Preserving Diversity in Auxiliary Objectives Provably Speeds Up Crossing Plateaus

Tatyana Polevaya, Maxim Buzdalov
ITMO University
49 Kronverkskiy prosp.
Saint-Petersburg, Russia, 197101
Email: tanusha2406@gmail.com, mbuzdalov@gmail.com

***Abstract*—Using auxiliary objectives often speeds up optimization in different conditions. However, actual results depend heavily on how exactly these auxiliary objectives are used.**

In one of the methods which uses auxiliary objectives, the EA+RL method which uses reinforcement learning algorithms to choose between objectives, the following use case was considered. When the target objective has plateaus, reinforcement learning is unable to learn which auxiliary objectives can be used to cross these plateaus, and if an auxiliary objective is simply a refined version of the target one, reinforcement learning hardly helps, if at all.

On the other hand, it is known that optimizing auxiliary objectives may help crossing plateaus, because it may introduce an additional slope which helps the optimizer to decide where to move. In this paper, we propose two ways to keep diversity in terms of auxiliary objectives while optimizing the target objective. We prove that simple duplicate elimination already speeds up optimization of XDIVK, a coarse-grained version of ONEMAX, when ONEMAX – or any injective function of unitation – is used as an auxiliary objective, provided the population size is large enough. When selection explicitly maximizes the spread of population according to ONEMAX, it is already fast when the population size is 2.

I. INTRODUCTION

Single-objective optimization can often benefit from multiple objectives [11], [12], [15], [16]. Different approaches are known from the literature. Some researchers introduce additional objectives to escape from the plateaus [1]. Decomposition of the primary objective into several objectives also helps in many problems [9], [12], [15]. Additional objectives may also arise from the problem structure [13], [14].

Different approaches may be applied to a problem with the "original" objective, which can be called the *target* objective, and some auxiliary objectives. The *multi-objectivization* approach is to optimize *all* auxiliary objectives at once using a multi-objective optimization algorithm [9], [12]. The *helper-objective* approach is to optimize simultaneously the target objective and some (one or more) auxiliary objectives, switching between them from time to time [11].

The approaches above are designed in the assumption that the auxiliary objectives are crafted to help optimizing the target objective. However, in some cases such auxiliary objectives are automatically synthesized [5], and it is not known a priori which of them are good or bad. For this case, with participation of one of co-authors of the current paper, the EA+RL method was developed [4] which chooses the objective to optimize (by a single-objective optimization algorithm) using one of the reinforcement learning algorithms [18].

While the EA+RL method was shown to be practically efficient in certain settings [5], and was theoretically proven to be efficient for the case of bad auxiliary objectives [3], the similar research for the case of a supporting objective [2] revealed certain problems. In EA+RL, the reward, which is used to distinguish good or bad objectives, is typically calculated as the difference between the best target objective values in subsequent generations. When the target objective function contains large plateaus, this difference is zero for most of time. This means that the reinforcement learning algorithm cannot learn anything while the algorithm is on the plateau; in particular, this makes it hard to escape from the plateau. In [2], the XDIVK problem is defined, which is a "coarse-grained" version of ONEMAX and has regular plateaus of width k, where k is the parameter of the problem. This problem is solved in $\Omega(n^{k-1})$ for constant k by $(1+1)$-type optimizers, so one can expect that the presence of ONEMAX as the auxiliary objective may help speeding up optimization. This indeed happens, but only to a small degree (only by a factor of 2^{k-2}). Additionally, the paper [2] shows that reinforcement learning never learns anything in the setting of this problem.

In this paper, we seek how to use auxiliary objectives such that plateaus can be efficiently crossed. The main idea is to use an elitist scheme for the target objective while preserving diversity in the worst individuals using auxiliary objectives. This idea, commonly known as *phenotypical diversity*, is typically used for the target objective; several techniques for this were analyzed in [8]. From this perspective, auxiliary objectives can be seen as a form of *phenotypic information*. It is known that using phenotypic information speeds up finding optima, see, for instance, a recent work by Dang et al. [6]. We present two diversity preserving schemes along with their theoretical analysis on the benchmark problem with the XDIVK target function and the ONEMAX auxiliary function.

II. PRELIMINARIES

The ONEMAX problem is probably the most classic test function in evolutionary computation. The problem instances

are functions $\textsc{OneMax}_{n,z}$, $z \in \{0,1\}^n$, which are defined on bit strings of size n by

$$\textsc{OneMax}_{n,z} : \{0,1\}^n \to \mathbf{R}; x \mapsto |\{i \in [1..n] \mid x_i = z_i\}|,$$

that is, $\textsc{OneMax}_{n,z}$ counts the number of bit positions in which x and z agree. All problem instances with the fixed n and every possible bit string z of length n constitute the OneMax problem of size n.

In [2], the problem XDivK was proposed. It has a parameter $k > 1$ and is defined for problems sizes $n = mk$ for integer m. Formally, XDivK is defined as follows:

$$\textsc{XDivK}_{n,z} : \{0,1\}^n \to \mathbf{R}; x \mapsto \left\lfloor \frac{\textsc{OneMax}_{n,z}(x)}{k} \right\rfloor.$$

For simplicity, we also assume that $k \neq n$, which means that $k \leq n/2$. This function has $m = n/k$ plateaus of width k. Typically, simple optimizers, such as randomized local search or the $(1 + 1)$-EA, have certain problems optimizing this function because of these plateaus. In particular, they typically need to flip $\Theta(k)$ bits at once to reach the optimum from the last plateau. Randomized local search finds an optimum of this function in time which is $O(n^k)$ and $\Omega(n^{k-1})$, see [2] for the details.

For simplicity, further in this paper we assume that the hidden optimum z is a string consisting of one-bits. This does not have any impact on the validity of the proofs for any other hidden optima.

III. Simple Duplicate Removal EA and its Runtime Analysis

In this section, we describe a simple $(\mu+1)$-EA which not only optimizes the given fitness function, but also attempts to keep the current solution set duplicate free according to the given auxiliary function.

We also analyze the running time of this algorithm on the XDivK problem, while OneMax is used to preserve diversity. We show that the performance of this algorithm depends on the relation of the population size μ to the problem parameter k: if $\mu \geq k$, the running time is $O(\mu n \log n)$, while if $\mu < k$, it becomes exponential in $k - \mu$.

A. The Algorithm

We define the $(\mu+1)$ *duplicate removal evolutionary algorithm*, or the $(\mu+1)$-DR-EA, as follows. The population size of this algorithm is μ, it uses a nullary individual generation operator δ_0 and an unary variation operator δ_1. We denote as f the fitness function to be maximized, and as g the auxiliary function which is used to preserve diversity. The algorithm uses the function g only to detect whether for the two given individuals p and q the equality $g(p) = g(q)$ holds.

Initially, the population P of the algorithm, $|P| = \mu$, consists of random individuals (the ones generated by δ_0). In every iteration, a new individual p is sampled (by uniformly choosing an individual from P and applying the mutation operator δ_1) and added to the population. Then, one individual has to be selected and removed from the population. The selection procedure works as follows:

- First, the set Q of individuals with the worst fitness value is generated.
- Second, the set of duplicates Q_D is generated as follows. If for an individual $p \in Q$ there exists another individual $p' \in Q$ such that $g(p) = g(p')$, then p is added to the set of duplicates Q_D. In other words, if this individual is not unique according to the auxiliary function, it is considered a "duplicate".
- If there are no duplicates ($Q_D = \emptyset$), a random element of Q is deleted from the population. Otherwise, a random element of Q_D is deleted from the population.

The algorithm is outlined in Fig. 1. In a sense, this is a simplest modification of the $(\mu + 1)$ evolutionary algorithm which uses an auxiliary function to preserve diversity, as this algorithm requires no knowledge about this function, except for the fact that it is not good to have equal values of this function in the population. In fact, the auxiliary function can be transformed with any equality preserving transformation without changing the algorithm's performance.

The diversity preserving mechanism is similar to the one introduced by Jansen and Wegener [10], where, together with

```
1:  procedure (μ + 1)-DR-EA(f, g, δ0, δ1, μ)
2:      – f: the fitness function to maximize
3:      – g: the auxiliary function to preserve diversity
4:      – δ0: the new solution generator
5:      – δ1: the mutation operator
6:      – μ: the population size
7:
8:      P ← ∅ – the population
9:      while |P| < μ do
10:         P ← P ∪ {δ0()}
11:     end while
12:     while true do
13:         q ← δ1(UniformlyRandom(P))
14:         P ← P ∪ {q}
15:         f_min ← min{f(p) | p ∈ P}
16:         Q ← {p | p ∈ P, f(p) = f_min}
17:         Q_D ← ∅
18:         for p ∈ Q do
19:             if |{p' ∈ Q | g(p) = g(p')}| > 1 then
20:                 Q_D ← Q_D ∪ {p}
21:             end if
22:         end for
23:         if Q_D = ∅ then
24:             P ← P \ {UniformlyRandom(Q)}
25:         else
26:             P ← P \ {UniformlyRandom(Q_D)}
27:         end if
28:     end while
29: end procedure
```

Fig. 1. The $(\mu + 1)$ Duplicate Removal EA

the use of crossover, it helps to optimize certain jump functions in polynomial time. However, in this paper it does not allow the duplicates to enter the population at all, which resembles the $(2+1)$ GA from Storch and Wegener [17]. Of course, the main difference is that this diversity preservation is applied to auxiliary objectives, not to the target one.

In the following section we show that large enough population sizes help this algorithm to cross plateaus, provided that the auxiliary function gives enough information to distinguish the solutions residing in these plateaus.

B. The Analysis

We analyze the running time of the $(\mu + 1)$-DR-EA on the fitness function XDIVK with the problem size n and the plateau size parameter k. Here, ONEMAX is used as the auxiliary function, but, due to the properties of the $(\mu+1)$-DR-EA, *any* injective function of unitation – that is, any function $f := x \mapsto g(|x|)$ where $g : \{0, 1, \ldots, n\} \to \mathbb{R}$ is injective – can be used as the auxiliary function without any change to the result.

In the first part of the analysis we consider an arbitrary initialization operator δ_0 and an arbitrary mutation operator δ_1 which makes it possible to solve ONEMAX for the $(1+1)$-EA. This can be, for example, the single bit mutation, or the standard mutation which flips each bit independently with the probability of $1/n$. We show that, for large enough population sizes $\mu \geq k$, this algorithm is only at most μ times slower than the $(1+1)$-EA on ONEMAX with the same mutation operator δ_1.

Theorem 1: Assume that the initialization operator δ_0 and the mutation operator δ_1 are such that the expected running time of the $(1+1)$-EA on ONEMAX is $T(n)$ for the problem size n. Consider the XDIVK problem with the problem size n and the plateau parameter k. If the $(\mu + 1)$-DR-EA uses the same operators δ_0 and δ_1, and $\mu \geq k$, then the considered XDIVK problem is solved by the $(\mu+1)$-DR-EA in expected time of at most $\mu \cdot T(n)$.

Proof: First we prove that, under the conditions of the theorem, at least one of the best individuals according to the value of ONEMAX survives the selection. Indeed, if there are at least two such individuals, one of them survives as only one individual is removed. If the best individual is unique, two cases are possible:

- Some duplicates exist. In this case, one of them is deleted, and the best individual survives.
- No duplicates exist. As there are $\mu+1$ individuals at the time of the selection, by the pigeonhole principle there exist at least one individual with the value of XDIVK worse than the one of the best individual, as there are only $k < \mu+1$ different ONEMAX values corresponding to the same XDIVK value. So the best individual does not appear in the set Q of the worst individuals according to XDIVK, and survives as well.

In each iteration, a random individual from the population is chosen to generate a new offspring. With the probability of at least $1/\mu$, this individual will be the best individual. Thus, with probability of at least $1/\mu$ one step of the $(1 + 1)$-EA optimizing ONEMAX is simulated, which proves the result. ■

TABLE I
RUNNING TIMES OF $(1 + 1)$-DR-EA ON XDIVK WITH ONEMAX WITH $k = 10$ AND $\mu = 20$. COLUMNS ARE NUMBERED FOR CONVENIENCE

n	Runtime	$en \ln en + 5e\mu n + \mu$	$\frac{\#2}{\#3}$	$\mu en \ln en$	$\frac{\#2}{\#5}$
#1	#2	#3	#4	#5	#6
1000	$2.210 \cdot 10^5$	$2.933 \cdot 10^5$	0.753	$4.299 \cdot 10^5$	0.514
2000	$4.893 \cdot 10^5$	$5.904 \cdot 10^5$	0.829	$9.352 \cdot 10^5$	0.523
5000	$1.353 \cdot 10^6$	$1.489 \cdot 10^6$	0.909	$2.587 \cdot 10^6$	0.523
10000	$3.026 \cdot 10^6$	$2.996 \cdot 10^6$	1.010	$5.551 \cdot 10^6$	0.545
20000	$6.621 \cdot 10^6$	$6.029 \cdot 10^6$	1.098	$1.186 \cdot 10^7$	0.559
50000	$1.821 \cdot 10^7$	$1.520 \cdot 10^7$	1.198	$3.213 \cdot 10^7$	0.567
100000	$3.745 \cdot 10^7$	$3.058 \cdot 10^7$	1.224	$6.803 \cdot 10^7$	0.550
200000	$7.914 \cdot 10^7$	$6.155 \cdot 10^7$	1.286	$1.436 \cdot 10^8$	0.551
500000	$2.171 \cdot 10^8$	$1.551 \cdot 10^8$	1.399	$3.839 \cdot 10^8$	0.565

A simple corollary of Theorem 1 is that for both single flip mutation and the standard bit flip mutation with $p = 1/n$, the expected running time of the $(\mu+1)$-DR-EA on XDIVK with the auxiliary ONEMAX is $O(\mu n \log n)$ when $\mu \geq k$. As this algorithm is clearly not faster than $(1 + 1)$-EA on ONEMAX, it runs in $\Omega(n \log n)$.

Our upper bound of $O(\mu n \log n)$ may seem weak, as Witt proved an $O(\mu n + n \log n)$ upper bound for a very similar case of the $(\mu + 1)$ EA optimizing ONEMAX [19]. However, there is empirical evidence that in our case the running time is actually $\Theta(\mu n \log n)$. To support this, we ran the experiments with $k = 10$, $\mu = 20$, $n = \{1, 2, 5\} \times \{10^3, 10^4, 10^5\}$, and the standard bit mutation. Results for all configurations were averaged over 100 runs. The results are presented in Table I. In this table we put the experimental results, as well as the $en \ln en + 5e\mu n + \mu$ bound proven by Witt for the $(\mu+1)$ EA and the value of $\mu en \ln en$ which follows from Theorem 1.

The fourth and sixth columns of Table I suggest that Witt's type of upper bound does not hold for the runtime of the $(\mu + 1)$-DR-EA on XDIVK with ONEMAX, as the actual runtime grows faster. In the same time, a more conservative $\Theta(\mu n \log n)$ expression demonstrates good estimation quality with a rather stable constant factor.

Now we show that the population size matters. The next theorem states that if $\mu < k$, the running time becomes exponential in $k - \mu$ for the single bit mutation operator. The proof of this theorem, due to its size, is given in the appendix.

Theorem 2: Consider the single bit mutation δ_1 and the XDIVK problem with the problem size n and the plateau parameter k, while ONEMAX is used as the auxiliary function. If $\mu < k$, the running time of the $(\mu + 1)$-DR-EA is $O(\mu^2 n^{k-\mu+1})$ and

$$\Omega\left(\left(\frac{n\sqrt{\mu}}{6k}\right)^{k-\mu+1} \frac{1}{6^{\mu-1}k \cdot \sqrt{\mu^3}}\right).$$

IV. SIMPLE SPREAD MAXIMIZING EA AND ITS RUNTIME ANALYSIS

In this section, we describe a simple modification of the $(\mu+1)$-DR-EA. It also uses the auxiliary function to determine

```
1: procedure (μ + 1)-SM-EA(f, g, δ0, δ1, μ)
2:     – f: the fitness function to maximize
3:     – g: the auxiliary function to preserve diversity
4:     – δ0: the new solution generator
5:     – δ1: the mutation operator
6:     – μ: the population size
7:
8:     P ← ∅ – the population
9:     while |P| < μ do
10:        P ← P ∪ {δ0()}
11:    end while
12:    while true do
13:        q ← δ1(UniformlyRandom(P))
14:        P ← P ∪ {q}
15:        f_min ← min{f(p) | p ∈ P}
16:        Q ← {p | p ∈ P, f(p) = f_min}
17:        s ← −∞
18:        Q_S ← ∅
19:        for p ∈ Q do
20:            S' ← S \ {p}
21:            s_p ← max_{q∈S'} g(q) − min_{q∈S'} g(q)
22:            if s_p > s then
23:                s ← s_p
24:                Q_S ← ∅
25:            end if
26:            if s_p = s then
27:                Q_S ← Q_S ∪ {p}
28:            end if
29:        end for
30:        P ← P \ {UniformlyRandom(Q_S)}
31:    end while
32: end procedure
```

Fig. 2. The $(\mu + 1)$ Spread Maximizing EA

which individual, among the worst, to remove, but does it in a different manner.

We also analyze the running time of this algorithm on the XDIVK problem, while ONEMAX is used to preserve diversity. We show that the running time is $O(\mu n \log n)$ not only for large population sizes, but for all $\mu \geq 2$.

A. The Algorithm

The $(\mu + 1)$ *spread maximizing evolutionary algorithm*, or $(\mu + 1)$-SM-EA, is defined as follows. The population size of this algorithm is μ, it uses a nullary individual generation operator δ_0 and an unary variation operator δ_1. We denote as f the fitness function to be maximized, and as g the auxiliary function which is used to preserve diversity.

By *spread* of a set of individuals Q we denote the value $\max_{q \in Q} g(q) - \min_{q \in Q} g(q)$. The bigger the spread, the larger portion of the search space the solution set is thought to occupy. The $(\mu + 1)$-SM-EA chooses one of the solutions with the minimum fitness value to remove such that the spread of the remaining population is maximum possible. All other details are similar to the $(\mu + 1)$-DR-EA. The algorithm is outlined on Fig. 2.

B. The Analysis

We analyze the running time of the $(\mu + 1)$-SM-EA on the fitness function XDIVK with the problem size n and the plateau size parameter k. Here, ONEMAX is used as the auxiliary function. We consider an arbitrary initialization operator δ_0 and an arbitrary mutation operator δ_1 which makes it possible to solve ONEMAX for the $(1 + 1)$-EA. We show that for all population sizes $\mu \geq 2$, this algorithm is only at most μ times slower than the $(1 + 1)$-EA on ONEMAX with the same mutation operator δ_1.

Theorem 3: Assume that the initialization operator δ_0 and the mutation operator δ_1 are such that the expected running time of the $(1+1)$-EA on ONEMAX is $T(n)$ for the problem size n. Consider the XDIVK problem with the problem size n and the plateau parameter k. If the $(\mu + 1)$-SM-EA uses the same operators δ_0 and δ_1, and $\mu \geq 2$, then the considered XDIVK problem is solved by the $(\mu+1)$-SM-EA in expected time of at most $\mu \cdot T(n)$.

Proof: It is easy to see that the selection scheme never deletes the best and the worst individuals, according to ONEMAX, from the set of individuals with the worst XDIVK value. Thus it never deletes the individual with the maximum ONEMAX value. The rest of the proof follows the same way as in Theorem 1. ■

As a corollary of Theorem 3, we can see that, for random initialization and common mutation operators (standard mutation or single bit mutation) the $(\mu + 1)$-SM-EA runs on the XDIVK fitness function and the ONEMAX auxiliary function in $O(\mu n \log n)$ for $\mu \geq 2$. So, contrary to $(\mu + 1)$-DR-EA, this algorithm is capable of running fast with small population sizes.

V. CONCLUSION

We presented two simple evolutionary algorithms which, apart from the fitness function to optimize, make use of the auxiliary function to preserve diversity. Both algorithms are $(\mu + 1)$ steady-state algorithms with population size μ. The first of them, the $(\mu+1)$ duplicate removal evolutionary algorithm, or $(\mu + 1)$-DR-EA, removes a single random duplicate (according to the auxiliary function), if exists, among the individuals with the worst fitness value. The second, the $(\mu+1)$ spread maximizing evolutionary algorithm, or $(\mu+1)$-SM-EA, removes a random individual in such a way that the *spread* of the remaining population is maximum possible according to the auxiliary function.

We analyzed these algorithms on a benchmark problem taken from our previous research [2]. In this problem, the fitness function is XDIVK, which is ONEMAX with all fitness values divided evenly by k, the problem's parameter, and the auxiliary function is ONEMAX. The XDIVK function has n/k plateaus of width k each and thus is quite hard to optimize: for example, both randomized local search and $(1 + 1)$-EA need $\Omega(n^{k-1})$ time to reach the optimum.

We proved the $(\mu+1)$-DR-EA algorithm, using certain initialization operator δ_0 and mutation operator δ_1, given large enough population size ($\mu \geq k$), runs on this benchmark problem at most μ times slower than $(1+1)$-EA on OneMax using the same operators. That is, $(\mu+1)$-DR-EA runs in $O(\mu n \log n)$ if $\mu \geq k$ and, for example, standard mutation operator is used. We also showed experimentally that this bound seems to be tight, i.e. should be read as $\Theta(\mu n \log n)$. However, for smaller population sizes ($2 \leq \mu < k$), this algorithm slows down: for the single bit mutation, it takes time which is $O(\mu^2 n^{k-\mu+1})$ and has a similar lower bound.

The second algorithm, $(\mu+1)$-SM-EA, appeared to be more efficient: it reaches the $O(\mu n \log n)$ running time already for $\mu \geq 2$, because its selection scheme never lets the individual with the largest OneMax value die.

These algorithms have several useful invariance properties for the auxiliary functions. For example, the behaviour of $(\mu+1)$-DR-EA does not change if any injective function of unitation is used instead of OneMax as the auxiliary function. The $(\mu+1)$-SM-EA is less invariant, however, it tolerates linear transformations, including those with negative linear coefficient. For example, OneMax can be changed to ZeroMax without any change in the running time.

The proposed algorithm schemes have several possible extensions to support multiple auxiliary objectives. For example, the selection scheme of $(\mu+1)$-SM-EA may delete the individual with the smallest crowding distance [7], or the least hypervolume contribution [20]. We think that the algorithmic ideas proposed in this paper may be helpful in combinatorial optimization, where the presence of plateaus is often an issue. Some theoretical future work may be necessary to justify certain design choices for using multiple auxiliary objectives.

Acknowledgments

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

References

[1] D. Brockhoff, T. Friedrich, N. Hebbinghaus, C. Klein, F. Neumann, and E. Zitzler. On the effects of adding objectives to plateau functions. *IEEE Transactions on Evolutionary Computation*, 13(3):591–603, 2009.

[2] 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*, pages 201–202. ACM, 2014.

[3] 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*, volume 1, pages 203–208. IEEE Computer Society, 2013.

[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*, volume 1, pages 150–155, 2012.

[5] 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*, number 8084 in Lecture Notes in Computer Science, pages 300–305. Springer, 2013.

[6] D.-C. Dang, T. Friedrich, T. Kötzing, M. S. Krejca, P. K. Lehre, P. S. Oliveto, D. Sudholt, and A. M. Sutton. Escaping local optima with diversity mechanisms and crossover. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 645–652, 2016.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[8] T. Friedrich, P. S. Oliveto, D. Sudholt, and C. Witt. Analysis of diversity preserving mechanisms for global exploration. *Evolutionary Computation*, 17(4):455–476, 2009.

[9] J. Handl, S. C. Lovell, and J. D. Knowles. Multiobjectivization by decomposition of scalar cost functions. In *Parallel Problem Solving from Nature – PPSN X*, number 5199 in Lecture Notes in Computer Science, pages 31–40. Springer, 2008.

[10] T. Jansen and I. Wegener. The analysis of evolutionary algorithms–a proof that crossover really can help. *Algorithmica*, 34:47–66, 2002.

[11] M. T. Jensen. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation: Evolutionary computation combinatorial optimization. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347, 2004.

[12] 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*, pages 269–283. Springer-Verlag, 2001.

[13] D. F. Lochtefeld and F. W. Ciarallo. Deterministic helper-objective sequences applied to Job-Shop scheduling. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 431–438. ACM, 2010.

[14] D. F. Lochtefeld and F. W. Ciarallo. Helper-objective optimization strategies for the Job-Shop scheduling problem. *Applied Soft Computing*, 11(6):4161–4174, 2011.

[15] F. Neumann and I. Wegener. Minimum spanning trees made easier via multi-objective optimization. *Natural Computing*, 5(3):305–319, 2006.

[16] F. Neumann and I. Wegener. Can single-objective optimization profit from multiobjective optimization? In *Multiobjective Problem Solving from Nature*, Natural Computing Series, pages 115–130. Springer Berlin Heidelberg, 2008.

[17] T. Storch and I. Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, 2004.

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

[19] C. Witt. Runtime analysis of the $(\mu+1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86, 2006.

[20] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the Strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Appendix

A. Proof of the Upper Bound in Theorem 2

Let an integer m be equal to n/k. We denote as q_i the number of individuals $x \in P$ such that $\textsc{XDivK}(x) = i$ for $i \in [0; m]$. We also denote as v the smallest i such that $q_i > 0$ – that is, the *worst* OneMax fitness in the population. Let $W(v)$ be the expected value of the number of steps to increase v by some non-zero amount, or until the search algorithm stops. One more quantity is $Z_v(q)$, for $1 \leq q \leq \mu$, which is the expected number of steps required for $q_v = q$ to eventually drop below q, probably by visiting states with $q_v > q$ and getting back.

A straightforward upper bound is $W(v) \leq \sum_{q=1}^{\mu} Z_v(q)$ for $v < m-1$, and $W(m-1) \leq Z_{m-1}(\mu)$. The latter is different because we require only one individual, and not the whole population, to reach the optimum. The total expected running time, as a function of n and k, can be bounded as:

$$T(n,k) \leq \sum_{v=0}^{m-1} W(v) \leq \sum_{v=0}^{m-2} \sum_{q=1}^{\mu-1} Z_v(q) + \sum_{v=0}^{m-1} Z_v(\mu).$$

The double sum corresponds to the phases of the optimization when there are at least two different values of XDivK in the population, while the single sum corresponds to the phases

when all individuals in the population have the same XDIVK value, i.e., when the entire population is located on the same plateau. We estimate these sums separately.

1) The Case of Different XdivK Values: In this part we estimate the value of $Z_v(q)$, $0 < q < \mu$, and the corresponding sum in the expression for the running time. The case of $q < \mu$ corresponds to the situation when there are q individuals with the XDIVK value of v, and $\mu - q > 0$ individuals with greater XDIVK values.

On every step, an individual is selected at random, then its offspring is added to the population, after that, survival selection is performed. So there are two possible cases:

1) If an individual x with $\textsc{XDIVK}(x) = v$ is selected, which happens with probability $\frac{q}{\mu}$, then at most q individuals with the XDIVK value of v will remain in the population after survival selection.
2) Otherwise, an individual x with $\textsc{XDIVK}(x) > v$ is selected, which happens with probability $\frac{\mu-q}{\mu}$. Its offspring may have the XDIVK value of v with the probability of at most $\frac{(v+1)k}{n}$, in this case, q individuals with the XDIVK value of v will remain in the population. Otherwise, the number of such individuals will be $q-1$.

This constitutes the following estimation for $Z_v(q)$:

$$Z_v(q) \le 1 + \frac{q}{\mu} Z_v(q) + \frac{\mu - q}{\mu} \frac{(v+1)k}{n} Z_v(q);$$
$$Z_v(q) \le \frac{n\mu}{(n - (v+1)k)(\mu - q)}.$$

The corresponding sum in the expression for the running time is thus at most:

$$\begin{aligned}\sum_{v=0}^{m-2} \sum_{q=1}^{\mu-1} Z_v(q) &\le \sum_{v=0}^{m-2} \sum_{q=1}^{\mu-1} \frac{n\mu}{(n-(v+1)k)(\mu-q)} \\ &= \frac{n\mu}{k} \left(\sum_{v=0}^{m-2} \frac{1}{m-(v+1)} \right) \left(\sum_{q=1}^{\mu-1} \frac{1}{\mu - q} \right) \\ &\le m\mu(1+\log m)(1+\log \mu) \\ &< n\mu(1+\log n)(1+\log \mu).\end{aligned}$$

2) The Case of Equal XdivK Values: In this part we estimate the value of $Z_v(\mu)$, which corresponds to the situation when all individuals in the population share the same value of XDIVK.

Denote the maximum ONEMAX value in the population as ξ. We denote as $X_v(j)$ the expected number of steps required for ξ to eventually become greater than $vk + j$, where $0 \le j < k$, provided that currently $\xi = vk + j$. Using this notation, we can prove an upper bound of the following form: $Z_v(\mu) \le \sum_{j=0}^{k-1} X_v(j)$. Indeed, when the best individual acquires the ONEMAX value of $vk + k$, it also increases its XDIVK value.

We continue by estimating $X_v(j)$. Two cases have to be considered: the case $0 \le j < \mu$ and the case $\mu \le j < k$.

Consider the **first case**. If a best individual (the one with the ONEMAX value of ξ) is chosen for reproduction, which happens with the probability of at least $1/\mu$, then ξ will be increased with the probability of $1 - \xi/n$. Otherwise, an offspring with a worse ONEMAX fitness appears. In this case, either the offspring has a worse value of XDIVK, so will not survive selection, or there is a duplicate in the population, so one of the duplicates will be removed. In any of the latter cases, at least one of the best individuals survive, so ξ remains the same. The overall probability of increasing ξ is at least $(1 - \xi/n)/\mu$, so:

$$X_v(j) \le \frac{\mu}{1 - \frac{\xi}{n}} = \frac{n\mu}{n - (vk + j)},\ 0 \le j < \mu.$$

Consider the **second case**, $\mu \le j < k$. We always consider the worst case when there is only one individual with the best ONEMAX value, since its deletion decreases ξ. There are two cases:

1) If the best individual is selected for reproduction (which happens with probability $1/\mu$), the following happens:
   - With probability $\frac{\xi}{n}$, one of the one-bits flips. Then with probability $\frac{1}{\mu+1}$, the best individual is removed, and its child becomes the best: $\xi \leftarrow \xi - 1$; otherwise, the best individual remains the same.
   - With probability $\frac{n-\xi}{n}$, one of the zero-bits flips. If the child is subsequently removed, which happens with probability $\frac{1}{\mu+1}$, the parent remains the best, otherwise the child takes over and $\xi \leftarrow \xi + 1$.
2) If some other individual is selected for reproduction (with probability of $1 - 1/\mu$), with probability $\frac{1}{\mu+1}$, the best individual is removed, and in the worst case $\xi \leftarrow vk$. Alternatively (with probability $\frac{\mu}{\mu+1}$), the best individual remains intact.

We can put a recursive upper bound as follows:

$$\begin{aligned}X_v(j) \le 1 &+ \frac{vk+j}{\mu n} \left(\frac{X_v(j-1) + X_v(j)}{\mu+1} + \frac{\mu \cdot X_v(j)}{\mu+1} \right) \\ &+ \frac{n-(vk+j)}{\mu n} \frac{1}{\mu+1} X_v(j) \\ &+ \frac{\mu-1}{\mu} \left(\frac{1}{\mu+1} \sum_{i=0}^{j} X_v(i) + \frac{\mu}{\mu+1} X_v(j) \right) \\ = 1 &+ \frac{\mu-1}{\mu(\mu+1)} \sum_{i=0}^{j-1} X_v(i) + \frac{vk+j}{\mu(\mu+1)n} X_v(j-1) \\ &+ \left(1 + \frac{1}{\mu} \frac{n-(vk+j)}{n} \frac{-\mu}{\mu+1} \right) X_v(j)\end{aligned}$$

After putting all $X_v(j)$ instances together, we get that:

$$\begin{aligned}X_v(j) &\le \frac{(\mu+1)n}{n-(vk+j)} \left(1 + \frac{\mu-1}{\mu(\mu+1)} \sum_{i=0}^{j-1} X_v(i) \right. \\ &\quad \left. + \frac{vk+j}{\mu(\mu+1)n} X_v(j-1) \right) \\ &\le \frac{(\mu+1)n}{n-(vk+j)} \left(1 + \frac{1}{\mu+1} \sum_{i=0}^{j-1} X_v(i) \right) \\ &\le \frac{2n\mu}{n-(vk+j)} + \frac{n}{n-(vk+j)} \sum_{i=0}^{j-1} X_v(i). \qquad (1)\end{aligned}$$

Now we prove the following lemma:

Lemma 1: The upper bound on $X_v(j)$ for $\mu \le j < k$ is:

$$X_v(j) \le \frac{n}{n-(vk+j)}\left(2\mu + \sum_{i=\mu}^{j-1}\frac{2n\mu\,(1+n)^{j-i-1}}{n-(vk+j)} + (1+n)^{j-\mu}\sum_{i=0}^{\mu-1}X_v(i)\right).$$

Proof: We prove this lemma by induction in a more general form, depending on the parameter τ, $\mu \le \tau \le j$:

$$X_v(j) \le \frac{n}{n-(vk+j)}\left(2\mu + \sum_{i=\tau}^{j-1}\frac{2n\mu\,(1+n)^{j-i-1}}{n-(vk+j)} + (1+n)^{j-\tau}\sum_{i=0}^{\tau-1}X_v(i)\right).$$

The induction base is (1), which is the inequation above for $\tau = j$, and the statement to be proven holds for $\tau = \mu$. The induction step is proven by expanding $X_v(\tau-1)$ using (1):

$$\begin{aligned}\sum_{i=0}^{\tau-1}X_v(i) &\le \sum_{i=0}^{\tau-2}X_v(i) + \left(\frac{2n\mu}{n-(vk+\tau-1)} + \frac{n}{n-(vk+\tau-1)}\sum_{i=0}^{\tau-2}X_v(i)\right)\\ &\le \frac{2n\mu}{n-(vk+j)} + (1+n)\sum_{i=0}^{\tau-2}X_v(i),\end{aligned}$$

where the first addend goes to the second addend of the sum in the induction statement, and the second addend forms the new third addend of this sum. ■

Using the first case results ($X_v(j) \le \frac{n\mu}{n-(vk+j)}$, $0 \le j < \mu$), we get that:

$$\begin{aligned}X_v(j) &\le \frac{n\cdot\left(2\mu + \sum_{i=\mu}^{j-1}\frac{2n\mu(1+n)^{j-i-1}}{n-(vk+j)}\right)}{n-(vk+j)} + \frac{n\cdot\left((1+n)^{j-\mu}\sum_{i=0}^{\mu-1}X_v(i)\right)}{n-(vk+j)}\\ &\le \frac{n\cdot\left(2n\mu + 2n\mu\sum_{i=\mu}^{j-1}(1+n)^{j-i-1}\right)}{(n-(vk+j))^2} + \frac{n\cdot\left((1+n)^{j-\mu}n\mu^2\right)}{(n-(vk+j))^2}\\ &= \frac{n\cdot\left(2n\mu + 2n\mu\frac{(1+n)^{j-\mu}-1}{n} + (1+n)^{j-\mu}n\mu^2\right)}{(n-(vk+j))^2}\\ &\le \frac{n\cdot\left(2n\mu(1+n)^{j-\mu} + n\mu^2(1+n)^{j-\mu}\right)}{(n-(vk+j))^2}\\ &\le \frac{(1+n)^{j-\mu}n^2(1+\mu)^2}{(n-(vk+j))^2}.\end{aligned}$$

Finally, we estimate $Z_v(\mu)$ as follows:

$$\begin{aligned}Z_v(\mu) &\le \sum_{j=0}^{k-1}X_v(j) \le \sum_{j=0}^{\mu-1}\frac{n\mu}{n-(vk+j)} + \frac{n^2(1+\mu)^2}{(n-(vk+k-1))^2}\sum_{j=\mu}^{k-1}(1+n)^{j-\mu}\\ &< \frac{n\mu^2}{n-(vk+\mu-1)} + \frac{n^2(1+\mu)^2(1+n)^{k-\mu}}{n(n-(vk+k-1))^2}.\end{aligned}$$

The sum of $Z_v(\mu)$ is thus at most:

$$\begin{aligned}\sum_{v=0}^{m-1}Z_v(\mu) &< \sum_{v=0}^{m-1}\frac{n\mu^2}{n-(vk+\mu-1)} + \sum_{v=0}^{m-1}\frac{n^2(1+\mu)^2(1+n)^{k-\mu}}{n(n-(vk+k-1))^2}\\ &< n\mu^2(\ln n+1) + \sum_{v=0}^{m-1}\frac{n^2(1+\mu)^2(1+n)^{k-\mu}}{n(n-(vk+k-1))^2}\\ &< n\mu^2(\ln n+1) + \frac{n^2(1+\mu)^2(1+n)^{k-\mu}}{n}\cdot\frac{\pi^2}{6}\\ &= O(\mu^2 n^{k-\mu+1}),\end{aligned}$$

which determines the running time of the algorithm. ■

B. Proof of the Lower Bound in Theorem 2

In this section, we prove the lower bound for the running time of $(\mu+1)$-DR-EA, when the problem size is n, the target function is XDIVK with the plateau parameter k, the auxiliary function is ONEMAX, and single-bit flip mutation is used.

It is clear that the best possible algorithm state among those which do not contain an individual with an optimum XDIVK value is the state with all individuals having the XDIVK value of $n/k-1$. The elitist selection ensures that, when the algorithm starts from such a state, no individual with XDIVK value less than $n/k-1$ will ever survive. In this section we describe the algorithm state as $\langle u, v\rangle$ which means that the maximum ONEMAX value in the population is $n-k+u$, and exactly v individuals have this value. By $E\langle u, v\rangle$ we denote the expected running time for the algorithm residing in state $\langle u, v\rangle$ to improve the best ONEMAX value in the population (and to have it survived). If $T(n,k)$ is the expected running time of the algorithm, then obviously $T(n,k) \ge \min_{v=1}^{\mu} E\langle k-1, v\rangle$.

We derive lower bounds for $E\langle u, v\rangle$ for all possible combinations of u and v. For states $\langle u, v\rangle$ where $0 \le u < \mu$, we simply put $E\langle u, v\rangle \ge \frac{\mu}{v}\frac{n}{k-u}$ as it is the minimum expected number of attempts to leave any state $\langle u, v\rangle$. For $\mu \le u < k$, we prove the lower bounds below.

In the following, we use the shortcut $E\langle u, 0\rangle = E\langle u, 1\rangle + \min_{v=1}^{\mu} E\langle u-1, v\rangle$, which means that if all individuals with ONEMAX value greater than or equal to u disappear, the level u has to be conquered again. We also denote as $[u = k-1]$ the function of u and k which equals one if $u = k-1$ and is zero otherwise. Consider two cases for the state $\langle u, v\rangle$:

1) A best individual is selected (with probability $\frac{v}{\mu}$). Then:

- With probability $\frac{k-u}{n}$, a zero-bit is flipped. If $u = k-1$, the new individual is the optimum, so we may safely think that this individual survives, and the new state is $\langle u+1, 1\rangle$. If $u < k-1$, the new individual survives with probability $\frac{\mu}{\mu+1}$, so the overall survival probability can be written as $\frac{\mu+[u=k-1]}{\mu+1}$. If the new individual does not survive, the state remains $\langle u, v\rangle$.
- With probability $\frac{n-k+u}{n}$, a one-bit is flipped. Then, if one of the best individuals dies (with probability $\frac{v}{\mu+1}$), the new state is $\langle u, v-1\rangle$, otherwise the state remains $\langle u, v\rangle$.

2) An individual which is not the best is selected (with probability $\frac{\mu-v}{\mu}$). Then:
   - With probability at most $\frac{k}{n}$, a zero-bit is flipped. The new individual may become one of the best individuals. With probability at most $\frac{\mu-v}{\mu+1}$, all best individuals, including the new one, survive, and the new state is $\langle u, v+1\rangle$. Otherwise, at most v individuals remain at the level u, and the state remains $\langle u, v\rangle$. Here, we optimistically set to zero the probability of decreasing the number of best individuals, which is possible if the new individual is not one of the best.
   - With probability at least $\frac{n-k}{n}$, a one-bit is flipped. Then, with probability $\frac{v}{\mu+1}$ one of the best individuals dies, and the new state is $\langle u, v-1\rangle$. Otherwise, the state remains $\langle u, v\rangle$.

Putting things together, in the best case the probabilities for new states are: $\frac{v(k-u)(\mu+[u=k-1])}{n\mu(\mu+1)} = \frac{P_A}{n\mu(\mu+1)}$ for the state $\langle u+1, 1\rangle$, $\frac{(\mu-v)^2 k}{n\mu(\mu+1)} = \frac{P_B}{n\mu(\mu+1)}$ for the state $\langle u, v+1\rangle$ and $\frac{v^2(n-k+u)+v(\mu-v)(n-k)}{n\mu(\mu+1)} = \frac{v(uv+n\mu-k\mu)}{n\mu(\mu+1)} = \frac{P_C}{n\mu(\mu+1)}$ for the state $\langle u, v-1\rangle$. So the lower bound on $E\langle u, v\rangle$ is:

$$\begin{aligned}
E\langle u, v\rangle &\geq \frac{1 + \frac{P_B}{n\mu(\mu+1)} E\langle u, v+1\rangle + \frac{P_C}{n\mu(\mu+1)} E\langle u, v-1\rangle}{\frac{P_A+P_B+P_C}{n\mu(\mu+1)}} \\
&\geq \frac{P_B E\langle u, v+1\rangle + P_C E\langle u, v-1\rangle}{P_A + P_B + P_C} \\
&\geq \frac{v(uv+n\mu-k\mu)E\langle u, v-1\rangle}{P_A + P_B + P_C} \\
&\geq \frac{v\mu(n-k)E\langle u, v-1\rangle}{v\mu(n-k) + k\mu(\mu+[u=k-1])} \\
&= \frac{v(n-k)E\langle u, v-1\rangle}{v(n-k) + k(\mu+[u=k-1])}.
\end{aligned}$$

For $v = 1$, we substitute the shortcut $E\langle u, 0\rangle$ by its value $E\langle u, 1\rangle + \min_{v=1}^{\mu} E\langle u-1, v\rangle$, which yields:

$$\begin{aligned}
E\langle u, 1\rangle &\geq \frac{(n-k)(E\langle u, 1\rangle + \min_{v=1}^{\mu} E\langle u-1, v\rangle)}{(n-k) + k(\mu+[u=k-1])} \\
E\langle u, 1\rangle &\geq \frac{(n-k)\min_{v=1}^{\mu} E\langle u-1, v\rangle}{((n-k) + k(\mu+[u=k-1]))\left(1 - \frac{n-k}{(n-k)+k(\mu+[u=k-1])}\right)} \\
&= \frac{n-k}{k(\mu+[u=k-1])} \min_{v=1}^{\mu} E\langle u-1, v\rangle.
\end{aligned}$$

We denote $\frac{v(n-k)}{v(n-k)+k\mu}$ as $\delta(v)$ and $\frac{v(n-k)}{v(n-k)+k(\mu+1)}$ as $\delta'(v)$. It is easy to see that for all v, $1 \leq v < \mu$, $\delta'(v) < \delta(v) < 1$. Now, using that $E\langle u, v\rangle \geq \frac{\mu}{v}\frac{n}{k-u}$ for $0 \leq u < \mu$ and all v, we can state that:

$$\begin{aligned}
E\langle \mu, 1\rangle &\geq \frac{n-k}{k\mu}\frac{\mu}{\mu}\frac{n}{k-\mu+1} = \frac{n-k}{k\mu}\frac{n}{k-\mu+1} \\
E\langle \mu, v\rangle &\geq \prod_{i=2}^{v}\delta(i)\frac{n-k}{k\mu}\frac{n}{k-\mu+1} \\
E\langle \mu, \mu\rangle &\geq \prod_{i=2}^{\mu}\delta(i)\frac{n-k}{k\mu}\frac{n}{k-\mu+1}.
\end{aligned}$$

As for $E\langle \mu, v\rangle$ the lower bound gets smaller as v grows, it holds that:

$$E\langle \mu+1, 1\rangle \geq \frac{n-k}{k\mu}\prod_{i=2}^{\mu}\delta(i)\frac{n-k}{k\mu}\frac{n}{k-\mu+1},$$

so, subsequently, we may show similarly that:

$$\begin{aligned}
E\langle u, \mu\rangle &\geq \left(\frac{n(n-k)}{k\mu(k-\mu+1)}\right) \times \left(\frac{n-k}{k\mu}\prod_{i=2}^{\mu}\delta(i)\right)^{u-\mu+1} \\
T(n,k) \geq E\langle k-1, \mu\rangle &\geq \left(\frac{n(n-k)}{k\mu(k-\mu+1)}\right) \\
&\quad \times \left(\frac{n-k}{k(\mu+1)}\prod_{i=2}^{\mu}\delta'(i)\right) \times \left(\frac{n-k}{k\mu}\prod_{i=2}^{\mu}\delta(i)\right)^{k-\mu-1}
\end{aligned}$$

The only remaining thing is to estimate $\prod_{i=2}^{\mu}\delta(i)$ and the same product for $\delta'(i)$ as well. For this we have (using Stirling's formula):

$$\begin{aligned}
\prod_{i=2}^{\mu}\delta(i) &= \prod_{i=2}^{\mu}\frac{i(n-k)}{i(n-k)+k\mu} \\
&\geq \prod_{i=2}^{\mu}\frac{i(n-k)}{n\mu} = \mu!\left(\frac{n-k}{n\mu}\right)^{\mu-1} \\
&\sim \sqrt{2\pi\mu}\cdot\left(\frac{\mu}{e}\right)^{\mu}\cdot\left(\frac{n-k}{n\mu}\right)^{\mu-1} \\
&\geq \sqrt{2\pi\mu}\cdot\frac{\mu}{e^{\mu}2^{\mu-1}} \geq \frac{\sqrt{6\pi\mu^3}}{6^{\mu}}.
\end{aligned}$$

As $\delta'(i)$ demonstrate the same asymptotic behavior, we conclude that:

$$\begin{aligned}
T(n,k) &\geq \frac{n(n-k)^{k-\mu}}{k^{k-\mu+1}\mu^{k-\mu}(\mu+1)(k-\mu+1)}\left(\frac{\sqrt{6\pi\mu^3}}{6^{\mu}}\right)^{k-\mu} \\
&= \Omega\left(\left(\frac{n}{k\mu}\right)^{k-\mu+1}\frac{\sqrt{6\pi}\mu^{3(k-u)/2}}{k6^{\mu(k-\mu)}}\right) \\
&= \Omega\left(\left(\frac{n\sqrt{\mu}}{6k}\right)^{k-\mu+1}\frac{1}{6^{\mu-1}k\cdot\sqrt{\mu^3}}\right). \quad \blacksquare
\end{aligned}$$