Runtime Analysis of a Population-based Evolutionary Algorithm with Auxiliary Objectives Selected by Reinforcement Learning

Denis Antipov
ITMO University
Saint-Petersburg, Russia
antipovden@yandex.ru

Arina Buzdalova
ITMO University
Saint-Petersburg, Russia
abuzdalova@gmail.com

Andrew Stankevich
ITMO University
Saint-Petersburg, Russia
stankev@gmail.com

ABSTRACT

We propose the method of selection of auxiliary objectives $(2 + 2\lambda)$-EA+RL which is the population-based modification of the EA+RL method. We analyse the efficiency of this method on the problem XdivK that is considered to be a hard problem for random search heuristics due to multiple plateaus. We prove that in the case of presence of a helping auxiliary objective this method can find the optimum in $O(n^2)$ fitness evaluations in expectation, while the initial EA+RL, which is not population-based, yields at least $\Omega(n^{k-1})$ fitness evaluations, where k is the plateau size. We also prove that in the case of presence of an obstructive auxiliary objective the expected runtime increases only by a constant factor.

CCS CONCEPTS

• **Theory of computation** → **Theory of randomized search heuristics**; • **Computing methodologies** → Control methods;

KEYWORDS

Runtime analysis, multiobjectivization, reinforcement learning

ACM Reference Format:
Denis Antipov, Arina Buzdalova, and Andrew Stankevich. 2018. Runtime Analysis of a Population-based Evolutionary Algorithm with Auxiliary Objectives Selected by Reinforcement Learning. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3205651.3208231

1 INTRODUCTION

Single-objective optimization can often be accelerated by the introduction of auxiliary objectives [1, 9]. There are plenty of ways how auxiliary objectives can be used [11]. In multiobjectivization [5, 6], auxiliary objectives are used to help the optimizers pass through plateaus of the target objective or escape its local optima.

Auxiliary objectives can be obtained in different ways, e.g. they can arise from decomposition of the target objective [1, 6, 8] or be automatically generated in the hope that they reflect some properties of the problem [3]. However, in the latter case it is a challenging task to learn, which of the objectives accelerate the optimization and which ones slow it down. Moreover, some objectives may change their helpfulness during the optimization, so it is important to dynamically learn, which objective is the most useful at every moment. It caused the uprise of methods of dynamic selection of auxiliary objectives [5, 7]. One of such methods is EA+RL [3].

The essence of the EA+RL method is in the interaction of an optimizer, that is usually an evolutionary algorithm (EA), and a reinforcement learning agent that chooses the objectives to be optimized [3]. The efficiency of this method has been proven both empirically [3] and theoretically [2, 10]. While most empirical studies of EA+RL consider various EAs as the optimizer, in theoretical works only simple non-populational EAs are analysed. This lack of theoretical research increases the risk that population-based heuristics are used in EA+RL without real understanding of their role.

In our work we show that using of a population of large size may decrease the asymptotic runtime of EA+RL method when there is at least one helpful auxiliary objective.

2 PROPOSED MODIFICATION OF EA+RL METHOD

The detailed pseudocode of the proposed $(2 + 2\lambda)$-EA+RL method is shown in Algorithm 1.

The EA+RL method consists of two iteratively interacting components. The first component is a learning agent that chooses an auxiliary objective on each iteration. The second component is an EA that receives an auxiliary objective from the learning agent, performs one iteration with respect to the chosen objective and calculates the reward r and the new state of the algorithm s. The learning agent uses the values of r and s to update its selection strategy. We define a state as the value of the target objective calculated on the best individual in the population (see Line 5 in Algorithm 1). The reward is defined as the difference in the sum of the target objective values over the population before and after the iteration (Line 15).

As the optimizer we propose the $(2 + 2\lambda)$-EA that has two individuals in the population and on each iteration creates λ offspring from each individual in the population by mutating one random bit (Line 8). After creating offspring, the algorithm selects two individuals to the next generation from the parents and the offspring. First it selects one of the individuals with the best value of auxiliary objective uniformly at random (u.a.r. for brevity, see Line 11). Note that the target objective may be selected as an auxiliary objective as well. Then the algorithm selects one of the rest individuals with the best value of target objective u.a.r. (Line 13).

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208231

Algorithm 1 The $(2 + 2\lambda)$-EA+RL with forgetting mechanism.

1: Individuals $x_1, x_2 \leftarrow$ two random bit strings
2: Population $Pop \leftarrow \{x_1, x_2\}$
3: $H \leftarrow$ {auxiliary objectives, target objective f}
4: $Q(s, h) \leftarrow 0$ for all states s and objectives $h \in H$
5: $s \leftarrow \max_{i\in[1..2]}(f(x_i))$
6: **while** $s < n$ div k **do**
7: **for** $i \in [1..2]$ and $j \in [1..\lambda]$ **do**
8: $Pop \leftarrow Pop \cup$ Mutate(x_i) (random bit inversion)
9: **end for**
10: Objective $h \leftarrow$ RandomElement$(\arg\max_{h'\in H} Q(s, h'))$
11: $x'_1 \leftarrow$ RandomElement$(\arg\max_{x\in Pop} h(x))$
12: $Pop \leftarrow Pop \setminus x'_1$
13: $x'_2 \leftarrow$ RandomElement$(\arg\max_{x\in Pop} f(x))$
14: New state $s' \leftarrow \max_{i\in[1..2]} f(x'_i)$
15: Reward $r \leftarrow (f(x'_1) + f(x'_2)) - (f(x_1) + f(x_2))$
16: **if** for C iterations: h was chosen & $r = 0$ & s has not changed **then**
17: $Q(s, h) \leftarrow 0$
18: **else**
19: $Q(s, h) \leftarrow (1 - \alpha)Q(s, h) + \alpha(r + \gamma \max_{h'\in H} Q(s', h'))$
20: **end if**
21: New generation $Pop \leftarrow \{x'_1, x'_2\}$
22: $s \leftarrow s'$
23: **end while**

The previous theoretically analysed versions of the EA+RL method employed only one parent and one child, and in all the previous versions the search was guided only by the selected objective. In the proposed method, the selection of two individuals by different objectives is the feature that became possible due to the two individuals in the population instead of one. It preserves the best individual as in [10], but also lets the algorithm to explore the search space with the second individual.

The learning agent in the proposed method uses Q-learning algorithm that stores the quality $Q(s, h)$ of each objective h for each state of the algorithm s. On each step it chooses one objective from the objectives with the largest Q-value in the current state of the algorithm u.a.r. After EA sends the reward to the learning agent, Q-learning updates the quality of the selected objective as in Line 19. Q-learning has two parameters α and γ. Their values are taken from the interval $(0, 1)$.

The proposed method also has a forgetting mechanism. If the learning agent chooses the same objective h for C iterations in a row and during these iterations the algorithm has not changed its state, then the quality of these objectives falls to zero (Line 17). C is a parameter of the algorithm that is considered as a constant. Note that if $r = 0$ and s does not change, then the quality $Q(s, h)$ is multiplied by $(1 - \alpha + \gamma\alpha) < 1$. Thus due to the precision of floating point arithmetics if the same objective with positive quality is chosen for many times and the reward is always zero, at some point its quality becomes zero. So the forgetting mechanism is actually just determinization of this process to simplify the analysis.

3 PROBLEM STATEMENT

We consider three objectives. The first objective is OneMax that is a classical function for theoretical analysis defined on all the bit-strings of length n. OneMax(x) is the number of one-bits in x, so it reaches its maximum value in $x^* = (1, \dots, 1)$.

The second objective is ZeroMax(x) that returns the number of zero-bits in x. The third objective we consider is XdivK that is usually defined as XdivK(x) = OneMax(x) div k. However, in our work we consider the modified version XdivK(x) = (OneMax(x) + $k - n \bmod k$) div k., so that this function has the unique optimum in $x^* = (1, \dots, 1)$. In the rest of the paper by XdivK we imply the modified version of this function.

The two research questions in our work are (i) whether the proposed population-based EA+RL method can solve hard problems faster in the case of presence of helpful objectives and (ii) whether obstructive objectives may significantly decrease its efficiency. To answer the first question, we consider (XdivK, OneMax) problem, where XdivK is the target objective and OneMax is an auxiliary objective. XdivK has a lot of plateaus where the algorithm does not receive any feedback from the objective. To increase the value of XdivK the optimizer has to flip up to k zero-bits. All these make XdivK hard to be optimized by EAs. At the same time, OneMax has the same optimum as XdivK, and OneMax is a linear function, so it is easier to optimize, and it is supposed to be a helpful objective. To answer the second question we added ZeroMax as an auxiliary objective. This objective is counter directed with XdivK: any improvement of XdivK leads to decrease in ZeroMax. Therefore, ZeroMax is considered as an obstructive objective.

To evaluate the efficiency of the algorithm we estimate its expected runtime. In this work by runtime we imply the number of iterations that algorithm performs before finding the optimum. However, we are also interested in the number of fitness evaluations that can be computed as the number of iterations multiplied by 2λ, which is the number of fitness evaluations per iteration.

4 THEORETICAL ANALYSIS

To find the expected runtime of the algorithm we use the fitness levels technique [12]. We first find how many iterations the $(2+2\lambda)$-EA+RL spends on each plateau in expectation. Then we sum up these values for all plateaus.

4.1 Notation

By plateau P_j we imply the set of all possible bit-strings with at least j one-bits, but less than $j + k$ one-bits. For this notation we use only j such that $(n - j) \mod k = 0$, so the value of XdivK is the same for all individuals in P_j.

To simplify our proofs we use the following notation. We say that *the algorithm enters (or reaches) the plateau* P_j, when the individual from P_j occurs in the population for the first time. *The time that the algorithm spends on the plateau* P_j is the number of iterations since the algorithm enters P_j until it reaches the next plateau P_{j+k}.

Pull-up is the iteration that starts with a population of two individuals from different plateaus and finishes with two individuals from the same plateau. We say that the algorithm *pulls up* during this iteration. We call it a pull-up, since the algorithm always accepts an individual with the best known value of the target objective. It implies that if the two individuals from the population are from plateaus P_a and P_b, then after the pull up they both will be on the plateau P_c, where $c \geq \max(a, b)$.

4.2 Analysis on (XdivK, OneMax) Problem

Theorem 4.1. *The expected runtime of $(2+2\lambda)$-EA+RL optimizing (XdivK, OneMax) problem is*

$$E[T] \le \sum_{m=0}^{n/k-1} \Big((1-(1-(n-mk)/n)^{\lambda})^{-1} + \\ + 2^{k-1} \prod_{i=mk}^{mk+k-1} \left(1-\left(1-\frac{n-i}{n}\right)^{\lambda}\right)^{-1}\Big) + n + \frac{Cn}{k}.$$

To prove this theorem we first have to prove Lemma 4.2.

Lemma 4.2. *If both individuals in the population are in P_j and the auxiliary objective is chosen u.a.r. from XdivK and OneMax, then the expected number of iterations that $(2+2\lambda)$-EA+RL spends on P_j is*

$$E[T_j] \le 2^{k-1} \prod_{i=j}^{j+k-1} \left(1-\left(1-\frac{n-i}{n}\right)^{\lambda}\right)^{-1} + k.$$

Proof. First notice that an individual from the previous plateau cannot appear in the population, as this individual is inferior in both objectives to the individuals in the population. This fact implies that the sum of the target objective values over the population does not change until the algorithm reaches the next plateau. Therefore, the learning agent does not receive any reward and proceed to choose objectives u.a.r.

We define *an improvement* as an iteration when the algorithm generates and accepts an individual with more one-bits than in the best individual in the current population. At every moment the algorithm can perform a series of not more than k improvements to reach the next plateau, since flipping k zero-bits increases the value of XdivK. The probability of such series $p(j)$ is not less than the product of the probabilities of k consequent improvements. When the best individual has exactly i one-bits, then the probability of the improvement is not less than the probability to create an individual with more one-bits $p_1(i) = (1-(1-(n-i)/n)^{\lambda})$ multiplied by the probability to accept this individual to the next population $p_2(i)$. If OneMax is chosen by the learning agent, then the individual with maximal number of one-bits will be accepted to the next population for sure, so $p_2(i) \ge 1/2$. Moreover, if $i = j+k-1$, then the individual with more one-bits will be accepted regardless to the chosen objective. Therefore,

$$p(j) \ge \frac{1}{2^{k-1}} \prod_{i=j}^{j+k-1} \left(1-\left(1-\frac{n-i}{n}\right)^{\lambda}\right).$$

The algorithm starts the series of improvements not later than after $1/p(j)$ iterations in expectation. After that it needs no more than k iterations to reach the next plateau. Hence,

$$E[T_j] \le 2^{k-1} \prod_{i=j}^{j+k-1} \left(1-\left(1-\frac{n-i}{n}\right)^{\lambda}\right)^{-1} + k. \qquad (1)$$

□

Now we are ready to prove Theorem 4.1

Theorem 4.1. Let T_j be the time that $(2+2\lambda)$-EA+RL spends on P_j.

If after the algorithm enters P_j both individuals in the population are in P_j, then the algorithm occurs in the conditions of Lemma 4.2. Hence, in this case the runtime is bounded as in (1).

Otherwise, after entering the plateau the population consists of one individual from P_j and one individual from the previous plateau. So the algorithm has to pull up. If the algorithm generates a superior offspring of the best individual, both this offspring and the best individual are better than any individual from the previous plateau. Hence, they both will be accepted to the next population. Thus, the probability of the pull-up is not less than $p_1(j) \ge (1-(1-(n-j)/n)^{\lambda})$. Therefore, the expected number of iterations before the pull-up is not greater than $(1-(1-(n-j)/n)^{\lambda})^{-1}$. After the pull-up, the sum of the target objective values over the population increases and the learning agent starts to choose the rewarded objective. In this case either the algorithm reaches the next plateau in C iterations, or after C iterations the forgetting mechanism is triggered, and the algorithm occurs in conditions of Lemma 4.2.

Therefore, the most pessimistic bound on the expected time that algorithm spends on P_j is

$$E[T_j] \le (1-(1-(n-j)/n)^{\lambda})^{-1} + C \\ + 2^{k-1} \prod_{i=j}^{j+k-1} \left(1-\left(1-\frac{n-i}{n}\right)^{\lambda}\right)^{-1} + k. \qquad (2)$$

Summing up $E[T_j]$ over all plateaus we prove the theorem. □

If $\lambda = 1$ we have expected runtime that is $O(n^k)$. However, the bound decreases with growth of λ. If $\lambda = n$, then $(1-(n-i)/n)^{\lambda} \le 1/e$, thus

$$E[T] \le \frac{n}{k}\left(1-\frac{1}{e}\right)^{-1} + 2^{k-1}\frac{n}{k}\left(1-\frac{1}{e}\right)^{-k} + n + \frac{Cn}{k} = O(n).$$

If $\lambda = n$, then the expected number of fitness evaluations is n times the expected number of iterations, that is $O(n^2)$. Even besides the leading constant is large, this result is significant, as it is the first modification of the EA+RL method that provenly needs asymptotically less than n^k fitness evaluations in expectation to solve XdivK problem. The previous result was $\Omega(n^{k-1})$ for both EA+RL and Random Local Search [2].

4.3 Analysis on (XdivK, OneMax, ZeroMax) Problem

The main result of this section is Theorem 4.3.

Theorem 4.3. *The expected runtime of $(2+2\lambda)$-EA+RL optimizing (XdivK, OneMax, ZeroMax) problem is*

$$E[T] \le \sum_{m=0}^{n/k-1} \Big(\frac{5}{3}(1-(1-(n-mk)/n)^{\lambda})^{-1} \\ + (3^k/2 + 2^{k-1}) \prod_{i=mk}^{mk+k-1} \left(1-\left(1-\frac{n-i}{n}\right)^{\lambda}\right)^{-1}\Big) + 2n + \frac{2Cn}{k}.$$

Overall, analysis of the algorithm on (XdivK, OneMax, ZeroMax) problem replicates the ideas of the analysis on (XdivK, OneMax) problem. However, because of the presence of ZeroMax the

algorithm can *fall from plateau* P_j, that is, having both individuals in P_j it can accept an individual from the previous plateau to the next population. This results into Lemma 4.4.

Lemma 4.4. *If both individuals in the population are in P_j and the auxiliary objective is chosen u.a.r. from XdivK, OneMax and ZeroMax, then the expected number of iterations that $(2+2\lambda)$-EA+RL spends on P_j is*

$$E[T_j] \le (3^k/2 + 2^{k-1}) \prod_{i=j}^{j+k-1} \left(1 - \left(1 - \frac{n-i}{n}\right)^{\lambda}\right)^{-1} + 2k + C + \left(1 - \left(1 - \frac{n-j}{n}\right)^{\lambda}\right)^{-1}.$$

Proof. Before reaching the next plateau the algorithm can go through up to two phases. In the first phase it selects objectives u.a.r. The first phase ends when the algorithm receives a non-zero reward. It happens when the algorithm either reaches the next plateau or if an individual from the previous plateau is accepted to the population. In the second case the second phase starts.

To finish the first phase the algorithm can perform a series of k improvements, as in Lemma 4.2. If OneMax is chosen (or XdivK is chosen in the end of the plateau), then a better individual will surely be accepted into the next population. The probability to choose one particular objective is $1/3$. So the probability of k improvements is

$$p(j) \ge \frac{2}{3^k} \prod_{i=j}^{j+k-1} \left(1 - \left(1 - \frac{n-i}{n}\right)^{\lambda}\right).$$

Therefore, the first phase takes not more than $1/p(j) + k$ iterations in expectation.

The second phase starts only if the individual from the previous plateau is accepted to the population. However, such individual can be accepted only if the learning agent selects ZeroMax. Thus, on the second phase ZeroMax is not selected by the learning agent, since it has received a negative reward. Hence, the algorithm is in the same situation as in the worst case in Theorem 4.1. Therefore, the expected runtime of the second phase is the same as in (2).

Summing up the expected runtime of the two phases we prove the lemma.

□

Now we are ready to prove Theorem 4.3.

Theorem 4.3. Like in Theorem 4.1, if after entering the plateau both individuals in the population are in P_j, then the auxiliary objectives are selected u.a.r. at least on the next iteration. Therefore, the algorithm is in conditions of Lemma 4.4

If after entering the plateau only one individual in the population is in P_j, then the algorithm has to pull up. After that, in the worst-case scenario it spends C iterations on the plateau, triggers the forgetting mechanism and gets into the conditions of Lemma 4.4. The difference with Theorem 4.1 is that the probability of pull-up should be multiplied by $\frac{2}{3}$, as it can occure only if ZeroMax was not selected.

Summing up the expected runtime of the worst-case scenario over all the plateaus we prove the theorem. □

Considering particular values of λ we can see that for $\lambda = 1$ the expected runtime is $O(n^k)$. However, for $\lambda = n$ we have $1 - (1 - (n-i)/n)^{\lambda} \ge 1 - 1/e$. Therefore,

$$E[T] \le \frac{5n}{3k}\left(\frac{e}{e-1}\right) + \frac{(3^k + 2^k)n}{2k}\left(\frac{e}{e-1}\right)^k + 2n + \frac{2Cn}{k} = O(n).$$

And the expected number of fitness evaluations is $O(n^2)$.

5 CONCLUSION

We proposed and analysed the $(2 + 2\lambda)$-EA+RL, a populational version of the EA+RL method, which selects auxiliary objectives in EAs with reinforcement learning. Compared to the previous versions of EA+RL, the proposed method solves the XdivK problem asymptotically faster in the presence of a helpful auxiliary objective. Inclusion of an obstructive objective increases the runtime only by a constant factor.

Since the choice of λ significantly affects the runtime, in the future it may be interesting to analyse different parameter control techniques applied to the parameter λ in the $(2+2\lambda)$-EA+RL. We feel optimistic that even simple 1/5-th rule may give an asymptotical improvement, like it did, e.g., in [4].

This work was partially financially supported by the Government of Russian Federation (Grant 08-08), and by RFBR according to the research project No. 16-31-00380 mol_a.

REFERENCES

[1] Dimo Brockhoff, Tobias Friedrich, Nils Hebbinghaus, Christian Klein, Frank Neumann, and Eckart Zitzler. 2009. On the Effects of Adding Objectives to Plateau Functions. *IEEE Transactions on Evolutionary Computation* 13, 3 (2009), 591–603.

[2] Maxim Buzdalov and Arina Buzdalova. 2014. OneMax Helps Optimizing XdivK: Theoretical Runtime Analysis for RLS and EA+RL. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*. ACM, 201–202.

[3] Maxim Buzdalov, Arina Buzdalova, and Irina Petrova. 2013. Generation of Tests for Programming Challenge Tasks Using Multi-Objective Optimization. In *Proceedings of Genetic and Evolutionary Computation Conference Companion*. ACM, 1655–1658.

[4] Benjamin Doerr and Carola Doerr. 2015. Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings. In *Proceedings of Genetic and Evolutionary Computation Conference*. 1335–1342.

[5] Mikkel T. Jensen. 2004. Helper-Objectives: Using Multi-Objective Evolutionary Algorithms for Single-Objective Optimisation: Evolutionary Computation Combinatorial Optimization. *Journal of Mathematical Modelling and Algorithms* 3, 4 (2004), 323–347.

[6] J. D. Knowles, R. A. Watson, and D. Corne. 2001. Reducing Local Optima in Single-Objective Problems by Multi-objectivization. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*. Springer-Verlag, 269–283.

[7] Darrell F. Lochtefeld and Frank W. Ciarallo. 2011. Helper-Objective Optimization Strategies for the Job-Shop Scheduling Problem. *Applied Soft Computing* 11, 6 (2011), 4161–4174.

[8] Darrell F. Lochtefeld and Frank W. Ciarallo. 2015. Multi-objectivization Via Decomposition: An analysis of helper-objectives and complete decomposition. *European Journal of Operational Research* 243, 2 (2015), 395–404.

[9] Frank Neumann and Ingo Wegener. 2008. Can Single-Objective Optimization Profit from Multiobjective Optimization? In *Multiobjective Problem Solving from Nature*. Springer Berlin Heidelberg, 115–130.

[10] I. Petrova and A. Buzdalova. 2017. Reinforcement learning based dynamic selection of auxiliary objectives with preservation of the best found solution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1435–1438.

[11] C. Segura, C. A. C. Coello, G. Miranda, and C. Léon. 2013. Using multi-objective evolutionary algorithms for single-objective optimization. *4OR* 3, 11 (2013), 201–228.

[12] Ingo Wegener. 2001. Theoretical Aspects of Evolutionary Algorithms. In *Proceedings of Automata, Languages and Programming, ICALP '01 (Lecture Notes in Computer Science)*, Vol. 2076. Springer, 64–78.