Chapter 2
A Gradient-Free Method for Multi-objective Optimization Problem

Nantu Kumar Bisui, Samit Mazumder, and Geetanjali Panda

Abstract In this chapter, a gradient-free method is proposed for solving the multi-objective optimization problem in higher dimension. The concept is developed as a modification of the Nelder-Mead simplex technique for the single-objective case. The proposed algorithm is verified and compared with the existing methods with a set of test problems.

Keywords Nelder-Mead simplex method · Multi-objective programming · Gradient-free method · n dimension simplex

2.1 Introduction

A general multi-objective optimization problem is stated as

$$(MOP) \quad \min_{x \in S \subset \mathbb{R}^n} F(x),$$

$F(x) = (f_1(x), f_2(x) \ldots, f_m(x))^T$, $f_j : \mathbb{R}^n \to \mathbb{R}$, $j = 1, 2, \ldots, m$, $m \geq 2$. In practice, (MOP) involves several conflicting and non-commensurate objective functions which have to be optimized simultaneously over $\mathbb{R}^n$. If $x^* \in \mathbb{R}^n$ minimizes all the objective functions simultaneously, then certainly an ideal solution is achieved. But in general, improvement in one criterion results in loss in another criterion, leading to the unlikely existence of an ideal solution. For this reason one has to look for the "best" compromise solution, which is known as an efficient or Pareto optimal

N. K. Bisui · S. Mazumder · G. Panda (✉)
Department of Mathematics, Indian Institute of Technology Kharagpur, Kharagpur 721302, India
e-mail: geetanjali@maths.iitkgp.ac.in

N. K. Bisui
e-mail: nantukrbisuimath@gmail.com

S. Mazumder
e-mail: samitmazumder599@gmail.com

V. Laha et al. (eds.), *Optimization, Variational Analysis and Applications*,
Springer Proceedings in Mathematics & Statistics 355,
https://doi.org/10.1007/978-981-16-1819-2_2

solution. The concept of efficiency arises from a pre-specified partial ordering on $\mathbb{R}^m$. The points satisfying the necessary condition for efficiency are known as critical points. Application of these kinds of problems are found in engineering design, statistics, management science, etc.

Classical methods for solving (MOP) are either scalarization methods or heuristic methods. Scalarization methods reduce the main problem to a single objective optimization problem using predetermined parameters. A widely used scalarization method is due to Geofforion [15], which computes proper efficient solutions. Geofforion's approach has been further developed by several researchers in several directions. Other parameter-free techniques use the concept of order of importance of the objective functions, which have to be specified in advance. Another widely used general solution strategy for multi-objective optimization problems is the ϵ-constrained method [10, 21]. All the above methods are summarized in [1, 8, 19, 23]. These scalarization methods are user-dependent and often have difficulties in finding an approximation to the Pareto front. Heuristic methods [7] do not guarantee the convergence property but usually provide an approximate Pareto front. Some well-known heuristic methods are genetic algorithms, particle swarm optimization, etc. NSGA-II [7] is a well-known genetic algorithm.

Recently, many researchers have developed line search methods for (MOP), which are different from the scalarization process and heuristic approach. These line search numerical techniques are possible extensions of gradient-based line search techniques for the single-objective optimization problem to the multi-objective case. In every gradient-based line search method, the descent direction at an iterative point x is determined by solving a subproblem at x, and a suitable step length α at x in this direction is obtained using the Armijio type condition with respect to each objective function to ensure $f_j(x + \alpha d) < f_j(x)$. Then a descent sequence is generated, which can converge to a critical point. Some recent developments in this direction are summarized below.

The steepest descent method, which is the first line search approach for (MOP), was developed by Fliege and Svaiter [12] in 2000 to find a critical point of (MOP). In this method, descent direction d at every iterating point x is the solution of the following subproblem,

$$\inf_{d\in\mathbb{R}^n} \max_j \nabla f_j(x)^T d,$$

which is same as

$$\begin{aligned} &\min_{t,d} \; t + \frac{1}{2} d^T d \\ \text{subject to} \quad &\nabla f_j(x)^T d - t \leq 0, \quad j = 1, 2, ..., m \\ &t \in \mathbb{R}, d \in \mathbb{R}^n. \end{aligned}$$

The Newton method for single-objective optimization problem is extended to (MOP) by Fliege et al. [11] in 2009, which uses convexity criteria. Newton direction for (MOP) at x is obtained by solving the following min-max problem, which

involves the quadratic approximation of all the objective functions.

$$\inf_{d\in\mathbb{R}^n} \max_{j\in\Lambda_m} \nabla f_j(x)^T d + \frac{1}{2} d^T \nabla^2 f_j(x) d$$

This is equivalent to the following subproblem.

$$\min_{t\in\mathbb{R}, d\in\mathbb{R}^n} t$$
$$\text{subject to} \quad \nabla f_j(x)^T d + \frac{1}{2} d^T \nabla^2 f_j(x) d - t \leq 0, \quad j = 1, 2, ..., m.$$

If every f_j is a strictly convex function, then the above subproblem is a convex programming problem. Using the Karush-Kuhn-Tucker (KKT) optimality condition, the solution of this subproblem becomes the Newton direction $d_N(x)$ as

$$d_N(x) = -[\Sigma_{j=1}^m \lambda_j(x) \nabla^2 f_j(x)]^{-1} \Sigma_{j=1}^m \lambda_j(x) \nabla f_j(x),$$

where $\lambda_j(x)$ are Lagrange multipliers. This iterative process is locally and quadratically convergent for Lipschitz continuous functions.

An extension of the Quasi-Newton method for (MOP) is studied by Qu et al. [27] in 2011 for critical point, which avoids convexity assumptions. The method proposed by Qu et al. [27] uses the approximate Hessian of every objective function. The subproblem in Qu et al. [27] is

$$\min_{t\in\mathbb{R}, d\in\mathbb{R}^n} t$$
$$\text{subject to} \quad \nabla f_j(x)^T d + \frac{1}{2} d^T B_j(x) d - t \leq 0 \quad j = 1, 2, ..., m,$$

where $B_j(x)$ is the approximation of $\nabla^2 f_j(x)$.

These individual $B_j(x)$ are replaced by a common positive definite matrix in Ansari and Panda [2] to reduce the complexity of the algorithm. In [2], the descent direction at every iterating point x is determined by solving the following subproblem which involves linear approximation of every objective function along with the common positive definite matrix $B(x)$ in place of individual matrices $B_j(x)$.

$$\min_{t,\ d} \ t + \frac{1}{2} d^T B(x) d$$
$$\text{subject to } \nabla f_j(x)^T d - t \leq 0 \ j = 1, 2, ..., m,$$
$$t \in \mathbb{R}, \ \ d \in \mathbb{R}^n.$$

Here, a sequence of positive definite matrices is generated during the iterative process like the quasi-Newton method for the single-objective case. The Armijo-

Wolfe type line search technique is used to determine the step length. A descent sequence is generated whose accumulation point is a critical point of (MOP) under some reasonable assumptions.

The above line search techniques are restricted to unconstrained multi-objective programming problems, which are further extended to constrained multi-objective problems. A general constrained multi-objective optimization problem is

$$(MOP_C): \begin{cases} \min\limits_{x\in\mathbb{R}^n} \ F(x) \\ \text{subject to } g_i(x) \leq 0, \ \ i = 1, 2, ..., p \end{cases}$$

Concept of the line search methods for single objective constrained optimization problems are extended to the multi-objective case in some recent papers; see [3, 4, 13]. A variant of the sequential quadratic programming (SQP) method is developed for inequality constrained (MOP_C) in the light of the SQP method for the single-objective case by Ansari and Panda [4] recently. The following quadratic subproblem is solved to obtain a feasible descent direction at every iterating point x, which involves linear approximations of all functions.

$$\begin{aligned} &\min_{t,\ d} \ t + \frac{1}{2}d^T d \\ &\text{subject to } \nabla f_j(x)^T d \leq t, \ \ j = 1, 2, ..., m, \\ &g_i(x) + \nabla g_i(x)^T d \leq t, \ \ i = 1, 2, ..., p, \\ &t \in \mathbb{R}, \ \ d \in \mathbb{R}^n. \end{aligned}$$

The same authors consider a different subproblem in [3] which involves quadratic approximations of all the functions, and use the SQCQP technique to develop a descent sequence. This subproblem is

$$\begin{aligned} &\min_{t,\ d} \ t \\ &\text{subject to } \nabla f_j(x)^T d + \tfrac{1}{2}d^T \nabla^2 f_j(x) d - t \leq 0, \ \ j = 1, 2, ..., m \\ &g_i(x) + \nabla g_i(x)^T d + \tfrac{1}{2}d^T \nabla^2 g_i(x) d \leq 0, \ \ i = 1, 2, ..., p \\ &t \in \mathbb{R}, d \in \mathbb{R}^n. \end{aligned}$$

With these subproblems, a non-differentiable penalty function is used to restrict constraint violations. To obtain a feasible descent direction, the penalty function is considered as a merit function with a penalty parameter. The Armijo type line search technique is used to find a suitable step length. Global convergence of these methods is discussed under the Slater constraint qualification.

The above iterative schemes are free from the burden of selection of parameters in advance, and also have the convergence property. These iterative schemes are gradient-based methods, and large-scale problems can be solved efficiently only if the gradient information of the functions is available. Some optimization software

packages perform the finite difference gradient evaluation internally. But this is inappropriate when function evaluations are costly and noisy. Hence there is a growing demand for derivative-free optimization methods which neither require derivative information nor approximate the derivatives. The reader may refer to the book Cohn et al. [5] for the recent developments on derivative-free methods for single-objective optimization problem.

Coordinate search is the simplest derivative-free method for the unconstrained single-objective optimization problem. It evaluates the objective function of n variables at $2n$ points around a current iterate defined by displacements along the coordinate directions, their negatives, and a suitable step length. The set of these directions form a positive basis. This method is slow but capable of handling noise and guarantees to converge globally. The implicit filtering algorithm is also a derivative-free line search algorithm that imposes sufficient decrease along a quasi-Newton direction. Here, the true gradient is replaced by the simplex gradient. This method resembles the quasi-Newton approach. The trust region-based derivative-free line search method is also in demand to address noisy functions. In this method, quadratic subproblems are formulated from polynomial interpolation or regression. The implicit filtering is less efficient than the trust region but more capable of capturing noise.

The next choice is the widely cited Nelder-Mead method [24], which is a direct search iterative scheme for single objective optimization problems. This evaluates a finite number of points in every iteration, which takes care of the function values at the vertices of the simplex $\{y^0, y^1, ..., y^n\}$ in n dimension, ordered by increasing values of the objective function which has to be minimized. Action is taken based on simplex operations such as reflections, expansions, and contractions (inside or outside) at every iteration. The Nelder-Mead method attempts to replace the simplex vertex that has the worst function value. In such iterations, the worst vertex y^n is replaced by a point in the line that connects y^n and y^c, where

$$y = y^c + \delta(y^c - y^n),\ y^c = \frac{1}{n}\sum_{i=0}^{n-1} y^i,\ \ \delta \in R.$$

$\delta = 1$ indicates a reflection, $\delta = 2$ an expansion, $\delta = 1/2$ an outer contraction, and $\delta = -1/2$ an inside contraction. Nelder-Mead can also perform shrink. Except for the shrinks, the emphasis is on replacing the worse vertex rather than improving the best. The simplices generated by Nelder-Mead may adapt well to the curvature of the function.

In this chapter, a derivative-free iterative scheme is developed for (MOP). The idea of the Nelder-Mead simplex method is imposed in a modified form using the Non-dominated Sorting algorithm to solve (MOP). This algorithm is coded in MATLAB(2019) to generate the Pareto front. The efficiency of this algorithm is justified through a set of test problems, and comparison with a scalarization method and NSGA−II is provided in terms of the number of iterations and CPU time.

2.2 Notations and Preliminaries

Consider that $\mathbb{R}^m$ is partially ordered by a binary relation induced by $\mathbb{R}^m_+$, the non-negative orthant of $\mathbb{R}^m$. For $p, q \in \mathbb{R}^m$,

$$p \leqq_{\mathbb{R}^m_+} q \iff q - p \in \mathbb{R}^m_+;$$

$$p \preceq_{\mathbb{R}^m_+} q \iff q - p \in \mathbb{R}^m_+ \setminus \{0\};$$

$$p \prec_{\mathbb{R}^m_+} q \iff q - p \in int\,(\mathbb{R}^m_+).$$

Definition 2.1 A point $x^* \in S$ is called a *weak efficient solution* of the (MOP) if there does not exist $x \in S$ such that $F(x) \prec_{\mathbb{R}^m_+} F(x^*)$. In other words, whenever $x \in S,\ F(x) - F(x^*) \notin -int\,(\mathbb{R}^m_+)$. In set notation, this becomes $(F(S) - F(x^*)) \cap -int\,(\mathbb{R}^m_+) = \phi$.

Definition 2.2 A point $x^* \in S$ is called an *efficient solution* of the (MOP) if there does not exist $x \in S$ such that $F(x) \preceq_{\mathbb{R}^m_+} F(x^*)$. In other words, whenever $x \in S,\ F(x) - F(x^*) \notin -(\mathbb{R}^m_+ \setminus \{0\})$. In set notation, this becomes $(F(S) - F(x^*)) \cap (-(\mathbb{R}^m_+ \setminus \{0\})) = \phi$. This solution is also known as the Pareto optimal or non-inferior solution. If X^* is the set of all efficient solutions, then the set $F(X^*)$ is called the Pareto front for (MOP).

Definition 2.3 For $x_1,\ x_2 \in \mathbb{R}^n$, x_1 is said to *dominate* x_2 if and only if $F(x_1) \preceq_{\mathbb{R}^m_+} F(x_2)$, that is, $f_j(x_1) \leq f_j(x_2)$ for all j and $F(x_1) \neq F(x_2)$. x_1 *weakly dominates* x_2 if and only if $F(x_1) \prec_{\mathbb{R}^m_+} F(x_2)$, that is, $f_j(x_1) < f_j(x_2)$ for all j. A point $x_1 \in \mathbb{R}^n$ is said to be *non-dominated* if there does not exist any x_2 such that x_2 dominates x_1.

This concept can also be extended to find a non-dominated set of solutions of a multi-objective programming problem. Consider a set of N points $\{x_1, x_2, ..., x_N\}$, each having $m(> 1)$ objective function values. So $F(x_i) = (f_1(x_i), f_2(x_i), ..., f_m(x_i))$. The following algorithm from Deb [6] can be used to find the non-dominated set of points. This algorithm is used in the next section to order the objective values at every vertex of the simplex.

Algorithm 1[6]
Step 0 : Begin with $i = 1$.
Step 1 : For all $j \neq i$, compare solutions x_i and x_j for domination using Definition 2.3 for all m objectives.
Step 2 : If for any j, x_i is dominated by x_j, mark x_i as "dominated".
Step 3 : If all solutions (that is, when $i = N$ is reached) in the set are considered, go to **Step** 4, else increment i by one and go to **Step** 1.
Step 4 : All solutions that are not marked "dominated" are non-dominated solutions.

2.3 Gradient-Free Method for *MOP*

In this section, we develop a gradient-free algorithm as an extension of the Nelder-Mead simplex method, which is a widely used algorithm for the single-objective case. The dominance property as explained in Algorithm 1 helps to compare $F(x)$ at different points in $\mathbb{R}^m$.

Consider a simplex of $n+1$ vertices in $\mathbb{R}^n$ as $Y = \{y^0, y^1, ..., y^n\}$ ordered by component-wise increasing values of F. To order the vertices component-wise, one can use the "Non-dominated Sorting Algorithm 1". The most common Nelder-Mead iterations for the single-objective case perform a reflection, an expansion, or a contraction (the latter can be inside or outside the simplex). In such iterations, the worst vertex y^n is replaced by a point in the line that connects y^n and y^c,

$$y = y^c + \delta(y^c - y^n),\ \delta \in \mathbb{R},$$

where $y^c = \sum_{i=0}^{n-1} \frac{y^i}{n}$ is the centroid of the best n vertices. The value of δ indicates the type of iteration. For instance, when $\delta = 1$ we have a (genuine or isometric) reflection, when $\delta = 2$ an expansion, when $\delta = \frac{1}{2}$ an outside contraction, and when $\delta = -\frac{1}{2}$ an inside contraction. A Nelder-Mead iteration can also perform a simplex shrink, which rarely occurs in practice. When a shrink is performed, all the vertices in Y are thrown away except the best one y^0. Then n new vertices are computed by shrinking the simplex at y^0, that is, by computing, for instance, $y^0 + \frac{1}{2}(y^i - y^0)$, $i = 1, 2, ..., n$. Note that the "shape" of the resulting simplices can change by being stretched or contracted, unless a shrink occurs.

2.3.1 Modified Nelder-Mead Algorithm

Choose an initial point of vertices $Y_0 = \{y_0^0, y_0^1, ..., y_0^n\}$. Evaluate F at the points in Y_0. Choose constants:

$$0 < \gamma^s < 1, -1 < \delta^{ic} < 0 < \delta^{oc} < \delta^r < \delta^e.$$

For $k = 0, 1, 2, ...$, set $Y = Y_k$.

1. Order the $n+1$ vertices of $Y = \{y^0, y^1, ..., y^n\}$ using Algorithm 1 so that

$$F(y^0) \preceq_{\mathbb{R}_+^m} F(y^1) \preceq_{\mathbb{R}_+^m} ... \preceq_{\mathbb{R}_+^m} F(y^n).$$

Denote $F(y^t) = F^t, t = 0, 1, \ldots, n$

2. Reflect the worst vertex y^n over the centroid $y^c = \sum_{i=0}^{n-1} \frac{y^i}{n}$ of the remaining n vertices:

$$y^r = y^c + \delta(y^c - y^n)$$

Evaluate $F^r = F(y^r)$. If F^0 dominates F^r and F^r dominates weakly F^{n-1}, then replace y^n by the reflected point y^r and terminate the iteration:

$$Y_{k+1} = \{y^0, y^1, ..., y^r\}.$$

3. If F^r dominates weakly F^0, then calculate the expansion point

$$y^e = y^c + \delta^r(y^c - y^n)$$

and evaluate $F^e = F(y^e)$. If F^e dominates F^r, replace y^n by the expansion point y^e and terminate the iteration:

$$Y_{k+1} = \{y^0, y^1, ..., y^e\}.$$

Otherwise, replace y^n by the reflected point y^r and terminate the iteration:

$$Y_{k+1} = \{y^0, y^1, ..., y^r\}.$$

4. If F^{n-1} dominates F^r, then a contraction is performed between the best of y^r and y^n.

   (a) If F^r dominates weakly F^n, perform an outside contraction

$$y^{oc} = y^c + \delta^{oc}(y^c - y^n)$$

   and evaluate $F^{oc} = F(y^{oc})$. If F^{oc} dominates F^r, then replace y^n by the outside contraction point y^{oc} and terminate the iteration:

$$Y_{k+1} = \{y^0, y^1, ..., y^{oc}\}.$$

   Otherwise, perform a shrink.

   (b) If F^n dominates F^r, perform an inside contraction

$$y^{ic} = y^c + \delta^{ic}(y^c - y^n)$$

   and evaluate $F^{ic} = F(y^{ic})$. If F^{ic} dominates weakly F^n, then replace y^n by the inside contraction point y^{ic} and terminate the iteration:

$$Y_{k+1} = \{y^0, y^1, ..., y^{ic}\}.$$

   Otherwise, perform a shrink.

5. Evaluate f at the n points $y^0 + \gamma^s(y^i - y^0)$, $i = 1, ..., n$, and replace $y^1, ..., y^n$ by these points, terminating the iteration:

$$Y_{k+1} = y^0 + \gamma^s(y^i - y^0), i = 0, ..., n.$$

2.4 Numerical Illustrations and Performance Assessment

In this section, the algorithm is executed on some test problems, which are collected from different sources and summarized in Table 2.1 (see Appendix). The results obtained by Algorithm 2.3.1 are compared with the existing methods: the scalarization method (Weighted sum) and NSGA-II. MATLAB code (R2017b) for these three methods is developed. The comparison is provided in Table 2.2 (see Appendix). In this table, "Iter" corresponds to the number of iterations and "CPU time" corresponds to the time for executing the Algorithms. Denote the algorithms in short term as

Algorithm 2.3.1—(NMSM)
Weighted Sum Method—(WSM)
NSGA-II

Pareto front: To generate the Pareto front by Algorithm 2.3.1, the **RAND** strategy is considered for selecting the initial point. 500 uniformly distributed random initial points between lower bound and upper bound are selected. Every test problem is executed 10 times with random initial points. The Pareto front of the test problem "BK1" in NSGA-II, NMSM, and WSM is provided in Fig. 2.1 with red, green, and blue stars, respectively.

2.5 Performance Profile

Performance profile is defined by a cumulative function $\rho(\tau)$ representing a performance ratio with respect to a given metric, for a given set of solvers. Given a set of solvers S and a set of problems P, let $\zeta_{p,s}$ be the performance of solver s on solving problem p. The performance ratio is then defined as $r_{p,s} = \zeta_{p,s}/\min_{s \in S} \zeta_{p,s}$. The cumulative function $\rho_s(\tau)$ $(s \in S)$ is defined as

$$\rho_s(\tau) = \frac{|\{p \in P : r_{p,s} \leq \tau\}|}{|P|}.$$

It is observed that the performance profile is sensitive to the number and types of algorithms considered in the comparison; see [16]. So the algorithms are compared pairwise. In this chapter, the performance profile is compared using purity, Γ, Δ spread metrics.

Purity metric: Let $P_{p,s}$ be the approximated Pareto front of problem p obtained by method s. Then an approximation to the true Pareto front P_p can be built by considering $\underset{s\in S}{\cup} P_{p,s}$ first and removing the dominated points. The purity metric for algorithms s and problem p is defined by the ratio

$$\bar{t_{p,s}} = |P_{p,s} \cap P_p|/|P_{p,s}|.$$

Clearly, $\bar{t_{p,s}} \in [0, 1]$. When computing the performance profiles of the algorithms for the purity metric, it is required to set $t'_{p,s} = 1/\bar{t_{p,s}}$. $t' = 0$ implies that the algorithm is unable to solve p.

Spread metrics: Two types of spread metrics (Γ and Δ) are used in order to analyze if the points generated by an algorithm are well-distributed in the approximated Pareto front of a given problem. Let $x_1, x_2, \ldots, x_N$ be the set of points obtained by a solver s for problem p and let these points be sorted by objective function j, that is, $f_j(x_i) \leq f_j(x_{i+1})$ $(i = 1, 2, \ldots, N-1)$. Suppose x_0 is the best known approximation of global minimum of f_j and x_{N+1} is the best known global maximum of f_j, computed over all approximated Pareto fronts obtained by different solvers. Define

$$\delta_{i,j} = f_j(x_{i+1}) - f_j(x_i).$$

Then Γ spread metric is defined by

$$\Gamma_{p,s} = \max_{j\in\Lambda_m} \max_{i\in\{0,1,\ldots,N\}} \delta_{i,j}.$$

Define δ_j to be the average of the distances $\delta_{i,j}$, $i = 1, 2, \ldots, N-1$. For an algorithm s and a problem p, the spread metric Δ is

$$\Delta_{p,s} = \max_{j\in\Lambda_m} \frac{\delta_{0,j} + \delta_{N,j} + \sum_{i=1}^{N-1} |\delta_{i,j} - \bar{\delta_j}|}{\delta_{0,j} + \delta_{N,j} + (N-1)\bar{\delta_j}}.$$

Result Analysis: A deep insight into Figs. 2.2, 2.3, and 2.4 clearly indicates the advantage of the proposed method (NMSM) to the existing methods WSM and NSGA−II. In RAND, NMSM has a better performance ratio in the Γ metric than WSM and NSGA−II and purity and δ metrics than NSGA−II in most of the test problems. Also from the computational details tables, one may observe that NMSM takes less number of iterations and time than WSM and NSGA−II in most of the test problems.

2.6 Conclusions

In this chapter, a Nelder-mead simplex method is developed for solving unconstrained multi-objective optimization problems. This method is modified from the existing Nelder-mead simplex method for single-objective optimization problems. Justification of this iterative process is carried out through numerical computations. This chapter can be further studied for the constrained multi-objective programming problem and for the better spreading technique to generate the Pareto points, which can be considered as the future scope of the present contribution.

Acknowledgements We thank the anonymous reviewers for the valuable comments that greatly helped to improve the content of this chapter.

Appendix

Table 2.1 Multi-objective test problems

Problem	Source	Problem	Source	Problem	Source
BK1	[17]	Jin2	[20]	TKLY1	[26]
Deb41	[6]	Jin3	[20]	LE1	[17]
Deb513	[6]	Jin4	[20]	I1	[17]
Deb521aa	[6]	lovison1	[22]	Far1	[17]
Deb521b	[6]	lovison2	[22]	SK1	[17]
DG01	[17]	lovison3	[22]	SK2	[17]
ex005	[18]	lovison4	[22]	SP1	[17]
ZDT3	[28]	LRS1	[17]	SSFYY1	[17]
Fonseca	[14]	MOP2	[17]	SSFYY2	[17]
Deb53	[9]	MHHM2	[17]	VFM1	[17]
GE5	[9]	MLF1	[17]	ZDT1	[28]
IKK1	[17]	MLF2	[17]	VU1	[17]
ZDT2	[28]	SCH1	[17]	VU2	[17]
Jin1	[20]	MOP1	[17]	KW2	[9]
OKA1	[25]	MOP3	[17]	MOP7	[17]
OKA2	[25]	MOP5	[17]	ZDT4	[28]
QV1	[17]	MOP6	[17]	CEC09_1	[3]
CEC09_2	[3]	CEC09_3	[3]	CEC09_7	[3]
CEC09_8	[3]	CEC09_10	[3]	Deb512a	[6]
Deb512b	[6]	Deb512c	[6]	DTLZ1	[6]
DTLZ1n2	[6]	DTLZ2	[6]	DTLZ2n2	[6]
DTLZ3	[6]	DTLZ3n2	[6]	DTLZ4	[6]
DTLZ4n2	[6]	DTLZ5_a	[6]	DTLZ5n2_a	[6]
DTLZ6	[6]	DTLZ6n2	[6]	FES1	[17]
FES2	[17]	FES3	[17]	IM1	[17]

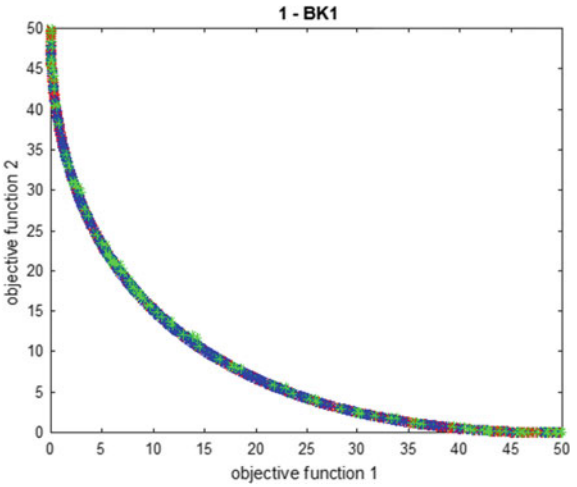

Fig. 2.1 Pareto front of BK1 for Weighted Sum, Nelder-Mead Simplex, and NSGA−II

Table 2.2 Computation details

Problem	NMSM		WSM		NSGA−II	
	Iter	CPU time	Iter	CPU time	Iter	CPU time
BK1	100	175.7102	1000	17.3119	1000	96.1962
CEC09_1	40	40.8337	39185	85.1831	1000	125.0616
CEC09_2	15	14.7780	29659	62.3473	1000	149.2667
CEC09_3	30	43.5034	17463	47.0575	1000	261.1938
CEC09_7	50	58.12	66763	115.43	1000	246.4144
CEC09_8	10	39.5513	11881	24.811	1000	414.9251
CEC09_10	1	7.1234	26521	61.1016	500	424.4517
Deb41	225	46.961	3451	13.1149	200	54.7721
Deb53	1	3.148	4779	16.182	1000	226.228
Deb512a	95	30.8398	6831	16.2914	1000	67.5017
Deb512b	1	1.6304	5665	22.007	500	236.7293
Deb512c	1	1.3482	4169	13.0784	1000	105.0796
Deb513	100	286.7849	1105	8.2348	500	233.5419
Deb521a	200	40.6678	2074	16.2494	500	178.9993
Deb521b	200	34.7894	1832	9.5663	500	86.927
DG01	100	383.6649	3527	10.0337	500	114.6849
DTLZ1	1	8.0783	829	11.3604	500	604.637
DTLZ1n2	50	51.4517	1533	10.5865	500	329.7814

(continued)

Table 2.2 (continued)

Problem	NMSM		WSM		NSGA−II	
	Iter	CPU time	Iter	CPU time	Iter	CPU time
DTLZ2	1	9.1068	4384	12.1977	500	201.0205
DTLZ2n2	30	92.2633	1705	9.2817	500	90.3009
DTLZ3	1	8.5647	546	11.3052	500	180.3397
DTLZ3n2	25	59.364	1489	12.3074	500	837.8227
DTLZ5	5	37.5631	2361	11.2001	500	263.9273
DTLZ5n2	4	19.6248	2417	10.2009	500	277.3301
DTLZ6	100	40.4558	3696	13.6023	500	159.34
DTLZ6n2	150	266.6435	2416	9.9722	500	186.7189
ex005	2	8.6635	2066	9.0123	500	280.0558
Far1	50	54.4216	6633	21.8326	500	122.3895
hline FES1	65	211.8147	22021	69.7231	500	293.3888
FES2	1	3.6761	24335	80.9286	500	146.9458
FES3	1	3.6302	26324	96.1848	500	142.7173
Fonseca	50	83.6762	3872	11.3395	500	95.5476
GE5	1	5.932	2412	9.7671	500	433.6827
I1	4	9.8997	1610	18.8042	500	115.2193
IKK1	1	2.9279	1539	8.1619	500	95.1506
IM1	1	4.971	2353	9.4475	500	185.6666
Jin1	50	35.5375	1091	7.9041	500	83.9951
Jin2	28	5.6778	3834	11.2386	500	152.2696

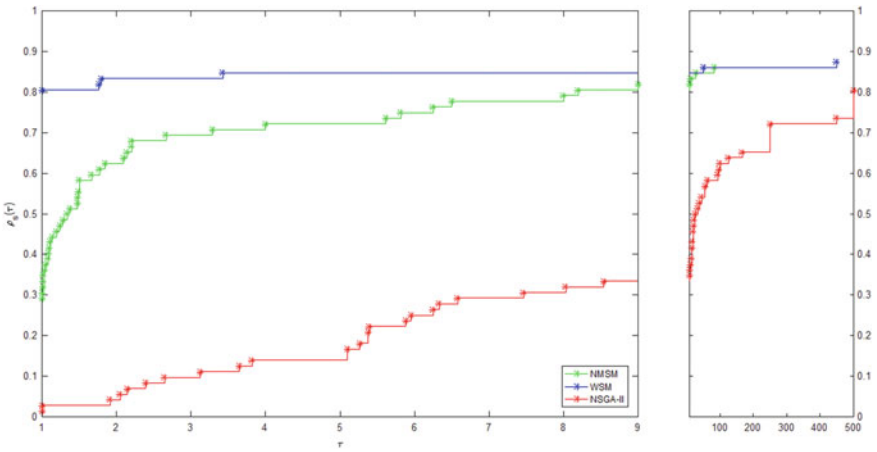

Fig. 2.2 Purity performance profile between NMSM,WSM, and NSGA-II

Table 2.3 Computation details continued

Problem	NMSM		WSM		NSGA−II	
	Iter	CPU time	Iter	CPU time	Iter	CPU time
Jin3	100	15.7487	1565	8.3628	500	196.8463
Jin4	150	24.6867	2815	10.8569	500	111.2916
KW2	50	89.2429	4885	15.7046	500	118.0532
LE1	50	240.3437	9798	19.4636	500	182.9293
Lovison1	1	2.968	1752	9.9745	500	111.5821
Lovison2	150	25.9414	2316	10.4402	500	207.9985
Lovison3	1	2.5782	2389	10.3237	500	206.382
Lovison4	1	4.0833	1949	10.1973	500	279.6307
LRS1	50	336.4876	1002	7.7188	500	142.3477
MHHM2	3	11.0745	3681	11.4456	500	344.4489
MLF1	50	122.7622	1927	9.1655	500	104.0309
MLF2	2	8.4472	4347	11.103	500	358.7373
MOP1	1	5.2652	1006	8.604	500	119.9713
MOP2	50	154.4144	3373	10.4286	500	99.4147
MOP3	2	4.551	4403	12.7004	500	95.9286
MOP5	1	1.2941	4007	10.4313	500	113.9832
MOP6	45	117.3724	1111	8.6546	500	278.6248
MOP7	1	5.1297	3765	12.0097	500	166.5825
OKA1	60	80.5031	5412	17.0589	500	118.3507
OKA2	50	33.6682	5753	16.9394	83	19.2191
QV1	55	251.9818	962	11.4491	500	76.7295
SCH1	50	61.1843	1387	9.228	500	97.6939
SK1	50	53.5915	2171	9.5896	500	98.0009
SK2	50	164.0477	2202	9.9194	500	146.5663
SP1	75	164.2625	3035	10.2907	500	133.1876
SSFYY1	100	323.6209	1002	7.9782	500	91.426
SSFYY2	50	186.4307	2745	10.4761	500	106.6671
TKLY1	55	29.4778	3106	10.9727	500	108.0413
VFM1	1	5.7106	1000	8.5332	500	121.1181
VU1	100	31.6459	1802	8.9375	500	138.3799
VU2	1	3.3975	2116	9.6255	500	400.1829
ZDT1	100	64.8696	4584	19.4772	500	252.9595
ZDT2	100	49.0452	1967	10.6646	500	361.0973
ZDT3	50	44.126	4499	13.469	500	480.667
ZDT4	50	83.5181	8527	19.4524	100	836.2467

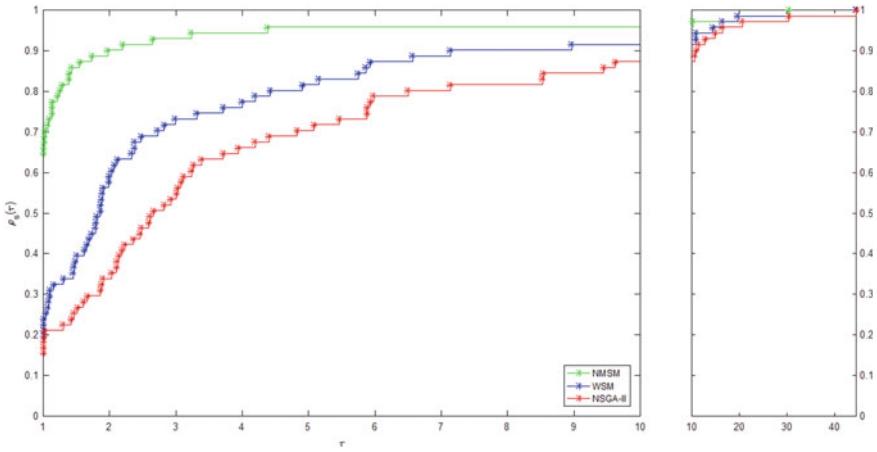

Fig. 2.3 Γ performance profile between NMSM,WSM, and NSGA-II

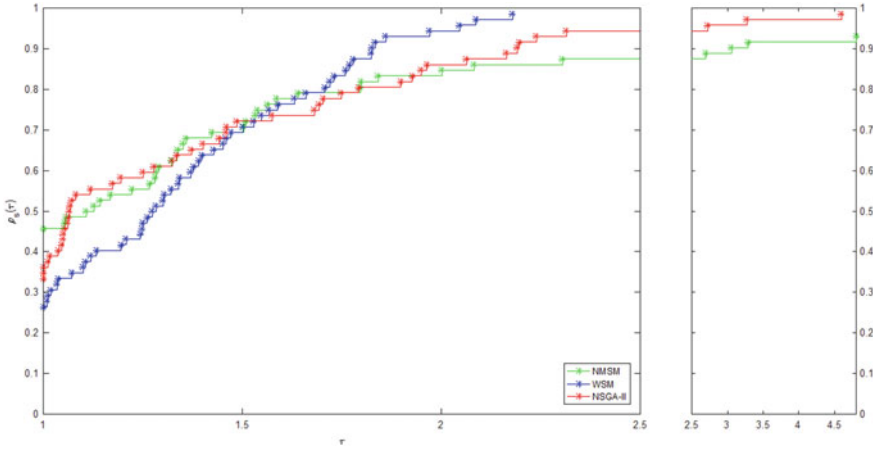

Fig. 2.4 Δ performance profile between NMSM,WSM, and NSGA-II

References

1. Ansari, Q.H., Köbis, E., Yao, J.C.: Vector Variational Inequalities and Vector Optimization. Springer (2018)
2. Ansary, M.A.T., Panda, G.: A modified quasi-newton method for vector optimization problem. Optimization **64**, 2289–2306 (2015)
3. Ansary, M.A.T., Panda, G.: A sequential quadratically constrained quadratic programming technique for a multi-objective optimization problem. Eng. Optim. **51**(1), 22–41 (2019). https://doi.org/10.1080/0305215X.2018.1437154
4. Ansary, M.A.T., Panda, G.: A sequential quadratic programming method for constrained multi-objective optimization problems. J. Appl. Math. Comput. (2020). https://doi.org/10.1007/s12190-020-01359-y

5. Conn, A.R., Scheinberg, K., Vicente, L.N.: Introduction to Derivative-Free Optimization. SIAM (2009)
6. Deb, K.: Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. Wiley India Pvt. Ltd., New Delhi, India (2003)
7. Deb, K.: Multi-objective Optimization Using Evolutionary Algorithms. Wiley India Pvt. Ltd., New Delhi, India (2003)
8. Ehrgott, M.: Multicriteria Optimization. Springer Publication, Berlin (2005)
9. Eichfelder, G.: An adaptive scalarization method in multiobjective optimization. SIAM J. Optim. **19**(4), 1694–1718 (2009)
10. Engau, A., Wiecek, M.M.: Generating ϵ-efficient solutions in multiobjective programming. Eur. J. Oper. Res. **177**, 1566–1579 (2007)
11. Fliege, F., Drummond, L.M.G., Svaiter, B.: Newton's method for multiobjective optimization. SIAM J. Optim **20**(2), 602–626 (2009)
12. Fliege, J., Svaiter, F.V.: Steepest descent methods for multicriteria optimization. Math. Methods Oper. Res. **51**(3), 479–494 (2000)
13. Fliege, J., Vaz, A.I.F.: A method for constrained multiobjective optimization based on SQP techniques. SIAM J. Optim. **26**(4), 2091–2119 (2016)
14. Fonseca, C.M., Fleming, P.J.: Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation. IEEE Trans. Syst. Man Cybern. Part A: Syst. Hum. **28**(1), 26–37 (1998)
15. Geofforion, A.M.: Proper efficiency and the theory of vector maximization. J. Optim. Theory Appl. **22**, 618–630 (1968)
16. Gould, N., Scott, J.: A note on performance profiles for benchmarking software. ACM Trans. Math. Softw. **43**, 15 (2016)
17. Huband, S., Hingston, P., Barone, L., While, L.: A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans. Evol. Comput. **10**(5), 477–506 (2006)
18. Hwang, C.L., Yoon, K.: Multiple Attribute Decision Making: Methods and Applications a State-of-the-Art Survey. Springer Science and Business Media (1981)
19. Jahn, J.: Vector Optimization. Theory, Applications, and Extensions. Springer, Berlin (2004)
20. Jin, Y., Olhofer, M., Sendhoff, B.: Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how? pp. 1042–1049 (2001)
21. Li, Z., Wang, S.: ϵ-approximate solutions in multiobjective optimization. Optimization **34**, 161–174 (1998)
22. Lovison, A.: Singular continuation: generating piecewise linear approximations to pareto sets via global analysis. SIAM J. Optim. **21**(2), 463–490 (2011)
23. Miettinen, K.M.: Nonlinear Multiobjective Optimization. Kluwer, Boston (1999)
24. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965). https://doi.org/10.1093/comjnl/7.4.308
25. Okabe, T., Jin, Y., Olhofer, M., Sendhoff, B.: On test functions for evolutionary multi-objective optimization. In: International Conference on Parallel Problem Solving from Nature, pp. 792–802. Springer (2004)
26. Preuss, M., Naujoks, B., Rudolph, G.: Pareto set and EMOA behavior for simple multimodal multiobjective functions, pp. 513–522 (2006)
27. Qu, S., Goh, M., Chan, F.T.S.: Quasi-newton methods for solving multiobjective optimization. Oper. Res. Lett. **39**, 397–399 (2011)
28. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: empirical results. Evol. Comput. **8**(2), 173–195 (2000)