Derivative-Free Optimization: Lifting Single-Objective to Multi-Objective Algorithm

Cyrille Dejemeppe(✉), Pierre Schaus, and Yves Deville

ICTEAM, Université Catholique de Louvain (UCLouvain),
Louvain-la-Neuve, Belgium
{cyrille.dejemeppe,pierre.schaus,yves.deville}@uclouvain.be

Abstract. Most of the derivative-free optimization (DFO) algorithms rely on a comparison function able to compare any pair of points with respect to a black-box objective function. Recently, new dedicated derivative-free optimization algorithms have emerged to tackle multi-objective optimization problems and provide a Pareto front approximation to the user. This work aims at reusing single objective DFO algorithms (such as Nelder-Mead) in the context of multi-objective optimization. Therefore we introduce a comparison function able to compare a pair of points in the context of a set of non-dominated points. We describe an algorithm, MOGEN, which initializes a Pareto front approximation composed of a population of instances of single-objective DFO algorithms. These algorithms use the same introduced comparison function relying on a shared Pareto front approximation. The different instances of single-objective DFO algorithms are collaborating and competing to improve the Pareto front approximation. Our experiments comparing MOGEN with the state-of the-art Direct Multi-Search algorithm on a large set of benchmarks shows the practicality of the approach, allowing to obtain high quality Pareto fronts using a reasonably small amount of function evaluations.

1 Introduction

Continuous optimization aims at minimizing a function $f(x)$ with $x \in \mathbb{R}^n$. When some information is known about the derivatives of f, one generally uses gradient based methods. For some other problems the function is black-box which means it can only be evaluated (for instance the evaluation is the result of a complex simulation model). Original algorithms have been imagined to optimize f by only relying on its evaluation. This family of techniques is generally called *derivative-free optimization* (DFO). One differentiates further the applicability of the derivative-free algorithms depending whether or not the function evaluation is costly to evaluate. Genetic algorithms obtain very good results for DFO benchmarks but they generally require a prohibitive number of evaluations. Finally, in many practical applications, considering a single-objective is not sufficient. In many cases, objectives are conflicting, which means they do not share the same optimum. As such, dedicated multi-objective optimization

© Springer International Publishing Switzerland 2015
L. Michel (Ed.): CPAIOR 2015, LNCS 9075, pp. 124–140, 2015.
DOI: 10.1007/978-3-319-18008-3_9

methods aim at finding a set of solutions being tradeoffs between the different objectives. This set of tradeoffs is called the Pareto front. This is the context of this work: *We are interested at optimizing multi-objective black-box functions costly to evaluate providing to the user a set of non-dominated points.*

Current state-of-the-art multi-objective optimization algorithms use the Pareto dominance to determine if new points can be added to the current Pareto front approximation. Our first contribution is the definition of a comparison function allowing to compare points with regards to a current Pareto front estimation.

Our second contribution is the definition of a framework, MOGEN, making use of this comparison function to solve multi-objective optimization problems. This framework uses *derivative-free optimization single-objective algorithms* (such as the Nelder-Mead algorithm) in which we substitute our new comparison function to the classical one. With this comparison function, these DFO single-objective algorithms are able to identify directions to discover new points potentially improving the current Pareto front optimization. This framework can be instantiated with several algorithms and performs elitism such that algorithms bringing the most improvement to the Pareto front approximation will be favoured. The aim of MOGEN is to solve multi-objective optimization problems using a limited amount of evaluations; such behaviour is desired to solve problems for which the evaluation is expensive in terms of computation time. For example, problems where each evaluation requires a costly simulation could be solved using MOGEN.

2 Background

A generic multi-objective optimization problem can be expressed as follows:

$$\begin{aligned} &\text{minimize } F(X) \equiv \{f_1(X),\ \ldots,\ f_m(X)\} \\ &\text{such that } X \in \Omega \end{aligned} \tag{1}$$

where $\Omega \subseteq \mathbb{R}^n$ is the feasible region and $f_i(X)$ are the objective functions. When $m = 1$, the problem is a single-objective optimization problem. For the rest of this paper, we consider the feasible region Ω defines upper and lower bounds on each dimension. In such case, $X \in \Omega$ can be translated as $\forall\, i \in \{1, \ldots, n\} : x_i \in [l_i, u_i]$ such that $\forall\, i \in \{1, \ldots, n\} : l_i < u_i$.

The *Pareto dominance* allows to evaluate if a point is better than another point with regards to several objective functions. Considering two points $x, y \in \Omega$, we say that the point x dominates the point y on functions $f_1, \ldots, f_m$, written $x \prec y$, if the two following conditions are satisfied:

$$x \prec y \equiv \begin{cases} \forall\, i \in \{1, \ldots, m\} : f_i(x) \leq f_i(y) \\ \exists\, i \in \{1, \ldots, m\} : f_i(x) < f_i(y) \end{cases}$$

Alternative dominance definitions exist, as those proposed in [11], [15] and [2], but they are not detailed in this article. These dominance definitions could also be used in the framework we define.

A point x is said to be Pareto optimal if it satisfies the following condition:

$$\nexists\, y \in \Omega : y \prec x$$

The Pareto optimal set is defined as the set of Pareto optimal points, i.e. the set of non-dominated points. Multi-objective optimization algorithms aim at finding an approximation of this Pareto optimal set.

3 Derivative-Free Optimization

In this section, we recall the concept of Derivative-Free optimization methods. Three popular DFO algorithms are described to illustrate this concept. These latter are used by the MOGEN framework described later in this article.

3.1 Derivative-Free Optimization Methods

Derivative-free optimization methods, as defined in [3] and [12], are optimization search techniques. These methods iteratively use comparisons between points to evaluate the search progress. This iterative design with comparisons is rather intuitive and many DFO algorithms rely on simple concepts and structures. The main advantage of these methods is that the only information they need is the comparison of evaluations of the objective functions.

DFO algorithms can be used to optimize *black-box* functions, i.e. functions for which the only information available is its evaluation. In many practical applications, it is desired to optimize black-box functions. There exists a huge range of single-objective DFO algorithms; several of them being described in [3]. In the following sections, we rapidly explain three single-objective DFO algorithms. These methods are designed to solve single-objective problems as defined in Equation 1 with $m = 1$. If new points x_i are discovered outside the box defined by Ω, they are replaced by the closest points in Ω to ensure the bound constraints are always respected.

The Directional Direct Search Algorithm. The Directional Direct Search Algorithm described in [3] converges to a local optimum by iteratively polling new points around the current iterate which is a point in Ω. This algorithm relies on a collection of unit vectors D and a step size α. At each iteration, for each direction $d_i \in D$, a new point around the current iterate, $x_{current}$ is created as follows: $x_i = x_{current} + \alpha \times d_i$. If a new point x_i is discovered such that it is better than $x_{current}$, i.e. $f(x_i) \leq f(x_{current})$, then x_i becomes the new iterate and a new iteration can begin. If no better point has been discovered at the end of an iteration, α is decreased. On the other hand, if the iteration was successful, α is either maintained or increased. By replacing the current iterate with better points, the algorithm eventually converges to a local minimum.

The Nelder-Mead Algorithm. The Nelder-Mead algorithm introduced in [13] is a popular single-objective DFO algorithm. This algorithm converges to a local optimum by iteratively applying transformations on a hypervolume (also called simplex). To solve a problem in $\Omega \subseteq \mathbb{R}^n$, the Nelder-Mead algorithm uses a hypervolume containing $n+1$ points. These points $y_0, \ldots, y_n$ are sorted such that $f(y_0) \leq f(y_1) \leq \ldots \leq f(y_{n-1}) \leq f(y_n)$. At each iteration, the worst point y_n is transformed into a new point y'_n such that $f(y'_n) \leq f(y_n)$. The transformations applied are, depending on the situation, reflection, expansion, inside and outside contraction. The Nelder-Mead transformations are applied around the centroid of all the hypervolume points but the worst. A 2D example of reflection of the worst point of a Nelder-Mead hypervolume is shown in Figure 1. In this example, the centroid of points y_0 and y_1 is y_c and the reflection of the worst point y_2 over the centroid is y_r.

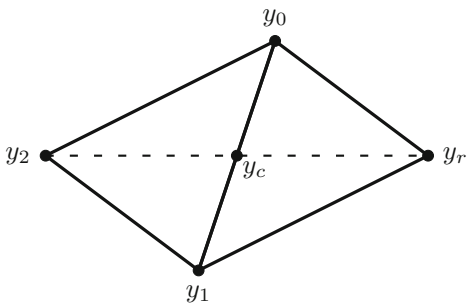

Fig. 1. Example of transformation of the worst point (y_2) from an hypervolume of the Nelder-Mead algorithm into its reflection (y_r) over the centroid (y_c)

If transformations fail to produce a new point better than y_n, the hypervolume is shrunk around y_0. By transforming the worst point of the hypervolume at each iteration, all points contained in the hypervolume are increasing in terms of quality eventually converging to a local minimum.

The MultiDirectional Search Algorithm. The MultiDirectional Search algorithm introduced in [7], similarly to the Nelder-Mead algorithm, applies transformation to a hypervolume structure to converge to a local optimum. This hypervolume contains $n+1$ points to solve problems in $\Omega \subseteq \mathbb{R}^n$. These points $y_0, \ldots, y_n$ are sorted such that $f(y_0) \leq f(y_1) \leq \ldots \leq f(y_{n-1}) \leq f(y_n)$. At each iteration, all points but the best are transformed into new points y'_i by applying transformations: reflection, expansion (these are different from those of the Nelder-Mead algorithm since they are applied on all points but the best of the hypervolume while only the worst point is transformed in the Nelder-Mead algorithm). If at least one of the new point y'_i is better than the former best point y_0, then the iteration is a success. Otherwise, the hypervolume is shrunk around

y_0. These successive transformations of the hypervolume eventually converge to a local optimum.

3.2 The Comparison Step

DFO algorithms rely on comparisons of the objective evaluations to decide whether a point allows the algorithm to progress and is worth to be kept. DFO methods need a comparison function, *cmp*, to assess if a point is better than another one according to the considered problem. For example, in the case of a minimization problem, the comparison function used would be $cmp_<(x_1, x_2) \equiv f(x_1) < f(x_2)$. Indeed, if we have $f(x_1) < f(x_2)$ where f is the objective function, then x_1 is better than x_2 in order to minimize f.

These comparison functions could be replaced by any comparison function according to the type of problem considered. For example some industrial applications would require to minimize a non-deterministic function. In this case, the comparative function $<$ is not sufficient. Hypothesis tests would be more relevant as comparative functions. As a consequence, it would be interesting to be able to adapt the comparison function used in a DFO search to the type of problem considered. DFO algorithms are parametrized with a comparison function $cmp(x_1, x_2)$ that returns a boolean which is true if x_1 is better than x_2, false otherwise. A formal definition of the comparison function used in DFO algorithms can be expressed as follows:

$$cmp: \quad \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{B}$$

A comparison function can be used in a DFO algorithm only if it is *transitive*. Using a non-transitive comparison function could lead to a non-productive looping search.

By using a comparison function as a parameter for a DFO algorithm, it is possible to solve different optimization problems declaratively. For example we could parametrize the Nelder-Mead algorithm as follows: $\text{NM}(f, cmp, x_0)$ where f is the objective to optimize, *cmp* is the comparison function to use (e.g. the function $cmp_<$ declared earlier) and x_0 is the starting point.

4 Multi-Objective Optimization Comparison Function

In this section we define a comparison function between two points in a multi-objective optimization context. The aim of this comparison function is to be used as a parameter in any DFO algorithm. Multi-objective optimization search techniques tend to iteratively improve their current approximation of the Pareto front. As stated in [5], the quality of a Pareto front approximation can be measured in terms of two criteria. First, it is desired that points in a Pareto front approximation are as close as possible to the optimal Pareto front. Then, it is desired that points in a Pareto front approximation span the whole optimal

Pareto front. The approximation of a Pareto front inside a multi-objective optimization algorithm is often referred to as an *archive*. An archive is a set of points such that no point is dominating (nor dominated by) any other point in the set.

The classical way of comparing two points with respect to several objectives is the Pareto dominance comparison function. In the context of an algorithm trying to improve its current archive, it is not sufficient. For example, comparing two points x_1 and x_2 with the Pareto dominance, could determine that no point is better than the other one (i.e. they don't dominate each other). On the other hand, one of these points could dominate more points in the archive. Therefore we introduce a new comparison function using the Pareto dominance between two points but applying it additionally to the whole archive. This comparison function compares two points x_1 and x_2 with respect to an archive A.

We define $dominates(x, A)$ as the set of points in A that x dominates. Similarly, $dominated(x, A)$ is the set of points in A by which x is dominated. From these two sets, we define the quantity $score(x, A)$ that represents the potential improvement brought by x in the archive A.

$$\begin{aligned} dominates(x, A) &= \{y \in A \mid x \prec y\} \\ dominated(x, A) &= \{y \in A \mid y \prec x\} \\ score(x, A) &= |dominates(x, A)| - |dominated(x, A)| \end{aligned}$$

The higher $score(x, A)$ is, the more promising x is to be included in the archive A. We base our comparison function $cmp_{\prec(A)}$ on this definition of *score*. It compares two points $x_1, x_2 \in \mathbb{R}^n$ with respect to a set of non-dominated points $A \subseteq \mathbb{R}^n$ and is defined as follows:

$$cmp_{\prec(A)}(x_1, x_2) = score(x_1, A') \geq score(x_2, A') \tag{2}$$

where $A' = A \cup \{x_1, x_2\}$. We use A' instead of A to guarantee that $cmp_{\prec(A)}$ is true if $x_1 \prec x_2$. Indeed, if the two points dominate or are dominated by the same number of points in the archive A, we would have $score(x_1, A) = score(x_2, A)$. In some cases, even if we obtain the same result for the evaluation of the *score* function on x_1 and x_2 with respect to A, x_1 dominates x_2, or vice versa. Figure 2 illustrates an example where both x_1 and x_2 dominate all the points in A but x_1 dominates x_2. In this situation, using the *score* function on A would not have been sufficient while using it on $A' = A \cup \{x_1, x_2\}$ would have shown that $x_1 \prec x_2$.

In [14], the transitivity of the Pareto dominance function is proven. It is then straightforward to prove that $cmp_{\prec(A)}$ defined in Equation (2) is transitive and could be used in DFO methods. $cmp_{\prec(A)}(x_1, x_2)$ has the behaviour desired to solve multi-objective optimization problems; it improves an archive A both in terms of distance to the real Pareto front and in terms of spreading of the points within A.

The next section defines a framework using single-objective DFO algorithms with $cmp_{\prec(A)}$ to perform multi-objective optimization.

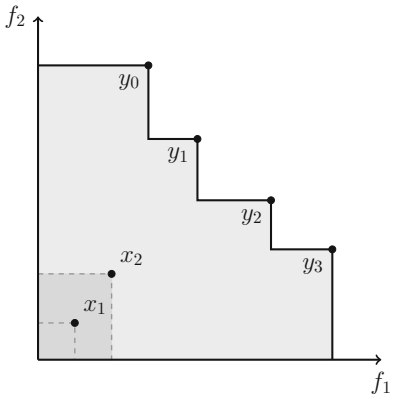

Fig. 2. The score values are equivalent: $score(x_1, A) = |A|$, $score(x_2, A) = |A|$ but $x_1 \prec x_2$. Need to use *score* with $A' = A \cup \{x_1, x_2\}$.

5 The MOGEN Algorithm

As stated in [4], most current state-of-the-art multi-objective optimization algorithms are *evolutionary/genetic* algorithms. These algorithms provide Pareto front approximations of high quality both in terms of spreading and in terms of distance to the real Pareto front but at the cost of an important number of function evaluations [9]. In many real world applications, the cost of evaluating the objective functions is too high to make those techniques practicable. Alternative multi-objective optimization methods should allow to find high quality Pareto front approximations using a small number of evaluations of the objective functions. The MOGEN algorithm attempts to perform efficient multi-objective optimization search using a limited amount of evaluations of the objective functions.

According to [4], only few multi-objective optimization search techniques which are not evolutionary/genetic algorithms are efficient. In particular, the Direct Multi-Search (DMS) algorithm introduced in [5] can be considered state-of-the-art. The Direct Multi-Search algorithm has the particularity that its archive contains triplets (x, D, α) where x is a point in the input space, D is a set of direction vectors and α is a real number. At each iteration, a triplet (x, D, α) is selected and new points are computed for a subset of $d \in D$ with $x_{new} = x + \alpha \times d$. The new points are then inserted as triplets (x_{new}, D, α) in the archive if they are not dominated (and dominated points are removed from the archive).

The MOGEN algorithm described in Algorithm 1 follows a similar approach since it associates to each point in the archive an algorithm and its current state of execution. MOGEN aims at using several single-objective DFO algorithms to perform multi-objective optimization. In MOGEN, the single-objective DFO algorithms are used to discover new points that are potentially inserted into a Pareto front approximation shared by all the single-objective DFO algorithms

used. It is possible to use these single-objective DFO algorithms by modifying the comparison function they usually use such that they favour points improving the shared Pareto front. All the data needed by an algorithm to continue its execution starting from where it has stopped is enclosed in its state of execution. This is illustrated on Figure 3. However MOGEN differs in two ways:

1. It may use several single-objective DFO algorithms. In Figure 3, three different DFO algorithms are used.
2. It uses the $cmp_{\prec(A)}$ comparison function, focusing on the possible improvements of the current archive.

To the best of our knowledge no other multi-objective optimization search method proposes to use single-objective DFO algorithms. In MOGEN, several single-objective DFO algorithms are in competition.

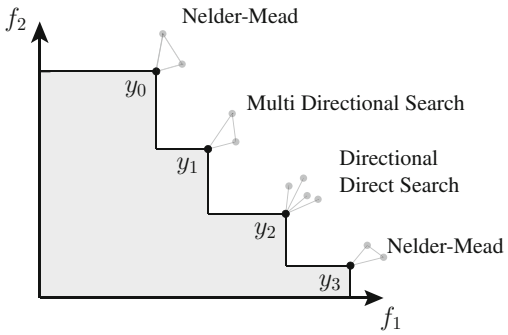

Fig. 3. An example of a MOGEN archive. The current state of execution is attached to each point of the archive. For example, y_1 is the best point of the simplex of a Nelder-Mead algorithm state.

Algorithm 1. MOGEN

Input: F A set of functions $f_i : \mathbb{R}^n \to \mathbb{R}$ to optimize
Input: Ω The feasible region ($\Omega \subseteq \mathbb{R}^n$)
Input: *cmp* A comparison function
Input: M A set of DFO algorithms
Output: *Arch* A set of non-dominated points in $\mathbb{R}^n$ which is an approximation of the Pareto front

```
1 Arch ← InitArchive(M, Ω)
2 while stopping criterion not met do
3 |   (x, a, s) ← SelectIterate(Arch)
4 |   (Y_new, s_new) ← Apply(x, a, s, cmp, Ω)
5 |   Arch ← AddAndClean(Arch, Y_new, a, s_new)
6 return Arch
```

5.1 The Algorithm

The MOGEN algorithm described in Algorithm 1 takes several parameters as input. The first parameter, F, is the set of functions to optimize. The second parameter, Ω, is the feasible region of the considered problem. The third parameter, *cmp*, is the comparison function used by DFO algorithms to compare points. This comparison function must be adapted to the type of problem considered (e.g. deterministic maximization, stochastic minimization, ...). In this article, we assume MOGEN uses the comparison function $cmp_{\prec(A)}$. The last parameter, $M = \{a_1, \ldots, a_m\}$, is a set of single-objective derivative-free algorithms in which the comparison function *cmp* can be used. A simplified version of MOGEN could consider the use of a single algorithm which would be the same for every element in the archive.

The `InitArchive` function in Algorithm 1 at Line 1 initializes the archive as a set of triplets (x, a, s) where $x \in \Omega$, a is an algorithm from the set of algorithms M and s is a state for the algorithm a in which x is the best point (according to *cmp*). This state s is needed to be able to start a new iteration of the algorithm from where it was paused before. The algorithm a considered can still remain a black-box algorithm; it only has to be able to perform a single iteration on demand. After the initialization, the elements in *Arch* are non-dominated points. Once *Arch* has been initialized, the algorithm updates it iteratively until a stopping criterion is met. This stopping criterion is somewhat arbitrary. Examples of stopping criteria are: a given number of iteration has been reached, a given number of evaluations has been exceeded or the running time of the algorithm exceeds a given threshold.

The first step performed at each iteration at Line 3 is the selection of a triplet (x, a, s) in the archive as current iterate. The `Apply` function at Line 4 allows to perform a single iteration of a DFO algorithm from a given state[1]. It takes five elements as parameters. The first parameter is x, the best point of the state s. The second parameter, a is the algorithm on which the iteration has to be performed. The third parameter is s, the state from which the algorithm a begins its iteration. The fourth parameter, *cmp*, is the comparison function used in a. The last parameter is Ω, the feasible region. `Apply` returns two elements. The first one is Y_{new}, the set of new points discovered during the iteration of a (the algorithm sometimes discovers multiple interesting points, for example in the case of a Shrink of the Simplex in the Nelder-Mead algorithm). The second one is s_{new}, the new state reached after one iteration of a was performed starting from state s.

Once Y_{new} has been computed, the function `AddAndClean` at Line 5 updates the archive. It takes four arguments. The first argument is the current archive, *Arch*. The second argument, Y_{new}, is the set of new points found by using the `Apply` function with x, a, s, *cmp* and Ω. The third argument is the algorithm that was used to find the points in Y_{new} and the fourth argument s_{new} is the state of algorithm a in which new points from Y_{new} were found. `AddAndClean`

[1] For instance a reflection operation applied to the simplex of a Nelder-Mead instance.

compares every point y_i in Y_{new} with elements from the archive. If an element in the archive is dominated by a point y_i, it is removed from $Arch$. If a point y_i is not dominated by any point in the archive $Arch$, it is added in $Arch$ as a triplet (y_i, a, s_{new}). When an element (y_i, a, s_{new}) is added to the archive, the state of execution s_{new} associated to y_i is a *clone* of the state of execution from which it was obtained; as such, each instance of the same algorithm in the archive has its own state of execution that is not shared. When the stopping criterion is met, the algorithm returns the current archive, $Arch$.

If we consider an archive containing triplets with several different DFO algorithms a_i, then MOGEN performs elitism on these algorithms. Indeed, an algorithm a_{bad} performing poorly and failing to find a lot of new non-dominated points will generate fewer new point triplets $(x_j, a_{\text{bad}}, s_j)$ in the archive. On the opposite, an algorithm a_{good} with good performances discovering many new non-dominated points will generate more new point triplets $(x_k, a_{\text{good}}, s_k)$ in the archive. Furthermore, the `AddAndClean` function will clean points which are further from the optimal Pareto front. Triplets $(x_j, a_{\text{bad}}, s_j)$ obtained with algorithms failing to get closer to the optimal Pareto Front will be eventually removed from the archive. Such algorithms will have less and less triplet representatives and potentially disappear from the archive. On the opposite, algorithms generating a lot of non-dominated points and getting closer to the optimal Pareto front will have more and more triplets representatives. As such, the MOGEN algorithm is elitist because it tends to use more and more often algorithms providing good results and to use less and less often (or even abandon) algorithms providing poor results.

5.2 Running Example

Let us consider a small running example of the MOGEN algorithm. This bi-objective problem has the following definition:

$$\begin{aligned}&\text{minimize } F(x_1, x_2) \equiv \left\{(x_1 - 1)^2 + (x_1 - x_2)^2), (x_1 - x_2)^2 + (x_2 - 3)^2\right\}\\&\text{such that } x_1, x_2 \in \Omega \equiv [-5, 5]\end{aligned}$$

We consider an initial archive containing only two triplets: one associated to the Nelder-Mead algorithm and the other one to the Directional Direct Search algorithm. The first triplet is (x_0, NM, s_0) where $x_0 = (2.0, 2.0)$, NM is the Nelder-Mead algorithm and $s_0 = \{x_0 = (2.0, 2.0); x_{0,1} = (0.0, 2.0); x_{0,2} = (2.0, 0.0)\}$ is the hypervolume (also called simplex) structure (i.e. the *state*) of the Nelder-Mead algorithm. This hypervolume is well sorted according to the $cmp_{\prec(A)}$ since its evaluations are: $\{F(x_0) = (1.0, 1.0); F(x_{0,1}) = (5.0, 5.0); x_{0,2} = (5.0, 13.0)\}$. The second triplet is (x_1, DDS, s_1) where $x_1 = (0.75, 1.5)$, DDS is the Directional Direct Search algorithm and $s_1 = (D_1, alpha_1)$ is the state of the DDS algorithm where $D_1 = \{(1, 0); (-1, 0); (0, 1); (0, -1)\}$ is the collection of unit vectors and $\alpha_1 = 0.5$ is the step size. The initial archive is thus $Arch = \{(x_0, NM, s_0), (x_1, DDS, s_1)\}$.

The first iteration begins with `SelectIterate`($Arch$). For this example, we consider that `SelectIterate` considers the archive as a FIFO queue; it selects

the first triplet in the archive. At the end of end of the iteration, the current iterate triplet is appended at the end of the archive with the new discovered points. The selected triplet is (x_0, NM, s_0). The iteration continues with $\texttt{Apply}(x_0, NM, s_0, cmp_{Arch}, \Omega)$ that applies an iteration of the Nelder-Mead algorithm starting from state s_0. The Nelder-Mead algorithm applies a reflection and an internal contraction discovering the new point $x_0^{new} = (1.5, 1.0)$ which evaluations are $F(x_0^{new}) = (0.5, 4.25)$. The `Apply` function returns a pair (Y_0^{new}, s_0^{new}) where $Y_0^{new} = \{x_0^{new} = (1.5, 1.0)\}$ is the set of new points discovered and $s_0^{new} = \{x_0 = (2.0, 2.0); x_0^{new} = (1.5, 0.5); x_{0,1} = (0.0, 2.0)\}$ is the new hypervolume for the Nelder-Mead algorithm associated to x_0. Then the iteration ends with $\texttt{AddAndClean}(Arch, Y_0^{new}, NM, s_0^{new})$ which inserts points from Y_0^{new} in $Arch$ such that they are associated to NM and s_0^{new}. The new archive is thus $Arch = \{(x_1, DDS, s_1), (x_0^{new}, NM, s_0^{new}), (x_0, NM, s_0^{new})\}$ at the end of the first iteration.

The second iteration begins with $\texttt{SelectIterate}(Arch)$ which returns the triplet (x_1, DDS, s_1). Then, $\texttt{Apply}(x_1, DDS, s_1, cmp_{Arch_0}, \Omega)$ applies an iteration of the Directional Direct Search algorithm starting from state s_1. Polling discovers a new point $x_1^{new} = (1.25, 1.5)$ which evaluation is $F(x_1^{new}) = (0.125, 2.3125)$. The `Apply` function returns a pair (Y_1^{new}, s_1^{new}) where $Y_1^{new} = \{x_1^{new} = (1.25, 1.5)\}$ and $s_1^{new} = (D_1, \alpha_1 = 1.0)$. Finally, $\texttt{AddAndClean}(Arch, Y_1^{new}, DDS, s_1^{new})$ inserts points from Y_1^{new} in $Arch$ to obtain $Arch = \{(x_0, NM, s_0^{new}), (x_1^{new}, DDS, s_1^{new})\}$. Note that x_0^{new} and x_1 have been removed from the archive since they were dominated by x_1^{new}.

6 Performance Assessment and Benchmarks

Intuitively it is desirable for a Pareto front estimation to contain points close to the optimal Pareto front and representing a diversified subset of the optimal Pareto front. Several measures exist to evaluate these two criteria (see in [17] and [16]). We present only the purity and delta metrics both used in [5] to evaluate the state-of-the-art Direct Multi-Search algorithm.

6.1 The Purity Metric

The *Purity* metric defined in [1] allows to compare several Pareto front approximations and define which one is the closest to the optimal Pareto front. Let us consider several different multi-objective DFO algorithms. Let $\mathcal{A}$ be the set of archives A_i produced by each algorithm i, and let A_{global} be the union of these archives A_i with dominated points removed:

$$A_{global} = \left\{ u \in \bigcup_{\mathcal{A}} A_i \;\middle|\; \forall\, v \neq u \in \bigcup_{\mathcal{A}} A_i : u \not\prec v \right\}$$

The Purity metric of an archive $A_i \in \mathcal{A}$ is then defined as the ratio of the number of points in both A_i and A_{global} and the number of points in A_{global}:

$$Purity(A_i) = \frac{|A_i \cap A_{global}|}{|A_{global}|}$$

The higher the Purity of an archive A_i is, the less points from A_i are dominated by other archives and the closer A_i is to the optimal Pareto front. As mentioned in [5], two similar solvers produce similar archives, which can decrease their performances for the Purity metric since many points from these approximations will dominate each other. Therefore a third solver could benefit from this effect and obtain a higher Purity metric performance than the other two. To avoid this phenomenon, we only compare solvers in pairs for the Purity metric.

6.2 The Delta Metric

The Delta metric, Δ, proposed in [6] and extended in [5], is a spreading metric like the Gamma metric (Γ) defined in [5]. The Γ metric is however ambiguous for problems with more than two objectives. For a given archive, F_i is an objective space containing k dimensions, the Delta metric $\Delta(A_i)$ is defined as follows:

$$\Delta(A_i) = \max_{j=1,\ldots,n} \left(\frac{\delta_{0,j} + \delta_{k,j} + \sum_{i=1}^{k-1} \left|\delta_{i,j} - \bar{\delta_j}\right|}{\delta_{0,j} + \delta_{k,j} + (k-1)\bar{\delta_j}} \right)$$

where $\bar{\delta_j}$ for $j = 1, \ldots, n$ is the average of the distances $\delta_{i,j} = f_{i+1,j} - f_{i,j}$ with $i = 1, \ldots, k-1$ (assuming the objective function values have been sorted by increasing value for each objective j). This metric allows to measure how well an archive is spread along several objective dimensions.

6.3 Benchmark Problems

Our goal is to assess the performances of the MOGEN algorithm on a wide range of multi-objective problems reported in the literature. We consider problems involving bound constraints on each dimension, i.e. problems where the input space is contained in a hyper-volume: $\Omega = [l, u]$ with $l, u \in \mathbb{R}^n$ and $l < u$.

In [5], a collection of 100 problems with bound constraints from the literature was modelled in AMPL (A Modelling Language for Mathematical Programming) [10]. This collection of problems, available at http://www.mat.uc.pt/dms, contains a wide range of problems with different dimensions and properties. We used this collection to assess the performances of our algorithm.

The results obtained on these benchmarks for the Purity and the Δ metrics are reported graphically using performance profiles as in [8] and [5]. Let $t_{p,s}$ represent the performance of the solver $s \in S$ on the problem p such that lower values of $t_{p,s}$ indicate better performance. The ratio of the performance is defined as follows: $r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s^*} | s^* \in S\}}$. A ratio $r_{p,s} = 1$ means that solver s obtained the best value on problem p. The performance profile $\rho_s(\tau) = \frac{1}{|P|} \times |\{p \in P | r_{p,s} \leq \tau\}|$ is a cumulative distribution of the performance of s compared to other solvers.

7 Results

Our results are divided in two parts. We first compare different MOGEN variants, then we compare MOGEN to the Direct Multi-Search algorithm.

7.1 Comparison of MOGEN Variants

The MOGEN algorithm can be instantiated in several ways since it involves several parameters and heuristics. We restrict our experimental comparison to different algorithm sets in the initial archive; the other (meta-)parameters are fixed and described as follows.

We decided to initialize all MOGEN variants with the *Line Multiple Point Initialization*. It selects n equidistant points on the line connecting l and u, respectively the lower and the upper bound of the input space $\Omega \subseteq \mathbb{R}^n$. The initial archive is defined as follows: $A_0 = \{l+(\frac{i}{n-1})(u-l)\}$ where $i = 0, \ldots, n-1$. The results obtained in this paper were all performed for $n = 10$ to ensure that all MOGEN variants start from the same initial archive. To select the iterate at the beginning of each iteration, we apply the (fair) *First In First Out Queue* selection heuristics. This means every iteration, a point is popped from the queue, an iteration is performed and the point (and possibly the new points found) is inserted back into the queue.

Finally, MOGEN variants may differ according the DFO algorithm associated with each point in the initial archive. We consider four different versions of MOGEN with the following algorithm proportions:

- **MOGEN(NM)** uses only the Nelder-Mead algorithm introduced in [13].
- **MOGEN(DDS)** uses only the Directional Direct Search algorithm as described in [3].
- **MOGEN(MDS)** uses only the MultiDirectional Search algorithm introduced in [7].
- **MOGEN(ALL)** uses the three algorithms described above; namely Nelder-Mead, Directional Direct Search and MultiDirectional Search. These algorithms are represented in the initial archive in identical proportions.

In Figure 4 we see the evolution of the average Purity metric on the benchmarks for the four MOGEN variants. As can be observed, MOGEN(NM) outperforms the other three variants. Even after a very small number of evaluations, MOGEN(NM) already has a high average Purity metric. It remains the case after a larger number of evaluations.

In Figure 5 we see the performance profiles of the Purity metric for the four MOGEN variants with a maximum budget of 20,000 evaluations. The performance profiles graph have to be read as follow: for a given solver s, a point on this graph in $(\tau, \rho(\tau))$ means that for a proportion $\rho(\tau)$ of the tested instances, the solver s was at worst τ times worse than the best performance obtained on all solvers represented in the graph. As such, the height $\rho(\tau)$ reached by a solver s in $\tau = 1$ represents the proportion $\rho(\tau)$ of instances on which solver s obtained

the best performance among all represented solvers. If the curve of a given solver s never reaches $\rho(\tau) = 1$ but stops at $\rho(\tau) = r$, it means that the solver obtained an infinite performance ratio for a proportion $(1 - r)$ of the instances.

As can be observed, MOGEN(NM) outperforms the three other variants. These curves even show that MOGEN(NM) is very efficient. Indeed, for $\tau = 1$, MOGEN(NM) obtains the best metric value for more than 85% of the problems. MOGEN(NM) is also very robust since it is able to find at least one non-dominated point for more than 95% of the problems (i.e. there were less than 5% of the instances for which MOGEN(NM) was not able to discover a single point in the global archive). MOGEN(ALL) is only the second best. This could be explained by the fact that, eventually, points in the archive associated to the MDS and DDS algorithms are dominated by points associated to NM, leading to an increasing proportion of points associated to NM, performing better.

In Figure 6 we see the evolution of the average Δ metric on the benchmarks for the four MOGEN variants. As can be observed, MOGEN(NM) outperforms the other three variants. Even after a very small number of evaluations, MOGEN(NM) already has a low average Δ metric. It remains the case after a larger number of evaluations as this metric stabilizes after around 1,000 evaluations.

Figure 7 shows performance profiles of the Delta metric for the four MOGEN variants. For the Δ metric, MOGEN(NM) seems to outperform the other variants. We can also see that it is again the MOGEN(DDS) variants that seems to have the worst performance for the Δ metric while the MOGEN(MDS) and the MOGEN(ALL) variants have similar performances. We can conclude that, according to the Purity and the Δ metrics, MOGEN(NM) is the MOGEN variant showing the best performance.

7.2 Comparison of MOGEN and Direct Multi-Search

In [5], the authors show that the Direct Multi-Search (DMS) algorithm outperformed state-of-the-art solvers. We compare the best MOGEN variant, MOGEN(NM), to DMS. The parameters used for MOGEN(NM) are the same that those used to compare MOGEN variants. In Figure 8, we compare the purity metrics for MOGEN(NM) and the DMS algorithm. MOGEN(NM) performs globally better than the DMS algorithm. Indeed, the performance profile reveals that MOGEN(NM) has the best purity metric value for more than 65% of the instances while the DMS algorithm only has the best purity metric value for less than 40% of the instances. The fact that the DMS algorithm seems to perform better than MOGEN(NM) for 8 instances between $\tau = 6$ and $\tau = 100$ only means that when the archive of DMS is largely dominated by another archive, it tends to have a few more non-dominated points than MOGEN(NM).

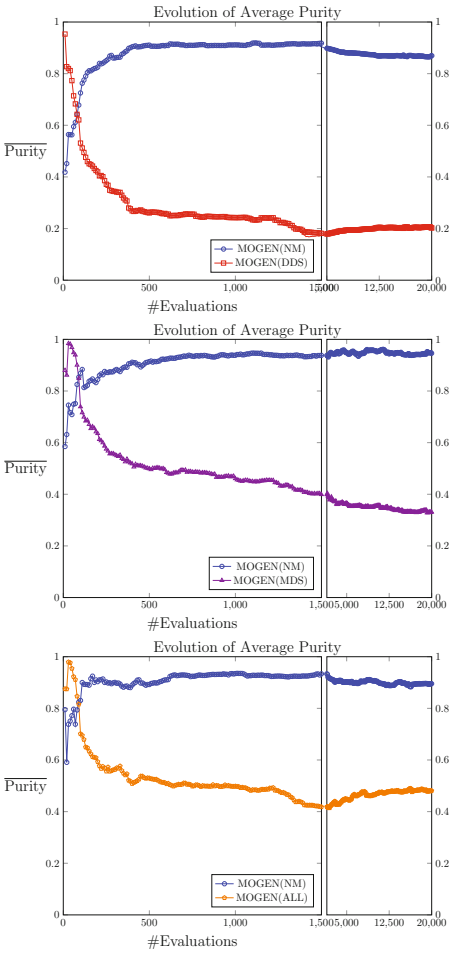

Fig. 4. Average Purity evolution - MOGEN

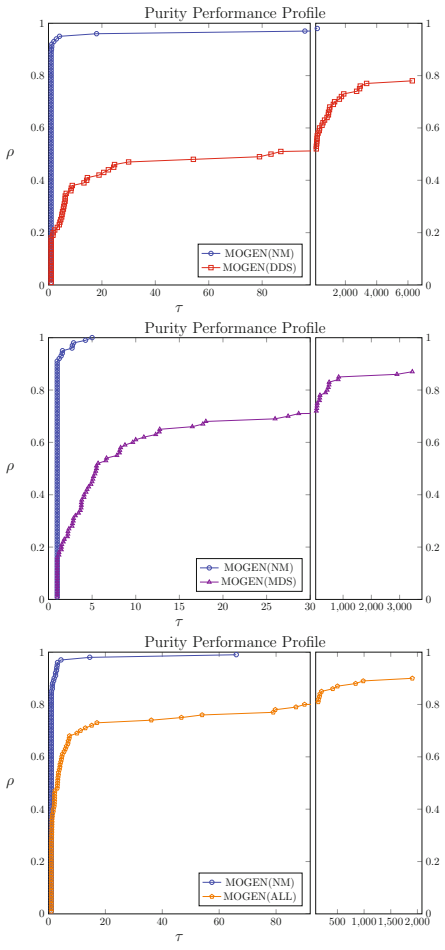

Fig. 5. Purity performance profiles - MOGEN

Figure 7 shows performance profiles of the Δ metric for the four MOGEN variants and DMS. For the Δ metric, DMS seems to outperform the MOGEN variants. However, a solver can have a very good Δ metric performance while its archive is completely dominated by the archives of other solvers. As such, DMS seems to diversify more than MOGEN(NM) but tends to produce archives dominated by those produced by MOGEN(NM). Similarly to what is done in DMS, a *search step* could be added at the beginning of each iteration of MOGEN to perform more diversification.

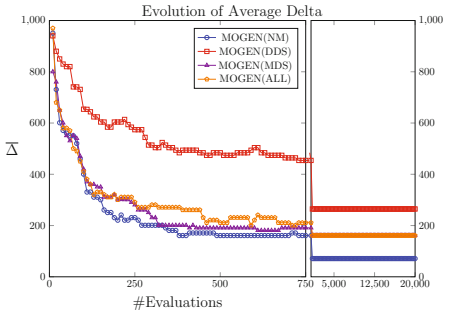

Fig. 6. Average Δ metric evolution - MOGEN

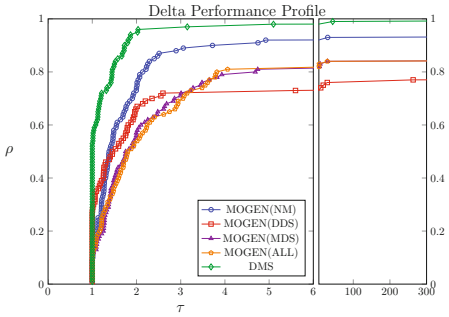

Fig. 7. Δ metric performance profiles

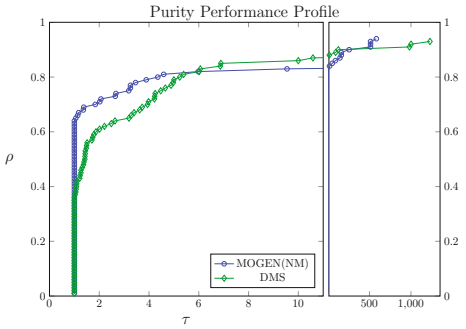

Fig. 8. Purity performance profiles - MOGEN(NM) & DMS

8 Conclusion

In this paper, we introduced a new comparison function, $cmp_{\prec(A)}$, allowing to compare points in a multi-objective optimization context with regards to an existing archive A. We defined a new generic multi-objective optimization framework, MOGEN, that uses single-objective DFO algorithms with $cmp_{\prec(A)}$. Several MOGEN variants using different sets of DFO algorithms have been compared and the one with the Nelder-Mead algorithm has obtained the best performances. The comparison between DMS and MOGEN revealed that the latter produces archives closer to the optimal Pareto front but tends to be less diversified.

Several research directions could be explored as future work. It would be interesting to use MOGEN with different comparison functions. Other algorithms could be used in the algorithm as well and it is possible that other single-objective DFO algorithms would improve greatly the performances obtained by MOGEN. The heuristics used in this algorithm should also be studied to reveal how much they impact the produced archive.

Acknowledgments. We thank the anonymous reviewers for their valuable comments. This research is partially supported by the UCLouvain Action de Recherche Concert?e ICTM22C1.

References

1. Bandyopadhyay, S., Pal, S., Aruna, B.: Multiobjective gas, quantitative indices, and pattern classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **34**(5), 2088–2099 (2004)
2. Ben Abdelaziz, F., Lang, P., Nadeau, R.: Dominance and efficiency in multicriteria decision under uncertainty. Theory and Decision **47**, 191–211 (1999)
3. Conn, A., Scheinberg, K., Vicente, L.: Introduction to Derivative-Free Optimization. Society for Industrial and Applied Mathematics, Mps-siam Series on Optimization (2009)
4. Custódio, A., Emmerich, M., Madeira, J.: Recent Developments in Derivative-Free Multiobjective Optimization (2012)
5. Custódio, A.L., Madeira, J.F.A., Vaz, A.I.F., Vicente, L.N.: Direct multisearch for multiobjective optimization. SIAM Journal on Optimization **21**(3), 1109–1140 (2011)
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
7. Dennis Jr., J.E., Torczon, V.: Direct search methods on parallel machines. SIAM Journal on Optimization **1**, 448–474 (1991)
8. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical programming **91**(2), 201–213 (2002)
9. Echagüe, E., Delbos, F., Dumas, L.: A global derivative-free optimization method for expensive functions with bound constraints. In: Proceedings of Global Optimization Workshop, pp. 65–68 (2012)
10. Fourer, R., Gay, D.M., Kernighan, B.W.: A modeling language for mathematical programming. Management Sci. **36**, 519–554 (1990)
11. Kollat, J., Reed, P., Kasprzyk, J.: A new epsilon-dominance hierarchical bayesian optimization algorithm for large multiobjective monitoring network design problems. Advances in Water Resources **31**(5), 828–845 (2008)
12. Mor, J., Wild, S.: Benchmarking derivative-free optimization algorithms. SIAM Journal on Optimization **20**(1), 172–191 (2009)
13. Nelder, J.A., Mead, R.: A simplex method for function minimization. In Comput. J. **7** (1965)
14. Voorneveld, M.: Characterization of pareto dominance. Operations Research Letters **31**(1), 7–11 (2003)
15. Zitzler, E., Brockhoff, D., Thiele, L.: The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 862–876. Springer, Heidelberg (2007)
16. Zitzler, E., Knowles, J., Thiele, L.: Quality assessment of pareto set approximations. In: Branke, J., Deb, K., Miettinen, K., Słowiński, R. (eds.) Multiobjective Optimization. LNCS, vol. 5252, pp. 373–404. Springer, Heidelberg (2008)
17. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., da Fonseca, V.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Transactions on Evolutionary Computation **7**(2), 117–132 (2003)