Noname manuscript No.
(will be inserted by the editor)

Splitting for Multi-objective Optimization

Qibin Duan · Dirk P. Kroese

Received: date / Accepted: date

Abstract We introduce a new multi-objective optimization (MOO) methodology based the splitting technique for rare-event simulation. The method generalizes the elite set selection of the traditional splitting framework, and uses both local and global sampling to sample in the decision space. In addition, an ε-dominance method is employed to maintain good solutions. The algorithm was compared with state-of-the art MOO algorithms using a prevailing set of benchmark problems. Numerical experiments demonstrate that the new algorithm is competitive with the well-established MOO algorithms and that it can outperform the best of them in various cases.

Keywords Splitting method · Multi-objective optimization · Pareto front · Pareto set · Benchmarking · Inverted Generational Distance

1 Introduction

Many optimization problems arising in science, engineering, economics, finance and logistics, have multiple, and often conflicting, objectives. It is natural to formulate such problems as *multi-objective optimization* (MOO) problems, also known as multi-criteria or vector optimization problems. In such problems, the goal is to optimize multiple conflicting objective functions simultaneously. Unlike single-objective optimization, there does not usually exist a single solution that optimizes all objective functions, and so one usually has a (possibly infinite) number of solutions with optimal trade-offs. The set of such solutions is called the *Pareto optimal front*. Research that focuses on solving MOPs usually aims to find the Pareto optimal front, which is a NP-hard problem.

It is, in principle, possible to obtain the Pareto front by solving many single-objective optimization problems; each corresponding to a specific preference ordering of the objectives. However, this is very time-consuming. Instead, it is often useful to take an evolutionary algorithm approach to find the Pareto front, as such algorithms can deal with the multitude

Qibin Duan · Dirk P. Kroese
School of Mathematics and Physics, The University of Queensland
Brisbane 4072, Australia
E-mail: q.duan@uq.edu.au
E-mail: kroese@maths.uq.edu.au

of candidate solutions simultaneously. Many multi-objective evolutionary algorithms (MOEAs) have been proposed over the last two decades; examples are the non-dominated sorting genetic algorithm (NSGAII)[7], Pareto-archive evolution strategy (PAES)[10], multi-objective particle swarm optimization (MOPSO) [5], multi-objective evolutionary algorithm based on decomposition (MOEA/D) [22], generalized differential evolution 3 (GDE3) [13], multi-objective cross-entropy method (MOCE)[21], and multi-objective artificial bee colony (MOABC) [1]. For more information on MOEAs, see a recent survey [25]. These algorithms usually originate from single-objective optimization counterparts.

In [8] a new optimization method was introduced based on the well-known *splitting* method for rare-event simulation; see, e.g., [19, Chapter 9] for a recent and detailed description of the splitting method. This optimization method, called SCO (Splitting for Continuous Optimization), has proved to be very successful for solving single-objective optimization problems. However, it was not clear if and how the method could be generalized to tackle MOO problems. Our aim in this paper is to extend the SCO algorithm to the multi-objective case. The resulting algorithm is called *Multi-Objective Splitting* (MOS).

Because there is a substantial difference between multi-objective and single-objective optimization, the methodology of the single-objective SCO method must be modified significantly to make the splitting idea work for MOO problems. First, when selecting the "elite set" the traditional sorting method is not suitable, so we provide a new method for constructing the elite set. Second, to obtain better candidates in the "splitting stage", we use a combination of the global sampling strategy used in [8] and a new local sampling strategy. Third, to keep track of all the good solutions ever find, we use an external "archive" of solutions.

The rest of the paper is organized as follows. In Section 2, we review the basic knowledge about Pareto optimality and give the main ideas of how the splitting method from rare-event simulation can be used as a technique for single-objective optimization. In Section 3, the details of the new MOS algorithm are given, which includes algorithms for the construction of the elite set, sampling strategies, and rules to update the archive solutions. In Section 4 we test the proposed method on a number of benchmark problems from the CEC'09 suite and compare it with state-of-the-art algorithms. Finally, in Section 5, we further analyze the results of the numerical experiments and discuss the parameter selection of the proposed methods.

2 Preliminaries

In this section, we review the basic facts about Pareto optimality and the splitting method.

2.1 Pareto optimality

An MOO problem is an optimization problem that involves multiple conflicting objective functions that are to be optimized simultaneously. Without loss of generality, we assume minimization throughout this paper. As in [6], a general MOO problem can be formulated as

$$\begin{aligned} &\min \mathbf{f}(\mathbf{x}) := (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_K(\mathbf{x})) \\ &\text{s.t.} \quad \mathbf{x} \in \mathscr{X}, \end{aligned} \tag{1}$$

where $f_i, i = 1, \ldots, K$ are the objective functions and $K \geqslant 2$ is the number of objectives. The vector $\mathbf{x} = (x_1, \ldots, x_D)$ is a D-dimensional decision vector and $\mathscr{X}$ is the feasible set of decision vectors that satisfy certain equality or inequality constraints. The minimization of function vector $\mathbf{f}(\mathbf{x})$ means that the K objective functions $f_1, \ldots, f_K$ need to be minimized simultaneously.

Having several objective functions as in Eq. (1), the aim of the optimization problem is not to find a single optimal solution, but the *Pareto optimal set* and *Pareto front*, which are defined based on the notion of *(Pareto) dominance*.

Definition 2.1 (**Pareto Dominance**): A vector $\mathbf{y} = (y_1, \ldots, y_k)$ is said to *dominate* another vector $\mathbf{y}' = (y'_1, \ldots, y'_k)$, denoted by $\mathbf{y} \preceq \mathbf{y}'$, if and only if $\mathbf{y}$ is partially less than $\mathbf{y}'$; that is, for all $i \in \{1, \ldots, k\}, y_i \leqslant y'_i$ and there exists at least one $i \in \{1, \ldots, k\}$ such that $y_i < y'_i$.

Definition 2.2 (**Pareto Optimality**): A solution $\mathbf{x}$ is said to be *Pareto optimal* with respect to $\mathscr{X}$ if and only if there is no $\mathbf{x}' \in \mathscr{X}$ such that $\mathbf{y}' = \mathbf{f}(\mathbf{x}')$ dominates $\mathbf{y} = \mathbf{f}(\mathbf{x})$.

In other words, $\mathbf{x}$ is Pareto optimal if there exists no feasible decision vector $\mathbf{x}'$ that would decrease the value of some objective functions without causing a simultaneous increase in the value of at least one other objective function.

Definition 2.3 (**Pareto Optimal Set and Pareto Front**): For a given MOO problem, the *Pareto optimal set* is defined as

$$\mathcal{P}^* := \{\mathbf{x} \in \mathscr{X} \ : \text{ there does not exist a } \mathbf{x}' \in \mathcal{X} \text{ such that } \mathbf{f}(\mathbf{x}') \preceq \mathbf{f}(\mathbf{x})\},$$

and the corresponding *Pareto Front* is defined as

$$\mathcal{PF}^* := \{\mathbf{y} = \mathbf{f}(\mathbf{x}) \ : \ \mathbf{x} \in \mathcal{P}^*\}.$$

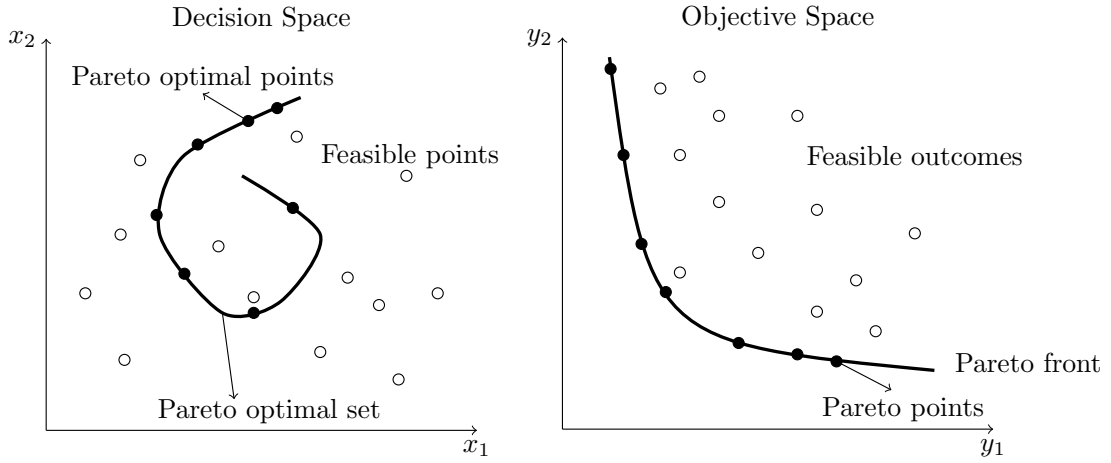

Fig. 1 Pareto optimal set and Pareto Front

In Figure 1, the thick curve in the decision space is the Pareto optimal set, and the corresponding curve in the objective space is the Pareto front. The solid points in the decision space are elements of the Pareto optimal set and their objective vectors lie on the Pareto front in the objective space. Hollow points are some feasible points in the decision space, which are associated with some feasible outcomes in the objective space.

In general, the Pareto front of a MOO problem can be convex or non-convex and continuous or discontinuous. Moreover, similar to single-objective optimization, there can be many *local* (suboptimal) Pareto fronts.

The complexity of large search spaces and the intricacy of Pareto optimality make it often difficult or impossible to find an analytical expression for the Pareto front. However, it is possible

to approximate the true Pareto front by sampling points on or close to the true Pareto front using a randomized algorithm. Such algorithms generally need to yield points that satisfy two requirements: they need to converge to the Pareto front and have a good diversity across the Pareto front. A good MOO solver thus should find a set of points that satisfies these two features.

2.2 Splitting methods

The problem of minimizing a complicated continuous or discrete real-valued function $f(\mathbf{x})$, $\mathbf{x} \in \mathscr{X}$ is closely related to the efficient estimation of rare-event probabilities of the form $\mathbb{P}(f(\mathbf{X}) \leqslant \gamma)$, where $\mathbf{X}$ is a random element of $\mathscr{X}$, distributed according to a given probability distribution, e.g., the uniform pdf on $\mathscr{X}$. This often involves efficient sampling from the γ *level set* $\{\mathbf{x} \in \mathscr{X} : f(\mathbf{x}) \leqslant \gamma\}$. By gradually decreasing γ, the level set becomes smaller and smaller until it only contains elements that lie close to the minimizer of f. For γ close to the minimum γ^*, the event $\{f(\mathbf{X}) \leqslant \gamma^*\}$ will be very rare. To find a minimizer $\mathbf{x}^*$ we could sample a random vector $\mathbf{X}$ conditional on the rare event $\{f(\mathbf{X}) \leqslant \gamma^*\}$. This is the philosophy of using rare-event simulation for optimization.

When γ^* is known, the estimation can be done using the *Generalized Splitting* (GS) method [4] by sampling iteratively from intermediate (increasingly rare) events $\{f(\mathbf{X}) \leqslant \gamma_t\}$, for levels $\infty = \gamma_0 \geqslant \gamma_1 \geqslant \ldots \geqslant \gamma_{T-1} \geqslant \gamma_T = \gamma^*$.

However, in an optimization setting γ^* is not known and therefore the sequence $\{\gamma_t\}$ needs to be determined adaptively, which can be one via the Adaptive Multilevel splitting (ADAM) algorithm, as in [12,2,3]. Having an initial sample set of vectors (often called particles) $\mathcal{X}_0 = \{\mathbf{x} \in \mathscr{X} : f(\mathbf{x}) \leqslant \gamma_0\}$, the ADAM algorithm executes the following two steps at each iteration $t = 0, 1, \ldots$:

(a) Calculate the function value $f(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{X}_t$ and sort these from smallest to largest. Let γ_{t+1} be the $\lceil N_t \varrho \rceil$-th smallest function value, where N_t is the size of $\mathcal{X}_t$ and ϱ is a fixed *rarity* parameter. Define the elite set $\mathcal{E}_{t+1} = \{\mathbf{x} \in \mathcal{X}_t : f(\mathbf{x}) \leqslant \gamma_{t+1}\}$.
(b) Split (diversify, enrich) the elite population in $\mathcal{E}_{t+1}$ via some random sampling mechanism, to create the next population $\mathcal{X}_{t+1}$. Increase t by one and go to Step (a) unless some stopping condition is met.

The splitting step can be implemented in different ways, e.g., by running a Markov chain from each of the elite elements. Often the sampling mechanism is such that each element is split into a fixed number of samples (fixed splitting factor). In other situations it is useful to keep the total sample size (the number of elements of $\mathcal{X}_t$) constant, say N. This is called splitting with a *Fixed Effort*. One way to "evenly" split N^{e} elite samples into N new samples is by defining random splitting factors $s_1, \ldots, s_{N^{\mathrm{e}}} \ldots$ as follows:

$$s_i = \left\lfloor \frac{N}{N^{\mathrm{e}}} \right\rfloor + B_i, \quad i = 1, \ldots, N^{\mathrm{e}}, \tag{2}$$

where $B_1, \ldots, B_{N^{\mathrm{e}}}$ are identically distributed Bernoulli random variables whose sum is $N \bmod N^{\mathrm{e}}$.

Figure 2 illustrates how the adaptive splitting method is performed on a typical single-objective optimization problem in 2-D space. Here, the initial sample set is $\mathcal{X}_0 = \{\mathbf{X}_1, \ldots, \mathbf{X}_5\}$. Suppose we set $\varrho = 0.4$, that is, 2 samples are selected to form the elite set. In the first iteration, $\mathbf{X}_1$ and $\mathbf{X}_2$ are selected and the function value of $\mathbf{X}_1$ becomes the first level parameter, γ_1. From both of the elite points we run a Markov chain, whose length is determined by Eq. (2), so that the two elite points are split into a total of five points. From this second generation of five points the best two points are selected as the new elite points, and the worst function value of these is

takes as the second level parameter, γ_2. If the procedure is replicated over and over again, the points will move toward the optimal level set.

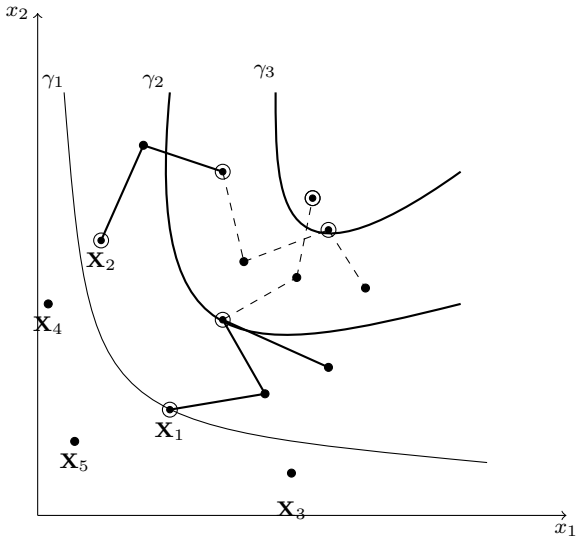

Fig. 2 Illustration of the ADAM algorithm in 2-D space with a fixed effort.

3 Splitting for Multi-objective Optimization

In this section, we will discuss how to adapt the splitting framework to solve MOO problems.

3.1 Splitting with multiple objectives

For MOO problems, there is not a unique criterion to determine the intermediate levels sequence and the corresponding level sets. Also, a good solver needs the simulated points to cover the true Pareto front in a uniform way. This contrasts with single-objective optimization where the simulated points are required to converge to a single solution only.

Nevertheless, the structure of the MOS algorithm is similar to that of the ADAM algorithm. Starting from an initial sample set $\mathcal{X}_0$, the MOS algorithm has the following two phases at each iteration:

(a) Calculate the function values $\mathbf{f}(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{X}_t$ and select the $\lceil N_t \varrho \rceil$ "best" samples as elite set $\mathcal{E}_{t+1}$, where N_t is the size of $\mathcal{X}_t$ and ϱ is a fixed rarity parameter.

(b) Split the elite population in $\mathcal{E}_{t+1}$ via some random sampling mechanism, to create the next population $\mathcal{X}_{t+1}$. Increase t by one and go to Step (a) unless some stopping condition is met.

In single-objective optimization, the elite set is formed by the top-ranking samples, measured by their function value. In a MOO scenario, the fitness of the samples in a population (and hence their relative rankings) is measured differently. A common approach, see, e.g., [9], is to express the fitness of a sample $\mathbf{x}$ in terms of (1) the number of samples that it dominates and (2) its *diversity*, given by

$$\left(\sum_{\mathbf{y}\,:\,\mathrm{dist}(\mathbf{x},\mathbf{y})<\varepsilon} \max\left\{ \frac{\varepsilon - \mathrm{dist}(\mathbf{x},\mathbf{y})}{\varepsilon}, 0 \right\} \right)^{-1},$$

where $\text{dist}(\mathbf{x}, \mathbf{y})$ is a normalized distance and $\varepsilon > 0$ is fixed. A low diversity of $\mathbf{x}$ indicates that all neighboring points lie close to $\mathbf{x}$. The overall ranking could, for example, be determined by scaling the number of dominated samples by their diversity, favoring both high dominance and high diversity.

We introduce a new method for the elite set selection that does not involve this type of scaling. All that is needed is to measure the normalized distance $\text{dist}(\mathbf{x}, \mathbf{y})$ between each pair $(\mathbf{x}, \mathbf{y})$ of samples in the population. How exactly the elite samples are determined in Step (a) will be detailed in Section 3.2.

The splitting step (b) is implemented using a similar randomized sampling scheme as in [8]. In particular, to split the elite set, a random-order Gibbs sampler is used to sample from a multivariate normal distribution for each elite point. The covariance matrix of the sampling distribution is diagonal with entries that depend on the other elite points. This sampling procedure has shown to work well in a single-objective setting. However, for multi-objective optimization we are not interested in finding a single optimal point, but the entire Pareto optimal set. If the the starting point in the splitting step (b) is already close to the Pareto optimal set and also has a reasonable diversity, the diagonal entries of the covariance matrix may be much larger than the distance to the Pareto optimal set. To improve the performance of the splitting algorithm for the multi-objective case, we will combine it with a local search technique. The details of the exact sampling strategy are given in Section 3.3.

To further illustrate the workings of the MOS algorithm, consider Figure 3.1, which depicts a problem with two variables and two objective functions. For simplicity of illustration, the fitness of a point is only determined by its current dominance-based rank. Non-dominated points in each generation are selected as elite points. The splitting step is simply implemented by running a random walk sampler on a continuous state space. However, a new state is only accepted when it is not dominated by the initial state.

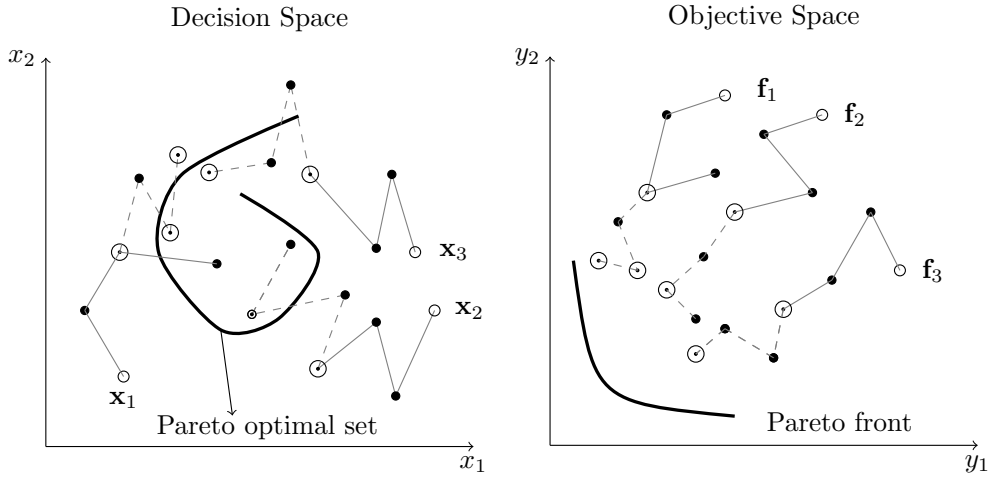

Fig. 3 Splitting in the decision space and objective space. Initial points are marked with the symbol ∘. The symbol • and ⊙ are splitting points. Solid lines indicate the first generation of splitting. Dashed lines indicate the second generation. The ⊙ points are the non-dominated points found in each generation.

To be specific, $\mathbf{x}_1, \mathbf{x}_2$ and $\mathbf{x}_3$ are the initial points in the decision space, and $\mathbf{f}_1, \mathbf{f}_2$ and $\mathbf{f}_3$ are the corresponding objective vectors in the objective space. From each initial point we run a Markov chain of length 3, where a new state is accepted with probability 1 if it is not dominated by its initial state. From the total of 9 splitting points we choose the non-dominated ones (3

$\odot$ points) as the elite points. These elite points are the starting points of the next iteration, where again we run Markov chains of length 3 in the same way. We obtain 9 splitting points and 4 of them are non-dominated. In the second iteration, the size of the elite set becomes 4. The corresponding evolution of the objective vectors can be observed in the objective space. As mentioned before, it is possible to keep a fixed sample size by using splitting with a fixed effort.

In addition to the selection of the elites and the specification of the splitting (sampling) step, it is also of importance to protect the diversity of simulated solutions. In the MOS algorithm, we use an external archive of fixed size to store good solutions that are found during the execution of the algorithm. This archive is maintained with a so-called *ε-dominance* method and updated whenever a good intermediate point is found. The details of the *ε-dominance* method are given in Section 3.3.

Having discussed the ingredients of the MOS algorithm, it is time to put forward its details, which are given in Algorithm 1. Basically, after initialization Algorithm 1 has two iterative steps:

1. Evaluate the values of the objective functions for each sample, and choose the elite set via Algorithm 2.
2. Split the elite set to create the next population using sampling methods given in Algorithm 3 and Algorithm 4, both of which update the archive using Algorithm 5.

Algorithm 1: Multi-objective Splitting (MOS)

Input: Sample size N, rarity parameter ϱ, initial external archive A, local search range vectors $\mathcal{V}_0 = [\mathbf{v}_1; \ldots; \mathbf{v}_N]$ and global search factor w, global search probability P_g, acceptance probability P_{ed}, upper bound $\mathbf{u}$ and lower bound $\mathbf{l}$ on the objective functions

Output: Final external archive A.

Generate $\mathcal{X}_0 = [\mathbf{x}_1; \ldots; \mathbf{x}_N]$ from the feasible domain $\mathscr{X}$. Set $t = 0$ and $N^{\mathrm{e}} = \lceil N\varrho \rceil$.
Evaluate the objective vectors $\mathcal{F}_t = [\mathbf{f}_1; \ldots; \mathbf{f}_N]$, where $\mathbf{f}_i = \mathbf{f}(\mathbf{x}_i)$ for each $i = 1, \ldots, N$.
while *the stopping criterion is not met* **do**
    Construct the elite set as in Algorithm 2 and output the elite set $\mathcal{E}_{t+1}$ and the corresponding objective vectors $\mathcal{F}^{\mathrm{e}}_{t+1}$ and local search range vectors $\mathcal{V}^{\mathrm{e}}_{t+1}$. Let $\mathbf{f}^{(i)}$ be the objective vector corresponding to $\mathbf{x}^{(i)}$
    Compute the splitting factors $s_{t+1,i}$ for each $\mathbf{x}^{(i)} \in \mathcal{E}_{t+1}$ as in Eq.(2), where $i = 1, \ldots, N^{\mathrm{e}}$.
    for $i = 1 : N^{\mathrm{e}}$ **do**
        Let $\mathbf{x} = \mathbf{x}^{(i)}$ and $\mathbf{f} = \mathbf{f}^{(i)}$
        Select R uniformly from the set $\{1, \ldots, N^{\mathrm{e}}\} \setminus \{i\}$ and compute $\boldsymbol{\sigma}^{(i)}$ via Eq.(4).
        for $j = 1 : s_{t+1,i}$ **do**
            Let the local search range vector be $\mathbf{v} = \mathbf{v}^{(i)} \in \mathcal{V}^{\mathrm{e}}_{t+1}$ and the indicator vector of improvement be $[I_1, \ldots, I_D] = [0, \ldots, 0]$.
            for try $= 1 :$ maxTry **do**
                Generate a random order $\mathbf{r} = [r_1, \ldots, r_D]$
                for $d = 1 : D$ **do**
                    if $I_{r_d} \neq 0$ **then** break the loop and update r_{d+1}
                    Generate a random number $U \sim \mathsf{U}(0, 1)$
                    if $U < P_g$ **then**
                        $[\mathbf{x}, \mathbf{f}, A, I_{r_d}] = \texttt{GlobalSearch}(\mathbf{x}, \mathbf{f}, \boldsymbol{\sigma}^{(i)}, r_d, P_{\mathrm{ed}}, A)$ (Algorithm 3)
                    else
                        $[\mathbf{x}, \mathbf{f}, A, I_{r_d}, \mathbf{v}] = \texttt{LocalSearch}(\mathbf{x}, \mathbf{f}, \mathbf{v}, r_d, P_{\mathrm{ed}}, A)$ (Algorithm 4)
            if $j \geqslant 2$ **then**
                Compute the distance $\mathrm{dist}\left(\mathbf{f}, \mathbf{f}^{(i)}\right)$ as in Eq.(3)
                if $\mathrm{dist}\left(\mathbf{f}, \mathbf{f}^{(i)}\right) \leqslant \gamma_t$ **then**
                    Set $\mathbf{x} = \mathbf{l} + (\mathbf{u} - \mathbf{l})\mathrm{diag}(\mathbf{U})$, where $\mathbf{U} \sim \mathsf{U}(0, 1)^D$. Let $\mathbf{f}$ be the corresponding objective vector.
            Add $\mathbf{x}$ to $\mathcal{X}_{t+1}$, $\mathbf{f}$ to $\mathcal{F}_{t+1}$, and $\mathbf{v}$ to $\mathcal{V}_{t+1}$.
    Set $t = t + 1$.

Algorithm 1 takes the following input. The sample size N and rarity parameter ϱ are control parameters of the splitting step. The size of the elite set N^{e} is determined by $N^{\mathrm{e}} = \lceil N\varrho \rceil$. At each iteration the sample set $\mathcal{X}_t$ is stored as a $N \times D$ matrix $[\mathbf{x}_1; \ldots; \mathbf{x}_N]$. The input vector $\mathbf{v}_i = [v_{i,1}, \ldots, v_{i,D}]$ is the initial local sampling range vector of $\mathbf{x}_i$ for $i = 1, \ldots, N$, where $v_{i,j}$ is the local sampling range of the j-th component of $\mathbf{x}_i$, $j = 1, \ldots, D$. A global search factor w is controlling the values of the standard deviation vectors of the global sampling distributions. The vectors $\mathbf{u} = [u_1, \ldots, u_D]$ and $\mathbf{l} = [l_1, \ldots, l_D]$ contain, for each dimension, the upper and lower bounds that any of the objective functions can take. Note that all vectors are *row* vectors. The external archive A is used to store good solutions, including both the decision vectors and the corresponding objective vectors, for example, $A = [\widehat{\mathbf{x}}_1, \widehat{\mathbf{y}}_1; \widehat{\mathbf{x}}_1, \widehat{\mathbf{y}}_1; \ldots]$. It is initialized by an empty collection.

In iteration $t = 0$, the initial sample set $\mathcal{X}_0$ is randomly and uniformly generated from the feasible domain $\mathscr{X}$. After evaluation of the objective functions for each sample, the decision variables $\mathbf{x}_i$ and objective values $\mathbf{f}_i$ are stored in the external archive A, for $i = 1, \ldots, N$. The elite set is selected according to Algorithm 2, where bad points are discarded based on distance and dominance rank, and the remaining ones are selected. In this implementation, we wish to use the framework with fixed effort, so the quota of N new samples is evenly allocated amongst the N^{e} elite samples.

For running the Markov chains, we use a Gibbs-like sampler, where each component of an elite sample is updated in a random order. For each component, there are two strategies to search better for updates: global sampling in Algorithm 3 and local sampling in Algorithm 4. The global sampling will be used with probability P_g and the local sampling with $1-P_g$, where P_g can simply be set to 0.5. For the global procedure, samples are drawn from a normal distribution specified by a standard deviation vector that depends on a randomly selected elite point, while for the local sampling procedure, the sampling range will be fixed, controlled by a local sampling range vector. The details will be elaborated in Section 3.3. Once good updates have been obtained, the intermediate (candidate) points will be checked to see if they could be put into the external archive. This is implemented in Algorithm 5. Finally, if in the Markov chain sampling the second splitting point is very close to the starting point, then the second point will be replaced with a random point so as to protect the diversity of the sample.

3.2 Construction of the elite set

The selection of the elite points is a very important ingredient in the MOS method. Its single-objective counterpart, the SCO algorithm, simply discards the samples with the worst objective function values and keeps the rest as the elite samples. However, for multi-objective optimization it is difficult to define the fitness of each individual in a population of points that are not on the Pareto front. In many multi-objective evolutionary algorithms (MOEA), scaling techniques are used to sort the individuals in the objective space before selection, based on the number of samples that are dominated by a solution, or the number of samples that the solution dominates. The computational order of these method is usually $O(N^2)$, where N is number of individuals. As for diversity preservation, there are various techniques available for MOEAs, including fitness sharing/niching, crowding/clustering, and relaxed dominance. The dominance ranking and diversity maintenance techniques usually work together to provide a diversity of points that are approximately uniformly distributed over the real Pareto front.

In the MOS algorithm we did not implement a dominance-based ranking criterion to select the elite samples, as we found that some points with a bad ranking at the beginning could still

split into points that reached the Pareto optimal set, while initially good ranking points could prove to be of little value later on. Instead, we introduce a new method of elite set selection.

Specifically, suppose we are given N samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ and their corresponding objective function vectors $\mathbf{f}_1, \ldots, \mathbf{f}_N$, where $\mathbf{f}_i = [f_{i,1}, \ldots, f_{i,K}]$, for $i = 1, \ldots, N$. We then evaluate the normalized Euclidean distance between each pair of solutions in the objective space as follows:

$$M(i,j) := \mathrm{dist}(\mathbf{f}_i, \mathbf{f}_j) := \sqrt{\sum_{k=1}^{K} \left(\frac{f_{i,k} - f_{j,k}}{f_{\max,k} - f_{\min,k}} \right)^2}, \tag{3}$$

where $f_{\max,k} = \max_{i=1,\ldots,N} f_{i,k}$ and $f_{\min,k} = \min_{i=1,\ldots,N} f_{i,k}$, $i = 1, \ldots, N-1, j = i+1, \ldots, N$ and $k = 1, \ldots, K$. All other entries are set to 0, so that matrix M is upper-triangular.

From matrix M we first find the pair of solutions $\mathbf{x}_i$ and $\mathbf{x}_j$ whose corresponding objective vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ have a distance smaller than any of the other pairs. We then compare the dominance relation between them. If one is dominated by another, then we discard the dominated one. If they do not dominate each other, then we randomly discard one. Next, we take the pair with the second shortest distance and repeat the procedure, and keep doing so until certain proportion of solutions are discarded. Finally, we set threshold γ_t as the distance between the last compared pair. Note that γ_t is therefore a completely different level threshold than in the single-objective SCO algorithm. It is primarily used to measure the crowdedness of consecutive points in the splitting step. The details are described in Algorithm 2.

Algorithm 2: Construction of the elite set $\mathcal{E}_{t+1}$

Input: The tth population $\mathcal{X}_t$ and the corresponding set of objective vectors $\mathcal{F}_t$
Output: The elite set $\mathcal{E}_{t+1}$, the corresponding set of objective vectors $\mathcal{F}^{\mathrm{e}}_{t+1}$, the local sampling ranges $\mathcal{V}^{\mathrm{e}}_{t+1}$, and the distance threshold γ_{t+1}

Compute the distance matrix M as in Eq.(3)
Set the initial row index as $\mathbf{I}_r = \{1, \ldots, N-1\}$, and the column index as $\mathbf{I}_c = \{i+1, \ldots, N\}$ for each $i \in \mathbf{I}_r$
for $k = 1 : \lfloor N(1-\varrho) \rfloor$ **do**
    Find the subscript $(i,j) = \mathrm{argmin}_{(i,j) \in \mathbf{I}_r \times \mathbf{I}_c} M(i,j)$ and set $\gamma_{t+1} = M(i,j)$
    Compare the dominance between $\mathbf{x}_i$ and $\mathbf{x}_j$
    if $\mathbf{f}_i \prec \mathbf{f}_j$ *or* $\mathbf{f}_j \prec \mathbf{f}_j$ **then**
        Discard the dominated one
    else
        Randomly choose one
    Let r denoted the index of the discarded point
    $\mathbf{I}_r = \mathbf{I}_r \setminus \{r\}$ and $\mathbf{I}_c = \mathbf{I}_c \setminus \{r\}$
Let $\mathbf{I}_e = \mathbf{I}_r \cup \mathbf{I}_c$ be the index of elements in $\mathcal{X}_t$ to be selected in elite set $\mathcal{E}_{t+1}$. Identify $\mathcal{F}^{\mathrm{e}}_{t+1}$ and $\mathcal{V}^{\mathrm{e}}_{t+1}$.

By using Algorithm 2 to form the elite sets, we maintain a high diversity of objective values in the objective space.

3.3 Sampling strategies

This section describes how an elite point $\mathbf{x}^{(i)}$ is randomly "split" into multiple points. Following [8], the idea is to split the initial point by updating only one of its components each time, using a multivariate normal distribution with mean vector $\boldsymbol{\mu}^{(i)} = \mathbf{x}^{(i)}$ and diagonal covariance matrix

$\text{diag}((\boldsymbol{\sigma}^{(i)})^2)$, where the vector of standard deviations $\boldsymbol{\sigma}^{(i)}$ is defined as follows:

$$\boldsymbol{\sigma}^{(i)} = w\,|\mathbf{x}^{(i)} - \mathbf{x}^{(R)}| := w \begin{pmatrix} |x_1^{(i)} - x_1^{(R)}| \\ |x_2^{(i)} - x_2^{(R)}| \\ \cdots \\ |x_n^{(i)} - x_n^{(R)}| \end{pmatrix}, \quad i = 1, \ldots, N^{\mathrm{e}}, \tag{4}$$

where w is a *scale factor*, and $\mathbf{x}^{(R)}$ is a uniformly selected elite point other than $\mathbf{x}^{(i)}$. As shown in numerical experiments, the SCO algorithm in [8] for single-objective optimization converges to a global optimum very efficiently and accurately, as a result of good balance between exploration and exploitation. Note that the sampling distribution for each elite point is randomly determined by the other elite points, such that it can guarantee that the mixture sampling distributions of all elite points cover the whole level set. If the corresponding standard deviation is relatively small, the algorithm samples locally, whereas a large standard deviation increases the chance of sampling globally. As the level set is shrinking, the standard deviations will decrease as well. This makes it easy to find the optimum in a region that contains the global optimum.

For multi-objective optimization, the above sampling strategy needs to be modified, because the algorithm usually keeps the solutions with larger distances when constructing the elite set in Algorithm 2. As a result, the standard deviations of sampling distributions are always relatively large and a local search cannot be achieved adequately.

Therefore, in the MOS algorithm, we combine the above "global" Gibbs sampling strategy of [8] with the "local" LS1 strategy in [20], where sampling is performed via a uniform distribution. Algorithm 3 describes how the dth component of a current decision vector $\mathbf{x} = [x_1, \ldots, x_D]$ is updated for the global strategy. The trial (proposal) vector $\widetilde{\mathbf{x}}$ is initialized as $\widetilde{\mathbf{x}} = \mathbf{x}$. Then, the dth component of $\widetilde{\mathbf{x}}$ is updated by setting

$$\widetilde{x}_d = x_d + \sigma_d^{(i)} Z, \quad Z \sim \mathsf{N}(0,1), \tag{5}$$

where $\sigma_d^{(i)}$ is the dth component of $\boldsymbol{\sigma}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ is computed according to Eq.(4). With some abuse of notation, let $\mathbf{f} = \mathbf{f}(\mathbf{x})$ and $\widetilde{\mathbf{f}} = \mathbf{f}(\widetilde{\mathbf{x}})$. If $\widetilde{\mathbf{f}} \prec \mathbf{f}$, then $\widetilde{\mathbf{x}}$ is accepted as an intermediate point. If $\widetilde{\mathbf{f}}$ and $\mathbf{f}$ do not dominate each other, then the $\widetilde{\mathbf{x}}$ has a probability P_{ed} ("ed" stands for "equal dominance) to be accepted as the next point.

Algorithm 3: The dth component is sampled from a Gaussian distribution

```
function GlobalSearch(x, f, σ, d, P_ed, A)
    Set x̃ = x, f̃ = f, and I_d = 0
    Update the dth component of x̃: set x̃_d = x_d + σ_d Z, where Z ~ N(0,1)
    if x̃_d < l_d or x̃_d > u_d then
        x̃_d = l_d + U (u_d − l_d), where U ~ U(0,1)
    Evaluate the new objective vector f̃ and check the dominance between x and x̃
    if f̃ ≺ f then
        Add [x̃, f̃] to A, and update A based on Algorithm 5
        Set x = x̃, f = f̃, and I_d = 1
    else if f and f̃ do not dominate each other then
        Add [x̃, f̃] to A, and update A based on Algorithm 5
        Draw U ~ U(0,1)
        if U < P_ed then set x = x̃, f = f̃ and I_d = 1
    return x, f, A, I_d
```

Algorithm 4 is used to update the dth component of a decision vector $\mathbf{x}$ in a different, more localized, manner. Instead of using a normal distribution as in Algorithm 3, here we use a

uniform distribution to sample updates. Vector $\mathbf{v} = [v_1, \ldots, v_D]$ is to control the parameters of the uniform distributions (one for each component), where each v_d can be negative or positive. If $v_d > 0$, the sampling distribution is $\mathsf{U}(x_d, x_d + v_d)$; otherwise, it is $\mathsf{U}(x_d + v_d, x_d)$. The candidate solutions are accepted in the same way as in Algorithm 3; that is, better solutions in terms of dominance are always accepted, and equally good solutions are accepted with probability P_{ed}. If the candidate solution is dominated by the current one, the sampling region will shrink a little (by a factor of 0.8 in the algorithm, unless $|v_d|$ is very small, when it will be reinitialized with $v_d = 0.4 \times (u_d - l_d)$) and go to its negative side.

Algorithm 4: The dth component is sampled from a Uniform distribution

```
function LocalSearch(x, f, v, d, P_ed, A)
    Set x̃ = x, f̃ = f, and I_d = 0
    Update the dth component of x̃: set x̃_d = x_d + v_d U, U ~ U(0,1)
    if x̃_d < l_d or x̃_d > u_d then
        x̃_d = l_d + (u_d − l_d)U, U ~ U(0,1)
    Evaluate the new objective vector f̃ and check the dominance between x and x̃
    if f̃ ≺ f then
        Add [x̃, f̃] into A, and update A based on Algorithm 5
        Set x = x̃, f = f̃ and I_d = 1
    else if f(x̃) and f(x) do not dominate each other then
        Add [x̃, f̃] into A, and update A based on Algorithm 5
        Draw U ~ U(0,1)
        if U < P_ed then set x = x̃_d, f = f̃ and I_d = 1
    else
        Set v_d = −0.8 × v_d
        if |v_d| < 10^−8 then set v_d = 0.4 × (u_d − l_d)
    return x, f, A, I_d, v
```

3.4 Archive update

The MOS algorithm is using a fixed-size external archive to store the good solutions, and we use the ε-*dominance* method to maintain the archive. The method has been introduced in [17]. Generally, in the ε-*dominance* method, each dimension of the objective space is sliced into intervals of size ε. When a new candidate comes into the external archive, the first thing to be done is to identify to which box (K-dimensional hypercube) it belongs. If the box is empty and not dominated by other non-empty boxes, then this candidate is put into the external archive. Dominance between boxes is similar to dominance between vectors. For example, in two dimensions, the box $(2\,\varepsilon, 2\,\varepsilon)$ dominates the boxes $(2\,\varepsilon, 3\,\varepsilon)$ and $(3\,\varepsilon, 2\,\varepsilon)$. If the box of the new candidate is dominated by some non-empty boxes, then reject it. If the box is not dominated by others and is non-empty, and if the candidate is closer to the left corner, then accept it to replace the worst candidate of that box; otherwise, reject it. The details are as in Algorithm 5. For more information about the ε-dominance method, we refer to [17,11].

Algorithm 5: Archive update using the ε-dominance method

Input: An ε-grid on the objective space, external archive A, a new solution $\mathbf{x}'$ and its objective vector $\mathbf{f}'$
Output: A

Identify the box b' that contains $\mathbf{f}'$
if *The box b' is empty* **then**
    Check the dominance between box b' and other non-empty boxes
    if *box b' is dominated by other non-empty boxes* **then**
        Reject b' and eliminate box b' from the grid
    else
        Accept $\mathbf{x}'$ and $\mathbf{f}'$, and eliminate the dominated non-empty boxes and the members insides
else
    Check the dominance between $\mathbf{x}'$ and other solutions in box b'
    if *one solution is dominated by another in the box* **then**
        Reject the dominated one
    else
        Keep the closest one to the left corner and remove other one

4 Numerical Experiments

4.1 Benchmarks and metrics

The CEC'09 benchmark has introduced a set of complex test problems with and without constraints for MOO competition. They are frequently used to assess the performance of MOO algorithms. In this paper, we use the ten unconstrained problems from the CEC09 test suite to evaluate the performance of the MOS algorithm. Among the ten problems, UF1-UF7 have two objectives and UF8-UF10 have three objectives. All the problems have 30 decision variables. The complete set of benchmark problems can be found in [24].

To assess the performance, we use the *Inverted Generational Distance* (IGD) value to measure the average Euclidean distance between the true Pareto front and the results obtained by MOS algorithm. Let $\widehat{\mathcal{PF}}^*$ be a set of uniformly distributed points along the true Pareto front $\mathcal{PF}^*$ in the objective space (which is known for this competition), and let $\mathcal{PF}'$ be the set of approximate points produced by MOO solvers. The IGD is defined as

$$\mathrm{IGD}(\mathcal{PF}', \widehat{\mathcal{PF}}^*) = \frac{\sum_{\mathbf{p}\in\widehat{\mathcal{PF}}^*} d\left(\mathbf{p}, \mathcal{PF}'\right)}{|\widehat{\mathcal{PF}}^*|}, \tag{6}$$

where $d\left(\mathbf{p}, \mathcal{PF}'\right)$ is the Euclidean distance between the vector $\mathbf{p}$ and its nearest point in the set $\mathcal{PF}'$.

The IGD metric has been used in the CEC'09 competition, so using IGD to measure performance of MOS algorithm allows us to compare with other methods easily.

4.2 Experiments and comparison

The numerical experiments will be run for two settings. The first setting is the same as in the CEC'09 competition, where the maximum number of function evaluations is 300,000. The performance is compared with methods in [1,23,13,20,16,15]. The second setting is the same as that used in [18]. That is, the maximum number of function evaluations for two and three-objective problems are 50,000 and 150,000 respectively. Moreover, the MOS algorithm is compared with state-of-the-art modifications of some famous methods, e.g., [7,22,14].

The experiments have been coded in MATLAB R2014b, and were conducted on a desktop personal computer with Intel(R) Core(TM) i7-4970 CPU @3.60GHz. Each experiment was independently repeated 30 times, and the average IGD value was reported. The control parameters of the MOS algorithm were set as follows: sample size $N = 100$, rarity parameter $\varrho = 0.9$, global search factor $w = 1$, and local search range vectors $\mathcal{V} = [\mathbf{v}_1; \ldots; \mathbf{v}_N]$, where $\mathbf{v}_i = [v_{i,1}, \ldots, v_{i,D}]$ and $v_{i,j} = 0.4 \times (u_j - l_j)$ for $i = 1, \ldots, N$ and $j = 1, \ldots, D$. The u_j and l_j are the upper and lower bounds of the j-th component.

The results for the compared methods were obtained from two key references [18,1], which are summarized in Tables 1 and 2 respectively. The best performances are shaded with gray and the top three performances are written in **bold**.

4.2.1 Experimental setting 1

In this setting, all methods have a maximum of 300,00 function evaluations. In Table 1, the IGD results of the MOS algorithm are shown in the first column, averaged over 30 independent runs. The other results in Table 1 were taken from [1]. The maximum number of non-dominated points used to calculate IGD value is 100 for seven two-objective problems and 150 for three three-objective problems. In the CEC'09 competition, the best 5 algorithms on a final ranking of the unconstrained problems are MOEA/D [23], MTS [20], DMOEADD [16], Liuli [15] and GDE3 [13], all of which are included in our comparison. We also include MOABC [1] since it performs well on the test functions and can beat other algorithms on some problems.

Table 1: Mean of the IGD values over 30 runs

Function	MOS	MOEA/D	GDE3	MTS	DMOEADD	Liuli Alg	MOABC
UF1	**0.00574**	**0.00435**	**0.00534**	0.00646	0.01038	0.00785	0.00618
UF2	**0.00605**	0.00679	0.01195	**0.00615**	0.00679	0.01230	**0.00484**
UF3	0.05096	**0.00742**	0.10639	0.05310	**0.03337**	**0.01497**	0.05120
UF4	**0.04280**	0.06385	**0.02650**	**0.02356**	0.04268	0.04350	0.05801
UF5	**0.06904**	0.18071	0.03928	**0.01489**	0.31454	0.16186	**0.07775**
UF6	**0.03602**	**0.00587**	0.25091	**0.05917**	0.06673	0.17555	0.06537
UF7	**0.00889**	**0.00444**	0.02522	0.04079	0.01032	**0.00730**	0.05573
UF8	**0.05491**	**0.05840**	0.24855	0.11251	0.06841	0.08235	**0.06726**
UF9	**0.03371**	0.07896	0.08248	0.11442	**0.04896**	0.09391	**0.06150**
UF10	**0.12310**	0.47415	0.43326	**0.15306**	0.32211	0.44691	**0.19499**

As shown in Table 1, MOEA/D produces the best results for four problems, i.e., UF1, UF3, UF6 and UF7, while MOS outperform the other methods on three problems, i.e., UF8, UF9 and UF10. The MTS algorithm performs best on UF4 and UF5, and MOABC gives the best results for UF2. Note that the MOS algorithm was ranked in the top three for nine of ten problems. By comparison, the MOEA/D, MOABC and MTS algorithms performed as one of the top three algorithms for five of the ten problems. The GDE3, DMOEADD and Liuli algorithms performed less well. Overall, in terms of IGD value, the MOS algorithm was competitive with, and sometimes improved on, the popular MOEA/D and MTS algorithms.

It is interesting to see that for all the *three-objectives* problems (UF8, UF9 and UF10) MOS obtained the best results. A possible reason could be that MOS uses a Gibbs-like sampler in the search scheme, which is particularly suitable for these three problems. And the good balance between exploration and exploitation in the sampling of the MOS algorithm makes it more successful than, for example, MOABC, which also updates the samples in a component-by-component manner.

4.2.2 Experimental setting 2

In this setting, the maximum number of function evaluations is 50,000 for the two-objective problems and 150,000 for the three-objective functions. Since [18] did not specify the maximum size of non-dominated set that was used to calculate IGD values, we keep here the same number of non-dominated points as in previous experiments. The results are summarized in Table 2. The means and standard deviations of the IGD values of the MOS algorithm are based on 30 independent replications, while all other data are extracted from [18].

Table 2: Mean and standard deviation of the IGD values over 30 runs

Function	MOS mean(std)	NSGA-II ACGDE mean(std)	MOEA/D ACGDE mean(std)	NSGA-II DE mean(std)	NSGA-II(S) SBX mean(std)	MOEA/D DE mean(std)	MOEA/D SBX mean(std)
UF1	**0.0226** (0.0026)	0.0528 (0.0148)	**0.0448** (0.0155)	0.0603 (0.0318)	0.1230 (0.0318)	**0.0475** (0.0372)	0.1568 (0.0652)
UF2	**0.0139** (0.0015)	**0.0205** (0.0027)	**0.0195** (0.0039)	0.0429 (0.0047)	0.0481 (0.0125)	0.0426 (0.0316)	0.0640 (0.0310)
UF3	0.1521 (0.0213)	**0.0947** (0.0139)	**0.1306** (0.0398)	0.1515 (0.0271)	0.2179 (0.0666)	**0.1513** (0.0688)	0.3064 (0.0300)
UF4	0.0580 (0.0026)	**0.0410** (0.0003)	**0.0436** (0.0014)	0.0723 (0.0078)	**0.0533** (0.0018)	0.0866 (0.0104)	0.0560 (0.0034)
UF5	**0.4053** (0.0543)	**0.2870** (0.0932)	0.4654 (0.0974)	0.8494 (0.1698)	**0.3257** (0.0943)	0.7643 (0.1307)	0.4318 (0.0812)
UF6	**0.2604** (0.0351)	**0.1576** (0.0849)	0.2893 (0.1694)	0.4181 (0.0819)	**0.2302** (0.0680)	0.4386 (0.2206)	0.4374 (0.1509)
UF7	**0.0507** (0.0245)	**0.0262** (0.0071)	0.0521 (0.0882)	**0.0437** (0.0422)	0.2359 (0.1447)	0.1018 (0.1648)	0.3536 (0.1552)
UF8	**0.0691** (0.0063)	0.1383 (0.0420)	**0.1045** (0.0112)	0.1520 (0.0300)	0.2194 (0.0098)	**0.0911** (0.0124)	0.148 (0.0358)
UF9	**0.0424** (0.0056)	0.1776 (0.1091)	**0.076** (0.0401)	0.1938 (0.0646)	0.1635 (0.0491)	**0.1065** (0.0452)	0.134 (0.0624)
UF10	**0.1900** (0.0188)	0.6440 (0.2715)	**0.2481** (0.066)	2.4308 (0.1848)	0.3236 (0.0703)	0.5826 (0.0716)	**0.2937** (0.1304)

As indicated in Table 2, both MOS and NSGA-II with ACGDE performed best in five out of ten problems. In eight cases MOS was ranked in the top three. In comparison, NSGA-II was in the top three in six cases and MOEA/D with ACGDE in seven cases. The other algorithms performed less well.

In this experimental setting, despite the fact that the performance of MOEA/D improved by combining it with the ACGDE operator, the MOS algorithms surprisingly outperformed other methods on problems UF1 and UF2. The conclusion is that MOS reaches a good solution sooner (with fewer iterations). However, after analyzing the evolution of the level sets in the MOS algorithm, we found that after certain iterations the improvement of the level sets became very minor. That is, few of new non-dominated points were found by increasing the function evaluation number. Therefore, after a certain number of function evaluations, MOEA/D will exceed MOS (if given enough function evaluations). On the problems UF8, UF9 and UF10, the MOS algorithm was still the best one among these methods.

5 Conclusion

In this paper, we have put forward a novel multi-objective splitting (MOS) method. The performance of MOS depends on several control parameters, which are very easy to tune. The local

search range vector could always be initialized with 40% of the feasible region for each dimension. The global search factor (w) can be set to 1 for multi-objective optimization problems. The sample size and rarity parameter should be adjusted correspondingly; for example, if the rarity parameter is small, then sample size should be relatively large. The grid size ε for the external archive is also of importance for controlling the diversity of the non-dominated solutions found. Through a set of popular benchmark problems, MOS has been compared with some famous and successful methods, and was shown to produce competitive and sometimes superior results.

Beside the numerical performance, more properties of MOS need to be studied in future work. For example, given a sufficient number of function evaluations, what is the convergence behaviour of the MOS algorithm? Also, the time and space complexities are interesting to analyze. A future study aims to investigate more closely the probabilistic reasons why MOS performs so well under certain conditions.

Acknowledgement

This work was supported by the Australian Research Council Centre of Excellence for Mathematical and Statistical Frontiers, under grant number CE140100049. Qibin Duan would also like to acknowledge the support from the University of Queensland through the UQ International Scholarships scheme.

References

1. R. Akbari, R. Hedayatzadeh, K. Ziarati, and B. Hassanizadeh. A multi-objective artificial bee colony algorithm. *Swarm and Evolutionary Computation*, 2:39–52, 2012.
2. Z. I. Botev. *The Generalized Splitting Method for Combinatorial Counting and Static Rare-Event Probability Estimation*. PhD thesis, The Unverisity of Queensland, 2009.
3. Z. I. Botev and D. P. Kroese. An efficient algorithm for rare-event probability estimation, combinatorial optimization, and counting. *Methodology and Computing in Applied Probability*, 10(4):471–505, 2008.
4. Z. I. Botev and D. P. Kroese. Efficient monte carlo simulation via the generalized splitting method. *Statistics and Computing*, 1(1–16), 2012.
5. C. A. C. Coello and M. S. Lechuga. MOPSO: A proposal for multiple objective particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 1051–1056, 2002.
6. C. A. C. Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary algorithms for solving multi-objective problems*, volume 242. Springer, 2002.
7. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.
8. Q. Duan and D. P. Kroese. Splitting for optimization. *Computers & Operations Research*, 73:119–131, 2016.
9. C. M. Fonseca and P. J. Fleming. Multiobjective genetic algorithms. In *Genetic algorithms for control systems engineering, IEE colloquium on*, pages 6–1. IET, 1993.
10. J. D Knowles and D. W Corne. The Pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, 1999.
11. J. D Knowles and D. W. Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary computation*, 8(2):149–172, 2000.
12. D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo Methods*. John Wiley & Sons, New York, 2011.
13. S. Kukkonen and J. Lampinen. GDE3: The third evolution step of generalized differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 443–450, 2005.
14. H. Li and Q. Zhang. Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. *IEEE Transactions on Evolutionary Computation*, 13(2):284–302, 2009.
15. H. Liu and X. Li. The multiobjective evolutionary algorithm based on determined weight and sub-regional search. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1928–1934, 2009.
16. M. Liu, X. Zou, Y. Chen, and Z. Wu. Performance assessment of DMOEA-DD with CEC 2009 MOEA competition test instances. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 2913–2918, 2009.

17. S. Mishra, K. Deb, and M. Mohan. Evaluating the -domination based multi-objective evolutionary algorithm for a quick computation of Pareto-optimal solutions. *Evolutionary Computation*, 13(4):501–526, 2005.
18. X. Qiu, J. Xu, K. C. Tan, and H. A. Abbass. Adaptive cross-generation differential evolution operators for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 20(2):232–244, 2016.
19. R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method*. John Wiley & Sons, third edition, 2017.
20. L. Tseng and C. Chen. Multiple trajectory search for unconstrained/constrained multi-objective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1951–1958, 2009.
21. A. Ünveren and A. Acan. Multi-objective optimization with cross entropy method: Stochastic learning with clustered Pareto fronts. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3065–3071, 2007.
22. Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.
23. Q. Zhang, W. Liu, and H. Li. The performance of a new version of MOEA/D on cec09 unconstrained MOP test instances. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 203–208, 2009.
24. Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari. Multiobjective optimization test instances for the CEC 2009 special session and competition. *University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report*, 264, 2008.
25. A. Zhou, B. Qu, H. Li, S. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.