Online Black-Box Algorithm Portfolios for Continuous Optimization

Petr Baudiš and Petr Pošík

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics
Technická 2, 166 27 Prague 6, Czech Republic
pasky@ucw.cz, petr.posik@fel.cvut.cz

Abstract. In black-box function optimization, we can choose from a wide variety of heuristic algorithms that are suited to different functions and computation budgets. Given a particular function to be optimized, the problem we consider in this paper is how to select the appropriate algorithm. In general, this problem is studied in the field of algorithm portfolios; we treat the algorithms as black boxes themselves and consider *online* selection (without learning mapping from problem features to best algorithms a priori and dynamically switching between algorithms during the optimization run).

We study some approaches to algorithm selection and present two original selection strategies based on the UCB1 multi-armed bandit policy applied to unbounded rewards. We benchmark our strategies on the BBOB workshop reference functions and demonstrate that algorithm portfolios are beneficial in practice even with some fairly simple strategies, though choosing a good strategy is important.

1 Introduction

Continuous black-box optimization concerns itself with the problem of finding a minimum value of a real-parameter function that has inaccessible analytical form. This is a rich area of research that produced many algorithms over the last 50 years — from the venerable Nelder-Mead simplex algorithm [1] to various gradient descent methods to population-based methods.

However, if a function is truly “black-box” and its features are hard to predict, the key question in the face of such variety is “which algorithm should I choose?” We can turn for help at a common platform for performance comparison — the currently accepted de-facto standard is the *COmparing Continuous Optimisers* COCO platform [2] [3] that was originally developed for the BBOB workshop series and which provides (most importantly) a set of diverse reference benchmark functions. But there is still a long way from previously published performance results on reference functions to a decision about which algorithm to use on an arbitrary function provided by the user. A method to automate the process is certainly desirable.

The problem of algorithm selection is not new [4] and was so far popular mainly when applied to combinatorial problem solvers [5]. In our work, we adopt

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 40–49, 2014.
© Springer International Publishing Switzerland 2014

the prism of algorithm portfolios [6]. Let us have a set of *heuristic algorithms* (each suitable for a different class of problem instances). Given a problem instance, we apply a *selection strategy* to pick and apply an algorithm from this portfolio. We can perform the selection once or along a fixed schedule based on features of the problem instance *(offline selection)*, or in multiple rounds allocating time to portfolio members based on their performance in previous rounds *(online selection)*. In successive rounds, algorithms can be either resumed from their previous state or restarted; we take the approach of resuming them, reserving the restart schedule to be an internal matter of each algorithm.

Using algorithm portfolios for continuous black-box optimization is still a fresh area of research. The main results so far lie either in modifications of population methods that combine a variety of genetic algorithms together, e.g. the *MultiEA* [7], *AMALGAM-SO* [8] and *PAP* [9] methods; or in offline methods based on exploratory landscape analysis [10] which were also recently applied to the BBOB workshop scenario [11]. We could also draw ideas from hyper-heuristics and adaptive strategies for operator selection within population-based algorithms [12] [13]. A Multi-Armed Bandit scenario has been already considered in the field of function optimization for determining online restart schedules [14].

In our work, we enforce the distinction between algorithms and selection strategies — we regard the algorithms as *black-box*, completely avoiding any modification and simply repeatedly resuming them and allowing them to make another optimization step. This allows any already implemented state-of-art algorithms to be easily combined in a single portfolio and also extends to other than population-based methods. Furthermore, we focus on *online* adaptive selection that selects algorithms based on their performance so far and does not require possibly expensive or brittle feature extraction and training.

We aim to confirm whether using an algorithm portfolio is advantageous compared to investing the whole budget in a single overally dominant algorithm, and how do selection strategies influence the portfolio performance.

In section 2, we investigate algorithm selection through the paradigm of the Multi-Armed Bandit Problem, motivating the proposed selection strategies. In section 3, we describe the exact strategies we have compared in our experiments. We present the experimental results in section 4 and draw our conclusions and outline some directions for future work in section 5.

2 Algorithm Selection as Multi-armed Bandit Problem

When considering an action selection strategy that operates in an initially unknown environment, we face the fundamental dilemma of balancing exploitative actions maximizing the reward based on our current model of the environment and exploratory actions that refine the model. Multi-Armed Bandit Problem [15] is a reference statistical problem that allows abstract study of the exploration-exploitation dilemma.

2.1 Multi-armed Bandit Problem

In its typical formulation [16], a K-armed bandit problem is defined by a sequence of random rewards $X_{it} \in [0,1], i = 1, \ldots, K, t \in \mathbb{N}$, where i is an index of the arm of a bandit (in other words, a gambling machine) and t denotes successive pulls of the arm. All rewards are independent random variables and rewards of a single arm follow an identical stationary distribution, but the distribution and expected value are originally unknown. A bandit policy π is then a function that selects the next arm to be pulled based on the sequence of rewards up to that point. The goal is to maximize cumulative reward over time; the policy aims to pull the arm with highest expected reward (exploitation), but needs to continually update its belief about which arm has the best reward (exploration).

The measure of policy performance is its "regret", i.e. cumulative reward loss compared to a hypothetical oracle policy. Let μ_i denote the true expected reward of arm i, μ^* the true expected reward of the optimal arm, $T_i(n)$ the number of times arm i has been pulled up to the n-th pull, and R_n the current regret:

$$R_n = n\mu^* - \sum_{i=1}^{K} \mathbb{E}\left[T_i(n)\right] \mu_i$$

It was proven early [16] that the lower asymptotic bound for the regret R_n is $\Omega(\ln n)$, and many (even very simple) policies achieve this bound.

Perhaps the simplest policy is the **epsilon-greedy policy**, simply choosing the arm with the highest estimated expectation with probability $1 - \varepsilon$ and a uniformly random arm with probability ε. Therefore, the ratio of exploration and exploitation actions is fixed and uniform exploration strategy is applied.

The **Upper Confidence Bound (UCB1) policy** [17] implicitly negotiates the exploration-exploitation dilemma by adding a relative measure of uncertainty *(bias)* to the estimated expectation; therefore, even low-expectation arms are occassionally explored when the uncertainty is too high compared to other arms:

$$\pi_{\mathrm{UCB1}}(n) = \operatorname{argmax}_i \left(\widehat{\mu_i}(n) + c\sqrt{\frac{2\ln n}{T_i(n)}} \right) \tag{1}$$

The policy quickly gained popularity as a reference Multi-Armed Bandit policy since it can be proven that the policy follows the logarithmic regret bound not just asymptotically but also uniformly (after a burn-in period) if the parameter c is tuned for the optimal exploration-exploitation ratio.

2.2 Action Rewards versus Optimization Performance

To apply the Multi-Armed Bandit Problem on algorithm selection, we (analogously to e.g. [14]) represent each algorithm as an arm and in each algorithm iteration decide which arm(s) to step once next. However, the key question is how to represent the reward estimates[1] used for the decision.

[1] In some of the algorithm selection literature, this is termed "credit" of the algorithm.

In the simplest form, the reward estimate $\widehat{\mu}$ may be represented simply by the negative of the **raw value** of the function in the current iteration of the algorithm — therefore, the algorithm currently closest to the optimum will be associated with the highest reward estimate.

This approach may be problematic if the reward needs to be bounded in a fixed interval; a normalization strategy is proposed below in Sec. 2.3. However, when the value does not approach the optimum smoothly, absolute value difference may not correspond well to algorithm performance difference. The approach of **value rank** [13] sidesteps the issue by ranking algorithms based on the values they yield in each round and using that rank (normalized by linear rescaling to $[0, 1]$) as the reward estimate.

An extension of these approaches is, instead of considering just the latest normalized reward, to use an exponentially decaying average of recent normalized rewards with an adaptation rate α [12], also known as the **exponentially weighted moving average (EWMA)**.

The Multi-Armed Bandit Problem assumes that the reward distributions are stationary and rewards are independent. But this is clearly not the case in our setting — as each algorithm proceeds through the functional landscape, its rate of improvement changes and previous results are tied to its future performance. These assumption violations may not be fatal in practice and we test the performance of considered algorithms without regard to them. Furthermore, it has been proven that the UCB1 policy can be used as-is for non-stationary distributions [18] (provided that the c parameter has been set correctly).

2.3 Raw Values and the UCB1 Policy

The UCB1 policy sums the reward estimate $\widehat{\mu}$ with a bias term (multiplied by a fixed constant). A key assumption here is that $\mu \in [0, 1]$ (being a reward expectation), but our raw values are entirely unbounded and exponentially skewed. A simple work-around is to use the value rank instead, but the actual difference between values may be useful during the decision, therefore we also propose a raw value normalization approach.

Every time we invoke the UCB1 policy, we re-normalize values of all arms, taking two assumptions. First, we use values relative to the *supposed optimum* by always assuming that we are just short of it, i.e. putting the $f_{opt?} = \min f - \Delta f$ where $\Delta f = 10^{-8}$ is the target precision of COCO. Secondly, we assume that the algorithms converge exponentially fast[2], therefore differences (relative to the supposed optimum $f_{opt?}$) between 10^3 and 10^2 should be considered on the same scale as differences between 10^{-6} and 10^{-7}.

With these two assumptions, a *log-rescaling* process is straightforward. First, we convert the absolute f_i values to values relative to the supposed optimum and rescale the values logarithmically:

$$g_i = \log(f_i - f_{opt?})$$

[2] A similar idea appears within the MetaMax algorithm [14] and is also supported by practical observations.

Second, we assign rewards by linear rescaling of the preprocessed values:

$$\mu_i = 1 - \frac{g_i - \min_j g_j}{\max_j g_j - \min_j g_j}$$

3 Algorithm Selection Strategies

The strategies and reward schemes outlined above offer a wide variety of possible combinations. To focus the scope of our research, we considered only combinations already proposed in the literature, in addition to a baseline strategy and two new applications of the UCB1 policy we propose. We performed a rough parameter tuning of each of the considered strategies on a portfolio of seven algorithms (see Sec. 4 and [19]); the performance is not very sensitive to exact values of the parameters. Our main loop consist of selecting an algorithm to step, then running it for a single iteration, then repeating the selection etc.

RR: As one baseline strategy, we used a round robin policy that samples each algorithm equally, in their portfolio order. (This is different from a "run in parallel" strategy in that the algorithms consume different budgets to sample a single iteration.)

EG: The epsilon-greedy policy with $\varepsilon = 0.5$.

RUCB: The UCB1 policy with EWMA-recent ranks as reward estimates ($\widehat{\mu_i}$ in Equation 1), with $c = 8$ and adaptation rate $\alpha = 0.9$.

LUCB: The UCB1 policy with EWMA-recent log-rescaled values as reward estimates ($\widehat{\mu_i}$ in Equation 1), with $c = 16$ and adaptation rate $\alpha = 0.7$.

We also tested Probability Matching and Adaptive Pursuit [12], Threshold Ascent [20], MetaMax variants [14] and UCB1 with Sum-of-Ranks and Area-Under-the-Curve rewards [13]. However, their results were not competitive with the strategies above and we cannot elaborate on them due to space limititations[3].

4 Experiments and Results

To benchmark against the BBOB testbed, we used the reference COCO framework [2] [3]. Our "COCOpf" extension [19] provides a common algorithm portfolio codebase, including algorithm stepping and publication reports generation.

We use the reference portfolio of seven optimization algorithms — the CMA-ES algorithm [21] and six numerical optimization algorithms distributed along the SciPy software package [22]. Description and comparison of the individual algorithms is detailed in [19].

Within the COCO framework, functions are classified based on their properties to separable, multi-modal, etc. Here, we deliberately did not adopt this classification as we do not study the behavior of individual algorithms[4], but the behavior of strategies that depend on the performance of portfolio members.

[3] Their results are included in an extended version of this paper, the raw datasets and generated reports at `http://pasky.or.cz/sci/cocopf-opt13`.

[4] See [19] for plots of performance of the portfolio members in the reference COCO function classes.

Table 1. The assignment of individual COCO benchmark functions to the classes we have devised, determined on the performance of our portfolio on the functions. Vertical lines delineate the standard function classes used by COCO.

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
By solvers (s/m)	m	s	s	s	m	s	s	m	m	s	s	s	s	m	s	s	s	s	s	s	s	s	s	m
By winner (g/b)	b	b	b	b	b	g	g	b	b	g	g	g	g	b	b	g	g	g	b	b	g	g	g	b
By converg. (s/v)	s	s	s	s	s	v	s	v	v	v	v	v	v	v	v	s	s	s	v	v	v	v	v	v

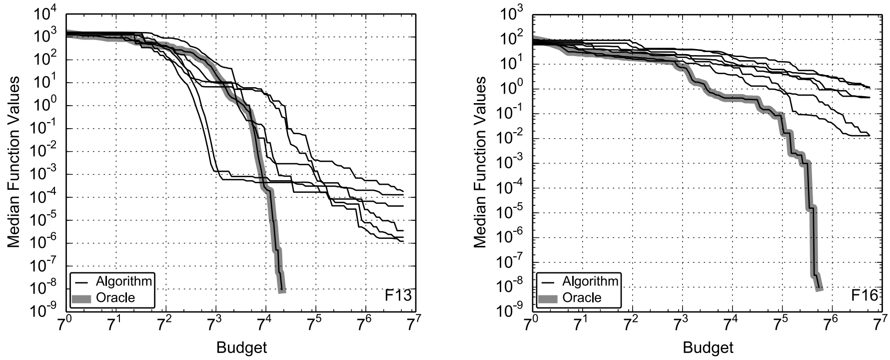

Fig. 1. Typical volatile (left) and stable (right) portfolio convergence. (We distinguish only the portfolio's best algorithm for each considered function.)

With respect to a particular portfolio, we propose the following function classification:

– **By solvers** based on whether a single algorithm dominated others in performance *(single-solver)*, or if multiple algorithms converged similarly quickly *(multi-solver)*. The latter favors strategies that do not focus sharply on a single algorithm. We consider algorithms to perform similarly if one takes at most twice as many evaluations to reach the optimum than another.
– **By winner** as the clear overall winning algorithm in our portfolio is CMA, converging first on most functions *(CMA-good)*, but there are many functions on which CMA actually performs relatively very poorly *(CMA-bad)*. Splitting functions along these lines allows us to quantify the loss caused by using a portfolio instead of a single algorithm.
– **By convergence** based on the algorithm behavior before finding optimum. In some cases, the algorithms that yield the best function values early continue to dominate throughout the convergence progression and eventually indeed converge first — these are the *stable* functions where strategies can early focus on the best algorithms. On the other hand, especially in a landscape rich on difficult-to-escape local optima, the pace of convergence of convential algorithms stays the same or even slows down throughout their run, while an algorithm (often CMA) that produced lukewarm results with smaller

Table 2. The average rank of portfolio algorithms and strategies (computed over both) by the order in which they converge

Solver	all	multi	single	volatile	stable	CMA-good	CMA-bad
CMA	6.4	11.2	4.8	4.7	9.2	1.3	11.4
BFGS	9.4	4.7	11.0	7.5	12.7	12.2	6.6
L-BFGS-B	9.7	3.8	11.7	8.1	12.5	13.0	6.5
SLSQP	10.9	6.0	12.6	9.5	13.2	13.5	8.3
N.-M.	12.0	10.0	12.7	9.9	15.6	12.0	12.0
Powell	12.9	12.9	12.9	15.9	7.9	14.6	11.3
CG	13.3	6.6	15.6	13.0	13.8	15.8	10.8
LUCB	6.5	10.5	**5.1**	7.2	5.3	**4.4**	8.5
RUCB	6.8	8.2	6.3	**6.5**	7.2	6.5	**7.1**
EG	7.1	**7.2**	7.1	8.3	**5.2**	6.6	7.6
RR	9.3	11.0	8.7	9.1	9.7	8.2	10.4

budget suddenly and unexpectedly improves in a dramatic way, achieving convergence early after[5] — we term these *volatile* functions. These are obviously much harder for purely online strategies. We consider a function to be volatile if the algorithm that converges first in budget $|\text{portfolio}|^k$ for some k was not the best algorithm in budget $|\text{portfolio}|^{k-2}$.

We illustrate the typical convergence progression on volatile vs. stable functions in figure 1[6]. Table 1 shows the classification of functions for our portfolio.

In all results, we show measurements with maximum function evaluation budget $\dim \cdot 10^5$ and use the performance on 5D functions as at least one algorithm converges within our budget for almost all functions in this dimension.

Table 2 shows the *average final rank* of each algorithm and portfolio strategy (in terms of budget required for convergence, i.e. 1 is best) averaged over the individual classes. Table 3 shows the *average*[7] *strategy log-slowdowns* in terms of difference between budget b required for certain algorithm or strategy to converge and budget b_o required for the oracle strategy (which runs only the single best algorithm for each function) to converge, computed as $\log_{|\text{portfolio}|}(b/b_o)$. I.e. a slow-down of 0 means perfect performance and slow-down of 1 means that it is as if we simply run all algorithms in parallel[8].

We can observe that the LUCB strategy performs best; while in total average it is superseded by the winner algorithm CMA, that is not very surprising as this algorithm performs best on its own on half of the functions and portfolios will always introduce an overhead. At the same time, the LUCB strategy exhibits

[5] We did not observe a situation where the initially best algorithm temporarily loses its first place only to eventually converge first.

[6] Interested readers may find portfolio convergence graphs for all functions as well as raw datasets at `http://pasky.or.cz/sci/cocopf-opt13`.

[7] Within a single function, the median instance is considered. Across functions within a class, the slowdown is averaged.

[8] Functions on which no algorithm converges in the assigned budget are not included in the average. We assign a log-slowdown of 3 to strategies not converging in time.

Table 3. The average log-slowdown of portfolio algorithms and strategies compared to oracle strategy (i.e. the best algorithm for each function)

Solver	all	multi	single	volatile	stable	CMA-good	CMA-bad
CMA	0.7	1.1	0.6	0.5	1.1	0.0	1.5
CG	2.3	0.8	2.7	2.3	2.3	2.8	1.7
BFGS	1.5	0.7	1.8	1.4	1.7	2.2	0.8
L-BFGS-B	1.9	0.6	2.3	1.8	1.9	2.5	1.1
Nelder-Mead	2.1	0.8	2.6	2.0	2.4	2.5	1.8
SLSQP	2.3	1.0	2.8	2.4	2.3	2.9	1.8
Powell	2.3	2.0	2.4	3.0	1.2	3.0	1.6
LUCB	0.9	0.9	0.9	1.3	**0.3**	1.1	0.8
RUCB	1.3	0.9	1.4	1.4	1.1	1.7	0.8
EG	1.4	1.0	1.6	1.6	1.1	2.0	0.9
RR	1.5	1.1	1.7	1.8	1.2	2.1	1.0

less performance variation (in terms of log-slowdown) from class to class, and on stable functions it outperforms all other strategies *and* algorithms by a large margin. The RUCB and EG strategies can also outdo the LUCB strategy on function-by-function basis in some classes, as is apparent from the average ranks.

5 Discussion and Conclusion

The results demonstrate a good case for the usage of algorithm portfolios for black-box optimization. Overally, our proposed LUCB and RUCB strategies perform the best, but the very simple EG strategy also performed very well. We belive both the LUCB and EG strategies are easy to implement and can be used as reference strategies in further research.

As expected, the portfolios were very beneficial especially in case of non-volatile functions. More work is clearly needed to deal with volatile functions. That will further benefit even non-volatile performance as all our strategies are currently very explorative — investment in even bad-looking algorithms is important in cases of volatile functions. Regardless of function classes, algorithm portfolios also offer a more stable performance than an individual algorithm in the face of unknown (as even CMA can fail miserably on some of the functions).

We can observe a significant stratification among the tested selection strategies. We can conclude that a selection strategy matters, and even a simple strategy like EG will bring a big improvement over a round-robin selection strategy which is still the de-facto standard for algorithm portfolios when they are used.

5.1 Future Work

We hope that performance can be improved by a future portfolio that is more diverse and balanced (either thanks to more algorithms or parameter tuning to refocus different algorithms to specific function classes). Another step in this direction is to study the influence of portfolio size and composition on preformance of various strategies[9].

Clearer measures for adaptation lag when the best algorithm changes in volatile functions or the suitability of using function value differences in strategies would be very desirable.

Many approaches to algorithm selection in terms of Multi-Armed Bandit Policies and reward assignment were proposed in the literature. We could not consider them all, but we think that especially performance-modeling approaches like modifications of the MultiEA [7] or GambleTA [23] algorithms are worth investigating in the future.

Aside of that, we attempted to give the problem a modular structure; this allows e.g. a full-scale comparison of reward assignment and bandit policy combinations on top of what has been proposed in the literature so far. Furthermore, the usage of UCB1 is not theoretically very sound and we assume it should be possible to develop a more suitable policy formula.

We have investigated just a purely online, black-box mode of action so far, but there is certainly a room to grow in the direction of previously introduced approaches. Offline learning can be combined with online methods at least to initialize them or detect function classes. Intermediate solutions could be shared and migrated between individual algorithms.

Acknowledgements. This research was supported by the CTU grant SGS14 /194/OHK3/3T/13 "Automatic adaptation of search algorithms". We would like to thank the Department of Applied Mathematics of the Charles University in Prague for providing the computational resources that made our experiments possible. Matteo Gagliolo's introduction to the research landscape helped us much in the beginning of our work.

References

1. Nelder, J.A., Mead, R.: A simplex method for function minimization. The Computer Journal 7(4), 308–313 (1965)
2. Hansen, N., et al.: Comparing continuous optimisers: Coco, `http://coco.gforge.inria.fr/`
3. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking 2012: Experimental setup. Technical report, INRIA (2012)
4. Rice, J.R.: The algorithm selection problem. Advances in Computers 15, 65–118 (1976)
5. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. AI Magazine (2014)
6. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1), 43–62 (2001)

[9] For example, it surprised us that it seems trimming the current portfolio by worst performing algorithms does not improve overall performance. However, we still consider this result preliminary.

7. Yuen, S.Y., Chow, C.K., Zhang, X.: Which algorithm should i choose at any point of the search: An evolutionary portfolio approach. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. GECCO 2013, pp. 567–574. ACM, New York (2013)
8. Vrugt, J.A., Robinson, B.A., Hyman, J.M.: Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Trans. on Evolutionary Computation 13(2), 243–259 (2009)
9. Peng, F., Tang, K., Chen, G., Yao, X.: Population-based algorithm portfolios for numerical optimization. IEEE Transactions on Evolutionary Computation 14(5), 782–800 (2010)
10. Muñoz, M.A., Kirley, M., Halgamuge, S.K.: The algorithm selection problem on the continuous optimization domain. In: Moewes, C., Nürnberger, A. (eds.) Computational Intelligence in Intelligent Data Analysis. SCI, vol. 445, pp. 75–89. Springer, Heidelberg (2013)
11. Bischl, B., Mersmann, O., Trautmann, H., Preuss, M.: Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference, pp. 313–320. ACM (2012)
12. Thierens, D.: Adaptive strategies for operator allocation. In: Lobo, F.G., Lima, C.F., Michalewicz, Z. (eds.) Parameter Setting in Evolutionary Algorithms. SCI, vol. 54, pp. 77–90. Springer, Heidelberg (2007)
13. Fialho, Á., Schoenauer, M., Sebag, M.: Toward comparison-based adaptive operator selection. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 767–774. ACM (2010)
14. György, A., Kocsis, L.: Efficient multi-start strategies for local search algorithms. J. Artif. Int. Res. 41(2), 407–444 (2011)
15. Robbins, H.: Some aspects of the sequential design of experiments. Bulletin of the American Mathematics Society 58, 527–535 (1952)
16. Lai, T.L., Robbins, H.: Asymptotically efficient adaptive allocation rules. Advances in Applied Mathematics 6, 4–22 (1985)
17. Auer, P., Bianchi, N.C., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning 47(2/3), 235–256 (2002)
18. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
19. Baudiš, P.: COCOpf: An algorithm portfolio framework. In: Poster 2014 — the 18th International Student Conference on Electrical Engineering, Czech Technical University, Prague, Czech Republic (2013)
20. Streeter, M.J., Smith, S.F.: A simple distribution-free approach to the max k-armed bandit problem. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 560–574. Springer, Heidelberg (2006)
21. Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J., Larranaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. STUDFUZZ, vol. 192, pp. 75–102. Springer, Heidelberg (2006)
22. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001)
23. Gagliolo, M., Schmidhuber, J.: Algorithm portfolio selection as a bandit problem with unbounded losses. Annals of Mathematics and Artificial Intelligence 61(2), 49–86 (2011)