

Progress Rate in Noisy Genetic Programming for Choosing λ

Jean-Baptiste Hooek¹ and Olivier Teytaud^{1,2}

¹ TAO (Inria), LRI, UMR 8623(CNRS - Univ. Paris-Sud),

bat 490 Univ. Paris-Sud 91405 Orsay, France, firstname.name@lri.fr

² Dept. of Computer Science and Information Engineering, National University of Tainan, Taiwan

Abstract. Recently, it has been proposed to use Bernstein races for implementing non-regression testing in noisy genetic programming. We study the population size of such a $(1 + \lambda)$ evolutionary algorithm applied to a noisy fitness function optimization by a progress rate analysis and experiment it on a policy search application.

1 Introduction

Genetic programming (GP) consists in automatically building a program for solving a given task. The fitness function quantifies the efficiency of a program for this task.

Non-regression testing consists in testing each new version of a program, in order to check that it is at least as good as the previous version. Non-regression testing is very difficult when the fitness function is noisy, as it is uncertain. Statistical tests have to take into account the high number of tests - this is a non-trivial issue. The present work originates in the tedious non-regression testing in a highly collaborative development, often poorly performed by humans. They don't care at the individual level of the global risk due to multiple testings. Tests had to be automatized, simultaneously with the development of improvements in a search module by automatic means in order to get rid of human biases in such developments.

Noisy GP is an important problem, related to many applications. In particular, direct policy search with symbolic controllers is noisy genetic programming, as well as the design of sorting algorithms faster on average on a huge number of instances. Bandits have been investigated very early for this problem[9, 7]. However, bandits address the problem of the load balancing between different possible mutations, but not the validation of the selection, i.e. the halting criterion for the offspring evaluation. Races[11] have the double advantage of considering the load balancing and the statistical validation. [11] considers the case in which we look for all good mutations. We might, in GP, be more interested in finding one good mutation, as μ good mutations do not necessarily cumulate to one better mutation. Yet, the case of selecting several good mutations is important also, but it was shown in [8] that there are frameworks in which $\mu = 1$ is more relevant and we focus on this case. In the case of

continuous optimization, races have been theoretically analyzed as a tool for ensuring optimal rates (within log factors) in [12]. A partial theoretical analysis, essentially ensuring consistency, was proposed for GP in [8]: we here extend this work by an analysis of progress rate. Importantly, the theoretical analysis proposes a choice for λ by optimisation of the progress rate.

Progress rate theory is classical in continuous optimization[2]. The progress rate will be here adapted to noisy GP. We will consider T (formally defined below), the number of fitness evaluations before finding a good mutation. The progress rate is then defined as $1/T$, the inverse time before finding a good mutation. We will then, following the continuous case, choose parameters that optimize the progress rate. Note that here, the progress rate is a success probability; this is equivalent to classical criteria [3] only in restricted settings (see assumptions in Section 3).

2 Framework and notations

Consider a program P , and a set M of possible mutations; let $P + m$ be the program after application of mutation m . Assume that the fitness is stochastic; $f(P + m)$ is a random variable with values in $[-1, 1]$. In all the paper we consider maximization. We consider $(1 + \lambda)$ genetic programming algorithms as in Alg. 1; this is a simple $(1 + \lambda)$ -algorithm[8].

RBGP algorithm.

Parameters: a risk level δ , a pop. size λ , a coefficient $c > 1$, a threshold $\epsilon > 0$.

Let $K = 1/(\sum_{n \geq 1} 1/n^c)$

Let P be a program to be optimized

Let M be a set of possible mutations on P

Let $n \leftarrow 0$

while There is some time left **do**

while no mutation accepted **do**

 Let $n \leftarrow n + 1$

 Randomly draw $pop = \{m_1, \dots, m_\lambda\}$ in M

 Perform a Bernstein race with risk $\delta_n = K\delta/n^c$ on pop and threshold ϵ

if A mutation m is selected **then**

$P \leftarrow P + m$

end if

end while

end while

Algorithm 1: One-plus-lambda Racing-Based Genetic Programming. We assume $c > 1$ so that K is finite.

All this section is written assuming that the range of fitness values is bounded in absolute value by 1, i.e. when we compute the fitness of a

point we get an answer between -1 and 1 . We assume that the fitness is unbiased, *i.e.* the expected value of the measurements is equal to the real fitness. This is standard in *e.g.* Monte-Carlo sampling, or in some randomized forms of Quasi-Monte-Carlo samplings.

Hoeffding and Bernstein bounds quantify the effect of noise on empirical averages. In noisy cases, the fitness $fitness(x)$ of a point x is unknown: we have only access to y_1, \dots, y_k , k real numbers, if $fitness(x)$ has been approximated k times; a natural estimate is $\widehat{fitness}(x) = \frac{1}{k} \sum_{i=1}^k y_i$. Hoeffding or Bernstein bounds state that with probability at least $1 - \delta$, $|fitness(x) - \widehat{fitness}(x)| < deviation$, where *deviation* is

$$deviation_{Hoeffding} = \sqrt{\log(2/\delta)/k}. \quad (1)$$

[1, 11] has shown the efficiency of using Bernstein's bound instead of Hoeffding's bound, in some settings. The deviation term is then:

$$deviation_{Bernstein} = \sqrt{\hat{\sigma}^2 2 \log(3/\delta)/n} + 3 \log(3/\delta)/k. \quad (2)$$

This equation depends on $\hat{\sigma}^2$, the empirical variance of the measurements y_1, \dots, y_k . An interesting feature of this equation is that using $\hat{\sigma}^2$ and not the real variance σ^2 is not an approximation: the inequality is rigorous with $\hat{\sigma}^2$ (contrarily to many asymptotical confidence intervals).

The Bernstein race is a typical one, following [11], except that we want to validate one and only one mutation, because in our framework it is known [8] that two good mutations do not necessarily cumulate, in the sense that sometimes:

$$\begin{aligned} \mathbb{E}f(P + m_1) &> \mathbb{E}f(P) \\ \mathbb{E}f(P + m_2) &> \mathbb{E}f(P) \\ \text{and yet } \mathbb{E}f(P + m_1 + m_2) &< \mathbb{E}f(P). \end{aligned}$$

In a $(1 + \lambda)$ -GP, we look for a mutation which provides a better success rate than the current parent, *i.e.* a higher fitness. However, we will here translate the fitness by subtracting the fitness of the parent; *i.e.* with a fitness function with values in $[-1, 1]$ and a parent z , we define

$$fitness'(x) = (fitness(x) - fitness(z))/2;$$

this just doubles the number of calls to the fitness. Therefore, in the Bernstein race, and without loss of generality, we look for a mutation which gives a positive fitness, instead of a mutation with $fitness(x) > fitness(z)$. The Bernstein race is as presented in Alg. 2, and the *computeBounds* function is defined as shown in Alg. 3.

where $\#$ denotes the cardinal operator.

Important properties of Bernstein's races as above, and which hold with probability at least $1 - \delta$, are the followings ([11]).

Properties of Bernstein races.

- *Property 1. The number of evaluations in a Bernstein race*
 - with population size $\#pop$;
 - with parameters δ and ϵ ;

```

BernsteinRace(pop, δ, ε)
while pop ≠ ∅ do
  for all m ∈ pop do
    Let n be the number of simulations of mutation m.
    Simulate m n more times (i.e. now m has been simulated 2n
    times).
    //this ensures nbTests(m) = O(log(n(m)))
    computeBounds(m, M, δ)
    if lb(m) > 0 then
      Return individual corresponding to mutation m.
    else if ub(m) < ε then
      pop = pop \ {m}           m is discarded.
    end if
  end for
end while
Return "no good individual in the offspring!"

```

Algorithm 2: Bernstein race for selecting good individuals in a population pop . M is the complete set of arms (global variable; see Alg. 1).

```

Function computeBounds(m, pop, δ)
Static internal variable: nbTests(m), initialized at 0.
Let n = nbTests(m).
Let r be the total reward over those n simulations.
nbTests(m) = nbTests(m) + 1
lb(m) =  $\frac{r}{n} - deviation_{Bernstein}(\delta / (\#pop \times \pi^2 nbTests(m)^2 / 6), n)$ .
ub(m) =  $\frac{r}{n} + deviation_{Bernstein}(\delta / (\#pop \times 2\pi^2 nbTests(m)^2 / 6), n)$ .

```

Algorithm 3: Function for computing a lower and an upper bound on arm m with confidence $1 - \delta$, where pop is the complete set of arms.

- with a population such that, with $p = \max(\epsilon, \max_{m \in \text{pop}} \mathbb{E}f(P + m))$, all expected fitness values are in $[-\Theta(\epsilon), p]$ (possibly all fitness values ≤ 0 , case in which there's no good mutation).

is

$$\text{Time}(\text{pop}, \delta, \epsilon) = \Theta(\text{Th}(\#\text{pop}, \delta, \epsilon, p))$$

where

$$\text{Th}(\#\text{pop}, \delta, \epsilon, p) = \left(\log\left(\frac{\#\text{pop} \log(1/p)}{\delta}\right) \right) \max(\sigma^2/p^2, 1/p) \quad (3)$$

with: σ^2 is an upper bound on the variance of fitness values:

$$\sigma^2 = \sup_m \mathbb{E}(f(P + m) - \mathbb{E}f(P + m))^2.$$

- Property 2. If a mutation is selected, then it has fitness > 0 .
- Property 3. If there is a mutation with fitness $\geq \epsilon$ then the race will return a mutation.

We will here focus on the case $\sigma^2 = \Theta(1)$ (which corresponds to the applicative framework in which variance does not decrease; this is consistent with many applicative fields, in particular when optimizing the parameters of a strategy with optimal success rate $< 100\%$ - yet, other cases might be considered in a future work), and therefore Eq. 3 can be replaced by Eq. 4 without significant loss:

$$\text{Th}(\#\text{pop}, \delta, \epsilon, p) = \left(\log\left(\frac{\#\text{pop} \log(1/p)}{\delta}\right) \right) / p^2. \quad (4)$$

Properties of RBGP. Eq. 4 and properties of Bernstein races above lead to the following properties of RBGP, which hold (all simultaneously) with probability at least $1 - \delta$: if there are

- $n - 1$ iterations of RBGP in which all mutations have expected fitness $\in [-\epsilon, 0]$;
- and thereafter, 1 iteration of RBGP with at least one mutation with expected fitness p and all other mutations with expected fitness in $[-\epsilon, p]$.

then

- Property A: RBGP will not return a bad mutation (i.e. a mutation with fitness ≤ 0);
- Property B: If there is a mutation with fitness $\geq \epsilon$ then a mutation will be found;
- Property C: And in that case the halting time is at most

$$O\left(\sum_{i=1}^{n-1} \text{Th}(\lambda, K\delta/i^c, \epsilon, p) + \text{Th}(\lambda, K\delta/n^c, p)\right) \quad (5)$$

Eq. 5 will be central in the progress rate analysis below.

3 Progress rate of Racing-based GP

We consider that randomly drawn mutation have fitness¹:

- $q > 0$ with probability $f > 0$;
- ≤ 0 otherwise.

We will study the behavior of $(1 + \lambda)$ -GP depending on q and f . The other relevant parameters are:

- λ (the offspring size);
- ϵ , the threshold of the Bernstein races (see Alg. 1);
- c , a parameter used in Alg. 1 and which is of moderate importance as shown below.

The different cases under analysis are (i) $q = \epsilon$ (ii) $q \gg \epsilon$. We will investigate the running time, i.e. the number T of fitness evaluations before finding a good mutation with probability at least $1 - 2\delta$. It is already known[8] that with probability at least $1 - \delta$,

- if there is a good mutation ($fitness \geq q$), it will be found;
- no bad mutation ($fitness < 0$) will be selected. Possibly, the race replies that it did not find any good mutation.

We will show (i) that a too mild rejection threshold ϵ has bad effects (section 3.1); (ii) that a good tuning provides significant improvements (section 3.2); (iii) that there is a parameter free version with ensures that there's no infinite loop (section 3.3).

3.1 Too mild rejection: $q \gg \epsilon$

Here, the precision required for rejecting a bad mutation is very small in front of the quality of good mutations. The following result shows that in that case the choice of λ is crucial: λ should be of the order

$$\log(\delta)/\log(1 - f) \tag{6}$$

Theorem 1: rejection pressure too small (ϵ too small). *Assume that $q > \epsilon$. Then,*

- *If $\lambda \geq \lceil \frac{\log(\delta)}{\log(1-f)} \rceil$, then $T = \Theta(\lambda \log(\lambda/(q\delta))/q^2)$.*
- *If $\lambda \leq \frac{\log(1/2)}{\log(1-f)}$, then $T = \Omega(\log(1/(f^2\epsilon\delta))/\epsilon^2)$.*

Proof: Eq. 5 shows that the computational cost is

$$O\left(\sum_{i=1}^{n-1} Th(\lambda, K\delta/i^c, \epsilon, q)Th(\lambda, K\delta/n^c, \epsilon, q)\right). \tag{7}$$

where n is the number of iterations before a good mutation is in an offspring.

The basic fact for the following is that the probability, for an offspring, to contain no good mutation is exactly $(1 - f)^\lambda$.

We distinguish the two cases of the theorem:

¹ We recall that fitnesses are translated as explained in section 2 so that the parent has fitness 0 and therefore “good” mutations are mutations with value > 0 .

– If $\lambda \geq \lceil \frac{\log(\delta)}{\log(1-f)} \rceil$, then

$$(1-f)^\lambda \leq \delta.$$

This implies that with probability at least $1 - \delta$, there's at least one good mutation in the first offspring. Therefore, $n = 1$; this and Eq. 7 yield the expected result.

– If $\lambda \leq \frac{\log(1/2)}{\log(1-f)}$, then

$$(1-f)^\lambda \geq \frac{1}{2}.$$

Therefore with probability at least $\frac{1}{2}$, there's no good mutation in the first offspring. Then, Eq. 7 (the first term) is enough for showing that

$$T = \Omega(\log(1/(f^2 \epsilon \delta))/\epsilon^2)$$

which is the expected result. \square

Remark: In the case $q \gg \epsilon$, this theorem implies that $\lambda \geq \lceil \frac{\log(\delta)}{\log(1-f)} \rceil$ is much better than $\lambda \leq \frac{\log(1/2)}{\log(1-f)}$.

3.2 Well tuned parameters: $q = \epsilon$

Theorem 2: population size with well tuned parameters. *Assume that $q = \epsilon > 0$. Then,*

$$T = \Theta\left(\frac{\lambda \log(\lambda \log(\frac{1}{\epsilon})/\delta)}{(1 - (1-f)^\lambda)\epsilon^2}\right). \quad (8)$$

Proof: We use Eq. 5, which shows that

$$O\left(\sum_{i=1}^{n-1} Th(\lambda, K\delta/i^c, \epsilon, q) + Th(\lambda, K\delta/n^c, \epsilon, q)\right). \quad (9)$$

Then, we compute n , the number of offsprings before we have a good mutation. A good mutation is found in an offspring with probability $1 - (1-f)^\lambda$. Therefore, n is a geometric random variable with parameter $z^{-1} = (1 - (1-f)^\lambda)$: its expected value is $\Theta(z)$, and its standard deviation is $\Theta(z)$. So, with probability $1 - \delta$, $n = \Theta(1/(1 - (1-f)^\lambda))$. This and Eq. 9 lead to

$$T = \Theta\left(\sum_{i=1}^n Th(\lambda, K\delta/i^c, \epsilon, q)\right). \quad (10)$$

This is equivalent to

$$T = \Theta(nTh(\lambda, K\delta/n^c, \epsilon, q)). \quad (11)$$

which in turn is equivalent to Eq. 8 for a fixed c . \square

Remark: Theorem 2 provides an evaluation of the cost

$$T = \Theta\left(\frac{\lambda \log(\lambda \log(\frac{1}{\epsilon})/\delta)}{(1 - (1-f)^\lambda)\epsilon^2}\right).$$

If we neglect all logarithmic factors,

- this is linear as a function of λ if $\lambda \simeq 1/n$, i.e. the overall cost is linear as a function of $1/(f\epsilon^2)$.
 - this is linear as a function of $1/(f\epsilon^2)$ if $\lambda = 1$.
- We therefore see that the population size does not matter a lot when the parameter ϵ is chosen so that it nearly matches q .

3.3 No prior knowledge

The analysis above has the weakness that it requires some knowledge on the fitnesses of possible mutations, in order to choose ϵ , the parameter used in the rejection rule. The main risk is a too strong rejection: $q \ll \epsilon$ would lead to the rejection of the best mutations. We here investigate results possible with no knowledge at all.

```

BernsteinRace(pop,  $\delta$ ,  $\epsilon$ )
while pop  $\neq \emptyset$  do
  for all  $m \in pop$  do
    Let  $n$  be the number of simulations of mutation  $m$ .
    Simulate  $m$ ,  $n$  more times
      (i.e. now  $m$  has been simulated  $2n$  times).
      //this ensures  $nbTests(m) = O(\log(n(m)))$ 
    computeBounds( $m, M, \delta$ )
    if  $lb(m) > 0$  then
      Return individual corresponding to mutation  $m$ .
    else if average fitness( $m$ )  $< 0$  then
      pop = pop  $\setminus \{m\}$             $m$  is discarded.
    end if
  end for
end while
Return "no good individual in the offspring!"

```

Algorithm 4: Variant of Bernstein race: a pattern is rejected as soon as its average fitness is below 0.

Theorem 3: population size with $q \ll \epsilon$. Consider RBGP with the Bernstein race as in Alg. 4. Then, with probability $1 - \delta$, no bad mutation is accepted, and if the probability of a good mutation is positive, then

$$T < \infty. \quad (12)$$

Proof: The fact that no bad mutation is accepted (with probability at least $1 - \delta$) follows the same lines as in other cases, as the acceptance criterion has not been modified.

We have to show $T < \infty$. This is proved as follows:

- A mutation with fitness $> q$ is selected infinitely often, as such a mutation is in the population with positive probability by assumption.
- Such a mutation has a non-zero probability of having always a positive empirical average. \square

4 Experimental results

The theoretical results above for choosing the population size are rather preliminary; the population size can be chosen optimally only if we have many informations. On the other hand, it proposes a criterion different from classical Bernstein races: acceptance is based on the same criterion as usual Bernstein races, but rejection is based on a simple naive empirical average, and with this criterion we have T finite without any prior knowledge. We here experiment rules a bit more complicated than algorithms above.

We have already tried the algorithm on the program MoGo (a software of Go) without real success against the full version of the software. We have decided to compare with another testbed. The testbed is Monte-Carlo Tree Search (MCTS [4, 6]) on the game NoGo [5]. It is a two-player board game. It is a variant of the game of Go. The rule is the following : the first player which captures one or several stone(s) has lost and the pass move is forbidden. This game has been designed by the Birs seminar on games as a nice challenge for game developpers. In all our experiments, we have worked on the size 7x7 of the game.

The baseline is the program NoGo (adapted from MoGo[10]) without mutation. In our experiments, we have added some rules in order to reject more rapidly some bad mutations, combining Algo. 2 (which requires some knowledge on the distributions) and Algo. 4 (which requires no knowledge). This leads to Algo. 5, which is somehow a combination of these two algorithms, empirically developped for our problem.

```

BernsteinRace(pop,  $\delta$ ,  $\epsilon$ )
while pop  $\neq$   $\emptyset$  do
  for all  $m \in$  pop do
    Let  $n$  be the number of simulations of mutation  $m$ .
    Simulate  $m$   $n$  more times (i.e. now  $m$  has been simulated  $2n$ 
    times).
    //this ensures  $nbTests(m) = O(\log(n(m)))$ 
    computeBounds( $m, M, \delta$ )
    if  $lb(m) > 0$  then
      Return individual corresponding to mutation  $m$ .
    else if average fitness( $m$ )  $< 0.004$  or (average fitness  $<$ 
     $0.006$  and  $n > 10^5$ ) then
      pop = pop  $\setminus$  { $m$ }
       $m$  is discarded.
    end if
  end for
end while
Return "no good individual in the offspring!"

```

Algorithm 5: Empirically modified version of Bernstein race for our problem. It is essentially Alg. 4, with a bit more of rejection for fastening the algorithm.

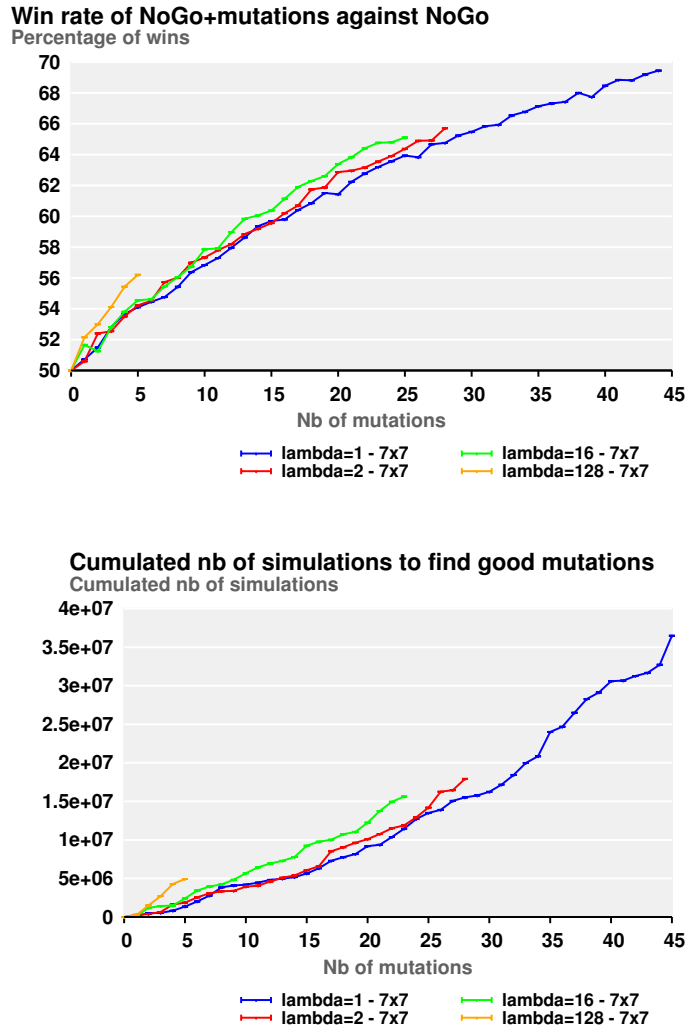


Fig. 1. Top: win rate of NoGo+mutations against the baseline (i.e. NoGo without mutation). Win rates are given with a precision of +/-0.3%. **Bottom:** the “running time“ (measured by the cumulated number of simulations) for finding a good mutation.

Fig. 1 shows a slightly better result for $\lambda = 16$, for a fixed number of mutations compared to $\lambda = 1$ and $\lambda = 2$; but more extensive experiments (possibly on toy datasets) are required; results are nearly the same for all λ in our real-world case. In all cases, we could get a nice curve, with a very significant (almost 70%) success rate against the baseline, which is still clearly increasing (yet, in a slower manner).

Another question is about the best population size λ in our experiments. It seems that we have generally a good mutation with frequency $1/15$. Using Eq. 6, with $\delta = 0.05$ and assuming $f \simeq 1/15$, we get $\lambda \simeq \log(0.05)/\log(1 - 1/15) \simeq 43$. Therefore our analysis suggests a population size $\lambda \simeq 43$.

5 Conclusion

The use of Bernstein races for rigorously performing non-regression testing was already proposed in [8]. We here investigate the natural question of the choice of the population size λ , and the modification of Bernstein races when no prior knowledge is available:

- **Choosing the population size.** The good news is that we find a formula for optimally choosing λ , equal to $\log(\delta)/\log(1 - f)$ where f is the frequency of good mutations and δ the risk level chosen by the user. Unfortunately, f is unlikely to be known unless the fitness improvement q that one can expect from good mutations is nearly known, and in this case the user is likely to choose ϵ of the order of q , and we show that in this case there is little to win by a good choice of λ : $\lambda = 1$ performs at least nearly as well as all values of λ .
- **What if we have no prior knowledge ?** We could propose a modified Bernstein race (Alg. 4) which has the advantage that it always converges ($T < \infty$), independently of all parameters.

In the experiments, we heuristically combined our various tools for optimizing the performance, proposing Algo. 5. Importantly, we got a very significant result on a new game, NoGo, recently proposed by the Birs seminar on games - the curve shows a regular improvement, for each parametrization of the algorithm. Importantly, we made this work with applications in minds; further investigations, on toy datasets for convenience and clarity, are necessary - so that we can see experimental results with confidence intervals, bridging the gap between our maths and our real-world experiments.

Acknowledgments.

The authors would like to thank Fabien Teytaud for the help to read the article. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see <https://www.grid5000.fr>).

References

1. J.-Y. Audibert, R. Munos, and C. Szepesvari. Use of variance estimation in the multi-armed bandit problem. In *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*, 2006.
2. A. Auger and N. Hansen. Reconsidering the progress rate theory for evolution strategies in finite dimensions. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 445–452, New York, NY, USA, 2006. ACM.
3. H.-G. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Heidelberg, 2001.
4. G. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik. Monte-Carlo Strategies for Computer Go. In P.-Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.
5. C.-W. Chou, O. Teytaud, and S.-J. Yen. Revisiting Monte-Carlo Tree Search on a Normal Form Game: NoGo. In *EvoGames 2011*, volume 6624 of *Lecture Notes in Computer Science*, pages 73–82, Turino, Italy, Apr. 2011. Springer-Verlag.
6. R. Coulom. 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*, 2006.
7. J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.*, 2(2):88–105, 1973.
8. J.-B. Hoock and O. Teytaud. Bandit-Based Genetic Programming. In *13th European Conference on Genetic Programming*, Istanbul, Turkey, 2010. Springer.
9. J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.
10. 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*, pages 73–89, 2009.
11. V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical Bernstein stopping. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA, 2008. ACM.
12. P. Rolet and O. Teytaud. Bandit-based Estimation of Distribution Algorithms for Noisy Optimization: Rigorous Runtime Analysis. In *Lion4*, Venice Italie, 2010.