Differential Evolution Algorithm Applied to Non-Stationary Bandit Problem

David L. St-Pierre
University of Liège
B-4000 Liège, Belgium
TAO, Inria, Univ. Paris-Sud, Paris, UMR CNRS 8623, FR
91190 Gif-sur-Yvette, France
Email: davidls@lri.fr

Jialin Liu
TAO, Inria, Univ. Paris-Sud, Paris, UMR CNRS 8623, FR
91190 Gif-sur-Yvette, France
Email: liu@lri.fr

Abstract—**In this paper we compare Differential Evolution (DE), an evolutionary algorithm, to classical bandit algorithms over the non-stationary bandit problem. First we define a testcase where the variation of the distributions depends on the number of times an option is evaluated rather than over time. This definition allows the possibility to apply these algorithms over a wide range of problems such as black-box portfolio selection. Second we present our own variant of discounted Upper Confidence Bound (UCB) algorithm that outperforms the current state-of-the-art algorithms for the non-stationary bandit problem. Third, we introduce a variant of DE and show that, on a selection over a portfolio of solvers for the Cart-Pole problem, our version of DE outperforms the current best UCB algorithms.**

I. Introduction

We are interested in a situation where an agent faces K possible options of unknown distribution(s) and is given T sequential evaluations, where T is not necessarily known. The agent is then asked to output $\tilde{\pi}$, a recommendation that corresponds to a probability distribution over K options, according to a performance criterion. In our case the performance criterion is the Simple Regret (SR) and we seek to maximize a reward.

This class of problems is well formalized through the bandit framework [**?**] and more specifically our setting is similar to [**?**]. Typically this framework tackles effectively the tradeoff between exploration and exploitation.

During the evaluation process $t \in T$, the agent selects an option $k \in K$ according to its selection policy $\pi(\cdot)$. A selection policy $\pi(\cdot) \in K$ is an algorithm that selects an option based on the information at hand. A representative example of a selection policy is UCB[**?**]. At the t^{th} evaluation, the option k is selected and a reward $r_{k,t}$ is computed. A detailed description of the selection policies and the distribution updates used in this paper are provided in Section **??**.

Such a process is repeated until the allocated number of evaluations T has been executed. Afterward, following the recommendation $\tilde{\pi}$, an option $\tilde{k}$ is chosen. The pseudo code for a generic bandit algorithm up to the recommendation of $\tilde{k}$ is provided in Algorithm **??** and the recommendation of $\tilde{k}$ used in this paper is provided in Section **??**. In this paper we focus on non-stationary distributions. A non-stationary bandit problem implies that the distribution of each option $k \in K$ changes. In such cases, most of the literature is focused over a variation of distributions based upon the number of evaluations t[**?**], [**?**], [**?**]. In this paper, we focus on a variation based upon the number of times we select an option. The reward distribution for an option improves as we select it more often. Such framework possesses a wide range of applications. The selection of an object from a portfolio is a good example and covers several testcases that justify such a definition.

Algorithm 1 Generic Bandit Algorithm. The problem is described through the "get stochastic reward", a stochastic method and the option sets. The "return" method is formally called the recommendation policy. The selection policy is also commonly termed exploration policy.

```
Require: $T > 0$: Computational budget
Require: $K$: Set of options
Require: $\pi$: Selection policy
  for $t = 1$ to $T$ do
      Select $k \in K$ based upon $\pi(\cdot)$
      Get stochastic reward $r_{k,t}$
      Update the information of $k$ with $r_{k,t}$
  end for
  Return $\tilde{\pi}$: Probability distribution over the set K
```

In this paper we are interested in comparing an evolutionary approach, namely Differential Evolution (DE)[**?**], to the classical bandit approach. DE is arguably one of the main evolutionary algorithms. It does not require heavy computations, performs well on many optimization testbeds [**?**], [**?**], [**?**] and possesses many variants including self-adaptive parameters [**?**], [**?**], [**?**], [**?**]. Our contribution is a variation of DE that outperforms UCB-Discounted[**?**] and UCB-Discounted2 (see Section **??**), both variants of UCB[**?**] for the non-stationary testcase.

Section **??** formulates the problem and introduces classic selection policies. Section **??** presents the DE algorithm and our variant. Section **??** shows the results and Section **??** concludes.

II. Problem Statement

In this section we formalize the bandit problem and introduce its related selection policy. Section **??** defines the non-stationary problem and presents two variants of UCB, one of the most popular selection policy for stationary bandit, UCB-Discounted, termed π_{UCBdt}, and our modified UCB-Discounted, termed π_{UCBdn}. These variants of UCB are specific for the non-stationary bandit problem. The first one is designed for problems where the reward distribution changes over time. The second one is a modification that we propose and where the distribution changes over the number of times an option is selected.

A. Non-Stationary Bandit Problem

In the non-stationary case under study, the reward distribution of the k^{th} option is given by $\mu_{k,n_{k,t}}$ and represents the expectation of rewards $r_{k,t}$, where $n_{k,t}$ is the number of times an option k has been selected after the t^{th} evaluation. $r_{k,t}$ is the reward of option k obtained at the t^{th} evaluation and given by $\sim Bernoulli(1, \mu_{k,n_{k,t}})$, with 1 being the number of Bernoulli trials.

At each time $t \in T$ the agent chooses an option $k \in K$ following a policy π and obtains a reward $r_{k,t}$. The reward is modeled from distributions unknown to the user. As such, the best option after t evaluations is $k^*_t = \operatorname{argmax}_{k \in K} \mu_{k,t}$.

The recommendation policy $\tilde{\pi}$ usually boils down to selecting the most played option

$$\tilde{k}_t = \operatorname*{argmax}_{k \in K} n_{k,t},$$

yet there exist several other recommendation policies[**?**].

We use Simple Regret as optimization criterion. Simple Regret is the difference between the average reward of the best option and the average reward obtained by the recommendation. As we focus on the non-stationary bandit problem, the Simple Regret is thus computed by $\mu_{k^*_T,T} - \mu_{\tilde{k}_t,n_{k,t}}$ for $t \in T$.

1) UCB-Discounted: UCBdt[**?**] is an attempt to adapt UCB to the non-stationary bandit problem. The idea is to partially forget past information. As such, the estimated value computed from rewards is changed from:

$$\bar{r}_{k,t} = \frac{\sum_{i=1}^{t} r_{k_i} : k_i = k}{n_{k,t}}, \tag{1}$$

to:

$$\bar{r}_{k,t} = \frac{\sum_{i=1}^{t} (\gamma^{t-i} \cdot r_{k_i}) : k_i = k}{\sum_{i=1}^{t} \gamma^{t-i}}$$

where $\gamma \in (0,1]$ and r_{k_t} is the reward of the selected option k_t obtained at the t^{th} evaluation. Note that if $\gamma = 1$ it becomes a UCB as defined for stationary bandit problems. The π_{UCBdt} is given by:

$$\operatorname*{argmax}_{k \in K} \left(\bar{r}_{k,t} + \sqrt{C \frac{\ln t(\gamma)}{n_k(\gamma)}} \right),$$

where $C > 0$, $n_k(\gamma) = \sum_{i=1}^{t} \gamma^{t-i} : k_i = k$ and $t(\gamma) = \sum_{k \in K} n_k(\gamma)$. Other padding functions are considered in [**?**], [**?**]. The main issue in this definition is that the discount depends on t rather than $n_{k,t}$.

2) UCB-Discounted2: We propose a second version of discounted UCB named UCBdn. UCBdn is similar to UCBdt but the discount depends on $n_{k,t}$, the number of time an option is selected. The idea is still to partially forget past information. The update rule becomes:

$$\bar{r}_{k,t} = \frac{\sum_{i=1}^{n_{k,t}} (\gamma^{n_{k,t}-i} \cdot r_{k_i})}{\sum_{i=1}^{n_{k,t}} \gamma^{n_{k,t}-i}}$$

where $\gamma \in (0,1]$. π_{UCBdn} is given by:

$$\operatorname*{argmax}_{k \in K} \left(\bar{r}_{k,t} + \sqrt{C \frac{\ln t(\gamma)}{n_k(\gamma)}} \right),$$

where $C > 0$, $n_k(\gamma) = \sum_{i=1}^{n_{n,t}} \gamma^{n_{k,t}-i}$ and $t(\gamma) = \sum_{k \in K} n_k(\gamma)$.

III. Differential Evolution Algorithm

To represent a bandit problem into a framework that can be optimized through an evolutionary approach is far from trivial and is the subject of Section **??**. Section **??** shows the protocol we use for the evaluation of the fitness and Section **??** presents a small variation proposed to greatly improve the performance of DE for bandit problem.

Let $f : \mathbb{R}^K \to \mathbb{R}$ be the fitness function to be maximized where K is the dimension of the problem. The DE algorithm takes as input a population $\mathcal{X}$ of individuals where each individual $x \in \mathbb{R}^K$ are candidate solutions. The goal can be viewed as finding a solution s such that $\forall\ x \in \mathbb{R}^K\ f(s) \geq f(x)$. Indeed this is for a maximization problem and the solution found is a global maximum. For a minimization, we can easily consider a fitness $h : -f$ instead. The variable t is increased every time a function evaluation is performed. The parameters $F \in [0,2]$, $Cr \in [0,1]$ and $|\mathcal{X}|$, the cardinality of $\mathcal{X}$, have a large impact on the performance of the optimization. Thus, the tuning process is an important part of the algorithm. Algorithm **??** presents our version of the algorithm.

A. Representation

In our DE algorithm, each individual $x \in \mathcal{X}$ represents a probability distribution over each option $k \in K$. Obviously, $\sum_{k \in K} x_k = 1$. These individuals are initialized with random probabilities as shown in Algorithm **??**. Moreover, after each $t \in T$ evaluations we keep in memory $\bar{r}$, a vector of $[0,1]^K$, representing the different $\bar{r}_{k,t}$ as defined in (**??**). In this paper, we add $\bar{r}$ as an individual in $\mathcal{X}$ and is updated at the end of each generation. In the following Section we further describe the evaluation of the fitness.

B. Evaluation of the fitness

When evaluating an individual x, we (i) optionally truncate (see Section **??**) and renormalize, (ii) randomly draw an option $k \in K$ (k is selected with probability x_k) and (iii) evaluate the selected option by getting a reward $r_{k,t}$ through t' Bernoulli trial(s), where $t' > 0$ is the number of re-evaluations.

For the evaluation of the fitness $f(x)$, we decide to compute it following

$$\sum_{k \in K} (x_k \cdot \bar{r}_{k,t}).$$

Note that x_k and $\bar{r}_{k,t}$ are two values accessible without any call to the problem at hand, there is no Bernoulli trial executed during the fitness evaluation. As such, it does not increment the evaluation budget $t \in T$.

C. Truncation

In order to make DE more competitive, we propose a modification similar in its essence to Bernstein pruning[**?**]. Since the recommendation $\tilde{\pi}$ consists in choosing only one option $\tilde{k}$ and we have access to the current approximation of $\mu_{k,n_{k,t}}$ through $\bar{r}_{k,t}$, we can truncate the options that are not likely to be recommended. This ensure that DE is solely focusing on the best options. The rule we used in the algorithm is given by:

$$x_k = \begin{cases} x_k & \text{if } \bar{r}_{k,t} > c_1 \cdot \max\limits_{k \in K} \bar{r}_{k,t} \text{ and } n_k > c_2 \cdot \max\limits_{k \in K} n_{k,t} \\ 0 & \text{otherwise,} \end{cases}$$

where $c_1,\ c_2 \in (0, 1]$. The truncation process is executed during the normalization process when the current number of evaluations is bigger than a given $M \geq 0$, as described in Algorithm **??**.

IV. Experiments

In this section we describe the various experiments. Section **??** defines the non-stationary problem at hand, Section **??** presents the tuning of the UCB variants, Section **??** shows the tuning of the DE algorithm and finally Section **??** compares the two algorithms.

A. Non-stationary data

We use a portfolio of solvers over the benchmark Cart-Pole problem[**?**] as a testbed. Each option of the bandit problem represents a black-box Direct Policy Search (DPS)[**?**]. More specifically for our case, we focus on three variants of black-box Noisy Optimization Algorithms (NOAs) combined with Neural Network (NN) (1, 2, 4, 6, 8 and 16 neurons). The three black-box NOAs are:

- a Self-Adaptive (μ, λ) Evolution Strategy with resamplings (RSAES)[**?**];
- a Fabian's stochastic gradient algorithm with finite differences[**?**], [**?**], [**?**];
- and a variant of Newton algorithm, adapted for noisy optimization, with gradient and Hessian approximated by finite differences and resamplings[**?**], termed Noisy Newton.

Algorithm 2 DE/best/1

```
Require: $K$: Dimension
Require: $T$: Computational budget
Require: $Truncation$: 1 if truncated, 0 otherwise
Require: $M$: Threshold if truncated
Require: $|\mathcal{X}|$: The size of the population $\mathcal{X}$
  Initialize each $x \in \mathcal{X}$ randomly $\in (0,1)^K$
  Normalize the values $x_{k \in K}$
  Initialize $t \leftarrow 1$
  Compute $f(x)$ for each $x \in \mathcal{X}$
  Set $a : f(a) \geq f(x) \in \mathcal{X}$
  while $t \leq T$ do
    if $Truncation$ and $t > M$ then
      Execute truncation
    end if
    for each $x \in \mathcal{X}$ do
      Evaluate $x$ $t'$ times
      $t \leftarrow t + t'$
      Pick randomly 2 distinct individuals $b$ and $c$ from $\mathcal{X}$
      Pick a random index $R \in K$
      Get a new position $y$ for the individual $x$ as follows:
      for each $k \in K$ do
        Pick a random number $rand \equiv \mathcal{U}(0,1)$
        if $rand < Cr$ or $k == R$ then
          Set $y_k = a_k + F(b_k - c_k)$
        else
          $y_k = x_k$
        end if
      end for
      Normalize the values $y_k,\ \forall k \in K$
      Compute $f(y)$
      if $f(y) > f(x)$ then
        replace $x$ by $y$
      end if
      if $f(x) > f(a)$ then
        replace $a$ by $x$
      end if
    end for
  end while
  Pick the agent $\tilde{x} \in \mathcal{X}$ using recommendation policy
  Return $\tilde{k}$
```

Basically at iteration n, the NOA chooses one (or more) new point(s) z_n in the search domain and gets a reward $r_{k,t}$. RSAES and Noisy Newton use resamplings, i.e. re-evaluations, to reduce the noise. Note that the budget is consumed accordingly with the number of resamplings. Table **??** describes the specifics of the NOAs that we use in our experiments. λ_I represents the number of points selected (more generally termed individuals size) during one iteration, r_n is the number of resamplings of each individual and σ_n is the stepsize. We refer to [**?**] for more details on the settings and the different algorithms. As shown in Table **??**, 24 solvers are used to resolve the Cart-Pole problem, thus resulting in 24 options in

TABLE I: Solvers used in the experiments, where d is dimension depending on the number of neurons; r_n is resampling number at iteration n for each individual. We refer to [?] for more details.

Number of neurons	Algorithm and parametrization
1, 2, 4, 6, 8, 16	RSAES with individual size $\lambda_I = 10d$, $\mu = 5d$, $r_n = 10n^2$.
1, 2, 4, 6, 8, 16	Fabian's solver with individual size $\lambda_I = 4$, stepsize $\sigma_n = 10/n^{0.1}$, $a = 100$, no resamplings.
1, 2, 4, 6, 8, 16	Noisy Newton's solver with individual size $\lambda_I = 4d^2 + 2d + 1$, stepsize $\sigma_n = 10/n$, $r_n = n^2$.
1, 2, 4, 6, 8, 16	Noisy Newton's solver with individual size $\lambda_I = 4d^2 + 2d + 1$, stepsize $\sigma_n = 100/n^4$, $r_n = n^2$.

the non-stationary bandit problem. Experiments are repeated 50 times and we present the mean of these results.

B. Tuning of bandit

In this section we present the tuning of the different UCBs under study. For the baseline UCB, the only parameter to tune is the C constant, which control the tradeoff between exploration and exploitation. Figure **??** presents a summary of the best combinations of parameters. The x-axis represents the number of evaluations and the y-axis shows the Simple Regret. For each point in Figure **??**, 50 independent trials are executed and we present the mean of these results.

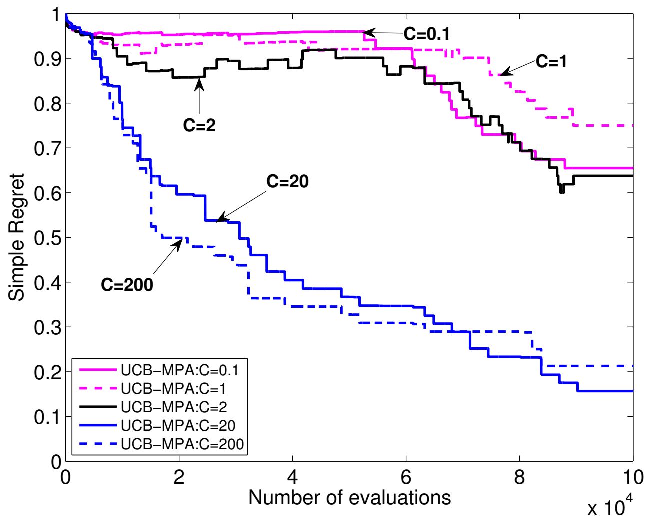


Fig. 1: Simple Regret for UCB. Tested parameters are $C \in [10^{-3}, 200]$. For the clarity of graph, we remove some similar results and only show results for $C = 0.1$, $C = 1$, $C = 2$, $c = 20$ and $C = 200$. UCBdn converges faster and depends less on C than UCB or UCBdt. The recommendation policy $\tilde{\pi}$ is Most Played Arm (MPA), where arm refer to option in our paper.

UCB does not possess the inherent quality to tackle non-stationary bandit problem. Instead, to artificially induce the ability to keep adapting to the variation in the distributions we expect higher exploration rate C. This is exactly what we observe in Figure **??** where the best C constant is 200 when the number of evaluations is below 7×10^4 and diminishes to 20 as the number of evaluations increases. This value is much higher than on stationary problem where typical empirical values are usually within $[0, 2]$.

For UCBdn and UCBdt, we look for the best combination of parameters C and γ. Figure **??** presents different C values for the best $\gamma = 0.85$ found for UCBdn (Top) and the best $\gamma = 0.99$ found for UCBdt (Bottom). The rest of the setting is similar to Figure **??**.

The best combination of parameters for UCBdn seems to be in the vicinity of $C = 2$ and $\gamma = 0.85$ for the given problem. The current γ value indicates that a fair amount of information is lost at each iteration but it is combined with a relatively small exploration factor $C = 2$. As opposed to UCBdn, UCBdt possesses a very high value for γ and C. Its best γ seems to indicate that UCBdt does not rely on forgetting information throughout the iterations. As such, it is conceivable that it relies on a higher exploration factor C. It appears that because UCBdt has a forget rate that is not related with the way the distributions evolve its best combination of parameters boils down to almost a normal UCB.

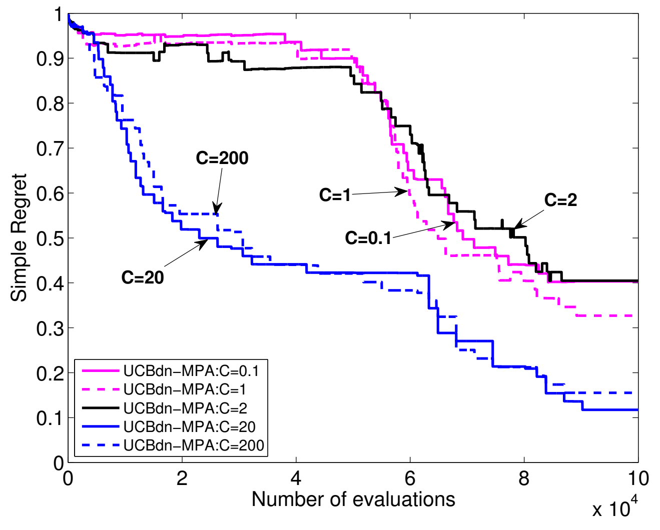
From Figure **??**, it clearly appears that UCBdn is the best variant of UCB for the settings describe in this paper.

C. Tuning of DE

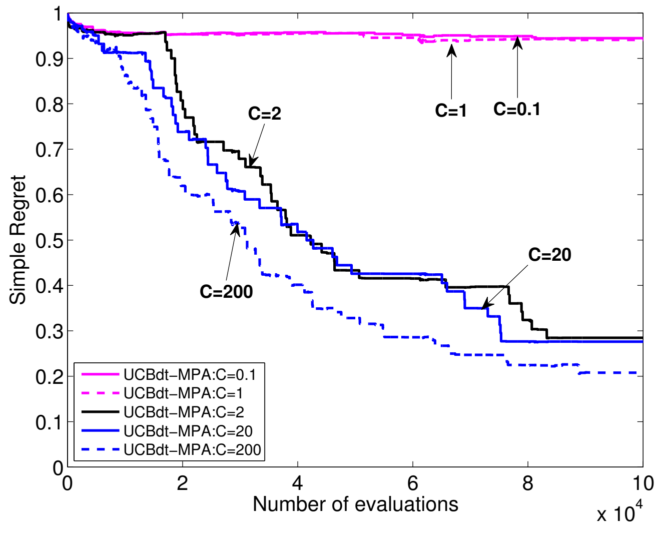
In this section we tune different variants of DE and compare them. There are 3 parameters to tune for DE and its variants: λ, F and Cr. Different settings of $F \in [0, 2]$ and $Cr \in [0, 1]$ are presented in combination with the 2 best population sizes that are found which are $\lambda = 5$ and $\lambda = 10$.

Figure **??** shows average performance of DE in terms of Simple Regret. The x-axis represents the number of evaluations and the y-axis shows the Simple Regret. For each point in Figure **??**, 50 independent trials are executed and we present the mean of these results.

First, it seems that a smaller population yields better performance. It is understandable since each particle consumes part of the evaluation budget and does not directly share information between them. As such, there is a loss in the efficiency of sampling which increases as the size of the population grows. The best parameter setting for $\lambda = 10$ (Bottom) is $F = 0.1$ and $Cr = 0.7$, that is a relatively small differential weight F that prevents the distribution to move too fast in a direction but with a relatively high crossover probability Cr. For $\lambda = 5$ (Top), the combination $F = 0.1$ and $Cr = 0.7$ still yields good results but there exists several other combinations that give similar results. After 9×10^4 evaluations, none of the combination of F and Cr for a $\lambda = 10$ reaches a simple regret lower than 0.5 whereas for the same number of evaluations, the combination $\lambda = 5$, $F = 0.1$ and $Cr = 0.7$ reaches 0.23.

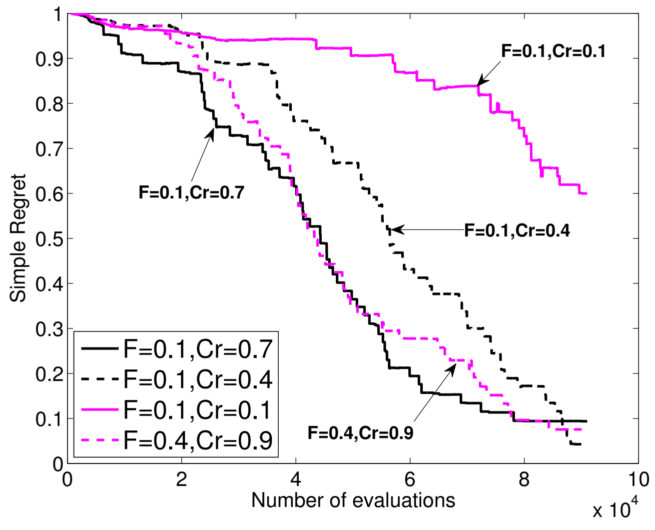


(a) UCB discounted2 (UCBdn) with $\gamma = 0.85$.

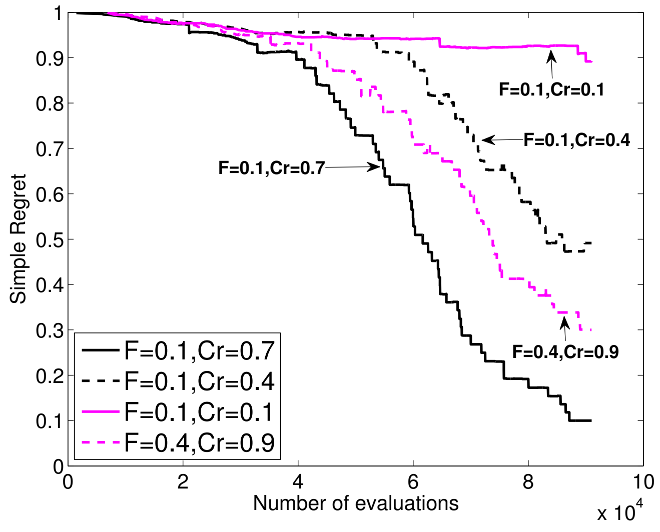


(b) UCB discounted (UCBdt) with $\gamma = 0.99$.

Fig. 2: Top: UCBdn performance with $\gamma = 0.85$; Bottom: UCBdt performance with $\gamma = 0.99$. Several values of γ and C are tested for UCBdn and UCBdt, we remove some results for the clarity of graphs. Only the best parameterization of γ are presented. UCBdt relies on a higher exploration factor C. UCBdn depends less on C and converges faster than UCBdt using the same constant C.

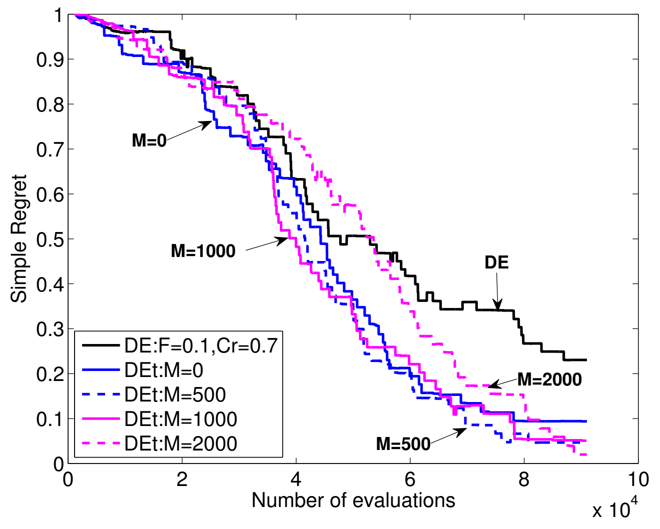


(a) Normal DE, $\lambda = 5$.

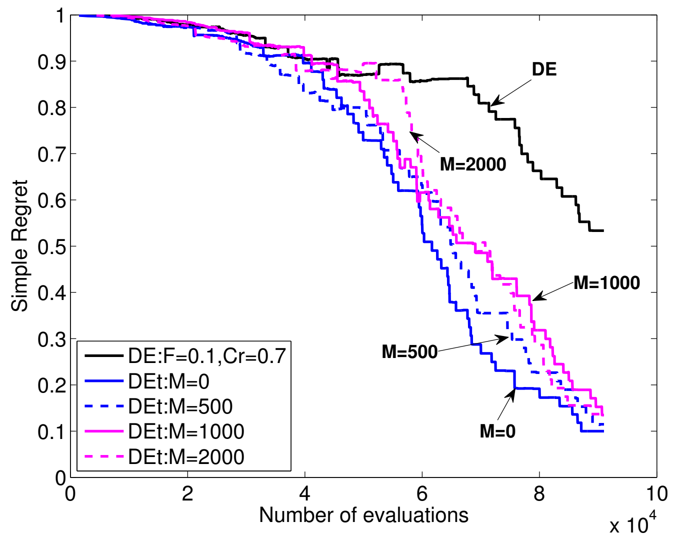


(b) Normal DE, $\lambda = 10$.

Fig. 3: Performance of different variants of DE in terms of Simple Regret. Number of options $K = 24$, population size $\lambda = 5$ (Top) and $\lambda = 10$ (Bottom). Here we only show a summary of results for the parameters. When $F = 0.1$, bigger Cr gets better performance. Normal DE can not converge when near to optimum, which is to be expected from an evolutionary algorithm.



(a) Truncated-DE compared to normal DE, $\lambda = 5$.



(b) Truncated-DE compared to normal DE, $\lambda = 10$.

Fig. 4: Performance of different variants of DE in terms of Simple Regret. Number of options $K = 24$, population size $\lambda = 5$ (Top) and $\lambda = 10$ (Bottom). Here we only show a summary of results for the parameters. Black solid curve is the best DE found in Figure **??**. Truncated-DE clearly outperforms the normal DE. M influences slightly on the convergence rate.

Figure **??** presents performance in terms of simple regret for different values of M for the truncated variant of DE presented in Section **??**. We used $c_1 = 0.7$ and $c_2 = 0.7$. The rest of the settings are similar to Figure **??**.

To facilitate the comparison of performance between normal DE and the truncated variant, we also plot the best normal DE found ($\lambda = 5$, $F = 0.1$, $Cr = 0.7$). Every tested variant of the truncated version outperform the normal DE. The best truncation factor for a population size $\lambda = 10$ appears to be $M = 0$ which indicates that early truncation are better.

From Figure **??**, it clearly appears that smaller population size λ yields better results which is in line with previous findings. For a population size $\lambda = 5$, the best value of M is 500 in early iterations ($< 9 \times 10^4$) and, as the number of evaluations grows ($> 9 \times 10^4$) the best M is 2 000. These results indicates that truncating early yields good results yet if the budget is large enough it is better to gather more information before starting the truncation process.

D. DE vs Bandit

This section compares the best UCBs with the best variants of DE. It is important to note that each algorithm are tuned individually instead of using generic constant for a family of algorithms. In this section, we also add another baseline which consists in choosing an option randomly and always pulling the same one. We call this baseline $Random$.

Figure **??** shows the average performance in terms of Simple Regret of the best UCBs and DEs. The x-axis represents the number of evaluations and the y-axis shows the Simple Regret. For each point, 50 independent trials are executed and we present the mean of these results. Standard deviation for each curve is of order of magnitude of 0.015.

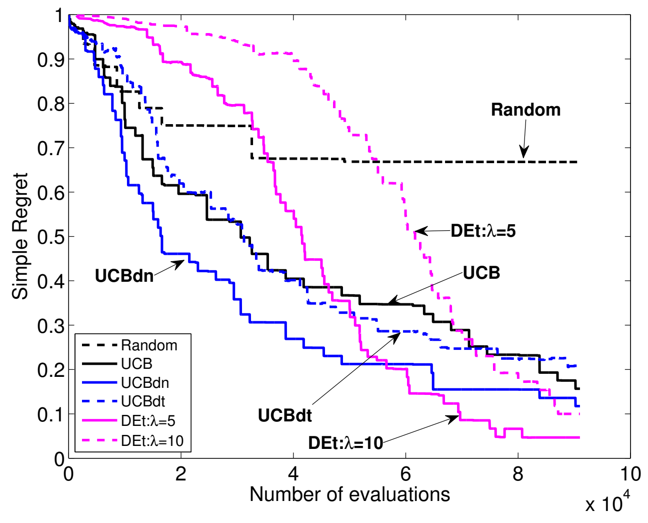


Fig. 5: Comparison of truncated-DE, UCB, UCBdn, UCBdt and $Random$. UCBdn outperforms UCB and UCBdt and is clearly the best algorithm if the budget is small. However, as the number of evaluations grows (5.7×10^4), truncated-DE gets better (smaller) Simple Regret. Standard deviation for each curve is of order of magnitude of 0.015.

Given a decent number of evaluations, every algorithm outperforms the baseline $Random$. As expected, the best of the UCB variant (UCBdn) is the best algorithm up to a budget of 5.7×10^4. Afterward, truncated-DE for $\lambda = 5$, $M = 500$ is the best available algorithm.

There are two important conclusions that we can infer from these experiments. First, the variant of UCB (UCBdn) presented in this paper is the best UCB algorithm for non-stationary bandit problem where the distribution varies with the number of times an option is selected. Second, Evolutionary Algorithm such as DE, given some adaptation, can outperform bandit specific algorithms on bandit problem.

V. Conclusion

In this paper, we compare an evolutionary algorithm, namely DE, to bandit algorithms on a black-box portfolio selection problem. Such problems can be formalized into a non-stationary bandit problems, where each option increases their performance as the number of times we evaluate it increases.

First, we introduce our definition of the non-stationary bandit problem and justify such an approach through a direct application (portfolio of solvers for the Cart-Pole problem). Second, we present the current state-of-the-art bandit algorithms for the given problems and propose our own variant (UCBdn) that outperforms all the other UCB-like algorithm on non-stationary distributions where each distribution changes according to the number of times the option is selected.

Third, we introduce DE, an evolutionary algorithm, and apply it over this specific non-stationary bandit problem. We also present our own variant of DE which outperforms the classic version.

Fourth and last, we compare the best variant of each category (UCBdn and truncated-DE). If the budget is smaller than 5.7×10^4 we propose the use of UCBdn and, as the number of evaluations increases, truncated-DE is better.

For future research, we intend to apply truncated-DE on a wider range of problems. Moreover, we would like to explore the idea of generic parameter tuning in the case of DE. As for the bandit algorithm part, to compute theoretical bound for UCBdn would provide interesting information on its generic performance.

Acknowledgment

This work is supported by the ADEME POST project. The authors would like to thank the reviewers for their comments.