2014 13th International Conference on Machine Learning and Applications

Improved Selection of Auxiliary Objectives using Reinforcement Learning in Non-Stationary Environment

Irina Petrova, Arina Buzdalova, Maxim Buzdalov
ITMO University
49 Kronverkskiy prosp.
Saint-Petersburg, Russia, 197101
Email: {petrova, afanasyeva, buzdalov}@rain.ifmo.ru

***Abstract*—Efficiency of evolutionary algorithms can be increased by using auxiliary objectives. The method which is called EA+RL is considered. In this method a reinforcement learning (RL) algorithm is used to select objectives in evolutionary algorithms (EA) during optimization. In earlier studies, reinforcement learning algorithms for stationary environments were used in the EA+RL method. However, if behavior of auxiliary objectives change during the optimization process, it can be better to use reinforcement learning algorithms which are specially developed for non-stationary environments. In our previous work we proposed a new reinforcement learning algorithm to be used in the EA+RL method. In this work we propose an improved version of that algorithm. The new algorithm is applied to a non-stationary problem and compared with the methods which were used in other studies. It is shown that the proposed method achieves optimal value more often and obtains higher values of the target objective than the other algorithms.**

I. Introduction

Important problems in evolutionary algorithms are avoiding local optima and increasing genetic diversity. Extra objectives can be used to overcome these difficulties [1], [2]. The corresponding approach is called multiobjectivization. General idea of multiobjectivization is described below.

A. Multiobjectivization

There are two methods of multiobjectivization. One of them is based on decomposing the target objective into several components, which are optimized simultaneously [1]. In this method the components should be independent, but it is not always easy or possible [1]. Another method is to use some additional objectives that are used in combination with the original objective [2]. This approach was successfully applied to some practical problems, for example, the Job Shop Scheduling Problem and the Traveling Salesman Problem [2], [3]. At each step of the algorithm, one or several auxiliary objectives are optimized [3]. There are methods of selecting auxiliary objectives at each step of optimization exist. One of them is to select auxiliary objectives in random order [2]. Another one is to use an ad-hoc heuristic [3]. The first approach is general, but does not take into account any features of an optimization problem, while the second one can be applied only to a specific problem. The EA+RL method was designed to deal with these issues [4].

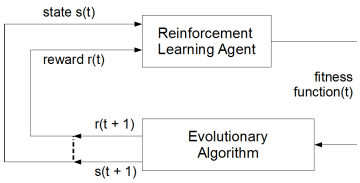

Fig. 1. EA+RL objective selection method

B. EA+RL Method

In the EA+RL method, reinforcement learning [5] is used to select an objective (fitness function in terms of an EA), which is optimized in the current iteration of an EA. In RL, an agent applies some action to an environment, then the environment returns some representation of its state and a numerical reward to the agent, and the process repeats. The environment in EA+RL method is the EA. Each action of the agent corresponds to an objective to be optimized at the current iteration. The agent selects an objective from a set of auxiliary objectives and the target objective. Note that we do not aim to maximize the auxiliary objectives, they are just used to increase the efficiency of the target objective optimization. Generally, the goal of RL is to maximize the total reward [5], calculated by formula $E[\sum_{t=0}^{\infty} \gamma^t r_t]$, where γ — discount factor. In the EA+RL method, the reward is based on the difference of the target objective values in two sequential iterations [6]. Therefore, the total reward which is roughly equivalent to the difference between the final and the initial values of the target objective, is maximized. So the target objective is maximized by RL and it is not necessary to explicitly optimize it by EA. Therefore, on each iteration only the selected objective is optimized by the EA. Hence, we can use a single-objective EA which typically runs faster than a multi-objective EA.

The EA+RL method is illustrated in Fig. 1, where t is the number of the current iteration of the EA. This method was previously shown to be efficient for a number of problems [4].

978-1-4799-7415-3/14 $31.00 © 2014 IEEE
DOI 10.1109/ICMLA.2014.99

In previous studies, it was implied that the environment was stationary. More precisely, the obtained reward depended only on the applied action and the state of the environment. Consequently, RL algorithms for stationary environments, such as Q-learning, were used. However, in the case when properties of auxiliary objectives depend on the stage of the optimization process, the reward that is obtained as a result of the action can be different in the same RL state. Therefore, RL algorithms for non-stationary environments should probably be used. In our previous work we analysed existing methods of reinforcement learning in non-stationary environment [7]. The Reinforcement Learning Context Detection (RLCD) algorithm is the closest to our needs [8]. We have tried to apply the RLCD algorithm in the EA+RL approach for solving the benchmark problem but the experiments showed that RLCD was not efficient. Therefore we proposed another RL approach.

C. Previous method for EA+RL in non-stationary environment

In our previous method ε-greedy Q-learning, which is a model-free RL algorithm [5], is used. As in the conventional Q-learning, at each iteration, the RL agent selects an action a and applies it to the environment, which is in state s. Then the expected reward estimation $Q(s,a)$ is updated using the obtained reward. The core idea of this method is to reset Q values sometimes, i.e. set $Q(s,a)$ equal to zero for all s and a. It occurs if the following condition is fulfilled: $|Q(s,a) - Q(s',a')| < \delta$ for some pair (s,a) and (s',a'), where δ is some constant, called *distinction factor*. This approach was tested on a benchmark problem. The achieved results were better than the results obtained using other considered methods.

In this work we propose improved version of the previous approach. The rest of the paper is organized as follows. First, the proposed method is described. Second, a new benchmark problem is proposed. Then the experiments are described. Finally, the proposed RL algorithm is compared with the methods previously used in EA+RL.

II. Method description

As in our previous research ε-greedy Q-learning is used. However, another condition for resetting the Q values is used. The pseudocode of the proposed approach is presented in Algorithm 1. There are two conditions of resetting Q values. The first condition is fulfilled if 1) the reward is less than or equal to zero during some consequent iterations and 2) during c_1 of these iterations the reward is less than zero, where c_1 is some constant. In this case Q values are reset.

The second condition works for fitness functions with plateaus. It is fulfilled if the reward is less than or equal to zero during some consequent iterations and in c_2 of these iterations the reward is equal to zero, where c_2 is some constant. In this case Q values are reset with probability equal to $\frac{optimal-current}{optimal}$, there $optimal$ is the optimal value of the target criterion, $current$ is a value of the target criterion in this iteration.

We assume the first condition indicates that the behavior of auxiliary objectives have changed. The second condition shows that the EA got stuck in local optima. Also note that in the end of optimization process finding a better solution takes more steps. Therefore, we decrease the probability of resetting Q values during optimization process. It prevents frequent resetting of Q values in the end of the optimization.

Algorithm 1 The proposed method

Form initial generation G_0
Initialize $Q(s,a) \leftarrow 0$ for each state s and action a
Initialize iteration counter: $k \leftarrow 0$
Initialize first reset counter: $reset_1 \leftarrow 0$
Initialize second reset counter: $reset_2 \leftarrow 0$
while (specified number of generations or maximum value of target objective not reached) **do**
  Evaluate current state s_k and pass it to agent
  Select action $a : Q(s,a) = \max_{a'} Q(s,a')$
  Generate next generation G_{k+1}
  Calculate reward r and the next state s'
  $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
  Initialize reset identifier: $reset \leftarrow false$
  if reward is less than zero: $r < 0$ **then**
    increment first reset counter: $reset_1 \leftarrow reset_1 + 1$
  if reward is equal to zero: $r = 0$ **then**
    increment second reset counter: $reset_2 \leftarrow reset_2 + 1$
  if reward is greater than zero: $r > 0$ **then**
    set first reset counter to zero: $reset_1 \leftarrow 0$
    set second reset counter to zero: $reset_2 \leftarrow 0$
  if first condition is fulfilled: $reset_1 = c_1$ **then**
    $reset \leftarrow true$
  if second condition is fulfilled: $reset_2 = c_2$ **then**
    set second reset counter to zero: $reset_2 \leftarrow 0$
    generate random number n from 0 to 1
    if $n \leq \frac{optimal-current}{optimal}$ **then**
      $reset \leftarrow true$
  if reset of learning is needed: $reset = true$ **then**
    set *Q(s, a)* for each s and a to zero : $Q(s,a) \leftarrow 0$
    set first reset counter to zero: $reset_1 \leftarrow 0$
    set second reset counter to zero: $reset_2 \leftarrow 0$
  Update iteration counter: $k \leftarrow k + 1$

III. Benchmark problem

We tested our method on the following non-stationary benchmark problem, which is harder than the previously used one [7]. In this problem, an individual is a bit string of length n. Let x be the number of bits in an individual which are set to one. Then the target fitness function (objective) is $g(x) = \lfloor \frac{x}{k} \rfloor$, where k is constant, $k < n$ and k divides n. The auxiliary objectives are $h_1(x)$ and $h_2(x)$, which are described below.

$$h_1(x) = \begin{cases} x, x \leq p_1 \\ -x + 2p_1, \\ \qquad p_1 < x \leq p_2 \\ x, p_2 < x \leq p_3 \\ \cdots \\ x, p_s < x \leq n \end{cases} \qquad h_2(x) = \begin{cases} -x, x \leq p_1 \\ x, p_1 < x \leq p_2 \\ -x + 2p_2, \\ \qquad p_2 < x \leq p_3 \\ \cdots \\ -x + 2p_s, \\ \qquad p_s < x \leq n \end{cases}$$

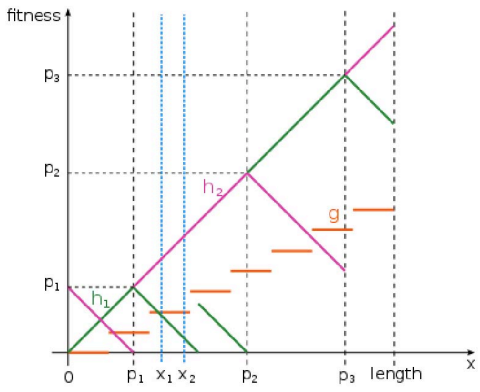

Fig. 2. Benchmark problem: fitness graph, g — the target objective, h_1, h_2 — auxiliary objectives

Here, each p_i is called a *switch point*, s is the number of switch points. Note that at each switch point the behavior of auxiliary objectives are changed. Objective h_i can be negative on $[2p_{k-1}, p_k)$ if $p_k - p_{k-1} > p_{k-1}$. Due to implementation needs fitness values should be positive, so the following transform is used: $h_i(x) \leftarrow h_i(x) + p_k - 2p_{k-1} = -x + 2p_{k-1} + p_k - 2p_{k-1} = -x + p_k$. The auxiliary objectives and the target objective are illustrated in Fig. 2. The h_1 objective helps optimization if $x \in [0, p_1], (p_2, p_3], ..., (p_s, n]$ while the h_2 objective is efficient in all other cases. Note that using the proper auxiliary objective allows to distinguish individuals with the same value of the target objective and give preference to the individual with higher x value. Such individual is more likely to produce a descendant with a higher target objective value. Ideally, for each EA iteration an auxiliary objective which is increasing at the current interval of x values should be selected. The main difference between this problem and the previous one [7] is that in the case of selection inefficient auxiliary objective the target objective value can be decreased. So it is harder to solve the new problem.

Let us describe why this problem is non-stationary. The RL state is a vector, the elements of the vector corresponds to the number of a generation, average generation target objective value and entropy of the generation fitness [9]. The number of a generation is binned into four intervals on a logarithmic scale. Average generation target objective value normalized by ideal generation target objective value and binned into four intervals. Entropy is binned into three equally sized discrete states. Therefore, state is a vector of three numbers — numbers of the corresponding intervals.

This state is independent of auxiliary objectives properties. Also note that a lot of target fitness values corresponds to one state. Therefore, the reward obtained as a result of selection auxiliary objective h can be different in the same state s.

For example, consider a generation consisting of two individuals, let the ideal target objective value be 100, let the maximal number of generations be 258. Consider the following optimization steps:

- The current generation number is 100. The target objective values of individuals in generation are 80 and 78. So the state is computed to 443 by formulas from [9]. The efficient auxiliary objective h_1 is chosen. The target objective values of individuals became 95 and 80, so the reward is 15 and the new state is 443.
- The current generation number is 101. The state is 443. The auxiliary objective h_1 changed its behavior and became inefficient. However, the Q values have not changed yet, so h_1 is chosen. The target objective values of individuals became 90 and 80, the reward is -5 the new state is still 443.

In this example the reward that is obtained as a result of the action h_1 can be 15 and -5 in the same state 443. So we showed that the same auxiliary objective can be efficient and inefficient at the same state. Consequently, the problem is non-stationary.

IV. Experiment Description

Several configurations of the formulated benchmark problem were solved using EA+RL with five different RL algorithms. All the considered algorithms were run 100 times on every problem instance, then the results were averaged. Different values of individual length n were considered. Configurations with 5 switch points and $k = 10$ were considered. The maximum number of iterations was set to 5000.

There were 100 individuals in a generation. Mutation operator flipped each bit with the probability of 0.001. Shift crossover [10] was applied with the probability of 0.7.

Switch points were chosen in the following way. The first point p_1 was chosen randomly from (0, n). The second point p_2 had been being chosen randomly from (p_1, n) until the difference between p_2 and the previously generated switch points was less than 100. The other points were generated similarly.

We tested five algorithms: the proposed method, our previous method [7], ε-greedy Q-learning [5], Delayed Q-learning [11] and RLCD [8]. The ε-greedy Q-learning and Delayed Q-learning algorithms were previously used in the EA+RL method for various problems. Their parameters were taken from [7]. For the ε-greedy Q-learning, learning rate was set to $\alpha = 0.6$, discount factor was $\gamma = 0.01$, exploration probability was $\varepsilon = 0.3$. In Delayed Q-learning the following parameters were used: update period $m = 5$, discount factor $\gamma = 0.01$, bonus reward $\varepsilon = 0.4$. Parameters of our previous method was set to $\alpha = 0.6$, $\gamma = 0.01$, $\delta = 0.001$. Parameters of the proposed method was set to $\alpha = 0.6$, $\gamma = 0.01$, $c_1 = 10$, $c_2 = 10$.

The Wilcoxon rank sum test was performed to check if the new algorithm and the other ones are distinguishable. Statistical significance was tested at $\alpha = 10^{-2}$ level.

V. Experiment Results

The average target objective values are shown in Table I as $\frac{optimal - average}{optimal} \cdot 100\%$, where *average* is the average fitness, *optimal* is the optimal value of the target objective. The first column contains the length of an individual. The next five columns contain the results achieved using new method, our

TABLE I
AVERAGE TARGET OBJECTIVE VALUES OBTAINED USING DIFFERENT ALGORITHMS IN PERCENT UNDER THE OPTIMAL VALUE

Length	New	Previous	Q	DQ	RLCD
750	0.56	1.60 (2.2×10^{-16})	0.64 (1.9×10^{-1})	11.43 (2.2×10^{-16})	5.23 (2.2×10^{-16})
1000	0.39	1.16 (2.2×10^{-16})	0.49 (7.7×10^{-2})	4.23 (3.6×10^{-16})	4.33 (2.2×10^{-16})
1250	0.34	1.02 (2.2×10^{-16})	1.44 (2.4×10^{-15})	15.28 (2.2×10^{-16})	4.76 (2.2×10^{-16})
1500	0.17	0.87 (2.2×10^{-16})	1.83 (2.2×10^{-16})	12.01 (2.2×10^{-16})	11.98 (2.2×10^{-16})
1750	0.27	1.14 (2.2×10^{-16})	1.97 (2.2×10^{-16})	16.56 (2.2×10^{-16})	8.37 (2.2×10^{-16})
2000	0.78	1.55 (2.2×10^{-16})	2.00 (2.2×10^{-16})	14.54 (2.2×10^{-16})	6.92 (2.2×10^{-16})
2250	1.07	1.59 (1.2×10^{-13})	1.98 (2.2×10^{-16})	21.91 (2.2×10^{-16})	15.39 (2.2×10^{-16})
2500	1.18	1.73 (2.2×10^{-16})	1.88 (2.2×10^{-16})	8.79 (2.2×10^{-16})	16.05 (2.2×10^{-16})
2750	1.35	1.68 (3.2×10^{-12})	1.96 (2.2×10^{-16})	12.32 (2.2×10^{-16})	8.34 (2.2×10^{-16})
3000	1.40	1.62 (1.2×10^{-11})	3.09 (2.2×10^{-16})	16.08 (2.2×10^{-16})	11.04 (2.2×10^{-16})
3250	1.45	1.52 (9.1×10^{-5})	3.70 (2.2×10^{-16})	5.37 (2.2×10^{-16})	18.06 (2.2×10^{-16})
3500	1.34	1.49 (3.0×10^{-7})	3.97 (2.2×10^{-16})	10.33 (2.2×10^{-16})	12.79 (2.2×10^{-16})
3750	1.32	2.03 (4.0×10^{-16})	4.13 (2.2×10^{-16})	13.31 (2.2×10^{-16})	20.01 (2.2×10^{-16})

previous method, ε-greedy Q-learning, Delayed Q-learning and RLCD. The grey background corresponds to be best result for each problem instance. The deviation of the average fitness in the case of using Delayed Q-learning is about 10%, the deviation in the case of using the other algorithms is about 0.5%. For all analyzed problem instances the new method outperforms previously used ones.

The p-values obtained when comparing the proposed method with the other algorithms are presented in parentheses. For our previous method, Delayed Q-learning and RLCD p-values are less than significance level α, so the new method is statistically distinguishable from these methods. For the ε-greedy Q-learning p-values are less than α for the instances of lengths greater than 1000, so the new method is statistically distinguishable from ε-greedy Q-learning in this case.

The number of runs of the considered algorithms when the optimal target objective value was reached is shown in Table II. The first column contains the length of an individual. The next five columns contain number of runs when the optimal target objective value was reached using new method, our previous method, ε-greedy Q-learning, Delayed Q-learning and RLCD. For the problem instances of lengths less than 2750 the new method outperforms previously used ones. On the other problem instances no algorithm reached optimal value.

TABLE II
NUMBER OF TIMES WHEN THE OPTIMAL VALUE OF THE TARGET OBJECTIVE WAS REACHED

Length	New	Previous	Q	DQ	RLCD
750	58	4	52	12	0
1000	61	0	51	15	0
1250	65	0	20	5	0
1500	76	0	0	1	0
1750	60	0	0	0	0
2000	60	0	0	0	0
2250	12	0	0	0	0
2500	2	0	0	0	0
2750	0	0	0	0	0

VI. CONCLUSION

An improved version of our previous reinforcement learning approach was proposed. The proposed approach was used to solve a benchmark problem. The Wilcoxon rank sum test showed that the performance of the proposed method is statistically distinguishable from the others. The achieved results are better than the results obtained with our previous method, ε-greedy Q-learning, Delayed Q-learning and RLCD algorithms.

This work was financially supported by the Government of Russian Federation, Grant 074-U01.

REFERENCES

[1] 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*. Springer-Verlag, 2001, pp. 269–283.

[2] 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.

[3] D. F. Lochtefeld and F. W. Ciarallo, "Deterministic Helper-Objective Sequences Applied to Job-Shop Scheduling," in *Proceedings of Genetic and Evolutionary Computation Conference*. ACM, 2010, pp. 431–438.

[4] A. Buzdalova and M. Buzdalov, "Increasing Efficiency of Evolutionary Algorithms by Choosing between Auxiliary Fitness Functions with Reinforcement Learning," in *Proceedings of the International Conference on Machine Learning and Applications*, vol. 1, 2012, pp. 150–155.

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

[6] S. D. Mller, N. N. Schraudolph, and P. D. Koumoutsakos, "Step size adaptation in evolution strategies using reinforcement learning," in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*. IEEE Press, 2002, pp. 151–156.

[7] I. Petrova, A. Buzdalova, and M. Buzdalov, "Selection of Extra Objectives using Reinforcement Learning in Non-Stationary Environment: Initial Explorations," in *Proceedings of 20th International Conference on Soft Computing MENDEL 2014*, Czech Republic, 2014, pp. 58–63.

[8] B. C. D. Silva, E. W. Basso, A. L. C. Bazzan, and P. M. Engel, "Dealing with Non-stationary Environments Using Context Detection," in *Proceedings of the 23rd International Conference on Machine Learning*. ACM Press, 2006, pp. 217–224.

[9] 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*, Dec 2010, pp. 74–79.

[10] V. Arkhipov, M. Buzdalov, and A. Shalyto, "Worst-Case Execution Time Test Generation for Augmenting Path Maximum Flow Algorithms using Genetic Algorithms," in *Proceedings of the International Conference on Machine Learning and Applications*, vol. 2. IEEE Computer Society, 2013, pp. 108–111.

[11] 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*, 2006, pp. 881–888.