Analysis of Q-Learning with Random Exploration for Selection of Auxiliary Objectives in Random Local Search

Maxim Buzdalov, Arina Buzdalova
ITMO University
49 Kronverkskiy av.
Saint-Petersburg, Russia, 197101
Email: {mbuzdalov, abuzdalova}@gmail.com

***Abstract*—We perform theoretical analysis for a previously proposed method of enhancing performance of an evolutionary algorithm with reinforcement learning. The method adaptively chooses between auxiliary objectives in a single-objective evolutionary algorithm using reinforcement learning. We consider the Q-learning algorithm with ε-greedy strategy ($\varepsilon > 0$), using a benchmark problem based on OneMax. For the evolutionary algorithm, we consider the Random Local Search. In our setting, OneMax problem should be solved in the presence of the obstructive ZeroMax objective. This benchmark tests the ability of the reinforcement learning algorithm to ignore such an inefficient objective. It was previously shown that in the case of the greedy strategy ($\varepsilon = 0$), the considered algorithm performs on the described benchmark problem in the best possible time for a conventional evolutionary algorithm. However, the ε-greedy strategy appears to perform in exponential time. Furthermore, every selection algorithm which selects an inefficient auxiliary objective with probability of at least δ is shown to be asymptotically inefficient when $\delta > 0$ is a constant.**

I. Introduction

In this paper we analyze a previously proposed method of selecting between auxiliary objectives in an evolutionary algorithm (EA). The selection is performed using reinforcement learning (RL), so the method is called EA+RL. The introduction starts with a brief overview of using auxiliary objectives to enhance single-objective optimization. Then we discuss the current state of the EA+RL research and give the motivation for analyzing EA+RL with $\epsilon-$greedy exploration strategy. The rest of the paper is dedicated to the theoretical analysis of EA+RL and to the discussion of its results.

A. Using Auxiliary Objectives in Single-Objective Optimization

To compare the solutions generated at each population of a conventional evolutionary algorithm, a *fitness function* is used. The fitness function is usually correlated with the objective being optimized [1]. Let us call the fitness function used in a conventional single-objective evolutionary algorithm a *target* one.

It is known that using some auxiliary objectives can enhance single-objective optimization performance [1]–[5]. In existing approaches termed as *multiobjectivization*, auxiliary objectives are designed manually and optimized simultaneously using a multi-objective evolutionary algorithm [6]–[8]. There also exists a dynamic *helper-objective* approach [9], [10]. In this approach a dynamically selected auxiliary objective, called helper-objective, is optimized along with the target one. The helper-objectives to be optimized at the current optimization stage are selected randomly from the set of all helper-objectives. In other papers, a problem dependent selection order of helper-objectives for the Job-shop scheduling problem is considered [11], [12]. Finally, there is an analysis of selection order for benchmark problems [13].

978-1-4799-7492-4/15/$31.00 ©2015 IEEE

B. EA+RL Method

The EA + RL method was proposed by the authors of this paper for adaptive selection of auxiliary fitness functions in single-objective evolutionary algorithms [14]. The selection of auxiliary fitness functions is performed during the running time of an evolutionary algorithm using reinforcement learning. The 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 and do not aim to optimize the auxiliary fitness functions themselves. 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. The reinforcement learning mechanism aims to select such auxiliary fitness functions, that provide speeding up of the target fitness function optimization. In other words, it should be able to select supporting fitness functions and to ignore obstructive ones.

One of the main differences between the EA + RL method and the multiobjectivization approach, as well as the helper-objective approach, is that EA + RL can be implemented

without using a multi-objective evolutionary algorithm. In EA + RL, auxiliary objectives still help to improve the target objective optimization performance, but it is not necessary to optimize them simultaneously with the target objective or with each other. They are used just to guide the search. Also notice, that no prior knowledge about the properties of auxiliary fitness functions is needed to apply EA + RL. Via its reinforcement learning mechanism, EA + RL should be able to select efficient (supporting) fitness functions and to ignore inefficient (obstructive) ones. On the other hand, in multiobjectivization approach auxiliary objectives are specially designed to be efficient, which implies sophisticated intellectual work [6].

C. Motivation and Comparison with Previous Papers

The EA + RL method was empirically shown to be efficient in applying to a real-world problem of test generation [15], as well as in solving a number of benchmark problems [14]. However, theoretical foundations are needed as well. Theoretical results could give insights on the method and suggest on the proper parameter and design settings for the experiments.

So far, we have proved some specific facts for the EA+RL implementation that uses Q-learning with greedy exploration strategy. According to this strategy, an action with the best Q-value is always selected. For the XDIVK problem, we showed that EA+RL with greedy strategy is able to select the ONEMAX supporting objective [16]. As a result, EA+RL asymptotically outperforms a conventional EA for this problem. We also considered the ONEMAX problem, which is a well-known benchmark in evolutionary computation theory. It was shown that EA+RL with greedy strategy is able to ignore obstructive ZEROMAX objective [17]. For this problem, EA+RL takes $\Theta(N \log N)$ iterations to reach the optimum, which is the best possible performance for solving ONEMAX using a conventional EA.

The mentioned results relate to the Q-learning with greedy exploration. In this paper we consider another popular exploration strategy called ε-greedy [18], [19]. 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 is chosen. This strategy should avoid stopping in local optima [18]. We investigate whether it is efficient for the ONEMAX problem with the ZEROMAX obstructive objective.

II. BENCHMARK PROBLEM AND ALGORITHM

In this section we consider a benchmark problem and an implementation of the EA + RL method used to solve it.

A. Description of Objectives

We consider two objectives: the obstructive objective ZEROMAX (f_0) and the target objective ONEMAX (f_1). The target objective f_1 is calculated as number of 1-bits in an individual of length N, while the obstructive objective f_0 is calculated as number of 0-bits, so optimizing f_0 decreases f_1. Let us call the described ONEMAX problem with the ZEROMAX obstructive objective a *modified* ONEMAX problem.

As it was mentioned in the introduction, for these objectives, EA + RL with greedy strategy performs as good as a conventional algorithm without an obstructive fitness function. In the next section we will investigate if $\epsilon-$greedy strategy is efficient here as well.

B. RLS + Q-learning Algorithm

For better understanding of the EA + RL mechanism, consider some basics of reinforcement learning [18], [19]. Reinforcement learning algorithms are designed to find an optimal behavioral strategy in an interactive environment. An agent chooses an action, applies it to the environment and receives the immediate reward for this action, as well as some representation of the environment state. The goal is to maximize the total reward. In the EA + RL we treat a single-objective evolutionary algorithm as an environment.

Consider the following EA + RL implementation. We use the Random Local Search (RLS) algorithm [6] for the EA and the Q-learning algorithm [18] for the RL algorithm. The scheme of the resulting algorithm is shown in Fig. 1

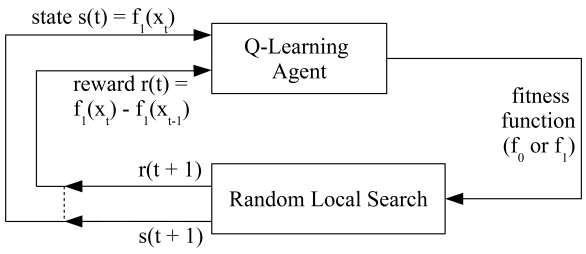

Fig. 1. RLS + Q-learning scheme

In this EA + RL implementation an environment state equals the number of 1-bits in the current individual generated with RLS. So there are $N+1$ possible states from s_0 to s_N. The immediate reward r equals the difference of the target fitness function values calculated on the two individuals, which are consequently generated with RLS. The Q-learning algorithm is designed to maximize the total reward: $E[\sum_{t=0}^{\infty} \gamma^t r_t] \to max$, where γ is a real valued parameter [18]. The efficiency of choosing some fitness function f in the state s is denoted by $Q(s, f)$.

The pseudocode of the considered EA+RL implementation is presented in Algorithm 1. Note that in the lines 11-18 an action to be applied is chosen according to the $\epsilon-$greedy exploration strategy.

III. RLS + ε-GREEDY Q-LEARNING ANALYSIS

In this section we construct a lower bound on the running time of the previously described algorithm, and show that it grows at least exponentially in N as soon as $\varepsilon > 0$.

Algorithm 1 RLS + 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
  With probability of $1 - \varepsilon$
  if $Q(s,0) > Q(s,1)$ **then**
    $f \leftarrow f_0$, $I \leftarrow 0$
  else if $Q(s,0) < Q(s,1)$ **then**
    $f \leftarrow f_1$, $I \leftarrow 1$
  end if
  With probability of ε
  $I \leftarrow$ random(0,1), $f \leftarrow f_I$
  if $f(Y) \geq f(X)$ **then**
    $X \leftarrow Y$
  end if
  $r \leftarrow f_1(X) - s$
  $s' \leftarrow f_1(X)$
  $Q(s,f) \leftarrow (1-\alpha)Q(s,f) + \alpha(r + \gamma \cdot \max_j Q(s', f_j))$
end while

A. The Case of "Ideal" ε-Greedy Algorithm

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 $(1-\delta)$, where $\delta = \varepsilon/2$, and f_0 with the probability of δ.

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 $(1-\delta)$ and remain in vertex 0 with the probability of δ. Thus, $Z(0) = (1-\delta) + \delta \cdot (1 + Z(0))$, from which by solving an equation we get:

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

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 $(1-\delta)$ and f_0 with the probability of δ. So we have four cases:

1) 0 to 1, f_0: the probability is $\delta(N-i)/N$, we remain in vertex i;
2) 0 to 1, f_1: the probability is $(1-\delta)(N-i)/N$, we go from vertex i to vertex $(i+1)$;
3) 1 to 0, f_0: the probability is $\delta i/N$, we go from vertex i to vertex $(i-1)$;
4) 1 to 0, f_1: the probability is $(1-\delta)i/N$, 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:

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

which is simplified to

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

The expected number of fitness evaluations for the algorithm, when started at x bits set to 1, is:

$$T_{R+\delta Q}(N,x) = \sum_{i=x}^{N-1} Z(i). \tag{3}$$

In the particular case of starting with $(N-1)$ bits, this turns to:

$$T_{R+\delta Q}(N, N-1) = Z(N-1). \tag{4}$$

In our previous work, we evaluated $T_{R+\delta Q}(N,0)$ experimentally on several values of N and δ [17]. The results are shown in Table I together with the exact values for $\delta = 0$.

TABLE I
VALUES OF $T_{R+\delta Q}(N)$ FOR DIFFERENT N AND δ: EXACT FOR $\delta = 0$, LOWER BOUNDS FOR $\delta > 0$

$\varepsilon \backslash N$	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 $\delta > 0$ the growth of lower bounds with N is close to an exponent. By trial and error we acquired an estimation $T_{R+\delta Q}(N) \approx 4\delta \times e^N \log N/N$ which holds with 10% of relative error for nearly all table entries.

In the current work, we theoretically estimate the lower bound for $T_{R+\delta Q}(N,x)$ by prooving the following theorem:

Theorem 1. $T_{R+\delta Q}(N,x) = \Omega\left(\left(1 + \frac{1}{1-\delta}\right)^N\right)$.

Proof: First, we construct a simple lower bound for $Z(i)$ for $0 \leq i < N$:

$$Z(i) = \frac{N + \delta \cdot i \cdot Z(i-1)}{(1-\delta)\cdot(N-i)} > \frac{N}{(1-\delta)\cdot(N-i)} > \frac{1}{1-\delta}. \tag{5}$$

Then, we define the boundary value $L = 1+\lceil N\cdot(1-\delta/2)\rceil$. Note that $L > N\cdot(1-\delta/2)$, and so $N-L < \delta\cdot N/2$. From these we conclude that:

$$\frac{\delta\cdot L}{(1-\delta)\cdot(N-L)} > \frac{\delta\cdot N\cdot(1-\delta/2)}{(1-\delta)\cdot\delta\cdot N/2} = \frac{2-\delta}{1-\delta} = 1+\frac{1}{1-\delta} > 1. \quad (6)$$

Using Eq. 6, we can deduce another lower bound for $Z(i)$, where $i \geq L$, which excludes another addend:

$$Z(i) = \frac{N+\delta\cdot i\cdot Z(i-1)}{(1-\delta)\cdot(N-i)} > \frac{Z(i-1)\cdot\delta\cdot i}{(1-\delta)\cdot(N-i)} \geq \frac{Z(i-1)\cdot\delta\cdot L}{(1-\delta)\cdot(N-L)} > Z(i-1)\cdot\left(1+\frac{1}{1-\delta}\right). \quad (7)$$

Finally, combining Eq. 5 and 7, we get:

$$Z(N-1) > Z(L-1)\cdot\left(1+\frac{1}{1-\delta}\right)^{N-L} > \frac{1}{1-\delta}\left(1+\frac{1}{1-\delta}\right)^{N-L}. \quad (8)$$

By definition of L, $L \geq 1+N\cdot(1-\delta/2)$ and $L < 2+N\cdot(1-\delta/2)$, from which follows $\delta\cdot N/2-2 < N-L \leq \delta\cdot N/2-1$, and we can conclude that $N-L = \Theta(N)$. From this and Eq. 4, 8 it finally follows that:

$$T_{R+\delta Q}(N, N-1) = Z(N-1) = \Omega\left(\left(1+\frac{1}{1-\delta}\right)^{N}\right). \quad (9)$$

If started with x bits set to 1, the algorithm never performs better, because, as follows from Eq. 3, $T_{R+\delta Q}(N,x) \geq T_{R+\delta Q}(N, N-1)$. ∎

B. A Lower Bound for Any RL Algorithm with Positive Lower Bound on the Probability of Selecting f_0

In this section, we prove that any reinforcement learning algorithm, which controls the RLS solving the modified ONEMAX problem with two fitness functions f_1 and f_0 and selects f_0 with the probability not less than δ, never performs better than the previously described strategy and, because of this, its running time is exponential. In other words, we prove that the previously described strategy is optimal.

To do that, we need to find a proper definition of *any reinforcement learning algorithm*, or even *any controlling algorithm*, which will be referred to as *strategy*. Such a strategy may make use of virtually anything, including time passed since the algorithm starts, external random number generator, and any knowledge of the history collected during the algorithm run. It cannot, however, access to the information hidden under fitness functions, such as which bits exactly are set to one in the individuals.

The scheme of the further analysis is as follows. We start with definitions to the *algorithm tree* and consider a strategy for a truncated algorithm tree, which is optimal by construction. Then we specify a certain version of this constructed strategy in Corollary 4 using a property from Theorem 2. Finally, in Theorem 5 we show that as the depth of the truncated tree approaches infinity, the considered version of the constructed optimal strategy fully coincides with the strategy from the previous section. Therefore, we prove that the strategy from the previous section is indeed the "ideal" strategy in the sense that the running time of the RLS controlled by it is the lower bound on the running times under the control of any RL algorithm with the probability for f_0 lower-bounded with δ.

Definition 1. *An **algorithm tree** is a tree whose nodes correspond to different sequences of decisions made during the algorithm run, and whose edges correspond to different decisions and have the probability of these decisions written on them.*

There are two types of decisions:

- selection of either f_0 or f_1 by the strategy;
- flipping of bit with the value of either 0 or 1 by the RLS.

For each algorithm iteration, these two events are independent, so we may consider them in either order, and it is more convenient for us to make the strategy to do the choice first, and then the RLS to make the flip. So the algorithm tree will have a layered structure with *even* layers corresponding to the nodes where the strategy does the choice, and *odd* layers where RLS does (the numeration of the layers starts with zero).

The even layers consist of the nodes of one type. Each of the nodes n have an associated *value* $v(n), 0 \leq v(n) \leq N$ equal to the number of bits set to one in the individual of RLS. These nodes have two children each, according to the choice of the strategy, except for the nodes with $v(n) = N$, because the algorithm stops when reaching these nodes.

The odd layers consist of the nodes of two types: *negative* nodes, which are the result of choosing f_0, and *positive* nodes, corresponding to the strategy choice of f_1. For a positive node n we define a value $v(n)$ to be x^+ if the individual has x bits set to one, and for a negative node the value will be x^-. These nodes have two children each, the node x^+ has the children with $v = x$ and $v = x+1$, the node x^- has the children with $v = x-1$ and $v = x$, except for 0^- and 0^+, which have only one child with $v = 0$ and $v = 1$, respectively.

The root of the algorithm tree is a vertex with the value of x, equal to the number of bits set to one when the algorithm starts. Some first layers of the tree with $x = 1$ are given in Fig. 2.

Note that in the same layer there may be several nodes with equal values, because these nodes correspond to different sequences of decisions that lead to the same number of ones in the individual, so the strategy may make different choices.

Definition 2. *A **probability decision strategy** is a strategy which for each node n selects the probability $\Delta(n)$ for selecting the f_0 for the RLS iteration (and $(1-\Delta(n))$ for the f_1) and then takes the decision randomly with the selected probability.*

Definition 3. *A **δ-bounded strategy** is a probability decision strategy in which for some $\delta > 0$, for all n holds $\Delta(n) \geq \delta$.*

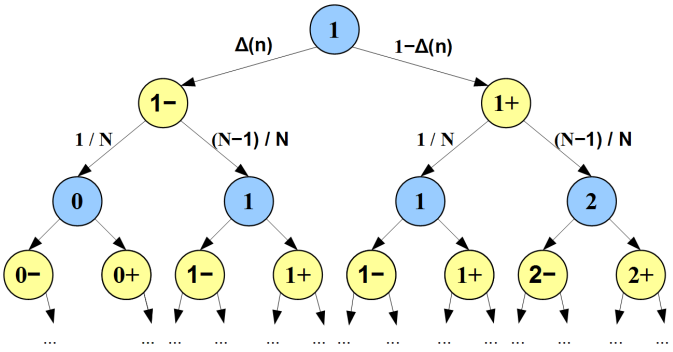

Fig. 2. The algorithm tree

In the rest of this section, we limit ourselves to δ-bounded strategies, although some of the statements may be extended to a more general class of strategies.

Definition 4. *The **expected subtree depth** $E(n)$ for a node n in the algorithm tree is the expected number of edges to visit until a node with the value of N is visited.*

It is clearly seen that for any node n with $v(n) = N$ holds $E(n) = 0$, and for any nonleaf node $E(n) = 1 + p_l \cdot E(l) + p_r \cdot E(r)$, where l and r are left and right child nodes, p_l and p_r are the probabilities of going to the left and the right child node, respectively.

The running time of an algorithm is $E(root)/2$, because each iteration of the algorithm corresponds to two edges in the algorithm tree. So a strategy minimizing the value of $E(root)$ is optimal.

The next step can be to construct an optimal δ-bounded strategy using the algorithm tree by setting the probabilities for each node n in the even layers that minimize $E(n)$. We can do it using induction by the algorithm tree, but only if the tree had a finite depth (this is not to be confused with $E(root)$, which is the expected depth in the space of all possible decisions), which is typically not true.

The approach that will be used is to consider a *truncated algorithm tree* for a certain depth D, construct the optimal strategy for that tree, and then see what happens if $D \to \infty$.

Definition 5. *The **truncated algorithm tree** with the depth D is the algorithm tree in which all nodes in the layers with the depth exceeding D are removed.*

Definition 6. *The **expected subtree depth** $E(n)$ for a node n in the **truncated** algorithm tree with the depth D is the expected number of edges to visit until a node with either the value of N **or the depth** D is visited.*

The algorithm to construct an optimal δ-bounded strategy for a truncated algorithm tree with the depth D can be outlined as follows:

- start with the deepest layer and proceed with the higher ones;
- for odd layers, simply compute $E(n)$ for each node n;
- for even layers, we first need to determine the optimal $\Delta(n)$ for each node n. As $E(n)$ is linear in $\Delta(n)$, we can consider the boundary values only: $\Delta(n) = 1$ and $\Delta(n) = \delta$. Let E^+ be the expected subtree depth for the positive children node of n, and E^- for the negative one. Then:
  - if $E^- > E^+$, then $\Delta(n) = \delta$;
  - if $E^- < E^+$, then $\Delta(n) = 1$.
  - if $E^- = E^+$, then set any $\Delta(n) \in [\delta, 1]$.

  Note that in the case of $E^- = E^+$, $\Delta(n)$ can be set to any value.

It is important to note that the above strategy is optimal by the construction. Actually, it is sufficient to always set $\Delta(n) = \delta$, because of the property formulated in the following theorem.

Theorem 2. *In any optimal δ-bounded strategy for a truncated algorithm tree with the depth of D, it is always true that $E^- \geq E^+$.*

Proof: Instead of the theorem, we will prove a stronger lemma:

Lemma 3. *The following statements hold for any optimal δ-bounded strategy for a truncated algorithm tree:*

1) *In each even layer, for any two nodes from that layer a and b it holds that if $v(a) \leq v(b)$, then $E(a) \geq E(b)$. This implies that if $v(a) = v(b)$ then $E(a) = E(b)$.*
2) *In each odd layer, for any two nodes from that layer a and b it holds that:*
   a) *if $v(a) = v(b)$, then $E(a) = E(b)$;*
   b) *if $v(a) = x^-$ and $v(b) = x^+$, then $E(a) \geq E(b)$;*
   c) *if $v(a) = x^-$ and $v(b) = (x+1)^-$, then $E(a) \geq E(b)$;*
   d) *if $v(a) = x^+$ and $v(b) = (x+1)^+$, then $E(a) \geq E(b)$.*

Proof:

The proof uses the induction by the tree, starting with the deepest layer.

The base. Consider two cases:

1) D is even. Then the deepest layer is the even layer. For every node n in that layer $E(n) = 0$, so the first induction statement holds. The second induction statement is not applicable for this case, and so is considered to be true.
2) D is odd. The deepest layer is the odd layer. For every node n in that layer $E(n) = 0$, so the second induction statement holds. The first induction statement is not applicable, and so is considered to be true.

The induction. Consider two cases:

1) The level is even. We need to prove the first induction statement, and we know by induction that the second induction statement holds for the next level.
   a) Consider a node n with the value of $x < N$. Its children are nodes n^- with the value of x^- and n^+ with the value of x^+. By induction, we know that $E(n^-) \geq E(n^+)$. Due to the construction algorithm, $\Delta(n) = \delta$ should be selected if

$E(n^-) > E(n^+)$, or any value may be selected if the equality holds.

b) Consider two nodes n and m with the values less than N and their children n^+, n^-, m^+ and m^-. If $v(n) = v(m)$, then $v(n^+) = v(m^+)$ and $v(n^-) = v(m^-)$, so by induction $E(n^+) = E(m^+)$ and $E(n^-) = E(m^-)$. Because the same formula is used for both n and m, thus $E(n) = E(m)$.

c) If $v(n) = v(m) + 1$, then $v(n^+) = v(m^+) + 1$ and $v(n^-) = v(m^-) + 1$. By induction, $E(n^+) \geq E(m^+)$ and $E(n^-) \geq E(m^-)$. Because the same formula is used for both n and m, but for n both components are not smaller than for m, so $E(n) \geq E(m)$.

d) If $v(n) < v(m)$, then $E(n) \geq E(m)$ can be proved by repeatedly applying the statement from (c).

e) For any node n with $v(n) = N$ $E(n) = 0$, while for any node m with $v(m) < N$ $E(m) > 0$. So any two nodes with $v = N$ have equal expected depths, and any node with $v = N$ has smaller expected depth than any node with $v < N$.

The first induction statement follows from the statements proved above.

2) The level is odd. We need to prove the second induction statement, and we know by induction that the first induction statement holds for the next level.

a) There are the separate cases for vertices with the values of 0^- and 0^+. For any node n for which $v(n) = 0^-$, there is only one child n^+ with $v(n^+) = 0$, so $E(n) = 1+E(n^+)$. For any node m for which $v(m) = 0^+$, there is only one child m^+ with $v(m^+) = 1$, so $E(m) = 1 + E(m^+)$, and because by induction $v(m^+) > v(n^+)$, $E(n) \geq E(m)$. Note that, by induction, the expected depths for all 0^+ nodes are the same and for all 0^- nodes are the same.

b) Consider any two nodes n and m where $v(n) = v(m) = x^+$ where $0 < x < N$. The node n has the children n^- with $v(n^-) = x$ and n^+ with $v(n^+) = x + 1$. The node m has the children m^- with $v(m^-) = x$ and m^+ with $v(m^+) = x + 1$. By induction, $E(n^-) = E(m^-)$ and $E(n^+) = E(m^+)$. As $E(n)$ and $E(m)$ are computed using the same formula, so $E(n) = E(m)$.

c) For nodes n and m where $v(n) = v(m) = x^-$, where $0 < x < N$, the proof of $E(n) = E(m)$ follows the same idea as in (b).

d) Consider any two nodes n and m where $v(n) = x^+$ and $v(m) = x^-$. The node n has the children n^- with $v(n^-) = x$ and n^+ with $v(n^+) = x + 1$. The node m has the children m^- with $v(m^-) = x - 1$ and m^+ with $v(m^+) = x$. As $E(n)$ and $E(m)$ are computed using the same formula, but $E(n^-) \leq E(m^-)$ and $E(n^+) \leq E(m^+)$ by induction, so $E(n) \leq E(m)$.

e) Consider any two nodes n and m where $v(n) = 0^+$ and $v(m) = 1^+$. The node n has the child n^+ with $v(n^+) = 1$. The node m has the children m^- with $v(m^-) = 1$ and m^+ with $v(m^+) = 2$. By induction, $E(n^+) = E(m^-)$ and $E(m^+) \leq E(m^-)$. As $E(n) = 1 + E(n^+)$ and $E(m) = 1 + E(m^-) \cdot 1/N + E(m^+) \cdot (N - 1)/N$, so $E(n) \geq E(m)$.

f) For nodes n and m where $v(n) = 0^-$ and $v(m) = 1^-$ the proof follows the same idea as in (e).

The second induction statement follows from the statements proved above.

So both the base and the induction step are proved, thus proving the lemma. ■

The theorem itself follows from the second component of the second part of the lemma. ■

Corollary 4. *The δ-bounded strategy for a truncated algorithm tree with the depth of D, where for any n $\Delta(n) = \delta$, is optimal.*

Proof: It was proved that for any node in an even layer, in the terms of the construction algorithm, $E^- \geq E^+$. So we must choose $\Delta(n) = \delta$ if $E^- > E^+$ and we may safely choose the same value if $E^- = E^+$. ■

Theorem 5. *A reinforcement learning algorithm described in Section III-A is an optimal δ-bounded strategy for an algorithm tree.*

Proof: Let us define the $E(root)$ for the optimal δ-bounded strategy for a truncated algorithm tree of depth D as $E_{low}(D)$. One can see that $E_{low}(D)$ is a monotonically increasing function over its argument D.

Consider a globally optimal δ-bounded strategy for a complete algorithm tree. Let us define as $E_{opt}(D)$ the expected depth for this globally optimal δ-bounded strategy when running it on the truncated algorithm tree of depth D. Clearly, $E_{opt}(D) \geq E_{low}(D)$. So the expression

$$E_{low} = \lim_{D \to \infty} E_{low}(D) \tag{10}$$

is a lower bound on E_{opt}, which is the expected depth for the globally optimal strategy on the (not truncated) algorithm tree.

However, the expected depth for the strategy from Section III-A is equal to E_{low}, because for all truncated algorithm trees it makes the same decisions as the optimal δ-bounded strategy from Corollary 4 for that tree in all its nodes. So from $E_{low} \leq E_{opt}$, which comes from the definition of optimality, and from $E_{low} \geq E_{opt}$, which follows from the fact that E_{low} is taken from an existing strategy, follows $E_{low} = E_{opt}$, which proves the theorem. ■

Therefore, there is no δ-bounded strategy which performs better than the strategy described in Section III-A, so any selection algorithm with δ-bounded strategy solves the considered problem in exponential time.

IV. Discussion

In the previous section, it is proven that using ε-greedy exploration strategy, it takes exponential time to find the optimum of the OneMax problem with an obstructive objective. At the same time, it was previously shown that this problem can be solved in $\Theta(N \log N)$ time using greedy exploration strategy, i. e. strategy with $\varepsilon = 0$. So the greedy strategy is proven to be significantly better than the ε-greedy strategy for the considered problem.

A. Fixed Budget Perspective May Differ

For a number of benchmark problems, the previously obtained experimental results are consistent with the fact that using Q-learning with greedy strategy ($\varepsilon = 0$) is more efficient than using some fixed nonzero ε [14]. At the same time, according to the experimental results for a harder problem of test case generation, ε-greedy strategy may outperform the greedy one [15]. This can be explained in the following way. In the test case generation problem, the aim of the optimization was not to find the real optimum, but to find a solution with the fitness better than some fixed value. So the optimization process was not run until the end. It appears that in the considered early stages of optimization ε-greedy strategy was efficient.

In the OneMax problem, the most difficult part of the optimization begins near the optimum, when it is hard to find the proper bit to be inverted, since nearly all the bits are already of value one. The same is true for the most previously considered problems. The exponential part in the running time appears from those late stages of optimization.

Therefore, despite of its asymptotically exponential behaviour, ε-strategy may outperform the greedy one in the early optimization stages, as the experiment results for the MH-IFF problem with an obstructive auxiliary objective suggest [14]. Thus, theoretical analysis from the fixed budget perspective is needed [20], as it would soundly show the performance of the ε-strategy at the different stages of optimization.

B. What if ε Depends on n?

The proofs in Sections III-A and III-B give a strong evidence that any algorithm, not necessarily a reinforcement learning one, that controls the RLS and leaves at least the fixed probability to select the fitness function f_0, performs exponentially bad.

However, looking at the structure of the proof from Section III-A, we can see that if $\varepsilon = O(1/N)$, then the proof stops working. It may be the case that there exists a function $F(N)$, such that if $\varepsilon = O(F)$, then the ε-greedy algorithm starts working in polynomial time.

V. Conclusion

In this paper, the following results are proved:

- Random Local Search, controlled with Q-learning algorithm with ε-greedy exploration strategy for which all states are already learned, solves modified OneMax problem using the following expected number of fitness function calls:
$$T_{R+\varepsilon Q}(N, N-1) = \Omega\left(\left(1 + \frac{1}{1-\varepsilon/2}\right)^N\right),$$
starting from exactly one bit set to zero.
- The previous result is a lower bound for all possible randomized strategies (not necessary ε-greedy reinforcement learning), which select the f_0 function with the probability of at least $\varepsilon/2$.

However, despite its asymptotically exponential performance, ε-greedy strategy may be efficient in the early stages of optimization process, as the previously seen experimental results suggest. What is more, some adaptive scheme of setting ε as a function of the problem size N may be tried in order to solve the considered problem in a polynomial time.

VI. Ackowledgments

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

References

[1] C. Segura, C. A. C. Coello, G. Miranda, and C. León, "Using multi-objective evolutionary algorithms for single-objective optimization," *4OR*, vol. 11, no. 3, pp. 201–228, 2013.

[2] S. J. Louis and G. J. E. Rawlins, "Pareto optimalityga-easiness and deception (extended abstract)," in *ICGA*, S. Forrest, Ed. Morgan Kaufmann, 1993, pp. 118–123.

[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] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler, "On the effects of adding objectives to plateau functions," *IEEE Trans. Evolutionary Computation*, vol. 13, no. 3, pp. 591–603, 2009.

[5] T. Brys, A. Harutyunyan, P. Vrancx, M. E. Taylor, D. Kudenko, and A. Nowé, "Multi-objectivization of reinforcement learning problems by reward shaping," in *2014 International Joint Conference on Neural Networks*, 2014, pp. 2315–2322.

[6] 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.

[7] 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.

[8] T.-D. Tran, D. Brockhoff, and B. Derbel, "Multiobjectivization with NSGA-II on the noiseless BBOB testbed," in *GECCO (Companion)*, C. Blum and E. Alba, Eds. ACM, 2013, pp. 1217–1224.

[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. 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.

[11] D. F. Lochtefeld and F. W. Ciarallo, "Deterministic helper-objective sequences applied to job-shop scheduling," in *GECCO*, M. Pelikan and J. Branke, Eds. ACM, 2010, pp. 431–438.

[12] ——, "Helper-objective optimization strategies for the job-shop scheduling problem," *Appl. Soft Comput.*, vol. 11, no. 6, pp. 4161–4174, 2011.
[13] ——, "Multiobjectivization via helper-objectives with the tunable objectives problem," *IEEE Trans. Evolutionary Computation*, vol. 16, no. 3, pp. 373–390, 2012.
[14] 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.
[15] M. Buzdalov and A. Buzdalova, "Adaptive selection of helper-objectives for test case generation," in *2013 IEEE Congress on Evolutionary Computation*, vol. 1, June 20-23 2013, pp. 2245–2250.
[16] ——, "OneMax Helps Optimizing XdivK: Theoretical Runtime Analysis for RLS and EA+RL," in *Proceedings of Genetic and Evolutionary Computation Conference Companion*. ACM, 2014, pp. 201–202.
[17] M. Buzdalov, A. Buzdalova, and A. Shalyto, "A first step towards the runtime analysis of evolutionary algorithm adjusted with reinforcement learning," in *Proceedings of the International Conference on Machine Learning and Applications, ICMLA 2013*. IEEE Computer Society, 2013, to be published.
[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[19] 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.
[20] T. Jansen and C. Zarges, "Fixed budget computations: A different perspective on run time analysis," in *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '12, 2012, pp. 1325–1332.