Can OneMax Help Optimizing LeadingOnes using the EA+RL Method?

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

***Abstract*—There exist optimization problems with the target objective, which is to be optimized, and several extra objectives, which can be helpful in the optimization process. The EA+RL method is designed to control optimization algorithms which solve problems with extra objectives. The method is based on the use of reinforcement learning for adaptive online selection of objectives.**

In this paper we investigate whether OneMax helps to optimize LeadingOnes when the EA+RL method is used. We consider LeadingOnes+OneMax problem where the target objective is LeadingOnes and the only extra objective is OneMax.

The following theoretical results are proven for the expected running times when optimization starts from a random vector in the case of randomized local search (RLS): $n^2/2$ for LeadingOnes, $n^2/3$ for LeadingOnes+OneMax when reinforcement learning state is equal to the LeadingOnes fitness or when random objective selection is performed, and $n^2/4+o(n^2)$ when there is one reinforcement learning state and the greedy exploration strategy is used. The case of starting with all bits set to zero is also considered.

So, OneMax helps, although not too much, to optimize LeadingOnes when RLS is used. However, it is not true when using the $(1+1)$ evolutionary algorithm, which is shown experimentally.

I. Introduction

Single-objective optimization can benefit from multiple objectives [1]–[4]. Different approaches are known from the literature. Some researchers introduce additional objectives to escape from the plateaus [5]. Decomposition of the primary objective into several objectives also helps in many problems [2], [3], [6]. Additional objectives may also arise from the problem structure [7], [8].

Different approaches may be applied to a problem with the "original" objective, which can be called the *target* objective, and some extra objectives. The *multi-objectivization* approach is to optimize *all* extra objectives at once using a multi-objective optimization algorithm [3], [6]. The *helper-objective* approach is to optimize simultaneously the target objective and some (not necessarily all, in some cases, only one is preferrable) extra objectives, switching between them from time to time [9].

The approaches above are designed in the assumption that extra objectives are crafted to help optimizing the target objective. However, this is not always true, especially when extra objectives are generated automatically [10], or their properties are unknown. In fact, extra objectives may support or obstruct the process of optimizing the target objective. The EA+RL method was developed to cope with such situations [11]. The idea of this method is to use a single-objective optimization algorithm and switch between the objectives (which include the target one and the extra ones). To find the most suitable objective for the optimization, reinforcement learning algorithms are used [12]–[14]. The EA+RL method can also be adapted to select the second objective for the helper-objective approach [10].

Some initial theoretical explorations of the EA+RL method are made for the case of an obstructive extra objective [15] and a fine-grained version of the target objective as an extra objective [16]. In [17] an objective selection method is proposed which delivers the running time proportional to the minimum of running times for the objectives.

This paper deals with the case of a supporting extra objective. The functions LeadingOnes and OneMax are chosen to be the object of research. The reason is that they are both simple to optimize by evolutionary algorithms with one individual, and they have only slightly differing computation complexities for this class of optimization algorithms: $\Theta(n \log n)$ for OneMax [18] and $\Theta(n^2)$ for LeadingOnes [19].

The main results of the paper are:

- when using randomized local search (RLS) as an optimizer, a single reinforcement learning state, and a greedy Q-learning agent, the running time of EA+RL on LeadingOnes+OneMax is $n^2/4 + o(n^2)$;
- when reinforcement learning states are determined by the value of LeadingOnes fitness, or when random reinforcement learning agent is used, the running time is $n^2/3 + o(n^2)$;
- if the $(1+1)$ evolutionary algorithm is used, the running time is almost always asymptotically worse than the running time of optimizing just LeadingOnes.

The rest of the paper is structured as follows. In Section II, we give some necessary definitions. In Section III, the running time of the randomized local search on LeadingOnes is analyzed in the case of random initialization. In Section IV, the same is done for the EA+RL method on LeadingOnes+OneMax. Section V is dedicated to the worst-case expected time analysis of both algorithms. Sec-

978-1-4799-7492-4/15/$31.00

tion VI contains comparison of the results for random and worst-case initializations in the case of the randomized local search, as well as the experimental evaluation of the $(1+1)$ evolutionary algorithm. The net result is that, in the case of single bit mutations, ONEMAX helps optimizing LEADINGONES, while in the case of independent bit mutations it does not help. Section VII concludes.

II. DEFINITIONS

The ONEMAX problem is often used in theoretical research on evolutionary algorithms [18]. The search space consists of all bit vectors of length n. The fitness function of a bit vector is the number of bits set to one. One needs to maximize the fitness function.

The LEADINGONES problem is also defined on bit vectors [19]. However, the fitness function is the length of the maximal prefix consisting of bits set to one.

We consider two simplistic optimization algorithms. The first one called "randomized local search", or simply RLS. It stores a bit vector x — the current candidate solution. In the beginning of the algorithm, x is initialized in some way — this paper assumes that the initial vector contains only zero bits. An iteration of this algorithm works this way:

- $y \leftarrow x$ with exactly one random bit flipped (mutation);
- if $f(y) \geq f(x)$ then $x \leftarrow y$.
- if $f(x)$ is the maximum, terminate.

Note that if $f(x) = f(y)$, the previous solution is discarded.

The second algorithm is called the "$(1+1)$ evolutionary algorithm", or the $(1+1)$-EA. The only difference between RLS and this algorithm is that the mutation is performed the following way: each bit is flipped with the probability of p.

Markov chains are often used in the proofs presented in this paper. We will distinguish between the states in the context of reinforcement learning, which we call *RL states*, and the states of Markov chains, which we call *Markov states*.

III. AVERAGE CASE RLS ANALYSIS

This section is dedicated to running time analysis of RLS on the LEADINGONES problem.

In the paper by Böttcher, Doerr and Neumann [19] the exact running time of the $(1+1)$ evolutionary algorithm for LEADINGONES problem was computed. It was shown to be equal to:

$$\frac{1}{2p^2}\left((1-p)^{1-n} - (1-p)\right), \tag{1}$$

where p is the probability of mutating each bit. To prove the results for RLS on LEADINGONES, we use the ideas of the proof from this paper. This section is structured as follows: first, we outline the proof from [19], and then we prove the similar formula for the running time of RLS.

A. Outline of the Original Proof for the (1+1)-EA

The authors of [19] assume that the optimization process is started from a random individual, i.e. every bit is set to one with the probability of $1/2$. Let us have the length of the bit vector equal to n. A Markov chain with $n+1$ states is used to model the optimization process. A state S_i of this Markov chain is defined as follows: first $(n-i)$ bits of the bit vector are set to one, the next one is set to zero, and $(i-1)$ bits in suffix are set to one independently with the probability of $1/2$. Thus, the LEADINGONES fitness of state S_i is $(n-i)$.

Each mutation, which is flipping each bit with the probability of p, retains the $(i-1)$ last bits to be of the same distribution, and may optionally flip other bits. If some of the first $(n-i)$ bits are flipped, the fitness is decreased, so mutations of this sort are not accepted. If first $(n-i+1)$ bits are not flipped, the state is not changed. The probability of fitness increase is $(1-p)^{n-i}p$, and the expected time of waiting for an improvement is $A_i = \frac{1}{(1-p)^{n-i}p}$. The probability that first k bits in the $(i-1)$ last bits are set to one is 2^{-k} for all $k < i-1$, so the expected running time starting from Markov chain state S_i until the optimization stops is

$$T_i = A_i + \sum_{j=1}^{i-1} 2^{j-i} T_j,$$

which is simplified down to

$$T_i = A_i + \frac{1}{2}\sum_{j=1}^{i-1} A_j.$$

We can compute the expected running time for the random initialization using the expression:

$$T = \frac{1}{2}\sum_{j=1}^{n} A_j.$$

This works by assigning the probability of $1/2$ to T_n, the probability of $1/4$ to T_{n-1} and so on, finally getting the same expression as for T_{n+1} with A_{n+1} set to zero. Finally, (1) is constructed from the expression for T by substituting $A_j = \frac{1}{(1-p)^{n-j}p}$.

B. Proof Update for RLS

The proof above can be easily updated to work with RLS instead of the (1+1)-EA. First, if each of the $(i-1)$ last bits is set to one independently with the probability of $1/2$, a single-bit mutation applied to these bits yields the same distribution, as it just switches the probability $p = 1/2$ to $1 - p = 1/2$. So, the expression for the expected running time T_i actually works in this case given A_i is computed correctly. However, for a single bit mutation, A_i is simply equal to n.

Finally, the expected running time for a random start is $T = \frac{1}{2}\sum_{j=1}^{n} n = \frac{n^2}{2}$, and the running time for T_n is $\frac{n(n+1)}{2}$.

IV. AVERAGE CASE EA+RL ANALYSIS

In this section we present the result of theoretical analysis for the EA+RL algorithm applied to the LEADINGONES+ONEMAX problem and different configurations of reinforcement learning.

In Section IV-A, the notation used in reinforcement learning algorithms is described. In Sections IV-B and IV-C, the proofs of the expressions for expected running time in different configurations with random initialization are given.

A. Reinforcement Learning

Reinforcement learning [12]–[14] uses the concepts of *state*, *action* and *reward*. A reinforcement learning algorithm is often thought to control an *agent* which interacts with a certain *environment*. The agent receives the current state from the environment as input. It should return an action to apply on the environment. For that action, it receives a reward. The aim of the reinforcement learning algorithm is to maximize the total reward by choosing appropriate actions in different states. The total reward can be treated as a sum of all rewards received by the algorithm, or as a *discounted* reward, when a reward for i-th step from the end is taken with a weight of γ^i, where $0 < \gamma < 1$.

In the EA+RL method, actions are objectives to choose, while states and rewards are defined depending on the problem. In this paper, the reward is always defined as the value of the target objective after the selection of an objective minus the value of the target objective before it. The sum of all these rewards is roughly equal to the final value of the target objective, so optimization of the reward leads to optimization of the target objective.

In this paper we consider a *random agent* — a somewhat degenerate reinforcement learning algorithm, which selects the actions at random. In some situations, however, every unbiased reinforcement learning algorithm (that is, an algorithm that chooses actions at random when there is no information about the expected rewards in the current state) behaves like the random agent. We also consider another reinforcement learning algorithm, the Q-learning algorithm with greedy exploration strategy [13], which we call a *greedy agent*.

In the rest of the paper, we often distinguish between the states in the context of reinforcement learning, which we call *reinforcement learning states*, and the states of Markov chains, which we call *Markov states*.

B. RLS, N+1 States

We consider the randomized local search (RLS) controlled by reinforcement learning. The target objective is LEADINGONES, and the extra objective is ONEMAX. There are $N + 1$ reinforcement learning states which correspond to the target objective values.

The reinforcement learning algorithm to use is not specified above. This is because the following lemma holds for the considered problem:

Lemma 1. *The EA+RL algorithm never returns to any state where some reward has been obtained.*

Proof. Consider a certain reinforcement learning state S_i, which corresponds to the LEADINGONES fitness being equal to i. There are only three possibilities for a single-bit mutation:

- Flip one of the first i bits. This decreases both the LEADINGONES and the ONEMAX fitnesses, so such mutation is always rejected. The reinforcement learning state is not changed, no reward is obtained.
- Flip the $(i + 1)$-th bit. This increases both fitnesses, so is always accepted. The positive reward is obtained, so the estimation of the reward in the current state is updated, but the state is changed to S_{i+1}.
- Flip one of the last $(n-i-1)$ bits. This does not change the LEADINGONES fitness and may alter the ONEMAX fitness. The reinforcement learning state is not changed, no reward is obtained.

To sum up, in the only case when some nonzero reward is gained, the state is changed to a previously unvisited one, so at any time the reinforcement learning agent must decide from no experience. □

If a reinforcement learning algorithm chooses actions at random when there is no reward obtained in the current state, it is equivalent to a random agent for this problem. We assume further in this section that the LEADINGONES fitness function is selected with the probability of q, and the ONEMAX fitness function is selected with the probability of $(1 - q)$.

In this section we try to follow the ideas of the proof from [19]. To do that, we need to invent some distribution of the tail of unknown bits which will be kept during mutations of the selected type and selection based on both target and auxiliary objectives. Fortunately, it is possible to do it in this case in a simple manner.

Lemma 2. *Assume that an individual has the form of $1^t0?^{n-t-1}$, where a symbol ? refers to unknown bits which are set to one independently with the probability of $p = 1/(1+q)$. After flipping a random bit and applying* LEADINGONES *with the probability of q and* ONEMAX *with the probability of $1-q$ the resulting individual will have a form of either $1^u0?^{n-u-1}$ or 1^n, where bits denoted by ? have the same probability of being set to one as above.*

Proof. Consider a bit flip. If a bit from the first group of ones is flipped, the mutation will be discarded. Otherwise, there are three possible situations:

- The bit was equal to one (so after mutation it is equal to zero) and LEADINGONES is selected. The result is zero, the probability is pq.
- The bit was equal to one and ONEMAX is selected. Then the mutation is discarded, the result is one, and the probability is $p(1 - q)$.
- The bit was equal to zero (so flipped to one). The result is one, the probability is $(1 - p)$.

In the latter case, the flipped bit could be the first zero bit. In this case, the first group of ones is extended until a first zero bit is found (then the remaining bits retain their original probability of being set to one), or until all bits are found to be one.

In all other cases, all unknown bits except for the flipped bit retain their probability of being equal to one, and the flipped bit is set to one with the probability $p(1-q)+(1-p) = 1-pq$, which is equal to p. □

By repeating the proof from [19] for this new probability, we get the following result:

Theorem 1. *If* LEADINGONES *is selected with the probability of* q *and* ONEMAX *with the probability of* $(1-q)$*, and if each bit is initially set to one independently with the probability of* $p = 1/(1+q)$*, then the expected running time starting with the* LEADINGONES *fitness equal to* $(n-i)$ *is* $T_i = n + n \cdot (i-1) \cdot \frac{q}{1+q}$*, and the average expected running time is* $n^2 \frac{q}{1+q}$*.*

Proof. For a sequence of t random bits set to one independently with the probability of p, the probability of first $u < t$ bits being equal to one and the next one being equal to zero is $p^u(1-p)$. So we have the following expression for T_i, $i > 0$:

$$T_i = A_i + \sum_{j=0}^{i-1} (1-p) p^{j-i-1} T_j.$$

By induction, we show that:

$$T_i = A_i + (1-p) \sum_{j=1}^{i-1} A_j.$$

The base for $i = 1$ is:

$$T_1 = A_1 + (1-p) \sum_{j=1}^{0} A_j = A_1.$$

Assume that, if $i > 1$, the induction statement holds for T_{i-1}. Then the induction statement can be proven for T_i the following way:

$$\begin{aligned}
T_i &= A_i + \sum_{j=0}^{i-1} (1-p) p^{j-i-1} T_j = \\
&= A_i + (1-p) T_{i-1} + p \sum_{j=0}^{i-2} (1-p) p^{j-i-2} T_j = \\
&= A_i + (1-p) T_{i-1} + p \left(T_{i-1} - A_{i-1} \right) = \\
&= A_i + T_{i-1} - p A_{i-1} = \\
&= A_i + \left(A_{i-1} + (1-p) \sum_{j=1}^{i-2} A_j \right) - p A_{i-1} = \\
&= A_i + (1-p) \sum_{j=1}^{i-1} A_j.
\end{aligned}$$

By substituting $A_i = n$ and $p = 1/(1+q)$, we get:

$$T_i = n + n(i-1) \frac{q}{1+q}.$$

The average running time is computed as explained in the end of Section III-A: in the expression for T_{n+1} the value of A_{n+1} is substituted by zero. □

This gives the same results as in [19] if $q = 1$, i.e. if we do not select ONEMAX at all. For the most common case, the probability of selecting LEADINGONES $q = 1/2$, so the probability of each bit from the tail to equal one is $p = 2/3$. In this case, the running time of the EA+RL algorithm using RLS on LEADINGONES+ONEMAX is $n^2/3$ in average and $n + n(n-1)/3$ from zero LEADINGONES fitness, which is asymptotically 1.5 times faster than RLS on LEADINGONES.

However, the results for RLS on LEADINGONES are given in the assumption that the unknown bits in the tail are equal to one with the probability of $1/2$, and the result for EA+RL using RLS on LEADINGONES+ONEMAX are given in the assumption of the same probability equal to $2/3$. Fortunately, further experiments show that the same ratio between the results is true in the case of the worst-case start for n being sufficiently large.

In the next sections we focus on the particular probability of random selection for LEADINGONES equal to $1/2$.

C. RLS, One State

In this section we consider the same setup as in the previous section, but with only one reinforcement learning state.

1) Random Agent: It is clear that for the reinforcement learning agent that selects the actions at random the behavior does not depend on the number of states, so the results will be the same: $n^2/3$ in average, $n + n(n-1)/3$ for the case of the initialization with zero LEADINGONES fitness. However, the probability of $2/3$ for each bit to equal one is implied.

2) Greedy Agent: Consider the greedy agent, whose definition is given in Section IV-A. In this situation, the reinforcement learning agent will select fitness functions at random until the first increase of the LEADINGONES function happens. The expected number of steps until this happens is n. Note that there will never be a decrease of the LEADINGONES function, because when the single-bit mutation is used, such a decrease will also result in the decrease of ONEMAX.

After the first increase of LEADINGONES, the fitness function that has just been selected will be used for the entire optimization process. That is, with the probability of $1/2$ LEADINGONES will be used, and the total optimization time will be at most $n + n(n-1)/2$. With the probability of another $1/2$ ONEMAX will be used. This happens because, due to the single-bit mutation, no choice of the fitness function yields negative reward, so whatever is the initial choice, it will be reinforced.

Assuming the probability of $1/2$ for each bit to be one, the upper bound on the expectation number of steps is:

$$T \leq n + \frac{1}{2} \left(\frac{n(n-1)}{2} + \sum_{i=1}^{n-1} \frac{n}{i} \right) \approx \frac{n^2}{4} + o(n^2). \quad (2)$$

This approaches $n^2/4$ as n grows. It can be said that this algorithm is roughly twice as fast as RLS on LEADINGONES. However, due to the fact that with the probability of $1/2$ it works the same way as RLS on LEADINGONES, the distribution of the number of steps has two peaks, one centered at roughly $n^2/2$ and one at $n \log n$.

V. WORST CASE RUNNING TIME

The proofs in the previous sections share a single drawback — they show the expected running time in the average case, not in the worst case. The expression for the state S_n, which corresponds to the minimum fitness value, show the

expected running time for the initialization which has the first bit set to zero and other bits initialized at random. However, the worst case is initialization with all zeros. In this case, last $n-1$ bits are not initialized at random, so the proofs cannot be applied.

The property that the proof from [19] relies on the random distribution of the tail also makes it difficult to adapt the proof to more general cases like the one in this paper, where, due to the presence of extra fitness functions, the distribution of bits in the tail may significantly differ from the random one.

In this section, we explain the method of computation of the worst-case expected running time for RLS optimizing LEADINGONES and for EA+RL with random agent optimizing LEADINGONES+ONEMAX. This method, however, can be extended by certain modifications to work with some other optimizers.

The idea to use a two-dimensional Markov chain: S_i^j is the state which corresponds to the bit vectors where the first $n-i$ bits are set to one, the next bit is set to zero, and from the last $i-1$ bits there are j bits set to one. All bit vectors matching this description have the same probability and belong to the same Markov state. For each state, we define the probabilities of moving to certain other states, and finally compute T_i^j — the expected number of steps needed to reach the terminal state.

Theorem 2. *Consider all possible bit vectors of length m with $k < m$ bits set to one, each taken with equal probability. The probability that the first $i \leq k$ bits are set to one and the next one is set to zero is:*

$$\frac{\binom{m-i-1}{m-k-1}}{\binom{m}{k}}.$$

Proof. There are $\binom{m}{k}$ equiprobable bit vectors. We want to count such vectors that the first i bits are set to one and the next bit is set to zero. After these bits, there are $m-i-1$ bits in the tail, from which there are $m-k-1$ zeros. So there are $\binom{m-i-1}{m-k-1}$ vectors of this type. □

When using the single-bit mutation, there is no chance for the LEADINGONES fitness to decrease. The probability of this fitness to increase is $1/n$, which is the probability of flipping the first zero bit. After this happens, there may be some other bits set to one after the flipped bit. The probability of t extra bits set to one follows from Theorem 2.

Let's define the probability of going from S_i^j to S_i^{j+1} as $P_{i,j}^+$, and the probability of going from S_i^j to S_i^{j-1} as $P_{i,j}^-$. Then the expected running time from the state S_i^j, $j < i-1$, can be written as follows:

$$T_i^j = \frac{1 + P_{i,j}^+ T_i^{j+1} + P_{i,j}^- T_i^{j-1} + \frac{1}{n} \sum_{x=i-j+1}^{i-1} V_{i,j,x}}{P_{i,j}^+ + P_{i,j}^- + \frac{1}{n}}, \quad (3)$$

where:

$$V_{i,j,x} = \frac{\binom{x-1}{i-j-2}}{\binom{i-1}{j}} T_x^{j-i+x+1}, \quad (4)$$

and for S_i^{i-1}:

$$T_i^{i-1} = \frac{1 + P_{i,j}^- T_i^{i-2}}{P_{i,j}^- + \frac{1}{n}}. \quad (5)$$

For RLS solving the LEADINGONES problem, $P_{i,j}^+ = (i-j-1)/n$ and $P_{i,j}^- = j/n$, because in the first case we need to flip one of the $(i-j-1)$ zero-bits in the tail, and in the second case we need to flip one of the j one-bits.

For EA+RL using RLS with the random agent solving the LEADINGONES+ONEMAX problem, $P_{i,j}^+ = (i-j-1)/n$ as before, and $P_{i,j}^- = j/(2n)$, because the ONEMAX fitness is selected with the probability of $1/2$, which in turn does not allow the one-to-zero bit flips to be accepted.

For EA+RL using RLS with the greedy agent and one reinforcement learning state, the probabilities are the same as in the case of plain RLS. However, in the case of T_n^j, we need to change $V_{n,j,x}$, because with the probability of $1/2$ ONEMAX is used. The expression for the changed $V_{n,j,x}$, which we define as $V'_{n,j,x}$, is as follows:

$$V'_{n,j,x} = 0.5 V_{n,j,x} + 0.5 \sum_{i=1}^{j} \frac{n}{i}.$$

As all T_i^j depend on their neighbors for the same i, we need to solve a system of linear equations to compute T_i^j for the fixed i, given the values for T_k^j, $k > i$ have been already computed. We cannot give the closed expressions for T_i^j for both LEADINGONES and LEADINGONES+ONEMAX cases. Instead, we compute these values numerically.

To compute the expected running time for all zero initialization, we just need to compute the value of T_n^0. Note that we do not count the fitness evaluation of the initial vector.

VI. EXPERIMENTAL EVALUATION

In this section, we present the results of experimental evaluation for each n from the set $\{2, 5, 10, 20, 30, 40, 50\}$.

This section is dedicated to two questions. First, we show that, in the case of RLS being used as an optimization algorithm, theoretical results for the average case approximate theoretical results for the worst case quite well. Second, we consider the case of the $(1+1)$-EA as an optimization algorithm, for which we don't yet have theoretical results about EA+RL, and show experimentally that, in general, ONEMAX does not help optimizing LEADINGONES in this case.

A. Results for RLS

First, we compute the "simple" expectation value (using an expression for T_n, which corresponds to random initialization with zero LEADINGONES fitness), the "exact" expectation value (computed using (3) and (5)) and the experimental results averaged over 1000 runs. The results for RLS optimizing the LEADINGONES problem are given in Table I. The results for EA+RL using RLS and the random agent are given in Table II, and for the greedy agent with one reinforcement learning state the results are given in Table III.

TABLE I
EXPERIMENTAL EVALUATION, RLS ON LEADINGONES FROM ALL ZEROS. COLUMN NAMES: "SIMPLE" IS THE SIMPLIFIED EXPECTED VALUE FOR RANDOM TAIL BITS, "EXACT" IS THE EXACT EXPECTED VALUE, "EXPERIMENT" IS THE AVERAGE OF THE EXPERIMENTAL VALUES.

n	simple	exact	experiment
2	3.000	3.333	3.204
5	15.000	16.547	17.094
10	55.000	58.427	59.496
20	210.000	216.930	215.397
30	465.000	475.397	472.594
40	820.000	833.863	816.506
50	1275.000	1292.329	1293.045

TABLE II
EXPERIMENTAL EVALUATION, EA+RL USING RLS ON LEADINGONES+ONEMAX FROM ALL ZEROS USING $N+1$ STATES. COLUMN NAMES HAVE THE SAME MEANING AS IN TABLE I.

n	simple	exact	experiment
2	2.667	3.200	3.204
5	11.667	14.497	14.210
10	40.000	46.906	47.939
20	146.667	161.242	163.648
30	320.000	341.961	341.308
40	560.000	589.295	587.090
50	866.667	903.287	905.810

TABLE III
EXPERIMENTAL EVALUATION, EA+RL USING RLS ON LEADINGONES+ONEMAX FROM ALL ZEROS USING ONE STATE. COLUMN NAMES HAVE THE SAME MEANING AS IN TABLE I.

n	simple	exact	experiment
2	3.500	3.200	3.237
5	15.208	14.194	14.094
10	46.645	44.493	43.978
20	150.477	146.109	145.868
30	306.925	300.369	302.389
40	515.071	506.330	494.827
50	774.480	763.554	764.321

TABLE IV
EXPERIMENTAL EVALUATION WITH THE $(1+1)$-EA FROM ALL ZEROS. COLUMN NAMES: "NO EA+RL" IS FOR LEADINGONES, "RANDOM" IS FOR EA+RL ON LEADINGONES+ONEMAX WITH RANDOM AGENT, "$N+1$ STATES" AND "ONE STATE" ARE FOR EA+RL WITH GREEDY AGENT.

n	no EA+RL	random	$N+1$ states	one state
2	3.859	3.906	3.869	3.942
5	24.325	23.123	23.011	24.845
10	93.607	98.028	93.877	92.571
20	364.358	614.730	494.653	345.059
30	787.643	4609.222	1794.575	775.359
40	1400.548	47724.651	5154.533	1345.653
50	2189.721	701151.808	12507.594	2118.676

The results given in tables show that the experimental results correspond well with the exact theoretical values. Moreover, one can see that the "simple" values approximate the exact values quite well. They are only slightly smaller in the cases of RLS and of the random agent, which shows that random initialization of the tail helps the optimization process. In the case of the greedy agent and one reinforcement learning state, the "simple" values are slightly larger. This can be explained that, after first LEADINGONES increase, there are quite many bits in the tail that are equal to one, while the expression for the ONEMAX assumes they are all zeros.

The overall impression is that, although the "simple" expressions are derived under some strange assumptions, they approximate the worst-case behavior quite good in all cases.

B. Results for the (1+1)-EA

It is possible to construct the exact expectation values using the ideas from Section V for the case of the $(1+1)$-EA used as an optimization algorithm. However, for all cases except for the $(1+1)$-EA solving the LEADINGONES problem, the LEADINGONES fitness may actually decrease when ONEMAX fitness increases. So the expressions for T_i^j are not as simple as in (3–5). Due to this fact, we present only the experimental results.

In Table IV, the results averaged over 1000 runs are presented for the following configurations: the $(1+1)$-EA optimizing LEADINGONES as "no EA+RL", the EA+RL algorithm using the $(1+1)$-EA and random agent as "random", the same algorithm with the greedy agent and $N+1$ reinforcement learning states as "$N+1$ states", the same configuration with one reinforcement learning state as "one state".

One can see in Table IV that the complexity of optimization with extra objectives is typically worse than the complexity of optimization of LEADINGONES. It can be explained by the fact that, in the case of multiple bit mutation, if ONEMAX is selected, the LEADINGONES fitness may decrease, and vice versa. When the random agent is used, optimization seems to take at least exponential time. When the greedy agent is used, it looks like the complexity is polynomial, but still higher than $\Theta(n^2)$.

The only exception is the use of the greedy agent and one reinforcement learning state. In this case, the performance of the EA+RL method on LEADINGONES+ONEMAX is just slightly better than of the $(1+1)$-EA on LEADINGONES. This differs from the case of RLS, where the quotient was roughly $1/2$. The reason is that, while the probability of increasing the LEADINGONES fitness if LEADINGONES was selected is still $1/n$, the same probability for ONEMAX is smaller, because there is a big chance that the number of bits flipping from one to zero in the rest of the bit vector will be at least two.

VII. CONCLUSION

We have presented some theoretical estimations and experimental evaluation for the expected running time of RLS and the $(1+1)$-EA solving the LEADINGONES+ONEMAX and LEADINGONES problems with or without the EA+RL method, correspondingly. The main results are:

- the expected running time of RLS solving LEADINGONES is $\frac{n^2}{2} + o(n^2)$;
- the expected running time of EA+RL using RLS with $N+1$ reinforcement learning states or with the random agent solving LEADINGONES+ONEMAX is $\frac{n^2}{3} + o(n^2)$;

- the expected running time of EA+RL using RLS with the greedy agent and one reinforcement learning state solving LEADINGONES+ONEMAX is $\frac{n^2}{4} + o(n^2)$;
- the expected running time of EA+RL using the $(1 + 1)$-EA with the random agent solving LEADINGONES+ONEMAX seems to be at least exponential;
- the expected running time of EA+RL using the $(1+1)$-EA with the greedy agent and $N + 1$ reinforcement learning states solving LEADINGONES+ONEMAX seems to be polynomial but worse than $\Theta(n^2)$;
- the expected running time of EA+RL using the $(1 + 1)$-EA with the greedy agent and one reinforcement learning state solving LEADINGONES+ONEMAX seems to be $\frac{n^2}{2} + o(n^2)$ and slightly better than just the $(1 + 1)$-EA.

The method for exact computation of the expected running time for LEADINGONES-like problems is presented. It works for algorithms using RLS as an optimizer and, with some modifications, for algorithms using the $(1 + 1)$-EA. The method is based on a two-dimensional Markov chain, the first dimension is for the LEADINGONES fitness, the second one for the number of bits set to one in the tail of the bit vector.

The approximations for the worst-case expected running times derived from somewhat unrealistic assumptions appeared to approximate the exact values quite closely, and the asymptotic behavior seems to be the same as of the exact values. The advantage of these approximations is that the expressions for them are very simple.

Even in the simple case of RLS as an optimizer, the exact computation method depends on solving tridiagonal systems of linear equations, which, at the current stage of the research, prevented us from making closed expressions for running times. However, we still believe that reasonably tight lower and upper bounds on all T_i^j are possible. This may not be the case for the $(1+1)$-EA, because the linear equations stop being tridiagonal and involve all Markov states instead of small regular groups of them.

As for the answer to the paper title, we can see that, if single bit mutation is used, ONEMAX can serve as a supporting objective for LEADINGONES, but with the use of currently available reinforcement learning methods, the "support" from ONEMAX is merely a multiplicative constant. This is not an improvement when compared with the method from [17] which delivers the $O(n \log n)$ running time for the considered problem. If a multiple bit mutation is used, ONEMAX and LEADINGONES can interfere with each other, slowing down the optimization progress. These results demonstrate existence of problems in current implementations of the EA+RL method. They also define several benchmark problems which should be considered when developing new objective selection methods.

VIII. ACKOWLEDGMENTS

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

REFERENCES

[1] F. Neumann and I. Wegener, "Can Single-Objective Optimization Profit from Multiobjective Optimization?" in *Multiobjective Problem Solving from Nature*, ser. Natural Computing Series. Springer Berlin Heidelberg, 2008, pp. 115–130.

[2] ——, "Minimum Spanning Trees Made Easier via Multi-objective Optimization," *Natural Computing*, vol. 5, no. 3, pp. 305–319, 2006.

[3] 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.

[4] 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.

[5] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler, "On the Effects of Adding Objectives to Plateau Functions," *Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 591–603, 2009.

[6] J. Handl, S. C. Lovell, and J. D. Knowles, "Multiobjectivization by Decomposition of Scalar Cost Functions," in *Parallel Problem Solving from Nature Parallel Problem Solving from Nature X*, ser. Lecture Notes in Computer Science. Springer, 2008, no. 5199, pp. 31–40.

[7] 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.

[8] ——, "Helper-Objective Optimization Strategies for the Job-Shop Scheduling Problem," *Applied Soft Computing*, vol. 11, no. 6, pp. 4161–4174, 2011.

[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] A. Buzdalova, M. Buzdalov, and V. Parfenov, "Generation of Tests for Programming Challenge Tasks Using Helper-Objectives," in *5th International Symposium on Search-Based Software Engineering*, ser. Lecture Notes in Computer Science. Springer, 2013, no. 8084, pp. 300–305.

[11] 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.

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

[13] 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.

[14] A. Gosavi, "Reinforcement Learning: A Tutorial Survey and Recent Advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, 2009.

[15] 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*, vol. 1. IEEE Computer Society, 2013, pp. 203–208.

[16] M. Buzdalov and A. Buzdalova, "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 Switch-and-Restart Algorithm with Exponential Restart Strategy for Objective Selection and its Runtime Analysis," in *Proceedings of the International Conference on Machine Learning and Applications*. IEEE Computer Society, 2014, pp. 141–146.

[18] 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.

[19] S. Böttcher, B. Doerr, and F. Neumann, "Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem," in *Parallel Problem Solving from Nature XI*, ser. Lecture Notes in Computer Science. Springer, 2010, no. 6238, pp. 1–10.