Theoretical Analyses of Multi-Objective Evolutionary Algorithms on Multi-Modal Objectives

Benjamin Doerr,[1*] **Weijie Zheng**[2*]

[1]Laboratoire d'Informatique (LIX), École Polytechnique, CNRS, Institut Polytechnique de Paris, Palaiseau, France
[2] Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China
doerr@lix.polytechnique.fr, zhengwj13@tsinghua.org.cn

Abstract

Previous theory work on multi-objective evolutionary algorithms considers mostly easy problems that are composed of unimodal objectives. This paper takes a first step towards a deeper understanding of how evolutionary algorithms solve multi-modal multi-objective problems. We propose the ONEJUMPZEROJUMP problem, a bi-objective problem whose single objectives are isomorphic to the classic jump functions benchmark. We prove that the simple evolutionary multi-objective optimizer (SEMO) cannot compute the full Pareto front. In contrast, for all problem sizes n and all jump sizes $k \in [4..\frac{n}{2}-1]$, the global SEMO (GSEMO) covers the Pareto front in $\Theta((n-2k)n^k)$ iterations in expectation. To improve the performance, we combine the GSEMO with two approaches, a heavy-tailed mutation operator and a stagnation detection strategy, that showed advantages in single-objective multi-modal problems. Runtime improvements of asymptotic order at least $k^{\Omega(k)}$ are shown for both strategies. Our experiments verify the substantial runtime gains already for moderate problem sizes. Overall, these results show that the ideas recently developed for single-objective evolutionary algorithms can be effectively employed also in multi-objective optimization.

Introduction

Many real-world applications contain multiple conflicting objectives. For such problems, a single best solution cannot be determined. Therefore, the task is to compute a set of solutions each of which cannot be improved without worsening in at least one objective (Pareto optima). With their population-based nature, multi-objective evolutionary algorithms (MOEAs) have been successfully applied here (Zhou et al. 2011). Similar to the situation in the theory of single-objective evolutionary algorithms, rigorous theoretical analyses of MOEAs fall far behind their successful applications in practice.

In order to reveal the working principles of MOEAs, the research has resorted to multi-objective, especially bi-objective, counterparts of well-analyzed single-objective benchmark functions used in evolutionary computation theory. For example, in the problems COCZ (Laumanns et al. 2002) and ONEMINMAX (Giel and Lehre 2010), the two objectives are both (conflicting) variants of the classic ONEMAX benchmark. The classic benchmark LEADINGONES was used to construct the LOTZ (Laumanns, Thiele, and Zitzler 2004) and WLPTNO (Qian, Yu, and Zhou 2013) problems. These multi-objective benchmark problems are among the most intensively studied (Giel 2003; Doerr, Kodric, and Voigt 2013; Doerr, Gao, and Neumann 2016; Bian, Qian, and Tang 2018; Huang et al. 2019; Huang and Zhou 2020; Osuna et al. 2020). We note that these problems are *unimodal* in the sense that from each set of solutions P a set P' witnessing the Pareto front can be computed by repeatedly selecting a solution from P, flipping a single bit in it, adding it to P, and removing dominated solutions from P. They are thus relatively easy to solve.

As in the theory of single-objective evolutionary computation, multi-modal problems are much less understood also in the theory of evolutionary multi-objective optimization. We defer a detailed discussion of the state of the art to **Section Related Works** and state here only that, to the best of our knowledge, there is not a single work discussing in detail how MOEAs cope with multimodality.[1] There are works that contain multimodal problems, but they are using these problems mainly to study other research questions or the multimodality is only minor (Brockhoff et al. 2007; Friedrich, Hebbinghaus, and Neumann 2010; Qian, Tang, and Zhou 2016; Li et al. 2016).

Our Contributions. This paper aims at a deeper understanding how MOEAs cope with multi-objective problems with natural, well-analyzed, multi-modal objectives. In the theory of single-objective evolutionary computation, the class of JUMP function is a natural and intensively used multi-modal function class (Droste, Jansen, and Wegener 2002) that has inspired many interesting results including that larger mutation rates, crossover, and estimation of distribution algorithms help in the optimization of multi-modal

*Both authors contributed equally to this work and both act as corresponding authors.

[1]To prevent misunderstandings, let us stress that by multi-modality, we refer to the fact that the optimization problems regarded are multi-modal, that is, have non-trivial local optima. Our work is not concerned with multi-modal optimization, which means finding all local optima of a problem.

functions (see, e.g., Doerr et al. 2017; Dang et al. 2018; Hasenöhrl and Sutton 2018). Hence, in this paper, we design a bi-objective counterpart of the JUMP function with problem size n and jump size k, called ONEJUMPZEROJUMP$_{n,k}$. It consists of one JUMP function w.r.t. the number of ones and one JUMP function w.r.t. the number of zeros. We compute its disconnected Pareto front in Theorem 5.

We prove for all n and $k \in [2..\frac{n}{2}]$ that the simple evolutionary multi-objective optimizer (SEMO) cannot find the Pareto front (Theorem 7), but that the global SEMO (GSEMO) finds the Pareto front in $O((n-2k)n^k)$ iterations in expectation (Theorem 8). We show a matching lower bound of $\Omega((n-2k)n^k)$ for $k \in [4..\frac{n}{2}-1]$ (Theorem 9). Here and in the remainder, the asymptotic notation only hides constants independent of n and k.

We also consider two approaches that showed advantages in single-objective multi-modal problems. Via the heavy-tailed mutation proposed by Doerr, Le, Makhmara, and Nguyen (2017), we improve the expected runtime of the GSEMO by a factor of $k^{\Omega(k)}$ to $O((n-2k)(en)^k/k^{k+0.5-\beta})$, where $\beta > 1$ is the power-law distribution parameter (Theorem 11). Via a suitable adaptation of the stagnation detection strategy from (Rajabi and Witt 2020) to multi-objective optimization, we obtain an expected runtime of $O((n-2k)(en)^k/k^k)$, again a $k^{\Omega(k)}$ factor improvement over the classic GSEMO and reducing the runtime guarantee for the heavy-tailed GSEMO by a (small) factor of $\Omega(k^{\beta-0.5})$, see Theorem 12. Our experiments show that these are not only asymptotic differences, but that roughly a factor-5 speed-up with heavy-tailed mutation and a factor-10 speed-up with stagnation detection can be observed already for jump size $k = 4$ and problem sizes n between 10 and 50.

Basic Definitions

A multi-objective optimization problem consists of maximizing multiple objectives simultaneously. This paper considers the maximization of bi-objective pseudo-Boolean problems $f = (f_1, f_2) : \{0,1\}^n \to \mathbb{R}^2$.

For any two search points x and y, we say that

- x *weakly dominates* y, denoted by $x \succeq y$, if and only if $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$;
- x *dominates* y, denoted by $x \succ y$, if and only if $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$ and at least one of the inequalities is strict.

We call $x \in \{0,1\}^n$ *Pareto optimal* if and only if there is no $y \in \{0,1\}^n$ such that $y \succ x$. All Pareto optimal solutions compose the *Pareto set*. The set of the function values of the Pareto set is called the *Pareto front*. For most multi-objective problems the objectives are at least partially conflicting and thus there is usually not a single Pareto optimum. Since a priori it is not clear which of several incomparable Pareto optima to prefer, the most common target is to compute the Pareto front, that is, compute a set P of solutions such that $f(P) := \{f(x) \mid x \in P\}$ is the Pareto front. This is our objective in this work as well. We note that if the Pareto front is excessively large, then one has to resort to approximating it in a suitable manner, but this will be not our problem here.

We will use $|x|_1$ and $|x|_0$ to denote the number of ones and zeros of the search point $x \in \{0,1\}^n$. We use $[a..b]$ to denote the set $\{a, a+1, \dots, b\}$ for $a, b \in \mathbb{Z}$ and $a \leq b$.

Related Works

Brockhoff, Friedrich, Hebbinghaus, Klein, Neumann, and Zitzler (2007) proposed the PLOM, PLZM, and PLATEAUS functions and used them to show that adding objectives can be both beneficial and detrimental. Friedrich, Hebbinghaus, and Neumann (2010) designed the PL function, which contains a larger plateau in one objective. They demonstrated that individuals generated from the solutions on the Pareto front may reset the undirected random walk on the plateau, which makes the multi-objective problem much harder than the two single objectives. In order to investigate the effect of mixing low-level heuristics, Qian, Tang, and Zhou (2016) designed the ZPLG and SPG functions (also containing plateaus), and showed that mixing fair selection w.r.t. the decision space and the objective space is beneficial for ZPLG, and that mixing 1-bit and 2-bit mutation is efficient for SPG. In the first theoretical study of decomposition-based MOEAs, Li, Zhou, Zhan, and Zhang (2016) besides analyses on classic unimodal multi-objective problems also defined two multi-modal problems and showed that the MOEA/D can solve both, whereas the two variants of the SEMO can only solve one of the two.

As is visible from this description, the few theoretical works in which a multi-modal multi-objective problem is regarded do this not with the intention of understanding how MOEAs cope with multimodality, but they use their multi-objective example problems to demonstrate particular strengths or weaknesses of different algorithms (that are not directly related to multimodality). Consequently, these multi-modal problems also are rather artificial as can be seen from the following description of the (only) three example problems which contain at least one multi-modal objective (where by multi-modal we mean that there is at least one local optimum that is separated by points of lower fitness from any global optimum). We note that in all these three examples, only one objective is multi-modal.

Definition 1 (ZPLG (Qian, Tang, and Zhou 2016))**.** *The function* $\text{ZPLG}(x) : \{0,1\}^n \to \mathbb{R} \times [0..2]$ *is defined by*

$$\begin{cases} (n+1,1), & \text{if } x = 1^i 0^{n-i}, i \in [1..\frac{3}{4}n-1]; \\ (n+2+i,0), & \text{if } x = 1^{\frac{3}{4}n+2i} 0^{\frac{1}{4}n-2i}, i \in [0..\frac{1}{8}n]; \\ (|x|_0, 2), & \text{else}. \end{cases}$$

Definition 2 (SPG (Qian, Tang, and Zhou 2016))**.** *The function* $\text{SPG}(x) : \{0,1\}^n \to \mathbb{R} \times \{0,1\}$ *is defined by*

$$\begin{cases} (-1,0), & \text{if } x = 1^i 0^{n-i}, i \bmod 3 = 1, i \in [1..n]; \\ (in,0), & \text{if } x = 1^i 0^{n-i}, i \bmod 3 \in \{0,2\}, i \in [1..n]; \\ (|x|_0, 1), & \text{else}. \end{cases}$$

Definition 3 (DEC-OBJ-MOP (Li et al. 2016))**.** *The function* DEC-OBJ-MOP$(x) : \{0,1\}^n \to \mathbb{R}^2$ *is defined by*

$$(n+1-|x|_0 \bmod n+1, n+|x|_0 \bmod n+1).$$

We note that the first objective of ZPLG has modals on $x = 1^{\frac{3}{4}n+2i}0^{\frac{1}{4}n-2i}, i \in [0..\frac{1}{8}n]$, the first objective of SPG has modals on $x = 0^n$ and $x = 1^i0^{n-i}, i \bmod 3 = 0, i \in [1..n]$, and the second objective of DEC-OBJ-MOP has two modals on $x \in \{0^n, 1^n\}$.

We note that there are also works on true combinatorial problems (e.g., Neumann 2007; Friedrich, Hebbinghaus, and Neumann 2010; Qian, Yu, and Zhou 2013, 2015; Qian, Zhang, Tang, and Yao 2018; Feng, Qian, and Tang 2019; Roostapour, Neumann, Neumann, and Friedrich 2019; Qian, Bian, and Feng 2020; Roostapour, Bossek, and Neumann 2020), some of which are multi-modal, but again these consider a particular optimization problem and give little general insight on how MOEAs cope with multimodality.

The ONEJUMPZEROJUMP Problem

To study via mathematical means how MOEAs cope with multimodality, we now define and analyze a class of bi-objective functions of scalable difficulty. This is strongly influenced by the single-objective JUMP function class proposed in (Droste, Jansen, and Wegener 2002), which is intensively used in the theory of single-objective evolutionary computation and which gave rise to many interesting results including that larger mutation rates help in the optimization of multi-modal functions (e.g., Doerr et al. 2017), that crossover can help to cope with multimodality (e.g., Jansen and Wegener 2002; Dang et al. 2018), and that estimation-of-distribution algorithms and the $(1 + (\lambda, \lambda))$ GA can significantly outperform classic evolutionary algorithms on multi-modal problems (e.g., Hasenöhrl and Sutton 2018; Doerr 2019; Antipov, Doerr, and Karavaev 2020; Antipov and Doerr 2020).

We recall that for all $n \in \mathbb{N}$ and $k \in [1..n]$, the jump function $\text{JUMP}_{n,k} : \{0,1\}^n \to \mathbb{R}$ is defined by $\text{JUMP}_{n,k}(x) = k+|x|_1$, if $|x|_1 \in [0..n-k] \cup \{n\}$ and $\text{JUMP}_{n,k}(x) = n-|x|_1$ otherwise. Hence for $k \geq 2$, this function has a valley of low fitness around its optimum, which can be crossed only by flipping k bits (or accepting solutions with very low fitness). We define the $\text{ONEJUMPZEROJUMP}_{n,k}$ function as a bi-objective counterpart of the function $\text{JUMP}_{n,k}$.

Definition 4 ($\text{ONEJUMPZEROJUMP}_{n,k}$). *Let $n \in \mathbb{N}$ and $k = [1..n]$. The function $\text{ONEJUMPZEROJUMP}_{n,k} = (f_1, f_2) : \{0,1\}^n \to \mathbb{R}^2$ is defined by*

$$f_1(x) = \begin{cases} k + |x|_1, & \text{if } |x|_1 \leq n-k \text{ or } x = 1^n, \\ n - |x|_1, & \text{else;} \end{cases}$$

$$f_2(x) = \begin{cases} k + |x|_0, & \text{if } |x|_0 \leq n-k \text{ or } x = 0^n, \\ n - |x|_0, & \text{else.} \end{cases}$$

Hence the first objective is just the classic $\text{JUMP}_{n,k}$ function. The second objective has a fitness landscape isomorphic to this function, but the roles of zeros and ones are exchanged, that it, $f_2(x) = \text{JUMP}_{n,k}(1^n - x)$. Figure 1 displays these two functions and in particular the two modals on $|x|_1 = n-k$ and $x = 1^n$ for the first objective and two modals on $|x|_1 = k$ and $x = 0^n$ for the second objective.

The following theorem determines the Pareto set and front of the $\text{ONEJUMPZEROJUMP}_{n,k}$ function. As Figure 1 suggests, the Pareto set consists of an inner region of all search points x with $|x|_1 \in [k..n-k]$ and the two extremal points 1^n and 0^n, as visualized in Figure 2.

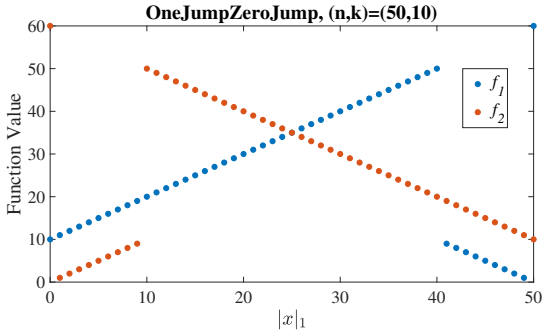

Figure 1: The values of two objectives in $\text{ONEJUMPZEROJUMP}_{n,k}$ with respect to $|x|_1$, the number of ones in the search point x.

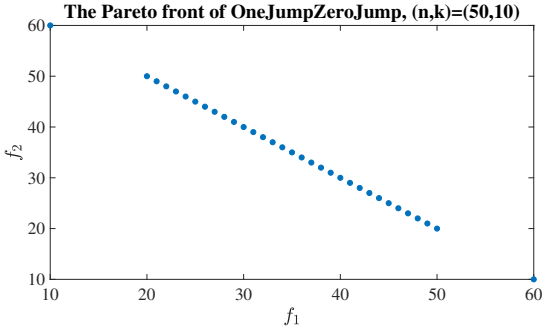

Figure 2: The Pareto front for the $\text{ONEJUMPZEROJUMP}_{n,k}$ function with $(n,k) = (50,10)$.

Theorem 5. *The Pareto set of the $\text{ONEJUMPZEROJUMP}_{n,k}$ function is $S^* = \{x \mid |x|_1 \in [k..n-k] \cup \{0,n\}\}$, and the Pareto front is $F^* = \{(a, 2k+n-a) \mid a \in [2k..n] \cup \{k, n+k\}\}$.*

For reasons of space, all mathematical proofs are omitted in this extended abstract. They can be found in the preprint (Doerr and Zheng 2020).

From Theorem 5, we easily obtain in the following corollary a general upper bound on the size of any set of solutions without pair-wise weak domination, and thus also on the size of the population in the algorithms discussed in this work.

Corollary 6. *Consider any set of solutions P such that $x \not\preceq y$ w.r.t. $\text{ONEJUMPZEROJUMP}_{n,k}$ for all $x, y \in P$ and $x \neq y$. Then $|P| \leq n - 2k + 3$.*

SEMO Cannot Optimize ONEJUMPZEROJUMP Functions

The simple evolutionary multi-objective optimizer (SEMO), proposed by Laumanns et al. (2002), is a well-analyzed basic benchmark algorithm in multi-objective evolutionary theory (Qian, Yu, and Zhou 2013; Li et al. 2016). It is a multi-objective analogue of the randomized local search (RLS) al-

Algorithm 1 SEMO

1: Generate $x \in \{0,1\}^n$ uniformly at random and $P \leftarrow \{x\}$
2: **loop**
3: Uniformly at random select one individual x from P
4: Generate x' via flipping one bit chosen uniformly at random
5: **if** there is no $y \in P$ such that $x' \preceq y$ **then**
6: $P = \{z \in P \mid z \not\preceq x'\} \cup \{x'\}$
7: **end if**
8: **end loop**

Algorithm 2 GSEMO

1: Generate $x \in \{0,1\}^n$ uniformly at random and $P \leftarrow \{x\}$
2: **loop**
3: Uniformly at random select one individual x from P
4: Generate x' via independently flipping each bit value of x with probability $1/n$
5: **if** there is no $y \in P$ such that $x' \preceq y$ **then**
6: $P = \{z \in P \mid z \not\preceq x'\} \cup \{x'\}$
7: **end if**
8: **end loop**

gorithm, which starts with a random individual and tries to improve it by repeatedly flipping a single random bit and accepting the better of parent and this offspring. As a multi-objective algorithm trying to compute the full Pareto front, the SEMO naturally has to work with a non-trivial population. This is initialized with a single random individual. In each iteration, a random parent is chosen from the population. It generates an offspring by flipping a random bit. The offspring enters the population if it is not weakly dominated by some individual already in the population. In this case, any individual dominated by it is removed from the population. The details of SEMO are shown in Algorithm 1.

In the following Theorem 7, we show that the SEMO cannot cope with the multimodality of the ONEJUMPZEROJUMP$_{n,k}$ function, since with one-bit flips alone it cannot traverse the valleys of low fitness.

Theorem 7. *For all $n, k \in \mathbb{N}$ with $k \in [2..\lfloor \frac{n}{2} \rfloor]$, the SEMO cannot optimize the* ONEJUMPZEROJUMP$_{n,k}$ *function.*

Runtime Analysis for the GSEMO

As the previous section showed, to deal with multi-modal problems a mutation-based algorithm needs to be able to flip more than single bits. The global SEMO (GSEMO), proposed by Giel (2003), is a well-analyzed MOEA that has this ability. Generalized from the $(1+1)$ EA algorithm, it uses the standard bit-wise mutation, that is, each bit is flipped independently with the same probability of, usually, $1/n$. The details are shown in Algorithm 2.

The standard bit-wise mutation in GSEMO ensures a $\Theta(n^{-k})$ probability of reaching the outer solution 0^n or 1^n from one closest inner solution. Together with the probability $\Omega(1/n)$ of selecting such an inner solution via Corollary 6, we obtain that the GSEMO can find the full Pareto front and does so in expected time $O(n^{k+1})$.

Theorem 8. *Let $n \in \mathbb{N}_{\geq 2}$ and $k \in [1..\lfloor \frac{n}{2} \rfloor]$. The expected runtime of the GSEMO optimizing the* ONEJUMPZEROJUMP$_{n,k}$ *function is at most $e(n-2k+3)(\frac{3}{2}n^k + 2n\ln\lceil \frac{n}{2} \rceil + 3)$.*

We finally show that this bound is very tight. For convenience, we only consider the case $k \geq 4$, but we are convinced that similar bounds can be shown also for $k = 2$ and $k = 3$.

Theorem 9. *Let $n \in \mathbb{N}_{\geq 8}$ and $k \in [4..\frac{n}{2} - 1]$. The expected runtime of the GSEMO optimizing the* ONEJUMPZEROJUMP$_{n,k}$ *function is at least $\frac{3}{2}e(n-2k+1)n^k \frac{n^k}{(n-1)^k}\left(1 - \frac{1}{n^{n/4-2}} - \frac{2}{n^{n-2k}} - \frac{e+3}{n} - \frac{4e(\ln n+1)}{n^{k-3}}\right)$.*

GSEMO with Heavy-Tailed Mutation

In the previous section, we have shown that the GSEMO can optimize our multi-modal optimization problem, but similar to the single-objective world (say, the optimization of JUMP functions via simple evolutionary algorithms (Droste, Jansen, and Wegener 2002)), the runtime increases significantly with the distance k a solution on the Pareto front can have from all other solutions on the front. As has been discussed in (Doerr et al. 2017), increasing the mutation rate can improve the time it takes to jump over such gaps. However, this work also showed that a deviation from the optimal mutation rate can be costly: A small constant-factor deviation from the optimal rate k/n leads to a performance loss exponential in k. For this reason, a heavy-tailed mutation operator was proposed. Compared to using the optimal (usually unknown) rate, it only loses a small polynomial factor (in k) in performance.

We now equip the GSEMO with the heavy-tailed mutation from Doerr et al. (2017) and observe similar advantages. We use the following discrete power-law distribution.

Definition 10 (Power-law distribution $D^\beta_{n/2}$)**.** *Let $n \in \mathbb{N}_{\geq 2}$ and $\beta > 1$. Let $C^\beta_{n/2} := \sum_{i=1}^{n/2} i^{-\beta}$. We say a random variable ξ follows the power-low $D^\beta_{n/2}$, written as $\xi \sim D^\beta_{n/2}$, if for all $\alpha \in [1..n/2]$, we have $\Pr[\xi = \alpha] = \left(C^\beta_{n/2}\right)^{-1} \alpha^{-\beta}$.*

The heavy-tailed mutation operator proposed by Doerr et al. (2017), in the remainder denoted by MUT$^\beta(\cdot)$, in each application independently samples a number α from the power-law distribution $D^\beta_{n/2}$ and then uses standard bit-wise mutation with mutation rate α/n, that is, flips each bit independently with probability α/n. Equipping the standard GSEMO with this mutation operator MUT$^\beta$ gives Algorithm 3, which we call GSEMO-HTM.

GSEMO-HTM on ONEJUMPZEROJUMP$_{n,k}$

We now analyze the runtime of the GSEMO-HTM on ONEJUMPZEROJUMP$_{n,k}$.

Theorem 11. *The expected runtime of the GSEMO-HTM optimizing the* ONEJUMPZEROJUMP$_{n,k}$ *function is at most $(n-2k+3)O(k^{\beta-0.5})C^\beta_{n/2}\frac{n^n}{k^k(n-k)^{n-k}}$.*

Algorithm 3 The GSEMO-HTM algorithm with power-law exponent $\beta > 1$

Generate $x \in \{0,1\}^n$ uniformly at random, and $P \leftarrow \{x\}$
loop
  Uniformly at random select one individual x from P
  Sample α from $D^\beta_{n/2}$ and generate x' via independently flipping each bit value of x with probability α/n
  if there is no $y \in P$ such that $x' \preceq y$ **then**
    $P = \{z \in P \mid z \not\preceq x'\} \cup \{x'\}$
  end if
end loop

Comparing Theorem 8 and Theorem 11, we could know the asymptotic expected runtime of the GSEMO-HTM on ONEJUMPZEROJUMP$_{n,k}$ is smaller than that of the GSEMO with the factor around $k^{k+0.5-\beta}/e^k$.

GSEMO with Stagnation Detection

In this section, we discuss how to adapt the stagnation detection strategy proposed by Rajabi and Witt (2020) to multi-objective optimization. We obtain a variant of the GSEMO that has a slightly better asymptotic performance on ONEJUMPZEROJUMP than the one with heavy-tailed mutation. However, we also speculate that this strategy may have difficulties with plateaus of constant fitness.

The Stagnation Detection Strategy of Rajabi and Witt

Rajabi and Witt (2020) proposed the following strategy to adjust the mutation rate during the run of the $(1+1)$ EA. We recall that the $(1+1)$ EA has a population size of one, that is, it generates in each iteration one offspring via mutation and accepts this if it is at least as good as the parent. The classic mutation operator for this algorithm is standard bit-wise mutation with mutation rate $1/n$, that is, the offspring is generated by flipping each bit of the parent independently with probability $1/n$.

The main idea of their approach is the following. Assume that the $(1+1)$ EA for a longer time, say at least $10n \ln n$ iterations, has not accepted any new solution. Then with high probability, it has generated (and rejected) all Hamming neighbors of the parent. Consequently, there is no use to generate these solutions again and the algorithm should better concentrate on solutions further away from the parent. This can be achieved conveniently by increasing the mutation rate (say, to $\frac{2}{n}$; this reduces significantly the rate of Hamming neighbors produced, but Hamming neighbors can still be generated, which is important in case we were unlucky so far and missed one of them).

More generally and more precisely, in the self-adjusting mutation rate strategy based on stagnation detection the $(1+1)$ EA maintains a counter ("failure counter") that keeps track of how long the parent individual has not given rise to a better offspring. This counter determines the current mutation rate. This dependency is governed by a safety parameter R which is recommended to be at least n. Then for $r = 1, 2, \dots$ in this order the mutation rate r/n is used for

$$T_r := \lceil 2(\tfrac{en}{r})^r \ln(nR) \rceil \tag{1}$$

iterations. When a strictly improving solution is found, the counter is reset to zero (and consequently, the mutation rate starts again at $\frac{1}{n}$).

Rajabi and Witt show that the $(1+1)$ EA with this strategy optimizes JUMP$_{n,k}$ with $k = o(n)$ in time $\Omega((\frac{en}{k})^k(1 - \frac{k^2}{n-k}))$ and $O((\frac{en}{k})^k)$. In particular, for $k = o(\sqrt{n})$, a tight (apart from constant factors independent of k and n) bound of $\Theta((\frac{en}{k})^k)$ is obtained. This is faster than the runtime of $\Theta(k^{\beta-0.5}(\frac{en}{k})^k)$ proven in (Doerr et al. 2017) for the $(1+1)$ EA with heavy-tailed mutation with power-law exponent $\beta > 1$ by a factor of $k^{\beta-0.5}$. For the recommended choice $\beta = 1.5$, this factor is $\Theta(k)$.

Adaptation of the Stagnation Detection Strategy to Multi-Objective Optimization

As the $(1+1)$ EA is an algorithm without a real population, it is clear that certain adaptations are required to use the stagnation detection strategy in multi-objective optimization.

Global or Individual Failure Counters The first question is how to count the number of unsuccessful iterations. The following two obvious alternatives exist.

Individual counters: From the basic idea of the stagnation detection strategy, the most natural solution is to equip each individual with its own counter. Whenever an individual is chosen as parent in the GSEMO, its counter is increased by one. New solutions (but see the following subsection for an important technicality of what "new" shall mean) entering the population start with a counter value of zero.

A global counter: Algorithmically simpler is the approach to use only one global counter. This counter is increased in each iteration. When a new solution enters the population, the global counter is reset to zero.

We suspect that for many problems, both ways of counting give similar results. The global counter appears to be wasteful in the sense that when a new individual enters the population, also parents that are contained in the population for a long time re-start generating offspring with mutation rate $\frac{1}{n}$ despite the fact that they have, with very high probability, already generated as offspring all solutions close by. On the other hand, often these "old individuals" do not generate solutions that enter the population anyway, so that an optimized choice of the mutation rate is less important.

For the ONEJUMPZEROJUMP problem, it is quite clear that this second effect is dominant. A typical run starts with some individual in the middle region of the Pareto front. In relatively short time, the whole middle region is covered, and for this it suffices that relatively recent solutions generate a suitable Hamming neighbor as offspring. The runtime is dominated by the time to find the two extremal solutions and this will almost always happen from the closest parent in the middle region of the front. For this reason, we analyze in the following the simpler approach using a global counter.

Dealing with Indifferent Solutions One question that becomes critical when using stagnation detection is how to deal with indifferent solutions, that is, which solution to put or keep in the population in the case that an offspring y has the same (multi-objective) fitness as an individual x already in the population. Since $f(x) = f(y)$, we have $x \preceq y$ and $y \preceq x$, that is, both solutions do an equally good job in dominating others and thus in approximating the Pareto front. In early works, e.g. (Laumanns et al. 2002) proposing the SEMO algorithm, such later generated indifferent solutions do not enter the population. This is partially justified by the fact that in many of the problems regarded in these works, search points with equal fitness are fully equivalent for the future run of the algorithm. We note that our ONEJUMPZEROJUMP problem also has this property, hence all results presented so far are valid regardless of how indifferent solutions are treated.

When non-equivalent search points with equal fitness exist, it is less obvious how to deal with indifferent solutions. In particular, it is clear that larger plateaus of constant fitness can be traversed much easier when a new indifferent solution is accepted as this allows to imitate a random walk behavior on the plateau (Brockhoff et al. 2007). For that reason, and in analogy to single-objective optimization (Jansen and Wegener 2001), it seems generally more appropriate to let a new indifferent solution enter the population, and this is also what most of the later works on the SEMO and GSEMO algorithm do (Friedrich, Hebbinghaus, and Neumann 2010; Friedrich, Horoba, and Neumann 2011; Qian, Tang, and Zhou 2016; Li et al. 2016; Bian, Qian, and Tang 2018; Osuna et al. 2020).

Unfortunately, it is not so clear how to handle indifferent solutions together with stagnation detection. In principle, when a new solution enters the population, the failure counter has to be reset to zero to reset the mutation rate to $1/n$. Otherwise, the continued use of a high mutation rate would prohibit finding good solutions in the direct neighborhood of the new solution. However, the acceptance of indifferent solutions can also lead to unwanted resets. For the ONEJUMPZEROJUMP problem, for example, it is easy to see by mathematical means that in a typical run, it will happen very frequently that an indifferent solution is generated. If this enters the population with a reset of a global failure counter (or an individual counter), then the regular resets will prevent the counters to reach interesting values. In a quick experiment for $n = 50$, $k = 4$, and a global counter, the largest counter value ever reached in this run of over 500,000,000 iterations was 5. Consequently, this SD-GSEMO was far from ever increasing the mutation rate and just imitated the classic GSEMO.

For that reason, in this work we regard the GSEMO with stagnation detection only in the variant that does not accept indifferent solutions, and we take note of the fact that thus our positive results on the stagnation detection mechanism will not take over to problems with non-trivial plateaus of constant fitness.

Adjusting the Self-Adjustment In the $(1+1)$ EA with stagnation detection, Rajabi and Witt (2020) increased the mutation rate from $\frac{r}{n}$ to $\frac{r+1}{n}$ once the rate $\frac{r}{n}$ has been used for T_r iterations with T_r as defined in (1). This choice ensured that any particular target solution in Hamming distance r is found in this phase with probability at least $1 - (nR)^{-2}$, see the proof of Lemma 3.1 in (Rajabi and Witt 2020). Since in a run of the GSEMO with current population size $|P|$ each member of the population is chosen as parent only an expected number of $T_r/|P|$ times in a time interval of length T_r, we need to adjust the parameter T_r. Not surprisingly, by taking

$$\tilde{T_r} = \lceil 2\,|P|\,(\tfrac{en}{r})^r \ln(nR) \rceil, \tag{2}$$

that is, roughly $|P|\,T_r$, the probability to generate any particular solution in Hamming distance r in phase r is at least

$$1 - \left(1 - \frac{1}{|P|}\left(\frac{r}{n}\right)^r \left(1 - \frac{r}{n}\right)^{n-r}\right)^{2|P|(en)^r \ln(nR)/r^r} \geq 1 - \frac{1}{(nR)^2},$$

which is sufficient for this purpose. Note that the population size $|P|$ changes only if a new solution enters the population and in this case the mutation rate is reset to $\frac{1}{n}$. Hence the definition of $\tilde{T_r}$, with the convention that we suppress $|P|$ in the notation to ease reading, is unambiguous.

Algorithm 4 SD-GSEMO with safety parameter R

```
Generate x ∈ {0,1}^n uniformly at random, P ← {x}
r ← 1 and u ← 0
loop
    Uniformly and randomly select x from P
    Generate x' via independently flipping each bit value of x with probability r/n
    u ← u + 1
    if there is no y ∈ P such that x' ⪯ y then
        P = {z ∈ P | z ⋠ x'} ∪ {x'}
        r ← 1 and u ← 0
    end if
    if u > 2|P|(en/r)^r ln(nR) then
        r ← min{r + 1, n/2} and u ← 0
    end if
end loop
```

The GSEMO with Stagnation Detection: SD-GSEMO Putting the design choices discussed so far together, we obtain the following variant of the GSEMO, called SD-GSEMO. Its pseudocode is shown in Algorithm 4.

SD-GSEMO on ONEJUMPZEROJUMP$_{n,k}$

We now analyze the runtime of the SD-GSEMO on the ONEJUMPZEROJUMP function class. This will show that its expected runtime is by a small polynomial (in k) factor smaller than the one of the heavy-tailed GSEMO (which was a factor of $k^{\Omega(k)}$ smaller than the one of the GSEMO).

Theorem 12. *The expected runtime of the SD-GSEMO optimizing the* ONEJUMPZEROJUMP$_{n,k}$ *function is at most* $(n-2k+3)(\frac{en}{k})^k(\frac{3}{2} + (\frac{4k}{n} + \frac{12}{nk})\ln(nR)) + 3e(n-2k+3)(n\ln n + 2(n-2)\ln(nR))$.

Assume that, as suggested in (Rajabi and Witt 2020), the control parameter R is set to n. Then the dominating element of the upper bound in Theorem 12 becomes $(n-2k+3)(\frac{en}{k})^k(\frac{3}{2}+8(\frac{k}{n}+\frac{3}{nk})\ln n)$. Hence if $k=O(\frac{n}{\ln n})$, then the runtime of SD-GSEMO on ONEJUMPZEROJUMP$_{n,k}$ is $O((n-2k)(\frac{en}{k})^k)$.

Experiments

To understand the performance of the algorithms discussed in this work for concrete problems sizes (for which an asymptotic mathematical analysis cannot give definite answers), we now conduct a simple experimental analysis. Since the SEMO cannot find the Pareto front, we did not include it in this investigation. We did include the variant of the SD-GSEMO, denoted by SD-GSEMO-Ind, in which each individual has its own failure counter (see the discussion in **Section GSEMO with Stagnation Detection**). Our experimental settings are the same for all algorithms.

- ONEJUMPZEROJUMP$_{n,k}$: jump size $k=4$ and problem size $n=10,14,\dots,50$.
- $\beta=1.5$ as suggested in (Doerr et al. 2017) for the power-law distribution in GSEMO-HTM.
- $R=n$ for SD-GSEMO and SD-GSEMO-Ind as suggested in (Rajabi and Witt 2020).
- 20 independent runs for each setting.

Figure 3 shows the mean number of function evaluations of GSEMO, GSEMO-HTM, SD-GSEMO, and SD-GSEMO-Ind on the ONEJUMPZEROJUMP$_{n,k}$ function. To see how the experimental results compare with our bounds, we also plot (i) the curve $1.5e(n-2k)n^k$ corresponding to the bounds for the GSEMO in Theorems 8 and 9, (ii) the curve $(n-2k)(en)^k/k^{k-1}$ for the GSEMO-HTM with $\beta=1.5$; since the leading constant in Theorem 11 is implicit, we chose a constant such that the curve matches the experimental data, and (iii) the curve $1.5(n-2k)(en)^k/k^k$ corresponding to the upper bound of SD-GSEMO with $R=n$ in Theorem 12.

We clearly see that these curves, in terms of shape and, where known, in terms of leading constants, match well the estimates of our theoretical runtime results. We also see, as predicted by informal considerations, the similarity of the performance of the SD-GSEMO and the SD-GSEMO-Ind. Finally, our experiments show that the different runtime behaviors are already visible for moderate (and thus realistic) problem sizes and not only in the asymptotic sense in which they were proven. In particular, we observe a performance improvement by a factor of (roughly) 5 through the use heavy-tailed mutation and by a factor of (roughly) 10 with the stagnation detection strategy.

Conclusion and Outlook

To increase the under-developed theoretical understanding how MOEAs cope with multimodality, we defined a class of bi-objective benchmark functions of scalable difficulty, ONEJUMPZEROJUMP$_{n,k}$. We proved that the SEMO cannot compute the full Pareto front. In contrast, for all problem

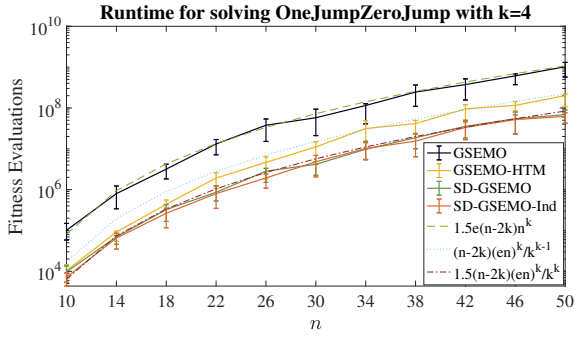

Figure 3: The mean number of function evaluations (with the first and third quartiles) of GSEMO, GSEMO-HTM, SD-GSEMO, and SD-GSEMO-Ind on ONEJUMPZEROJUMP$_{n,k}$ with $k=4$ and $n=10:4:50$ in 20 independent runs.

sizes n and jump sizes $k\in[4..\frac{n}{2}-1]$, the GSEMO covered the Pareto front in $\Theta((n-2k)n^k)$ iterations in expectation.

This paper also introduced two approaches that showed advantages in single-objective multi-modal problems into the toolbox of MOEAs. One is to use a heavy-tailed mutation operator in the GSEMO, the other is to self-adapt the mutation rate based on a stagnation detection strategy in the GSEMO. For both approaches we proved a runtime improvement over the standard GSEMO by a factor of $k^{\Theta(k)}$, with the self-adjusting GSEMO slightly ahead by a small polynomial factor in k. Our experiments confirmed this performance ranking already for moderate problem sizes, with the self-adjusting GSEMO more ahead than what the asymptotically small advantage suggests. On the downside, adapting the stagnation detection mechanism to MOEAs needs taking several design choices, among which the question how to treat indifferent solutions could be difficult for problems having larger plateaus of constant fitness.

Overall, this work suggests that the recently developed ideas to cope with multimodality in single-objective evolutionary optimization can be effective in multi-objective optimization as well. In this first work in this direction, we only concentrated on mutation-based algorithms. The theory of evolutionary computation has also observed that crossover and estimation-of-distribution algorithms can be helpful in multi-modal optimization. Investigating to what degree these results extend into multi-objective optimization is clearly an interesting direction for future research.

Also, we only covered very simple MOEAs in this work. Analyzing more complex MOEAs such as the successful decomposition-based MOEA/D (Zhang and Li 2007; Li et al. 2016; Huang et al. 2019; Huang and Zhou 2020) would be highly interesting. This would most likely require an adaptation of our benchmark problem. Since the difficult-to-find extremal points of the front are just the solutions of the single-objective sub-problems, and thus the two problems that naturally are part of the set of subproblems regarded by the MOEA/D, this algorithm might have an unfair advantage on the ONEJUMPZEROJUMP problem.

Acknowledgments

This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

This work was also supported by Guangdong Basic and Applied Basic Research Foundation (Grant No. 2019A1515110177); Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386); Shenzhen Peacock Plan (Grant No. KQTD2016112514355531); and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

References

Antipov, D.; and Doerr, B. 2020. Runtime Analysis of a Heavy-Tailed $(1+(\lambda, \lambda))$ Genetic Algorithm on Jump Functions. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*, 545–559. Springer.

Antipov, D.; Doerr, B.; and Karavaev, V. 2020. The $(1 + (\lambda, \lambda))$ GA is even faster on multimodal problems. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, 1259–1267. ACM.

Bian, C.; Qian, C.; and Tang, K. 2018. A general approach to running time analysis of multi-objective evolutionary algorithms. In *International Joint Conference on Artificial Intelligence, IJCAI 2018*, 1405–1411. IJCAI.

Brockhoff, D.; Friedrich, T.; Hebbinghaus, N.; Klein, C.; Neumann, F.; and Zitzler, E. 2007. Do additional objectives make a problem harder? In *Genetic and Evolutionary Computation, GECCO 2007*, 765–772. ACM.

Dang, D.; Friedrich, T.; Kötzing, T.; Krejca, M. S.; Lehre, P. K.; Oliveto, P. S.; Sudholt, D.; and Sutton, A. M. 2018. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation* 22: 484–497.

Doerr, B. 2019. A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost. In *Genetic and Evolutionary Computation Conference, GECCO 2019*, 1488–1496. ACM.

Doerr, B.; Gao, W.; and Neumann, F. 2016. Runtime analysis of evolutionary diversity maximization for ONEMINMAX. In *Genetic and Evolutionary Computation Conference, GECCO 2016*, 557–564. ACM.

Doerr, B.; Kodric, B.; and Voigt, M. 2013. Lower bounds for the runtime of a global multi-objective evolutionary algorithm. In *IEEE Congress on Evolutionary Computation, CEC 2013*, 432–439. IEEE.

Doerr, B.; Le, H. P.; Makhmara, R.; and Nguyen, T. D. 2017. Fast genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2017*, 777–784. ACM.

Doerr, B.; and Zheng, W. 2020. Theoretical Analyses of Multi-Objective Evolutionary Algorithms on Multi-Modal Objectives. *arXiv preprint arXiv:2012.07231* .

Droste, S.; Jansen, T.; and Wegener, I. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* 276: 51–81.

Feng, C.; Qian, C.; and Tang, K. 2019. Unsupervised feature selection by Pareto optimization. In *AAAI Conference on Artificial Intelligence, AAAI 2019*, volume 33, 3534–3541. AAAI.

Friedrich, T.; Hebbinghaus, N.; and Neumann, F. 2010. Plateaus can be harder in multi-objective optimization. *Theoretical Computer Science* 411(6): 854–864.

Friedrich, T.; Horoba, C.; and Neumann, F. 2011. Illustration of fairness in evolutionary multi-objective optimization. *Theoretical Computer Science* 412(17): 1546–1556.

Giel, O. 2003. Expected runtimes of a simple multi-objective evolutionary algorithm. In *IEEE Congress on Evolutionary Computation, CEC 2003*, volume 3, 1918–1925. IEEE.

Giel, O.; and Lehre, P. K. 2010. On the effect of populations in evolutionary multi-objective optimisation. *Evolutionary Computation* 18(3): 335–356.

Hasenöhrl, V.; and Sutton, A. M. 2018. On the runtime dynamics of the compact genetic algorithm on jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2018*, 967–974. ACM.

Huang, Z.; and Zhou, Y. 2020. Runtime analysis of somatic contiguous hypermutation operators in MOEA/D framework. In *AAAI Conference on Artificial Intelligence, AAAI 2020*, volume 34, 2359–2366. AAAI.

Huang, Z.; Zhou, Y.; Chen, Z.; and He, X. 2019. Running time analysis of MOEA/D with crossover on discrete optimization problem. In *AAAI Conference on Artificial Intelligence, AAAI 2019*, volume 33, 2296–2303. AAAI.

Jansen, T.; and Wegener, I. 2001. Evolutionary algorithms - how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation* 5: 589–599.

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

Laumanns, M.; Thiele, L.; and Zitzler, E. 2004. Running time analysis of multiobjective evolutionary algorithms on pseudo-boolean functions. *IEEE Transactions on Evolutionary Computation* 8(2): 170–182.

Laumanns, M.; Thiele, L.; Zitzler, E.; Welzl, E.; and Deb, K. 2002. Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In *International Conference on Parallel Problem Solving from Nature, PPSN 2002*, 44–53. Springer.

Li, Y.-L.; Zhou, Y.-R.; Zhan, Z.-H.; and Zhang, J. 2016. A primary theoretical study on decomposition-based multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 20(4): 563–576.

Neumann, F. 2007. Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning

tree problem. *European Journal of Operational Research* 181(3): 1620–1629.

Osuna, E. C.; Gao, W.; Neumann, F.; and Sudholt, D. 2020. Design and analysis of diversity-based parent selection schemes for speeding up evolutionary multi-objective optimisation. *Theoretical Computer Science* 832: 123–142.

Qian, C.; Bian, C.; and Feng, C. 2020. Subset selection by Pareto optimization with recombination. In *AAAI Conference on Artificial Intelligence, AAAI 2020*, 2408–2415. AAAI.

Qian, C.; Tang, K.; and Zhou, Z.-H. 2016. Selection hyper-heuristics can provably be helpful in evolutionary multi-objective optimization. In *International Conference on Parallel Problem Solving from Nature, PPSN 2016*, 835–846. Springer.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2013. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence* 204: 99–119.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015. Subset selection by Pareto optimization. In *Advances in Neural Information Processing Systems, NIPS 2015*, 1774–1782. Curran Associates, Inc.

Qian, C.; Zhang, Y.; Tang, K.; and Yao, X. 2018. On Multiset Selection With Size Constraints. In *AAAI Conference on Artificial Intelligence, AAAI 2018*, 1395–1402. AAAI.

Rajabi, A.; and Witt, C. 2020. Self-Adjusting Evolutionary Algorithms for Multimodal Optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, 1314–1322. ACM.

Roostapour, V.; Bossek, J.; and Neumann, F. 2020. Runtime analysis of evolutionary algorithms with biased mutation for the multi-objective minimum spanning tree problem. In *Genetic and Evolutionary Computation Conference, GECCO 2020*, 551–559. ACM.

Roostapour, V.; Neumann, A.; Neumann, F.; and Friedrich, T. 2019. Pareto optimization for subset selection with dynamic cost constraints. In *AAAI Conference on Artificial Intelligence, AAAI 2019*, volume 33, 2354–2361. AAAI.

Zhang, Q.; and Li, H. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11(6): 712–731.

Zhou, A.; Qu, B.-Y.; Li, H.; Zhao, S.-Z.; Suganthan, P. N.; and Zhang, Q. 2011. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation* 1(1): 32–49.