1	36.6	69.4	46.1	71.6	57.2	38.1	38.1	94.9	44.8	41.2
Formula-1	**1.6**	**3.3**	**7.2**	**14.8**	**26.7**	**43.7**	**2.1**	**8.0**	**8.0**	**7.5**	**18.2**	**19.3**
Formula-2	**1.1**	**1.9**	**5.7**	**10.2**	**21.0**	**38.3**	**1.9**	**4.9**	**4.9**	**8.2**	**14.4**	**15.1**
Formula-3	4.8	11.2	16.6	28.6	46.1	59.4	9.6	17.0	17.0	27.5	27.3	26.9
Formula-4	14.6	12.0	47.3	60.4	46.1	88.3	139.8	46.6	46.6	91.8	42.4	45.9
Formula-5	**2.2**	**3.8**	**7.0**	**18.7**	**28.6**	**45.1**	**3.6**	**7.5**	**7.5**	**10.1**	**17.4**	**18.1**
Formula-6	**1.1**	**2.0**	**5.3**	**10.6**	**20.8**	**38.1**	**3.3**	11.1	**7.8**	**10.5**	**20.7**	**21.4**
Formula-7	**1.2**	**2.0**	**5.6**	**10.7**	**19.6**	**39.0**	**2.4**	**7.1**	**5.6**	**7.8**	**17.7**	**17.7**

Table 2. Regret of all policies on the 12 discrete bandit problems with horizon $T = 1000$. The scores in bold indicate learned or discovered policies that outperform all baseline policies.

leads to a regret which very rapidly degrades when the parameter moves away from its optimal value. However, its optimal parameter value is always located in the interval [*expected reward of the second best arm, expected reward of the optimal arm*]. Actually, we could have anticipated – at least to some extent – this behaviour by looking carefully at this formula. Indeed, we see that if the value of its parameter is in this interval, the policy will be more and more likely to play the best arm as the number of plays goes on, while otherwise, it is not necessarily the case.

For Formula-7, which was the second best formula when optimized, the regret degrades less rapidly when moving away from the optimal parameter value. However, it is difficult to explain here the location of its optimum. For Formula-5, it is quite easy to find a value for its parameter that works well on each test problem. For example, with its parameter chosen equal to 3.5, performances which are always better than those obtained with the *optimized* version of UCB1 are observed. It is worth noticing that this Formula-5 is actually very close to UCB1. Indeed, it also associates to an arm an index value which is the sum of $\overline{r}_k$ plus a term that could also been seen as an upper confidence bias. Contrary to UCB1, this term decreases in t_k rather than in $\sqrt{t_k}$ but, more importantly, it does not increase explicitly with t.

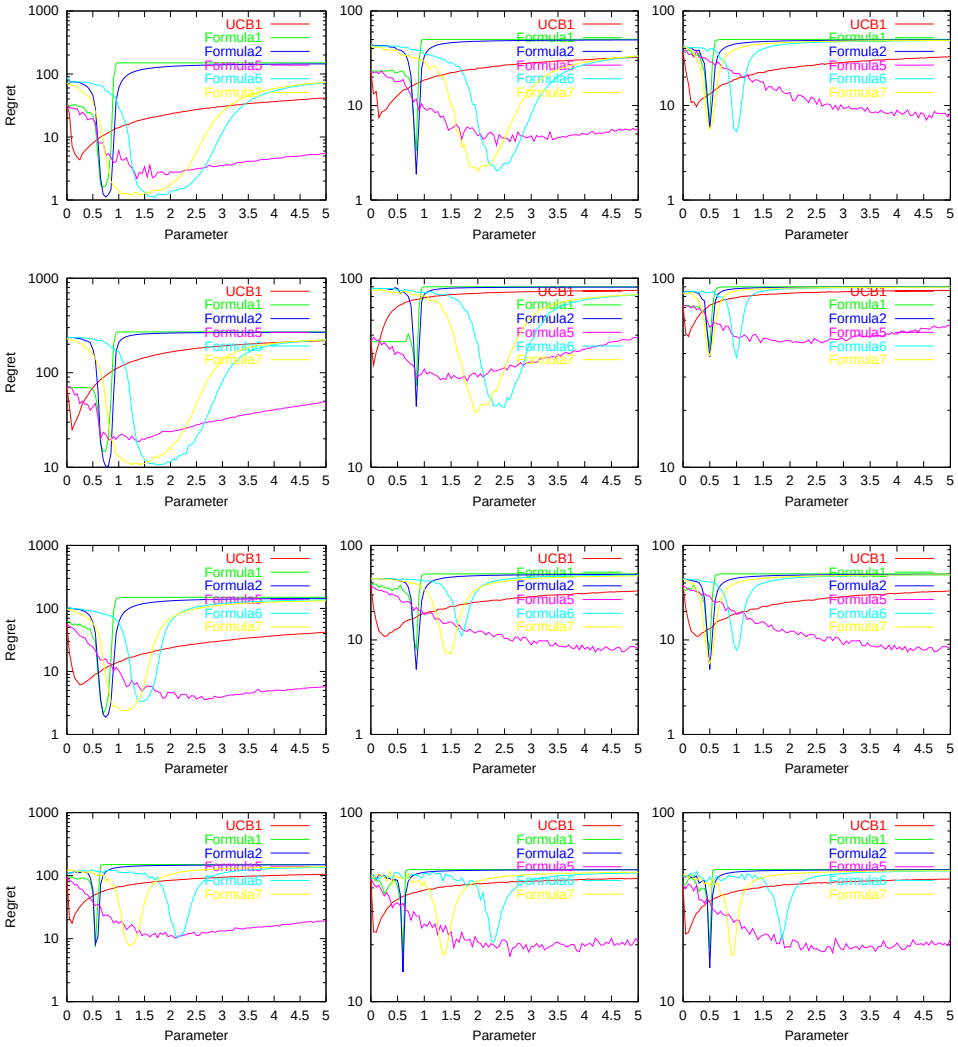

Fig. 2. First row: 2-Bernoulli problems. Second row: 10-Bernoulli problems. Third row: 2-Gaussian problems. Last row: 2-Clamped Gaussian problems.

5 Conclusions

We have proposed in this paper an approach for automatically identifying simple arm ranking formulas for playing with multi-armed bandits. This approach has lead us to discover several well performing formulas. Interestingly, some of them share similarities with those previously proposed in the literature. However, several formulas found to perform very well are behaving in a completely different way. For example, they yield indexes that are not guaranteed to converge (in probability) to a monotonic transformation of the expected rewards of the arms.

These newly found formulas are quite intriguing and we believe that it would be worth investigating their analytical properties. In particular, it would be interesting to better understand on which class of bandit problems they may be advantageous to use.

It is likely that one key issue that should be addressed in this analysis is the influence of the parameter values inside these formulas on the performance of the policy. Indeed, those that were found to be the best when their parameter was optimized were also found to be those for which we could not find a parameter value leading to a policy working well on every bandit problem of our test set.

Investigating how our approach for discovering policies could be adapted so as to discover policies that could work well on a large class of problems for some default values of their parameters is also an interesting research direction. This could be done for example by scoring the policies in a different way inside our search algorithm, for example by using as score their average performances on a large enough set of problems rather than scoring them on a single instance as we did in this paper.

Acknowledgements

Damien Ernst acknowledges the financial support of the Belgian National Fund of Scientic Research (FNRS) of which he is a Research Associate. This paper presents research results of the European excellence network PASCAL2 and of the Belgian Network BIOMAGNET, funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office.

References

1. J.Y. Audibert, R. Munos, and C. Szepesvari. Tuning bandit algorithms in stochastic environments. *Algorithmic Learning Theory*, pages 150–165, 2007.
2. P. Auer, P. Fischer, and N. Cesa-Bianchi. Finite-time analysis of the multi-armed bandit problem. *Machine Learning*, 47:235–256, 2002.
3. D. Chakrabarti, R. Kumar, F. Radlinski, and E. Upfal. Mortal multi-armed bandits. In *Neural Information Processing Systems*, pages 273–280, 2008.
4. C. Gonzalez, J.A. Lozano, and P. Larrañaga. *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.
5. C.S. Lee, M.H. Wang, G. Chaslot, J.B. Hoock, A. Rimmel, O. Teytaud, S.R. Tsai, S.C. Hsu, and T.P. Hong. The computational intelligence of MoGo Revealed in Taiwan's computer Go tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, 2009.
6. M. Looks. *Competent Program Evolution*. PhD thesis, Washington University in St. Louis, 2006.
7. F. Maes, L. Wehenkel, and D. Ernst. Learning to play k-armed bandit problems. *Submitted*, 2011.
8. M. Pelikan and H. Mühlenbein. Marginal distributions in evolutionary algorithms. In *Proceedings of the International Conference on Genetic Algorithms Mendel*, 1998.
9. R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.