2013 12th International Conference on Machine Learning and Applications

A First Step towards the Runtime Analysis of Evolutionary Algorithm Adjusted with Reinforcement Learning

Maxim Buzdalov, Arina Buzdalova, Anatoly Shalyto
Saint-Petersburg National Research University
of Information Technologies, Mechanics and Optics
49 Kronverkskiy prosp.
Saint-Petersburg, Russia, 197101
Email: {mbuzdalov, abuzdalova, anatoly.shalyto}@gmail.com

***Abstract*—A first step towards analyzing runtime complexity of an evolutionary algorithm adaptively adjusted using reinforcement learning is made. We analyze the previously proposed EA + RL method that enhances single-objective optimization by selecting efficient auxiliary fitness functions. Precisely, Random Mutation Hill Climber adjusted with Q-learning using greedy exploration strategy is considered. We obtain both lower and upper bound for the number of fitness function evaluations needed for this EA + RL implementation to solve a modified OneMax problem. It turns out that EA + RL with an inefficient auxiliary fitness function performs on par with a conventional evolutionary algorithm, namely in $\Theta(N \log N)$ fitness function evaluations, where N is the size of the OneMax problem. In other words, we show that reinforcement learning successfully ignores inefficient fitness function. A lower bound for the ε-greedy exploration strategy for $\varepsilon > 0$ is analyzed as well.**

I. Introduction

The EA + RL method was previously proposed by the authors of this paper for adaptive selection of auxiliary fitness functions in single-objective evolutionary algorithms [1], [2]. The selection is performed during the running time of an evolutionary algorithm (EA) using reinforcement learning (RL). The auxiliary fitness functions can arise during multi-objectivization by decomposition [3]–[5], or can be taken from the application domain [6]–[8]. EA + RL also can be used to dynamically select helper-objectives [9], [10] in multi-objective evolutionary algorithms [6], [7].

The method was empirically shown to be efficient in solving a number of model problems [1], [2], [11], [12], as well as in applying to a real-world problem [6], [7]. However, there are no theoretical foundations of this method yet. In this paper we present the first theoretical result considering the runtime analysis of EA + RL applied to a modified OneMax problem.

As long as we know, there are several other techniques of adjusting evolutionary algorithms with reinforcement learning in different ways (i. e. choosing evolutionary operators or adjusting real valued parameters, such as mutation rate) [13]–[16], but there is no runtime analysis of these methods as well. In reinforcement learning field itself, analyzing computational complexity is an open problem [17]. There are works on convergence and convergence rates of reinforcement learning algorithms [18]–[23], mostly of Q-learning for some special problems. There is also a huge work on analyzing computational complexity of evolutionary algorithms [24]. But even if we know the runtime complexity of a reinforcement learning algorithm and an evolutionary one, the runtime complexity of this evolutionary algorithm controlled by reinforcement learning is still an open question. In this article, a first step towards analyzing such combination of machine learning algorithms is made.

II. Previous Results and Motivation

In this section we describe the EA + RL method, consider an example of previous empirical results and give motivation for the formulation of the theoretical result.

A. EA + RL Method

Firstly, let us describe the concepts of EA + RL in more detail [1]. Consider a single-objective evolutionary algorithm. Let us call the fitness function used in this algorithm the *target* fitness function. EA + RL method is used to maximize the target fitness function in less number of generations than the initial evolutionary algorithm does.

For each population of the evolutionary algorithm, reinforcement learning agent selects the most efficient fitness function from a set, which consists of the target fitness function and some *auxiliary* ones. So we use the auxiliary fitness functions to enhance optimization of the target fitness function. Generally, there is no prior knowledge about these functions. Some of them can be *supporting*, in other words, selecting these functions speeds up the target fitness function optimization. Others can be *obstructive*, which means that selecting these functions slows down the target fitness function optimization.

B. Previous Results Example

Consider the following example. The EA + RL method was applied to generation of tests against solutions of programming challenge tasks [6], [7], [25]. Here the target fitness function

978-0-7695-5144-9/13 $31.00 © 2013 IEEE
DOI 10.1109/ICMLA.2013.42

was the running time of the solution, since our goal was to generate performance tests, which made the solution exceed some predefined time limit. But this fitness function was inefficient, because it is noisy, quantified and platform dependent. So we implemented some auxiliary fitness functions, namely, some counters placed in the solution code, that correlate with the running time of the solution.

The results of the corresponding experiment is shown in the Table I [6]. T stands for the target fitness function, Q, I, L are the auxiliary ones. We used a genetic algorithm with each of these functions and we also applied the EA + RL method to this algorithm. The run is called *successful*, if a test that makes the solution to exceed the time limit is generated.

TABLE I
RESULTS EXAMPLE

Algorithm	FFs	Success, %	Generations	
			Mean	σ
GA	Q	95	3815	3466
GA + RL	all	80	5817	6160
GA	I	54	12669	12873
GA	L	51	13755	14082
GA	T	0	–	–

Using the target fitness function turned to be completely inefficient, while using the Q fitness function provided with good tests in 95 % of runs. So the Q fitness function is a supporting one. The rest of fitness functions gave moderate results. However, the key idea of using EA + RL is that we do not want to test each auxiliary fitness function by performing a separate run of an evolutionary algorithm. We just take EA + RL, pass all the functions to it, and it selects the most efficient ones automatically during the single run of an evolutionary algorithm. According to the Table I, the EA + RL was successful in 80% of runs. It manages to select the efficient fitness functions and allows not to test all of them separately.

C. Motivation

Generally, we are interested in proving that EA + RL enhances the runtime complexity of the adjusted evolutionary algorithm if there is at least one supporting fitness function. It is also important to prove that EA + RL performs not worse than an evolutionary algorithm even if there are only obstructive ones. In this paper we prove that EA + RL performs equally well with the adjusted evolutionary algorithm if there is only one auxiliary fitness function and it is an obstructive one. In other words, we illustrate on a model problem that EA + RL manages to eliminate influence of an obstructive fitness function.

III. MODEL PROBLEM AND ALGORITHM

In this section we consider a model problem and an implementation of EA + RL method used to solve it. Later we will prove the runtime estimation for this algorithm.

We modify OneMax problem to get a problem with one target fitness function and one obstructive fitness function. Our goal is to prove that EA + RL method allows to eliminate influence of an obstructive fitness function. In other words, we will later show for the considered example that EA + RL implementation with obstructive fitness function performs as good as a conventional algorithm without an obstructive fitness function.

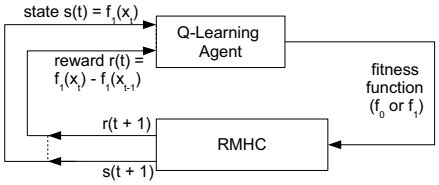

Fig. 1. RMHC + Q-learning scheme

A. Modified OneMax Problem

Consider a modified OneMax Problem. There are two fitness functions: the obstructive fitness function f_0 and the target fitness function f_1. The target fitness function f_1 is calculated as number of 1-bits in an individual of length N, while the obstructive fitness function f_0 is calculated as number of 0-bits, so optimizing f_0 decreases f_1.

B. RMHC + Q-learning Algorithm

Consider the following EA + RL implementation. We use the Random Mutation Hill Climber (RMHC) algorithm [5] for the EA and the Q-learning algorithm [26] for the RL algorithm. The scheme of the resulting algorithm is shown in Fig. 1

In this EA + RL implementation an environment state equals to the number of 1-bits in the current individual generated with RMHC. So there are $N + 1$ possible states from s_0 to s_N.

The immediate reward r equals to the difference of the target fitness function values calculated on the two individuals, which are consequently generated with RMHC. The Q-learning algorithm is designed to maximize the total reward: $E[\sum_{t=0}^{\infty} \gamma^t r_t] \rightarrow max$, where γ is a real valued parameter [26].

The efficiency of choosing some fitness function f in the state s is denoted by $Q(s, f)$. We use the greedy exploration strategy [26], [27], which means that in the state s the fitness function with the highest $Q(s, f)$ is chosen.

The pseudocode of the considered algorithm is shown in the Listing 1.

IV. MAIN RESULT

Theorem 1. *Random Mutation Hill Climber controlled with Q-learning algorithm with greedy exploration strategy solves modified OneMax problem in $\Theta(N \log N)$ fitness function calls.*

Proof: The outline of the proof is as follows. First, the Learning Lemma is formulated and proved. Using this lemma, we construct a Markov chain of the optimization process performed with the RMHC + Q-learning algorithm. Then we estimate lengths of some loops in this chain, which allows us

Algorithm 1 RMHC + Q-learning Algorithm

$X \leftarrow$ current individual, vector of N zeros
$Q \leftarrow$ transition quality matrix, $N \times 2$, filled with zeros
$f_1 \leftarrow$ target function: number of ones in an individual
$f_0 \leftarrow$ obstructive function: number of zeros in an individual
$Mutate(X) \leftarrow$ mutation operator: inverts random bit
$\alpha \in (0;1)$, $\gamma \in (0;1)$ — Q-learning parameters
while $f_1(X) < N$ **do**
  $s \leftarrow f_1(X)$
  $Y \leftarrow Mutate(X)$
  f, I: chosen fitness function and its index
  if $Q(s,0) > Q(s,1)$ **then**
    $f \leftarrow f_0$, $I \leftarrow 0$
  else if $Q(s,0) < Q(s,1)$ **then**
    $f \leftarrow f_1$, $I \leftarrow 1$
  else
    $I \leftarrow$ random(0,1), $f \leftarrow f_I$
  end if
  if $f(Y) \geq f(X)$ **then**
    $X \leftarrow Y$
  end if
  $r \leftarrow f_1(X) - s$
  $Q(s,f) \leftarrow (1-\alpha)Q(s,f) + \alpha(\gamma \cdot r + \max_j Q(s,f_j))$
end while

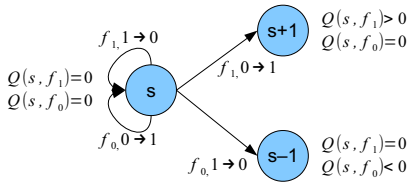

Fig. 2. Transitions from a state visited for the first time

to replace this chain with the linearized one. Using the linear chain, we calculate the runtime of the algorithm. Finally, we compare the obtained runtime estimation with the runtime of the conventional Random Mutation Hill Climber and get the main result.

A. Learning Lemma

First of all, let us formulate the lemma, which will be used to construct the Markov chain for the considered algorithm.

Lemma 1. *Assume that the Q-learning agent visits a state s and leaves it. Then the optimal fitness function f_1 will be chosen in s in all next visits.*

Proof: Consider a state s visited for the first time. For such state, $Q(s,0) = Q(s,1) = 0$. So either f_0 or f_1 can be chosen. In each case, mutation operator can flip either 1-bit or 0-bit. Consequently, there can be four transitions from the state s, as illustrated in Fig. 2:

1) Fitness function f_0 is chosen, mutation operator flips 0 to 1. Since the mutated individual has lower f_0 value, the algorithm stays in the state s. The reward is zero, Q values do not change.
2) Fitness function f_0 is chosen, mutation operator flips 1 to 0. Since the mutated individual has higher f_0 value, the algorithm moves to the state $s-1$. The reward is negative, $Q(s,f_0)$ becomes negative after the update.
3) Fitness function f_1 is chosen, mutation operator flips 0 to 1. Since the mutated individual has higher f_1 value, the algorithm moves to the state $s+1$. The reward is positive, $Q(s,f_1)$ becomes positive after the update.
4) Fitness function f_1 is chosen, mutation operator flips 1 to 0. Since the mutated individual has lower f_1 value, the algorithm stays in the state s. The reward is zero, Q values do not change.

The Q-learning agent leaves the considered state in the cases 2, 3. In both cases, the Q-values have different signs after the update and $Q(s,f_1) > Q(s,f_0)$. So f_1 will be chosen in the state s during the next visit. The inequality between $Q(s,f_1)$ and $Q(s,f_0)$ will not change, since the agent will receive either positive or zero reward after applying f_1 in the state s. So the optimal fitness function f_1 will be chosen in s in all next visits. ■

B. Markov Chain

Consider a Markov chain of the optimization process performed with the considered algorithm. The Markov chain is shown in Fig. 3. The numbers written on the vertexes of the chain correspond to the number of 1-bits in an individual. We assume that the first individual is a string of zeros, so the starting vertex of the chain is marked with a zero.

There are three groups of vertexes in the chain. The first group consists of vertexes visited for the first time after visiting some lower vertexes, the second group consists of vertexes visited after the agent had chosen the obstructive fitness function f_0 and the third group of vertexes corresponds to recovering after this choice.

The chain is constructed using Lemma 1. The vertexes from the second and the third groups correspond to the already visited states. So in this vertexes the agent will always choose the efficient fitness function and eventually move to the higher vertexes.

C. Markov Chain Linearization

In this subsection we simplify the obtained Markov chain by replacing some loops with edges of the same *length*. The length of an edge is defined as the expected number of fitness function evaluations needed to pass this edge. Consider a typical part of the chain illustrated in Fig. 4. The chosen fitness functions, as well as mutation results and corresponding transition probabilities are written on the edges.

Let us calculate the expected number of fitness evaluations needed to get from the vertex B_{i-1} to the vertex A_{i+1} through the vertex C_i.

Fist of all, the expected number of fitness evaluations needed to get from the vertex B_{i-1} to the vertex C_i is evaluated:

$$E(B_{i-1} \to C_i) = 1 \times \frac{N-i+1}{N} + (1 + E(B_{i-1} \to C_i)) \times \frac{i-1}{N}$$

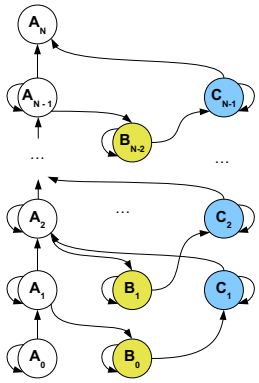

Fig. 3. Markov chain of the optimization process performed with RMHC + Q-learning

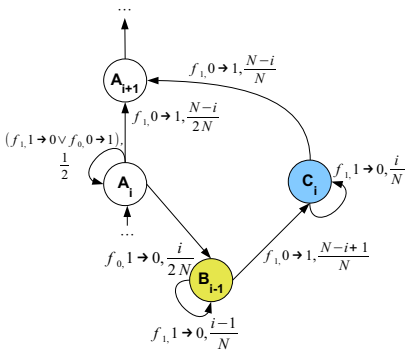

Fig. 4. A typical loop from the Markov chain on the Fig. 3

$$E(B_{i-1} \to C_i) = \frac{N}{N-i+1}$$

Then, the expected number of fitness evaluations needed to get from the vertex C_i to the vertex A_{i+1} is evaluated:

$$E(C_i \to A_{i+1}) = 1 \times \frac{N-i}{N} + (1 + E(C_i \to A_{i+1})) \times \frac{i}{N}$$

$$E(C_i \to A_{i+1}) = \frac{N}{N-i}$$

Finally, we obtain the result by summing up the intermediate estimations:

$$E(B_{i-1} \to A_{i+1}) = E(B_{i-1} \to C_i) + E(C_i \to A_{i+1})$$

$$E(B_{i-1} \to A_{i+1}) = \frac{N}{N-i+1} + \frac{N}{N-i}$$

Now we can replace the loop with a single edge of a corresponding length from the vertex A_i to the vertex A_{i+1}. The probability of passing this edge is equal to the probability of choosing the $A_i \to B_{i-1}$ edge, which is $p = \frac{i}{2N}$. Once the edge $A_i \to B_{i-1}$ is chosen with probability p, its length equals 1. So the length of the new edge between A_i and A_{i+1} is $1+E(B_{i-1} \to A_{i+1})$ as illustrated in Fig. 5. Replacing each loop in the same way, we obtain the linear Markov chain.

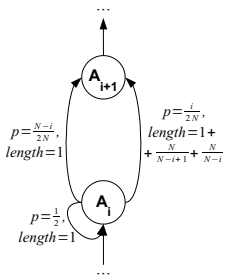

Fig. 5. Linearized part of the Markov chain

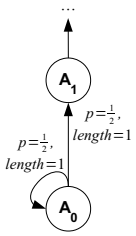

Fig. 6. Two first vertexes of the linearized Markov chain

D. Runtime Analysis

In this subsection we analyze the expected number of fitness function evaluations needed to get from the first vertex of the linearized Markov chain to the last one. In other words, we compute expected runtime of the considered algorithm.

Let $Z(i)$ be the expected number of fitness evaluations needed to get to the $(i+1)$-th vertex from the i-th one. Using the transition probabilities and lengths shown in the Fig. 5, we obtain the following equation:

$$Z(i) = \frac{1 + Z(i)}{2} + \frac{N-i}{2N} + \left(1 + \frac{N}{N-i+1} + \frac{N}{N-i}\right) \times \frac{i}{2N}$$

Solving the equation above, we get a closed formula for $Z(i)$:

$$Z(i) = 2 + \frac{i}{N-i+1} + \frac{i}{N-i} \tag{1}$$

Consider the special case of the starting vertex, which is illustrated in Fig. 6.

Analogically, we solve an equation for $Z(0)$ to get a closed formula:

$$Z(0) = 1 \times \frac{1}{2} + (1 + Z(0)) \times \frac{1}{2}$$

$$Z(0) = 2$$

Notice that if we substitute $i = 0$ to the Eq. 1, we also get $Z(0) = 2$. So we can compute the total expected number of fitness evaluations needed to get from the 0-th vertex to the N-th vertex as the following sum of $Z(i)$:

$$T_{R+Q}(N) = \sum_{i=0}^{N-1} \left(2 + \frac{i}{N-i+1} + \frac{i}{N-i}\right) \tag{2}$$

Now let us estimate the obtained result. Consider the expected number of fitness function evaluations needed to

solve the conventional OneMax problem using conventional Random Mutation Hill Climber:

$$T_O(N) = \sum_{i=0}^{N-1} \left(1 + \frac{i}{N-i}\right) \tag{3}$$

$$T_O(N) = \Theta(N \log N) \tag{4}$$

Let us analyze the argument of the sum in the Eq. 2:

$$T_{R+Q}(N) = \sum_{i=0}^{N-1} \left(2 + \frac{i}{N-i+1} + \frac{i}{N-i}\right)$$

$$\begin{aligned} 1 + \frac{i}{N-i} &< 2 + \frac{i}{N-i+1} + \frac{i}{N-i} < \\ &< 2 + 2\frac{i}{N-i} = 2(1 + \frac{i}{N-i}) \end{aligned}$$

Comparing the inequalities above with the runtime of the conventional RMHC (Eq. 3, 4), we obtain the upper and lower bounds for the runtime of the considered algorithm:

$$T_O(N) < T_{R+Q}(N) < 2 \cdot T_O(N),$$

$$T_{R+Q}(N) = \Theta(N \log N).$$

So it is finally proved that the expected number of fitness evaluations needed to solve the OneMax problem with the RMHC controlled with Q-learning is $\Theta(N \log N)$. ■

V. On the Computation Time of $\varepsilon-$greedy Q-learning

Previously in this paper, we used the greedy exploration strategy to choose between fitness functions (see Section III-B). Now let us consider another popular exploration strategy called ε-greedy [26], [27]. According to this strategy, the most efficient action (in our case, fitness function) is chosen with probability $(1-\varepsilon)$, while with probability ε a random action (fitness function) is chosen. This strategy should avoid stopping in local optima [26], but is it efficient for the considered problem?

In this section we construct a lower bound on the running time of the same algorithm as in Section IV, but with the use of ε-greedy exploration strategy, and show that it grows at least exponentially in N as soon as $\varepsilon > 0$.

To estimate the lower bound, we assume that ε-greedy Q-learning algorithm has learned the most efficient action in each state *before* the algorithm starts. That is, the algorithm will choose f_1 with the probability of $(1-\varepsilon)$ and f_0 with the probability of ε. Note that in a real implementation the probability for f_1 will always be less than the "ideal" one.

Let, as above, $Z(i)$ be the expected number of fitness evaluations needed to get to the $i+1$-th vertex from the i-th one. For $Z(0)$, every mutation flips 0 to 1 and increases target fitness value. So we go from vertex 0 to vertex 1 with the probability of $(1-\varepsilon)$ and remain in vertex 0 with the probability of ε. Thus, $Z(0) = (1-\varepsilon) + \varepsilon \cdot (1 + Z(0))$, from which by solving an equation we get:

$$Z(0) = \frac{1}{1-\varepsilon}. \tag{5}$$

In vertex i, $i > 0$, mutation flips 0 to 1 with the probability of $(n-i)/n$ and 1 to 0 with the probability of i/n. Independently, we select f_1 with the probability of $(1-\varepsilon)$ and f_0 with the probability of ε. So we have four cases:

1) 0 to 1, f_0: the probability is $\varepsilon\frac{n-i}{n}$, we remain in vertex i;
2) 0 to 1, f_1: the probability is $(1-\varepsilon)\frac{n-i}{n}$, we go from vertex i to vertex $(i+1)$;
3) 1 to 0, f_0: the probability is $\varepsilon\frac{i}{n}$, we go from vertex i to vertex $(i-1)$;
4) 1 to 0, f_1: the probability is $(1-\varepsilon)\frac{n-i}{n}$, we remain in vertex i.

In the third case, the fitness is decreased. However, we know by induction that the expected number of steps from vertex $(i-1)$ back to i is $Z(i-1)$. So the equation for $Z(i)$ has the following form:

$$\begin{aligned} Z(i) &= \frac{(1-\varepsilon)(n-i)}{n} + \frac{\varepsilon \times i}{n}\left(1 + Z(i-1) + Z(i)\right) + \\ &+ \left(\frac{(1-\varepsilon)\times i}{n} + \frac{\varepsilon \times (n-i)}{n}\right)(1 + Z(i)) \end{aligned}$$

which is simplified to

$$Z(i) = \frac{n + \varepsilon \times i \times Z(i-1)}{(1-\varepsilon) \times (n-i)} \tag{6}$$

As follows from Eq. 5, 6, the expected number of fitness evaluations for ε-greedy Q-learning is:

$$T_{R+\varepsilon Q}(N) = \frac{1}{1-\varepsilon} + \sum_{n=1}^{N-1} Z(n). \tag{7}$$

We have not found a closed expression for both $T_{R+\varepsilon Q}(N)$ and its asymptotic behavior yet. However, we evaluated it on several values of N and ε. The results are shown in Table II together with the exact values for $\varepsilon = 0$.

TABLE II
Values of $T_{R+\varepsilon Q}(N)$ for different N and ε: exact for $\varepsilon = 0$, lower bounds for $\varepsilon > 0$

$\varepsilon \backslash N$	4	16	64	256
0.0	1.58×10^1	9.66×10^1	5.49×10^2	2.89×10^3
0.1	1.12×10^1	1.00×10^2	1.05×10^4	5.37×10^{12}
0.2	1.42×10^1	2.75×10^2	8.56×10^6	3.27×10^{25}
0.3	1.89×10^1	1.27×10^3	2.84×10^{10}	1.52×10^{40}
0.4	2.71×10^1	1.01×10^4	4.04×10^{14}	1.56×10^{57}
0.5	4.37×10^1	1.42×10^5	3.75×10^{19}	2.32×10^{77}
0.6	8.34×10^1	4.07×10^6	4.95×10^{25}	1.25×10^{102}

It can be seen in the table that for $\varepsilon > 0$ the growth of lower bounds with N is close to an exponent. By trial and error we acquired an estimation $T_{R+\varepsilon Q}(N) \approx 4\varepsilon \times e^N \log N/N$ which holds with 10% of relative error for nearly all table entries. Compared to $\Theta(N \log N)$ for $\varepsilon = 0$, this clearly shows that ε-greedy exploration strategy is inefficient for the OneMax model problem.

VI. Conclusion

We analyzed EA + RL method, which was empirically shown to be effective for a number of model problems and a real-world application in previous research. We proved that

Random Mutation Hill Climber controlled by Q-learning with greedy exploration strategy solves the modified OneMax problem equally well with the conventional evolutionary algorithm without an obstructive fitness function, namely in $\Theta(N \log N)$ fitness function evaluations. We also analyzed the lower bound for the ε-greedy exploration strategy for $\varepsilon > 0$.

In this work a first step towards better theoretical understanding of how reinforcement learning adjusts evolutionary algorithms is made. Further work should involve analyzing broader range of evolutionary algorithms (first of all, $(1 + 1)$-evolutionary strategy), as well as reinforcement learning algorithms. It is also very important to formulate a model problem with a supporting fitness function and show that EA + RL outperforms the adjusted evolutionary algorithm on this problem. The long-term goal is to obtain more general results for class of problems and algorithms, as well as to work out a theoretical framework for the runtime analysis of evolutionary algorithms adjusted with reinforcement learning in different ways.

VII. Acknowledgments

The research was partially supported by the Ministry of Education and Science of Russian Federation in the framework of the federal program"Scientific and scientific-pedagogical personnel of innovative Russia in 2009-2013" (contract 16.740.11.0455, agreement 14.B37.21.0397),by the University ITMO development program in 2012-2018 and by the University ITMO research project 610455. The authors also would like to thank Fedor Tsarev for his useful ideas.

References

[1] A. Buzdalova and M. Buzdalov, "Increasing efficiency of evolutionary algorithms by choosing between auxiliary fitness functions with reinforcement learning," in *ICMLA (1)*. IEEE, 2012, pp. 150–155.

[2] A. Afanasyeva and M. Buzdalov, "Optimization with auxiliary criteria using evolutionary algorithms and reinforcement learning," in *Proceedings of 18th International Conference on Soft Computing MENDEL 2012*, Brno, Czech Republic, 2012, pp. 58–63.

[3] J. Handl, S. Lovell, and J. Knowles, "Multiobjectivization by decomposition of scalar cost functions," in *Parallel Problem Solving from Nature PPSN X*, ser. Lecture Notes in Computer Science, G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume, Eds. Springer Berlin Heidelberg, 2008, vol. 5199, pp. 31–40.

[4] F. Neumann and I. Wegener, "Can single-objective optimization profit from multiobjective optimization?" in *Multiobjective Problem Solving from Nature*, ser. Natural Computing Series, J. Knowles, D. Corne, K. Deb, and D. Chair, Eds. Springer Berlin Heidelberg, 2008, pp. 115–130.

[5] J. D. Knowles, R. A. Watson, and D. Corne, "Reducing local optima in single-objective problems by multi-objectivization," in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, ser. EMO '01. London, UK: Springer-Verlag, 2001, pp. 269–283.

[6] M. Buzdalov and A. Buzdalova, "Adaptive selection of helper-objectives for test case generation," in *2013 IEEE Conference on Evolutionary Computation*, vol. 1, June 20-23 2013, pp. 2245–2250.

[7] M. Buzdalov, A. Buzdalova, and I. Petrova, "Generation of Tests for Programming Challenge Tasks Using Multi-Objective Optimization," in *GECCO (Companion)*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 1655–1658.

[8] D. Greiner, J. Emperador, G. Winter, and B. Galvan, "Improving computational mechanics optimum design using helper objectives: An application in frame bar structures," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, Eds. Springer Berlin Heidelberg, 2007, vol. 4403, pp. 575–589.

[9] M. T. Jensen, "Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation: Evolutionary computation combinatorial optimization," *Journal of Mathematical Modelling and Algorithms*, vol. 3, no. 4, pp. 323–347, 2004.

[10] D. F. Lochtefeld and F. W. Ciarallo, "Helper-objective optimization strategies for the job-shop scheduling problem," *Applied Soft Computing*, vol. 11, no. 6, pp. 4161 – 4174, 2011.

[11] A. Afanasyeva and M. Buzdalov, "Choosing best fitness function with reinforcement learning," in *Proceedings of the Tenth International Conference on Machine Learning and Applications, ICMLA 2011*, vol. 2. Honolulu, HI, USA: IEEE Computer Society, 2011, pp. 354–357.

[12] A. Buzdalova and M. Buzdalov, "Adaptive selection of helper-objectives with reinforcement learning," in *ICMLA (2)*. IEEE, 2012, pp. 66–67.

[13] A. E. Eiben, M. Horvath, W. Kowalczyk, and M. C. Schut, "Reinforcement learning for online control of evolutionary algorithms," in *Proceedings of the 4th international conference on Engineering self-organising systems ESOA'06*. Springer-Verlag, Berlin, Heidelberg, 2006, pp. 151–160.

[14] S. Müller, N. N. Schraudolph, and P. D. Koumoutsakos, "Step size adaptation in evolution strategies using reinforcement learning," in *Proceedings of the Congress on Evolutionary Computation*. IEEE, 2002, pp. 151–156.

[15] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta, "A method to control parameters of evolutionary algorithms by using reinforcement learning," in *Signal-Image Technology and Internet-Based Systems (SITIS), 2010 Sixth International Conference on*, 2010, pp. 74–79.

[16] J. E. Pettinger and R. M. Everson, "Controlling genetic algorithms with reinforcement learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 692–.

[17] A. Gosavi, "Reinforcement learning: A tutorial survey and recent advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, 2009.

[18] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policyreinforcement-learning algorithms," *Mach. Learn.*, vol. 38, no. 3, pp. 287–308, Mar. 2000.

[20] C. Szepesvári, "The asymptotic convergence-rate of q-learning," in *Proceedings of the 1997 conference on Advances in neural information processing systems 10*, ser. NIPS '97. Cambridge, MA, USA: MIT Press, 1998, pp. 1064–1070.

[21] M. Kearns and S. Singh, "Finite-sample convergence rates for q-learning and indirect algorithms," in *Proceedings of the 1998 conference on Advances in neural information processing systems II*. Cambridge, MA, USA: MIT Press, 1999, pp. 996–1002.

[22] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1–25, Dec. 2004.

[23] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC Model-free Reinforcement Learning," in *Proceedings of the 23rd International Conference on Machine Learning (ICML 2006)*, 2006, pp. 881–888.

[24] P. S. Oliveto, J. He, and X. Yao, "Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results," *International Journal of Automation and Computing*, vol. 4, no. 3, pp. 281–293, 2007.

[25] M. Buzdalov, "Generation of Tests for Programming Challenge Tasks Using Evolution Algorithms," in *Proceedings of the 2011 GECCO Conference Companion on Genetic and Evolutionary Computation*, New York, US, ACM, 2011, pp. 763–766.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[27] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.