UNIVERSITY OF PISA

MASTER'S DEGREE IN
ARTIFICIAL INTELLIGENCE AND DATA ENGINNERING

Symbolic and Evolutionary Artificial Intelligence

On the Impact of Scalarization in MOEA/D: Performance across Convex, Non-Convex, and Disconnected Pareto Fronts

Professor:
Marco Cococcioni

Student:
Giuseppe Soriano

GitHub Repository
https://github.com/GiuseppeSoriano/SEAI_Project

ACADEMIC YEAR 2024/2025

Index

1. Introduction

Multi-objective optimization involves the simultaneous optimization of multiple, often conflicting, objective functions. In this context, the goal is not to identify a single optimal solution, but rather to approximate the set of Pareto-optimal solutions, each representing a different trade-off among the objectives. Evolutionary algorithms are widely adopted for this task due to their population-based nature, which allows exploration of diverse regions of the solution space in a single run.

Among the most studied approaches is the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D), which decomposes a multi-objective problem into a set of scalar optimization subproblems. Each subproblem is optimized simultaneously, typically using information from its neighboring subproblems. A key component of MOEA/D is the scalarization function, which transforms the multi-objective formulation into scalar ones. The most common scalarization techniques include the **weighted sum**, the **Chebyshev (Tchebycheff) approach**, and **boundary intersection methods**.

However, it is well known that the performance of scalarization techniques can vary significantly depending on the geometry of the Pareto front. In particular, convex scalarizations such as the weighted sum tend to perform poorly on non-convex Pareto fronts, potentially failing to identify large regions of the solution set. This motivates an empirical investigation into the behavior of different scalarization functions when applied to problems with non-convex Pareto fronts.

This work presents a comparative analysis of MOEA/D configured with two distinct scalarization strategies—Chebyshev and linear (weighted sum)—in contrast with the baseline performance of NSGA-II, a widely adopted dominance-based algorithm that does not rely on scalarization. The objective is to evaluate the strengths and limitations of each approach in addressing diverse Pareto front geometries, with a particular focus on how scalarization choices influence convergence, diversity, and overall solution quality in multi-objective optimization contexts. The comparison also includes a variant of MOEA/D where the traditional mutation operator is complemented by the Simulated Binary Crossover (SBX), aiming to investigate the impact of crossover-based genetic variation on the algorithm's effectiveness.

2. Background and Related Work

Multi-objective optimization deals with problems where two or more conflicting objectives must be optimized simultaneously. Instead of a single optimum, the outcome is a set of **Pareto-optimal solutions**, none of which can improve one objective without worsening another. The image of all Pareto-optimal objective vectors is called the **Pareto front**. Numerous algorithms, broadly termed **multi-objective evolutionary algorithms (MOEAs)**, have been developed to approximate the Pareto front. These algorithms can be categorized into different families based on how they balance the objectives:

- **Pareto dominance-based MOEAs:** Rely on the Pareto dominance relation to guide the search (e.g., NSGA-II, SPEA2, PAES).
- **Indicator-based MOEAs:** Use performance indicators (like hypervolume) to drive selection (e.g., SMS-EMOA, HypE).
- **Decomposition-based MOEAs:** Decompose the multi-objective problem into many single-objective subproblems (e.g., MOEA/D, MOGLS).

In this work, we focus on two representative algorithms: **NSGA-II** (a Pareto-based approach) and **MOEA/D** (a decomposition-based approach). These two methods are foundational in the literature and well-suited to highlight the contrast between dominance-based and decomposition-based strategies. Below, we detail the mechanics of each algorithm, including their mathematical underpinnings, and reference the seminal works where they were introduced.

2.1 NSGA-II: Non-dominated Sorting Genetic Algorithm II

NSGA-II is one of the most influential Pareto-based MOEAs, introduced by Deb *et al.* in 2002 [1] as an improvement over the earlier NSGA algorithm. NSGA-II's design addressed several shortcomings of first-generation MOEAs and became a standard baseline for multi-objective optimization. Key features of NSGA-II include:

- **Elitism:** It preserves the best solutions found so far by always keeping non-dominated individuals from generation to generation, rather than relying on an external archive.
- **Fast non-dominated sorting:** NSGA-II ranks individuals by Pareto dominance efficiently, without requiring any user-defined niche radius (unlike its predecessor).
- **Crowding distance diversity:** It uses a **crowding distance** metric to maintain solution diversity along the Pareto front, eliminating the need for a sharing parameter.
- **Simple parameter set:** NSGA-II does not introduce new parameters beyond the standard genetic algorithm ones (population size, crossover/mutation rates, etc.), making it easy to implement and tune.

Pareto Ranking (Non-dominated Sorting): In NSGA-II, the population is sorted into layers (Front 1, Front 2, etc.) based on dominance. Front 1 consists of all **non-dominated** individuals (Pareto-optimal within the population). Front 2 contains individuals that are dominated only byz individuals in Front 1, and so on. Each solution is assigned a rank (1 for the first front, 2 for the second, etc.). This ranking guides selection: lower-rank (better) fronts are prioritized for survival.

Crowding Distance: Within each Pareto front, NSGA-II computes a crowding distance for each solution to estimate how isolated it is from its neighbors in objective space. This promotes a spread-out Pareto front. The crowding distance for a solution is the sum of normalized objective function differences between its nearest neighbors on that front. For example, if solutions in a given front are sorted by objective m, the crowding distance d_i for solution i can be calculated as:

$$d_i \;=\; \sum_{m=1}^{M} \frac{f_m(i+1) - f_m(i-1)}{f_m^{\max} - f_m^{\min}} \,,$$

where $f_m(i+1)$ and $f_m(i-1)$ are the m-th objective values of the neighboring solutions in the sorted order, and $f_m^{\max}$ and $f_m^{\min}$ are the maximum and minimum values of the m-th objective in that front. Boundary solutions (highest and lowest value in each objective) are assigned an infinite crowding distance to always preserve extreme trade-offs. A larger d_i means solution i resides in a less crowded region, which is desirable for maintaining diversity.

Selection and Elitist Survival: NSGA-II uses a combined population approach each generation. Given a parent population P_t of size N and an offspring population Q_t of size N (produced by crossover and mutation), NSGA-II proceeds as follows to form the next generation P_{t+1}:

1. **Combine Populations:** Form $R_t = P_t \cup Q_t$ (of size $2N$) and perform non-dominated sorting on R_t to identify fronts $f_1, f_2, \ldots$.
2. **Rank-based Selection:** Initialize $P_{t+1} = \emptyset$. Fill P_{t+1} with the solutions from f_1, then f_2, etc., until adding another entire front would exceed N. Let f_k be the last front that *partially* fits.
3. **Diversity Selection:** If P_{t+1} has fewer than N solutions after filling whole fronts up to f_{k-1}, sort the solutions in f_k by descending crowding distance and select the most widely spaced solutions to fill the remaining slots (up to N). This ensures a well-distributed selection from the last included front.
4. **Next Generation:** Discard the rest and set P_{t+1} as the new parent population. Then generate Q_{t+1} from P_{t+1} via genetic operators and repeat.

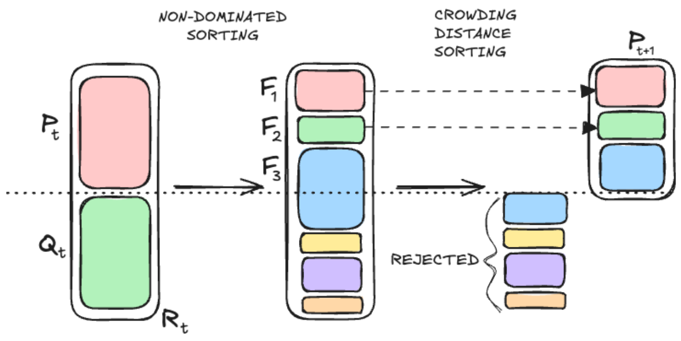


Figure 1: Schematic representation of the NSGA-II algorithm.

By always preferring lower-rank (non-dominated) solutions and secondarily those in less crowded regions, NSGA-II strikes a balance between **convergence** (favoring Pareto-optimal solutions) and **diversity** (covering the Pareto front evenly). Deb *et al.* report that these mechanisms allow NSGA-II to find a well-spread set of Pareto-optimal solutions in a single run [1]. Since its introduction, NSGA-II has been widely adopted as a benchmark and has inspired many subsequent algorithms in the Pareto-based MOEA family (including extensions like NSGA-III for many-objective problems).

2.2 MOEA/D: Multi-Objective Evolutionary Algorithm based on Decomposition

MOEA/D was proposed by Zhang and Li in 2007 [2] as a new paradigm for MOEAs based on **problem decomposition**. Instead of relying on Pareto dominance comparisons, MOEA/D decomposes a multi-objective problem into many scalar optimization subproblems and solves them concurrently. The algorithm maintains a population of N individuals, each associated with a distinct **weight vector** $\lambda^{(i)} = (\lambda_1, \ldots, \lambda_M)$ (where M is the number of objectives). These weight vectors define different trade-off directions, effectively specifying N scalarized objective functions.

Scalarization Functions: A core idea in MOEA/D is that any Pareto-optimal solution can be optimal for some scalarization of the objectives. Common scalarization strategies used in MOEA/D include:

- *Weighted Sum:* Each subproblem optimizes a weighted linear combination of objectives. The scalar objective is $g^{WS}(x \mid \lambda) = \sum_{i=1}^{M} \lambda_i f_i(x)$, with $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$. The weight vector λ emphasizes certain objectives over others. Weighted sum is simple, but it can only attain solutions on **convex** sections of the Pareto front; if the true Pareto front is concave (non-convex), some Pareto-optimal solutions will never be the minimum of any linear combination of objectives (they are unreachable by weighted sum optimization).
- *Tchebycheff (Chebyshev) approach:* Each subproblem minimizes the maximum weighted deviation from the ideal objective values. Let $\mathbf{z}^* = (z_1^*, \ldots, z_M^*)$ be the **ideal point** (best attainable value for each objective, often the minimum of $f_i(x)$ in the current population). The Chebyshev scalarization is: $g^{Cheb}(x \mid \lambda, \mathbf{z}^*) = \max_{1 \leq i \leq M}\{\, \lambda_i \,|\, f_i(x) - z_i^* \,|\, \}$. Each solution is evaluated by its worst (relative) objective performance, with weights λ_i. By adjusting λ, this formulation can target different parts of the Pareto front. An important property is that **for any Pareto-optimal solution, there exists a weight vector** λ (often in combination with an appropriate z^*) **that makes that solution optimal for the Chebyshev metric**. Thus, unlike the weighted sum, the Chebyshev approach can discover **non-convex Pareto fronts** (including concave regions) given a well-distributed set of weight vectors. This is a major motivation for using Chebyshev (and related boundary intersection methods) in MOEA/D when tackling problems like ZDT2 which have a concave front.

MOEA/D works by assigning each weight vector $\lambda^{(i)}$ to one individual in the population, which is intended to converge to a Pareto-optimal solution favoring that weight profile. A notion of **neighborhood** in the weight space is used: for each weight vector, a set of T closest weight vectors (usually in Euclidean distance) is defined as its neighbors. The subproblems corresponding to neighboring weight vectors are expected to have optimal solutions in nearby regions of objective space. Therefore, MOEA/D encourages **collaboration among neighboring subproblems** by mainly allowing genetic variation and solution replacement within those neighborhoods.

Algorithm Outline: Initially, a set of N weight vectors $\{\lambda^{(1)}, \ldots, \lambda^{(N)}\}$ is chosen (e.g., uniformly spread over the objective simplex for balanced coverage). The initial population $\{x^{(1)}, \ldots, x^{(N)}\}$ is typically generated at random or by some initialization heuristic, and the **ideal point** $\mathbf{z}^*$ is set to the component-wise minima of the objective values found. MOEA/D then evolves the population for a number of generations. At each generation, for each subproblem $i = 1, \ldots, N$:

1. **Mating Selection:** Randomly pick a few indices from the neighbor set $B(i)$ of subproblem i (which typically includes i itself and its T nearest weight indices). Using the corresponding parent solutions, generate an **offspring** solution (call it y) by applying crossover and mutation.
2. **Update Ideal Point:** Evaluate $f(y) = (f_1(y), \ldots, f_M(y))$. Update the ideal point z^* such that for each objective j: if $f_j(y) < z_j^*$, then set $z_j^* = f_j(y)$ (i.e. record any new best objective values found).
3. **Solution Replacement:** For each subproblem j in the neighbor set $B(i)$ (including i itself), **compare the new solution y to the current solution** $x^{(j)}$ using *j's scalarization function.* That is, compute the scalarized objective $g(x^{(j)} \mid \lambda^{(j)}, \mathbf{z}^*)$ and $g(y \mid \lambda^{(j)}, \mathbf{z}^*)$. If y is *better* (lower) in this scalar value, then replace $x^{(j)}$ with y as the new solution for subproblem j.

By restricting replacement to a subproblem's neighborhood, MOEA/D maintains diversity across different weight vectors while still allowing good solutions to propagate locally. Each subproblem searches in a semi-isolated manner, but the **sharing of offspring among neighbors** helps disseminate useful genetic material to nearby regions of the Pareto front. Notably, MOEA/D's computational cost per generation is relatively low, since each offspring is compared only with a limited set of neighbors rather than the entire population. Indeed, Zhang and Li note that MOEA/D has **lower per-generation complexity than NSGA-II**, which involves $O(N^2)$ sorting operations. Empirical results in the original MOEA/D paper showed that this algorithm could outperform or at least equal NSGA-II in various test problems, especially as the number of objectives grows [2].

Weight Vector Design and Scalarization Choice: The performance of MOEA/D heavily depends on a good spread of weight vectors and an appropriate scalarization method for the problem at hand. If the Pareto front is convex, a uniform spread of weight vectors with the simple weighted sum may suffice to approximate the front. For problems with **non-convex fronts (e.g., ZDT2)**, using the Chebyshev approach (or other advanced methods like Penalty Boundary Intersection) is beneficial because it can locate solutions in concave regions that weighted sum would miss. In practice, MOEA/D can incorporate various scalarization techniques; the chosen two in this work (linear weighted sum and Chebyshev) provide a clear contrast in how well they handle concave trade-offs. The **Chebyshev MOEA/D** is expected to cover the Pareto front more completely on non-convex problems, whereas **Weighted-Sum MOEA/D** may struggle to find evenly distributed solutions in the non-convex middle part of the front – this is precisely the issue under investigation.

MOEA/D has become a cornerstone in multi-objective optimization research. Its modular framework allows hybridization and extension; for example, researchers have studied alternative ways to update neighbors, dynamic weight adjustment, or embedding local search. The original MOEA/D formulation by Zhang and Li (2007) [2] demonstrated that even a straightforward decomposition (using fixed weights and basic genetic operators) is highly effective. That work opened up a new line of research into decomposition-based MOEAs, distinguishing MOEA/D as a **milestone algorithm** comparable to NSGA-II in its influence.

3. Experimental Setup

The experimental setup was designed to perform a rigorous comparison between NSGA-II and MOEA/D algorithms. Particular care was taken to maintain architectural consistency between the implementations in order to ensure fairness and modularity. Specifically, the MOEA/D algorithm was implemented from scratch by the author, reusing the structural conventions and modular decomposition adopted in the NSGA-II codebase provided by Prof. Marco Cococcioni. This included consistent interfaces, modular function decomposition (e.g., objective evaluation, initialization, crossover, mutation), and data structures for individuals and populations.

3.1 Benchmark Problems

To evaluate the performance of the algorithms under study, three well-established benchmark problems from the ZDT family have been selected. All of them are formulated as bi-objective minimization tasks defined over the domain $[0, 1]^n$, with $n = 30$ decision variables. Despite their similar structure, each problem exhibits a distinct shape of the Pareto front, allowing the investigation of how different algorithms and scalarization strategies cope with varying levels of complexity in multi-objective optimization.

The true Pareto fronts for all problems were generated analytically by sampling 1,000 uniformly spaced solutions in the decision space according to known conditions.

ZDT1

ZDT1 defines a convex Pareto front and is commonly used to assess the ability of algorithms to maintain a uniform spread of solutions across a smooth and continuous front. The problem is defined as follows:

$$\begin{aligned} f_1(x) &= x_1, \\ g(x) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1}, \\ f_2(x) &= g(x) \cdot \left(1 - \sqrt{\frac{x_1}{g(x)}}\right). \end{aligned}$$

The optimal front corresponds to the condition $x_1 \in [0, 1]$ and $x_i = 0$ for $i = 2, \ldots, n$.

ZDT2

ZDT2 presents a non-convex Pareto front, making it particularly useful for evaluating the ability of algorithms to approximate regions that cannot be reached via simple linear combinations of objectives. The formulation is given by:

$$\begin{aligned} f_1(x) &= x_1, \\ g(x) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1}, \\ f_2(x) &= g(x) \cdot \left(1 - \left(\frac{x_1}{g(x)}\right)^2\right). \end{aligned}$$

As in ZDT1, the Pareto-optimal set is obtained when $x_1 \in [0, 1]$ and all other variables are zero.

ZDT3

ZDT3 introduces an additional layer of complexity by featuring a **disconnected Pareto front**, composed of multiple non-contiguous segments. This structure challenges algorithms to ensure diversity across several isolated regions. The problem is defined as:

$$\begin{aligned} f_1(x) &= x_1, \\ g(x) &= 1 + 9 \cdot \frac{\sum_{i=2}^{n} x_i}{n-1}, \\ f_2(x) &= g(x) \cdot \left(1 - \sqrt{\frac{x_1}{g(x)}} - \frac{x_1}{g(x)} \cdot \sin(10\pi x_1)\right). \end{aligned}$$

The optimal solutions satisfy $x_1 \in [0, 1]$, $x_i = 0$ for $i = 2, \dots, n$, and lie on a front with five disconnected segments.

These three problems collectively cover a range of Pareto front geometries—convex, non-convex, and discontinuous—providing a robust basis for performance comparison.

3.2 Algorithmic Structure and Design Choices

NSGA-II

The NSGA-II implementation provided by the instructor was used without conceptual modification. This version follows the canonical structure proposed by Deb et al. (2002), with the following components:

- **Fast Non-Dominated Sorting**: Used to classify the population into Pareto fronts.
- **Crowding Distance**: Used to ensure diversity within each front.
- **Binary Tournament Selection**: Based on rank and crowding distance.
- **Simulated Binary Crossover (SBX)** and **Polynomial Mutation**: Used to generate offspring.
- **Elitist Replacement**: Parent and offspring populations are merged and sorted; the best N individuals are retained.

MOEA/D

This MOEA/D implementation strictly follows the decomposition-based optimization paradigm as described in the original MOEA/D paper. Its main components are:

- **Weight Vectors (λ_i)**: Each subproblem corresponds to a different weight vector in the objective space, uniformly generated for the bi-objective case.
- **Neighborhood Structure ($B(i)$)**: For each subproblem i, a neighborhood of $T = 10$ nearby subproblems is defined using Euclidean distance between weight vectors.
- **Ideal Point (z^*)**: Represents the best known objective values, updated throughout the evolution.
- **Scalarization**: Two strategies were employed:
  - **Chebyshev Scalarization**:

$$\text{Cheby}(f, z, \lambda) = \max_j \lambda_j \cdot |f_j - z_j|$$

  - **Linear Scalarization**:

$$\text{Linear}(f, z, \lambda) = \sum_j \lambda_j \cdot |f_j - z_j|$$

- **Variation Operators**: Polynomial mutation applied to a parent selected from the neighborhood $B(i)$.
- **Localized Replacement**: Offspring may replace neighboring solutions if they improve the scalarized objective with respect to their associated weight vector.
- **Archive Management**: An archive of non-dominated solutions of size N is maintained and updated at each generation. If the archive exceeds size N, solutions are removed based on crowding distance to preserve diversity. The choice of setting the archive size to N was made to ensure a fair comparison with NSGA-II, which does not use an external archive. However, if one wishes to experiment with an unbounded archive, as originally proposed in the official paper, it suffices to set `N_max` $= +\infty$, in which case no pruning will occur.

MOEA/D with Crossover

This variant extends the base MOEA/D algorithm by incorporating crossover in the variation stage, while still adhering strictly to the original MOEA/D formulation.

- **Weight Vectors (λ_i)**: Same as in MOEA/D.
- **Neighborhood Structure ($B(i)$)**: Same as in MOEA/D.
- **Ideal Point (z^*)**: Same as in MOEA/D.
- **Scalarization**: Same as in MOEA/D.
- **Variation Operators**: Simulated Binary Crossover (SBX) followed by polynomial mutation, applied to parents selected from the neighborhood $B(i)$.
- **Localized Replacement**: Same as in MOEA/D.
- **Archive Management**: Same as in MOEA/D.

Note on Polynomial Mutation Implementation

The **polynomial mutation** operator used throughout all algorithms in this study (NSGA-II, MOEA/D, MOEA/D with crossover) was adapted from the formulation originally proposed by Deb and Goyal (1996). The implementation adopted herein applies the perturbation to each decision variable individually, ensuring respect of the true variable bounds through normalization. This design choice differs from simplified textbook implementations that often assume variables are normalized to [0, 1], which may lead to inconsistent behavior when the search space has heterogeneous scales.

Implementation Details

1. **Normalization with Respect to Bounds:** For each decision variable x_j, the distances to the lower and upper bounds are computed and used to adjust the mutation magnitude. This ensures that the polynomial mutation operates consistently regardless of the absolute range of the decision variable.
2. **Perturbation via Polynomial Distribution:** A random perturbation Δ_q is computed according to the polynomial distribution controlled by the distribution index η_m, following the procedure described in Hamdan (2012). This perturbation is scaled by the width of the variable's feasible interval.
3. **Bound Enforcement:** After applying the mutation, the resulting variable is clamped within its prescribed bounds.

This version ensures that the mutation effect scales properly with the variable's bounds, unlike simplified versions that apply the delta directly assuming normalized ranges.

The snippet below exemplifies the implementation used:

```
for j = 1 : V
    r = rand(1);

    % Normalize with respect to bounds
    delta1 = (child_3(j) - l_limit(j)) / (u_limit(j) - l_limit(j));
    delta2 = (u_limit(j) - child_3(j)) / (u_limit(j) - l_limit(j));

    % Compute perturbation
    if r <= 0.5
        delta_q = (2*r + (1 - 2*r)*(1 - delta1)^(mum+1))^(1/(mum+1)) - 1;
    else
        delta_q = 1 - (2*(1 - r) + 2*(r - 0.5)*(1 - delta2)^(mum+1))^(1/(mum+1));
    end

    % Scale and apply mutation
    child_3(j) = child_3(j) + delta_q * (u_limit(j) - l_limit(j));

    % Enforce bounds
    child_3(j) = min(max(child_3(j), l_limit(j)), u_limit(j));
end
```

The **original NSGA-II code provided in the course materials** adopted a simplified variant of the polynomial mutation, shown below. In this version, the mutation delta is computed without normalization to the actual decision bounds, and is directly added to the current value. This approximation assumes implicitly that all variables lie in the range [0, 1], which is not generalizable to broader decision spaces:

```
% Simplified mutation (used in the provided NSGA-II code):
for j = 1 : V
    r = rand(1);
    if r < 0.5
        delta = (2*r)^(1/(mum+1)) - 1;
    else
        delta = 1 - (2*(1 - r))^(1/(mum+1));
    end
    child_3(j) = child_3(j) + delta;
end
```

While this formulation is computationally efficient, it breaks **scale invariance** and may lead to inconsistent behavior when decision variable ranges differ or extend beyond [0, 1].

Motivation for the Adopted Implementation The more rigorous mutation operator was adopted uniformly across all algorithms for the following reasons:

- **Consistency across decision spaces** with arbitrary variable bounds.
- **Correctness in problem formulation** for constrained and bounded problems.
- **Fair comparison across algorithms**, avoiding artificial differences caused by inconsistent mutation effects.

3.3 Parameter Configuration

For all algorithms, the following parameters were fixed:

- Population size: $N = 200$
- Generations: $G = 2000$
- Number of independent runs: 30 (random seed fixed from 1 to 30)
- Crossover and mutation indices: $\mu = 20$, $\mu_m = 20$

In NSGA-II and MOEA/D with crossover, the following genetic operators were used:

- Crossover probability: 90%
- Mutation probability: 10%

while in MOEA/D without crossover, only mutation was applied with 100% probability.

Neighborhood size $T = 10$ was used in both versions of MOEA/D, ensuring that updates are localized but sufficiently diverse.

3.4 Evaluation Metrics

The performance of each algorithm was assessed using four widely adopted multi-objective optimization metrics: **Generational Distance (GD)**, **Inverted Generational Distance (IGD)**, the **Δ diversity indicator**, and the **Hypervolume (HV)**. These metrics provide complementary insights into the convergence and diversity of the obtained solution sets. At the end of each run, only the first non-dominated front was retained for evaluation. The hypervolume computation is proposed in two versions, the approach derived from the *PlatEMO* framework and, for two-objective problems, the rectangle-based computation method.

A detailed mathematical definition of these metrics, including discussion on their significance and how they were computed in this work, is provided in Section 4.1.

3.5 Reproducibility

All code was executed in MATLAB with modularized folder structures (`+kpi/`, `+moead/`, `+moead_modified/`, `+nsga2/`, `+utility/`), allowing isolated evaluation of each method. Seeds were fixed per run, and all outputs (including final solution fronts and metric values) were saved for further inspection.

This consistent and transparent setup ensures both **fairness** in comparison and **reproducibility** of all results obtained in this experimental study.

4. Results and Discussion

This section presents the evaluation of the algorithms under comparison on the selected benchmark problems. Each algorithm was independently executed for 30 runs, and the results were assessed in terms of convergence and diversity using a standard set of multi-objective performance indicators. The first non-dominated front obtained at the end of each run was extracted and evaluated.

4.1 Evaluation Metrics

To ensure a comprehensive analysis of performance, four widely used metrics in evolutionary multi-objective optimization were employed: **Generational Distance (GD)**, **Inverted Generational Distance (IGD)**, **Diversity Indicator (Δ)**, and **Hypervolume (HV)**. Each of these provides insight into a specific aspect of the quality of the approximated Pareto front. All metric values were computed consistently across all 30 independent runs per algorithm. The average values for each metric were stored and used for analysis in the subsequent discussion.

Generational Distance (GD)

The **Generational Distance** measures how close the obtained approximation front A is to the true Pareto front Z, and is defined as:

$$GD(A) = \frac{1}{n}\left(\sum_{i=1}^{n} d_i^p\right)^{1/p}$$

Where:

- $A = a_1, \ldots, a_n$ is the set of non-dominated solutions produced by the algorithm;
- $Z = z_1, \ldots, z_m$ is the reference Pareto front;
- d_i is the Euclidean distance between a solution $a_i \in A$ and the nearest point in Z;
- p is typically set to 2 (Euclidean norm).

A lower GD indicates better convergence to the optimal front.

Inverted Generational Distance (IGD)

The **Inverted Generational Distance** instead evaluates how well the reference front is covered by the approximation set A, and is defined as:

$$IGD(A) = \frac{1}{m}\left(\sum_{i=1}^{m} d_i^p\right)^{1/p}$$

In this case, d_i is the distance between a reference point $z_i \in Z$ and the closest point in A. A low IGD reflects both good convergence and distribution.

Diversity Indicator (Δ)

The **Diversity Indicator**, denoted as Δ, was computed as the maximum of the GD and IGD values for each run:

$$\Delta = \max(\mathrm{GD}, \mathrm{IGD})$$

This simplified formulation emphasizes the worst-case degradation in either convergence or spread, penalizing algorithms that fail on one of the two aspects.

Hypervolume (HV)

The **Hypervolume (HV)** indicator quantifies the portion of the objective space that is weakly dominated by a set of solutions and bounded by a fixed **reference point** (commonly referred to as the *nadir point*). It is widely recognized for capturing both convergence and diversity of the approximation set.

In this study, two complementary methods were employed to compute the HV:

1. Rectangle-Based Method (Exact for $M = 2$) For **bi-objective problems**, HV can be computed **exactly** via a geometric decomposition of the dominated region into **rectangles**. This method assumes that:

- The input front is **non-dominated** and sorted in **ascending order** with respect to the first objective.
- The **reference point** lies in the **worst corner** of the objective space (typically beyond the last solution in each objective).

The intuition is as follows:

- Between each pair of consecutive points on the Pareto front, a **rectangle** is defined whose width is the difference in the first objective, and whose height is the distance to the reference point along the second objective.
- A final rectangle closes the region between the **last point** and the **reference point**, completing the dominated area.
- The **total hypervolume** is the **sum of the areas** of these axis-aligned rectangles.

This method is:

- **Deterministic** and **fast** for $M = 2$,
- Used in this work as a **reference computation** for two-objective problems,
- Particularly valuable when a large number of comparisons need to be made (e.g., across multiple runs).

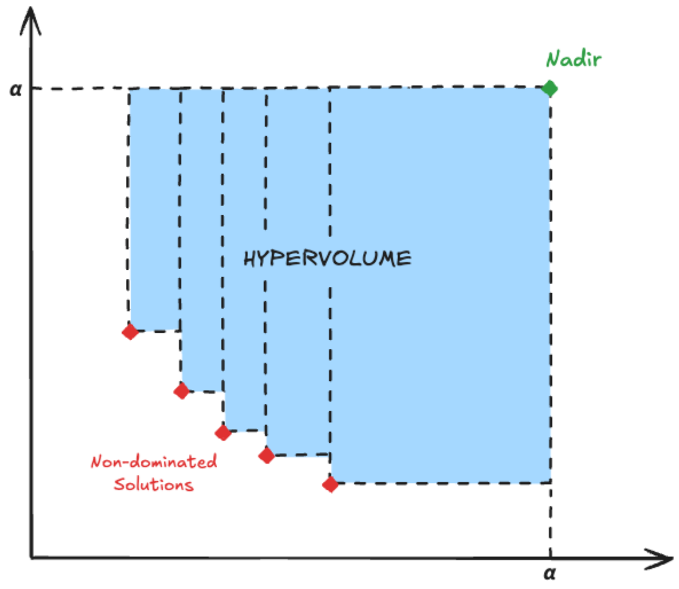


Figure 2: Hypervolume computation using rectangles for bi-objective problems.

2. PlatEMO-Based Method (General Case) To ensure accurate and consistent computation of the Hypervolume (HV) across different problem dimensions, this study employs a dual-strategy method derived from the PlatEMO platform [3]. The implementation automatically selects the most appropriate strategy depending on the dimensionality of the objective space.

Case 1: Monte Carlo Estimation for $M > 2$

For problems with more than two objectives, PlatEMO adopts a Monte Carlo-based estimation technique. This method approximates the hypervolume by generating a large number of uniformly distributed random points within the objective space bounded by the ideal and reference points. Specifically:

- A number of n_{sample} points are sampled uniformly between the component-wise minimum of the current approximation set (the ideal point) and the reference point (typically a nadir).
- For each sample point, the number of solutions that dominate it is counted.
- The contribution of each solution is then estimated based on the number of samples it dominates, weighted by coefficients α_i computed analytically as:

$$HV \approx \sum_{i=1}^{N} \alpha_i \cdot \text{Vol}_i$$

Where:

  - α_i accounts for the combinatorial weight of the subset in the decomposition,
  - Vol_i is the estimated hypervolume contribution of the i-th solution.

This method strikes a balance between **efficiency and scalability**, making it suitable for high-dimensional problems where exact computation is prohibitively expensive. While inherently approximate, the large sample size and the analytical form of the weights ensure good accuracy for comparative evaluations.

Case 2: Exact Computation for $M = 2$ via the HSO Algorithm

The implementation of hypervolume calculation for $\mathbf{M = 2}$ in the function `compute_HV_platemo` is explicitly designed around the properties of bi-objective spaces. It simplifies the **Hypervolume Slicing Objectives (HSO)** principle, originally proposed by White et al. [4], by leveraging the geometric triviality of slicing a 2D Pareto front along one objective.

The function `hypesub` operates by first sorting the points along the second objective dimension through the line:

```
[S, i] = sortrows(A, M);
```

This allows constructing **vertical slabs** in the objective space, whose widths are given by the difference between consecutive points:

```
extrusion = S(i+1, M) - S(i, M);  % Between current and next point
```

or between the last point and the reference bound:

```
extrusion = bounds(M) - S(i, M);
```

For $\mathbf{M = 2}$, this leads to a recursive reduction into $\mathbf{M = 1}$, where the contribution of each slab is trivially computed via the precomputed `alpha` weights:

```
h(pvec(1:i)) = h(pvec(1:i)) + extrusion * alpha(i);
```

This structure, although efficient and exact for $\mathbf{M = 2}$, inherently assumes that slices orthogonal to one axis fully characterize the dominated region in the reduced subspace. Once dimension $\mathbf{M = 1}$ is reached, contributions are accumulated directly because the intervals are scalar and ordered.

This assumption fails when $\mathbf{M > 2}$. After slicing on one dimension, the remaining **(M-1)**-dimensional space does not reduce to ordered segments but requires further handling of non-trivial Pareto subsets. The recursion:

```
h = h + extrusion * hypesub(l, S(1:i, :), M-1, bounds, pvec(1:i), alpha, k);
```

naively passes the first `i` points in the sorted order without ensuring they represent valid, non-overlapping slices in the reduced subspace. In higher dimensions, dominance relations between points within the slice cannot be ignored, and these points do not generally partition the subspace cleanly. Overlaps, holes, and complex front shapes arise, which the current recursion cannot capture.

Moreover, the allocation of contributions directly to indices `pvec(1:i)` becomes meaningless when slices no longer correspond to axis-aligned intervals but to irregular regions within **M-1** dimensions. This simplification would lead to incorrect volume assignments.

The use of this method for $\mathbf{M > 2}$ would require redefining the recursive structure to track the Pareto dominance relations within each slice explicitly and build complex subregions recursively, something this code is not prepared to handle. As a result, the `compute_HV_platemo` method correctly switches to Monte Carlo estimation when $\mathbf{M > 2}$.

For these reasons, this method is limited to $\mathbf{M} = \mathbf{2}$. A rigorous and general approach suitable for any $\mathbf{M}$ is provided in the following section, where a faithful implementation of HSO is presented.

3. HSO Method (General Case) The following implementation of the **Hypervolume Slicing Objectives (HSO)** method is adapted from https://github.com/BIMK/PlatEMO/blob/master/PlatEMO/Metrics/HV.m and provides an exact computation of the hypervolume indicator for problems with $\mathbf{M} < \mathbf{4}$ objectives. This method recursively decomposes the dominated region of the objective space through orthogonal **slicing** along one objective at a time. The recursion proceeds until the one-dimensional case is reached, where contributions can be computed directly. This approach reflects the original formulation proposed by White et al. (2007) for hypervolume computation.

The entry point is the function `compute_HV_HSO`, which starts by normalizing the points within the **unit hypercube** $[\mathbf{0}, \mathbf{1}]$^$\mathbf{M}$:

```
fmin = min(min(points,[],1),zeros(1,M));
fmax = ref_point;
points = (points - fmin) ./ ((fmax - fmin));
points(any(points > 1, 2), :) = [];
RefPoint = ones(1, M);
```

This ensures the reference point is `[1, 1, ..., 1]` and standardizes the objective space. Points outside these bounds are discarded. If no points remain, the function returns zero hypervolume.

For $\mathbf{M} = \mathbf{2}$, this method essentially reduces to the **rectangle method** described earlier. Specifically, the slicing reduces to computing horizontal strips between consecutive solutions ordered on the first objective. The final contributions are integrated through differences with the reference point, matching the same intuition and geometry as the rectangle-based method.

For $\mathbf{M} = \mathbf{3}$, the method constructs slices recursively. The outer loop slices the objective space along the first objective, collecting slabs with thickness corresponding to differences along the first dimension. Inside each slab, the procedure repeats on the remaining dimensions via the `Slice` function.

The `Slice` function performs slicing over the current dimension `k+1`:

```
cell_ = {abs(p(k)-p_(k)), ql};
```

Each slice accumulates its width along dimension `k`, and the remaining points are maintained in non-dominated form through the `Insert` function, which removes dominated points in dimensions `k+1:M`. This insertion logic ensures that the recursive decomposition proceeds only on relevant subsets of points, thereby avoiding redundant contributions.

At the final dimension, contributions are aggregated through:

```
hv = hv + S{i,1} * abs(p(M) - RefPoint(M));
```

The recursion guarantees that contributions from each orthogonal "strip" are properly accumulated. The total hypervolume is scaled back through:

```
hv = hv * prod(ref_point);
```

Thus, the method constructs the hypervolume as a sum of disjoint volumes, consistent with the theoretical formulation of HSO.

It is important to note that this recursive approach cannot trivially be extended beyond $\mathbf{M = 3}$ due to the exponential growth in the number of recursive slices and the combinatorial complexity of managing non-dominated subsets at each recursion level.

Comparison with Monte Carlo Estimation

To validate this exact HSO implementation, experiments were performed on problems with $\mathbf{M = 3}$. Results show that the **exact HSO computation closely matches the Monte Carlo-based approximation employed by PlatEMO**. The observed discrepancies between the two methods are consistently below **0.001** in hypervolume units. This confirms the correctness of the HSO implementation and the reliability of the Monte Carlo estimation when properly configured.

An example of the obtained results for **30 independent runs** is reported below on the **DTLZ1** problem with $\mathbf{M = 3}$ objectives and **V=2** decision variables, a very simple problem with a known Pareto front. These results clearly show that, while small variations exist (as expected from Monte Carlo estimation), the two methods converge to similar values:

Algorithm	GD	IGD	Δ	HV Platemo	HV HSO
moead_linear	0.0077	0.0183	0.0183	0.1900	0.1902
moead_cheby	0.0078	0.0185	0.0185	0.1880	0.1882
moead_mod_linear	0.0076	0.0192	0.0192	0.1900	0.1898
moead_mod_cheby	0.0079	0.0193	0.0193	0.1871	0.1874
nsga2	0.0078	0.0277	0.0277	0.1859	0.1860

The minor differences visible in `HV_Platemo` are attributable to the stochastic nature of Monte Carlo sampling, confirming the soundness and accuracy of the HSO method for $\mathbf{M = 3}$.

4.2 Results Analysis

The comparative evaluation of MOEA/D with Chebyshev scalarization, MOEA/D with linear scalarization, and NSGA-II was carried out on three classical benchmark problems: ZDT1, ZDT2, and ZDT3. Each algorithm was executed over 30 independent runs, using the same set of random seeds across all configurations to ensure consistency and comparability. The performance was assessed using four key metrics—Generational Distance (GD), Inverted Generational Distance (IGD), Hypervolume (HV) (both Platemo version and Rectangles method), and the Delta metric, whose definitions and properties have been introduced in the previous section.

The following table reports the average values of each metric across the 30 runs for every algorithm and problem:

Problem	Algorithm	GD	IGD	Δ	HV Platemo	HV Rect
ZDT1	moead_linear	0.0015	0.0022	0.0027	1.0004	0.8739
	moead_cheby	0.0018	0.0023	0.0028	1.0338	0.8738
	moead_sbx_linear	0.0004	0.0022	0.0022	0.8738	0.8738
	moead_sbx_cheby	0.0009	0.0025	0.0026	0.9225	0.8735
	nsga2	0.0006	0.0023	0.0023	0.8738	0.8738
ZDT2	moead_linear	0.0005	0.0060	0.0060	0.5354	0.5354
	moead_cheby	0.0016	0.0024	0.0029	0.7105	0.5405
	moead_sbx_linear	0.0007	0.0161	0.0161	0.5208	0.5208
	moead_sbx_cheby	0.0005	0.0026	0.0026	0.5519	0.5403
	nsga2	0.0004	0.0024	0.0024	0.5405	0.5405
ZDT3	moead_linear	0.0034	0.1908	0.1908	1.4654	1.3304
	moead_cheby	0.0035	0.1907	0.1907	1.5142	1.3304
	moead_sbx_linear	0.0022	0.1917	0.1917	1.3302	1.3302
	moead_sbx_cheby	0.0024	0.1907	0.1907	1.3433	1.3303
	nsga2	0.0023	0.1907	0.1907	1.3303	1.3303

In the case of **ZDT1**, which features a convex and continuous Pareto front, all algorithms perform consistently well in terms of convergence and diversity. NSGA-II exhibits the best GD and Δ values, indicating its ability to place solutions very close to the true front while maintaining a uniform distribution. The modified MOEA/D with linear scalarization (`moead_sbx_linear`) performs exceptionally, achieving the best overall values across all metrics, including GD, IGD, and HV (identical across both versions). This suggests that the addition of crossover significantly mitigates the limitations of linear scalarization on convex fronts. Standard MOEA/D variants without crossover show slightly inferior IGD values, confirming the beneficial effect of recombination. Interestingly, the Hypervolume computed via PlatEMO overestimates the performance of MOEA/D-linear, likely due to better coverage near the extreme regions, while the rectangle-based HV—aligned with true front coverage—supports a more consistent ranking.

When we shift our focus to **ZDT2**, characterized by a **non-convex** Pareto front, the behavior of the algorithms diverges. As expected, linear scalarization struggles to reconstruct non-convex shapes: `moead_linear` shows poor IGD and HV values despite a good GD, revealing that some solutions converge, but the overall coverage is poor. This limitation also affects the sbx-enhanced version (`moead_sbx_linear`), which maintains low GD but has the highest IGD and lowest HV. In contrast, both NSGA-II and MOEA/D-Chebyshev (standard and sbx) provide superior performance across all metrics. NSGA-II achieves the best GD and IGD values, whereas MOEA/D-Chebyshev attains the highest HV (especially using PlatEMO), underlining its robustness in dealing with front curvature and shape complexity. The crossover variant (`moead_sbx_cheby`) also brings minor improvements over the pure mutation version, suggesting additional benefit from recombination.

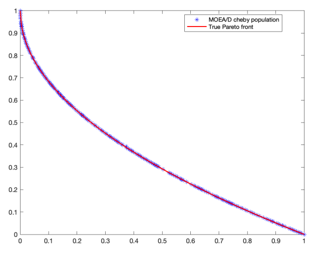
The scenario becomes even more challenging with **ZDT3**, due to the **disconnected** nature of the Pareto front. Here, convergence and diversity are harder to maintain across multiple disjoint regions. All algorithms show an increase in GD and IGD as expected. NSGA-II and `moead_sbx_linear` perform best in terms of GD, with values around 0.002, suggesting better convergence to the various front segments. However, MOEA/D with Chebyshev scalarization (standard and sbx) achieves the highest Hypervolume, demonstrating its superior ability to **spread solutions** across multiple disconnected segments. Linear MOEA/D variants, while performing well in convergence (especially when combined with crossover), fail to achieve high HV values, reflecting limited diversity across all regions. This confirms that scalarization-based methods require careful design to avoid biasing solutions toward specific areas of the front.

It is important to interpret the Δ **metric** carefully. By definition, $\Delta = \max(\mathrm{GD}, \mathrm{IGD})$ for each run. However, the table reports **average values over 30 runs**, and the reported Δ values are the mean of the per-run maximums. This means that the average Δ may not match the greater of average GD and average IGD. For example, in ZDT1 for `moead_cheby`, $\Delta = 0.0028$, which does not correspond exactly to $\max(0.0018, 0.0023) = 0.0023$, since in some runs GD was higher, and in others IGD was. This nuance is essential for correct interpretation of aggregated results.

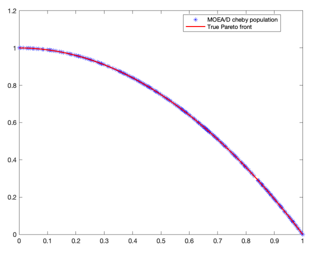
Overall, the experiments confirm theoretical expectations. **NSGA-II** consistently demonstrates strong performance across all problems due to its dominance-based selection and global elitism mechanisms. **MOEA/D with Chebyshev scalarization** adapts well to non-convex and disconnected fronts, leveraging its decomposition strategy to maintain spread and diversity. The **addition of crossover** (SBX) in MOEA/D proves beneficial, especially for linear scalarization, where it compensates for the inability to handle non-convexity and improves performance even on convex and discontinuous problems. **MOEA/D with linear scalarization**, while effective on ZDT1, consistently underperforms on ZDT2 and ZDT3 due to its intrinsic limitation in spanning the entire front geometry, unless recombination is introduced to increase diversity.

Visual Comparison of Pareto Fronts

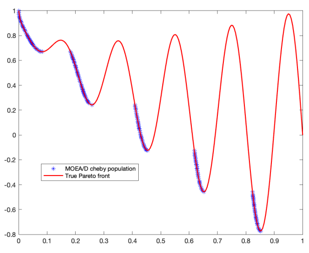
The graphical analysis includes a series of plots that compare the true Pareto front (shown in red) with the non-dominated solutions obtained by each algorithm. For each benchmark problem, three images are shown—one per algorithm—allowing direct visual inspection of coverage and convergence. These plots provide valuable qualitative support to the numerical metrics, especially in identifying regions of the front that are poorly approximated or entirely missed by certain algorithms.



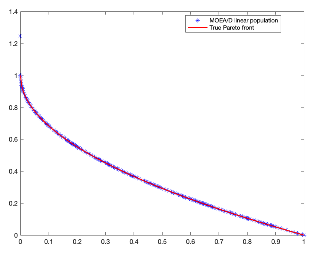
ZDT1 - MOEA/D-Chebyshev



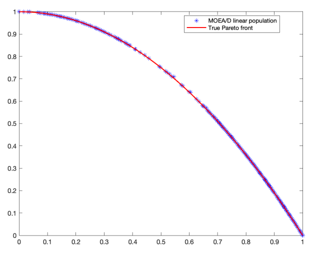
ZDT2 - MOEA/D-Chebyshev



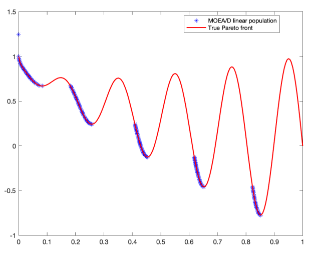
ZDT3 - MOEA/D-Chebyshev



ZDT1 - MOEA/D-Linear

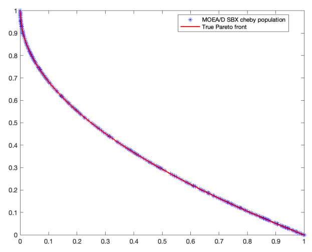


ZDT2 - MOEA/D-Linear

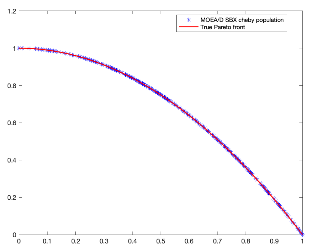


ZDT3 - MOEA/D-Linear

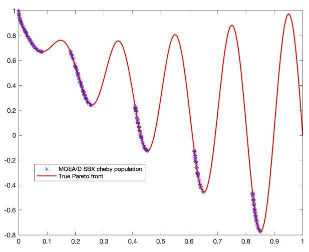
Figure 3: Comparison of final Pareto front approximations obtained by each algorithm on ZDT1–ZDT3. (1/2)



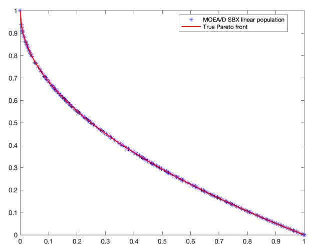
ZDT1 - MOEA/D-Chebyshev with SBX Crossover



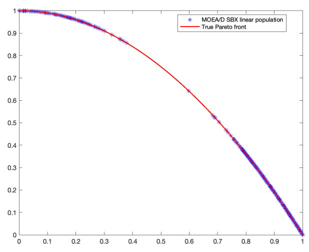
ZDT2 - MOEA/D-Chebyshev with SBX Crossover



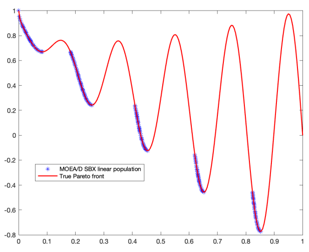
ZDT3 - MOEA/D-Chebyshev with SBX Crossover



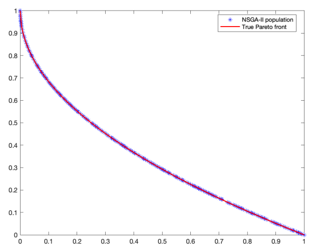
ZDT1 - MOEA/D-Linear with SBX Crossover



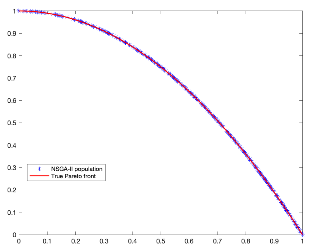
ZDT2 - MOEA/D-Linear with SBX Crossover



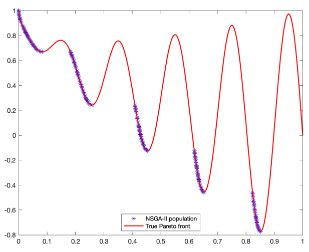
ZDT3 - MOEA/D-Linear with SBX Crossover



ZDT1 - NSGA-II



ZDT2 - NSGA-II



ZDT3 - NSGA-II

Figure 4: Comparison of final Pareto front approximations obtained by each algorithm on ZDT1–ZDT3. (2/2)

Appendix A

The numerical results from each of the 30 independent runs are reported per problem and per algorithm. This data provides insight into the variability of the algorithmic behavior and supports more fine-grained statistical analysis.

ZDT1 Results:

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	1	0.0013	0.0022	0.0022	0.9412	0.8739	0.8739
moead_cheby	1	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_linear	1	0.0005	0.0021	0.0021	0.8740	0.8740	0.8740
moead_mod_cheby	1	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	1	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
moead_linear	2	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	2	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_linear	2	0.0004	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	2	0.0006	0.0025	0.0025	0.8736	0.8736	0.8736
nsga2	2	0.0006	0.0023	0.0023	0.8737	0.8737	0.8737
moead_linear	3	0.0085	0.0022	0.0085	1.6637	0.8738	0.8738
moead_cheby	3	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_linear	3	0.0005	0.0023	0.0023	0.8736	0.8736	0.8736
moead_mod_cheby	3	0.0006	0.0023	0.0023	0.8737	0.8737	0.8737
nsga2	3	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	4	0.0024	0.0022	0.0024	1.1677	0.8738	0.8738
moead_cheby	4	0.0006	0.0023	0.0023	0.8737	0.8737	0.8737
moead_mod_linear	4	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	4	0.0006	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	4	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_linear	5	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	5	0.0007	0.0022	0.0022	0.8739	0.8739	0.8739
moead_mod_linear	5	0.0004	0.0023	0.0023	0.8735	0.8735	0.8735
moead_mod_cheby	5	0.0006	0.0028	0.0028	0.8731	0.8731	0.8731
nsga2	5	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
moead_linear	6	0.0015	0.0023	0.0023	0.9724	0.8738	0.8738
moead_cheby	6	0.0030	0.0022	0.0030	1.1305	0.8739	0.8739
moead_mod_linear	6	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	6	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	6	0.0005	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	7	0.0006	0.0022	0.0022	0.8740	0.8740	0.8740
moead_cheby	7	0.0007	0.0022	0.0022	0.8740	0.8740	0.8740
moead_mod_linear	7	0.0005	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	7	0.0006	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	7	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	8	0.0047	0.0022	0.0047	1.6160	0.8738	0.8738

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_cheby	8	0.0006	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_linear	8	0.0005	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_cheby	8	0.0006	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	8	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	9	0.0006	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	9	0.0033	0.0022	0.0033	1.1417	0.8738	0.8738
moead_mod_linear	9	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	9	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
nsga2	9	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	10	0.0006	0.0022	0.0022	0.8740	0.8740	0.8740
moead_cheby	10	0.0020	0.0022	0.0022	1.0936	0.8739	0.8739
moead_mod_linear	10	0.0005	0.0022	0.0022	0.8737	0.8737	0.8737
moead_mod_cheby	10	0.0006	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	10	0.0006	0.0022	0.0022	0.8738	0.8738	0.8738
moead_linear	11	0.0005	0.0022	0.0022	0.8740	0.8740	0.8740
moead_cheby	11	0.0007	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_linear	11	0.0004	0.0023	0.0023	0.8735	0.8735	0.8735
moead_mod_cheby	11	0.0014	0.0025	0.0025	0.9292	0.8735	0.8735
nsga2	11	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	12	0.0006	0.0023	0.0023	0.8739	0.8739	0.8739
moead_cheby	12	0.0024	0.0023	0.0024	0.9040	0.8738	0.8738
moead_mod_linear	12	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_mod_cheby	12	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	12	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
moead_linear	13	0.0006	0.0021	0.0021	0.8740	0.8740	0.8740
moead_cheby	13	0.0018	0.0022	0.0022	1.0645	0.8738	0.8738
moead_mod_linear	13	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	13	0.0041	0.0025	0.0041	1.2163	0.8735	0.8735
nsga2	13	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
moead_linear	14	0.0004	0.0021	0.0021	0.8740	0.8740	0.8740
moead_cheby	14	0.0029	0.0022	0.0029	1.2716	0.8738	0.8738
moead_mod_linear	14	0.0005	0.0021	0.0021	0.8740	0.8740	0.8740
moead_mod_cheby	14	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
nsga2	14	0.0006	0.0022	0.0022	0.8739	0.8739	0.8739
moead_linear	15	0.0006	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	15	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_linear	15	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	15	0.0007	0.0023	0.0023	0.8737	0.8737	0.8737
nsga2	15	0.0005	0.0023	0.0023	0.8739	0.8739	0.8739
moead_linear	16	0.0005	0.0021	0.0021	0.8739	0.8739	0.8739
moead_cheby	16	0.0025	0.0022	0.0025	1.2033	0.8739	0.8739
moead_mod_linear	16	0.0005	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	16	0.0006	0.0026	0.0026	0.8734	0.8734	0.8734
nsga2	16	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	17	0.0005	0.0022	0.0022	0.8738	0.8738	0.8738
moead_cheby	17	0.0011	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_linear	17	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	17	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	17	0.0005	0.0025	0.0025	0.8737	0.8737	0.8737
moead_linear	18	0.0005	0.0021	0.0021	0.8739	0.8739	0.8739
moead_cheby	18	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_linear	18	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	18	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	18	0.0006	0.0025	0.0025	0.8736	0.8736	0.8736
moead_linear	19	0.0006	0.0022	0.0022	0.8738	0.8738	0.8738
moead_cheby	19	0.0020	0.0023	0.0023	1.0779	0.8738	0.8738
moead_mod_linear	19	0.0006	0.0021	0.0021	0.8740	0.8740	0.8740
moead_mod_cheby	19	0.0006	0.0023	0.0023	0.8737	0.8737	0.8737
nsga2	19	0.0005	0.0024	0.0024	0.8737	0.8737	0.8737
moead_linear	20	0.0067	0.0023	0.0067	1.8562	0.8737	0.8737
moead_cheby	20	0.0060	0.0022	0.0060	1.6925	0.8738	0.8738
moead_mod_linear	20	0.0004	0.0022	0.0022	0.8739	0.8739	0.8739
moead_mod_cheby	20	0.0005	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	20	0.0005	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	21	0.0052	0.0022	0.0052	1.1948	0.8739	0.8739
moead_cheby	21	0.0007	0.0024	0.0024	0.8737	0.8737	0.8737
moead_mod_linear	21	0.0005	0.0023	0.0023	0.8737	0.8737	0.8737
moead_mod_cheby	21	0.0007	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	21	0.0006	0.0025	0.0025	0.8736	0.8736	0.8736
moead_linear	22	0.0006	0.0022	0.0022	0.8740	0.8740	0.8740
moead_cheby	22	0.0075	0.0022	0.0075	1.7303	0.8739	0.8739
moead_mod_linear	22	0.0004	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	22	0.0006	0.0026	0.0026	0.8734	0.8734	0.8734
nsga2	22	0.0005	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	23	0.0021	0.0022	0.0022	1.0965	0.8739	0.8739
moead_cheby	23	0.0006	0.0022	0.0022	0.8739	0.8739	0.8739
moead_mod_linear	23	0.0005	0.0021	0.0021	0.8739	0.8739	0.8739
moead_mod_cheby	23	0.0006	0.0025	0.0025	0.8736	0.8736	0.8736
nsga2	23	0.0005	0.0024	0.0024	0.8737	0.8737	0.8737
moead_linear	24	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	24	0.0046	0.0024	0.0046	1.2803	0.8736	0.8736
moead_mod_linear	24	0.0004	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	24	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	24	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	25	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	25	0.0006	0.0025	0.0025	0.8735	0.8735	0.8735
moead_mod_linear	25	0.0004	0.0020	0.0020	0.8740	0.8740	0.8740
moead_mod_cheby	25	0.0059	0.0024	0.0059	1.9446	0.8735	0.8735

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
nsga2	25	0.0005	0.0025	0.0025	0.8735	0.8735	0.8735
moead_linear	26	0.0014	0.0021	0.0021	0.9903	0.8739	0.8739
moead_cheby	26	0.0006	0.0023	0.0023	0.8737	0.8737	0.8737
moead_mod_linear	26	0.0004	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	26	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
nsga2	26	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_linear	27	0.0006	0.0023	0.0023	0.8739	0.8739	0.8739
moead_cheby	27	0.0005	0.0022	0.0022	0.8739	0.8739	0.8739
moead_mod_linear	27	0.0005	0.0021	0.0021	0.8740	0.8740	0.8740
moead_mod_cheby	27	0.0006	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	27	0.0006	0.0025	0.0025	0.8736	0.8736	0.8736
moead_linear	28	0.0006	0.0022	0.0022	0.8739	0.8739	0.8739
moead_cheby	28	0.0047	0.0023	0.0047	1.6954	0.8737	0.8737
moead_mod_linear	28	0.0005	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	28	0.0007	0.0025	0.0025	0.8735	0.8735	0.8735
nsga2	28	0.0006	0.0024	0.0024	0.8737	0.8737	0.8737
moead_linear	29	0.0005	0.0022	0.0022	0.8740	0.8740	0.8740
moead_cheby	29	0.0006	0.0024	0.0024	0.8736	0.8736	0.8736
moead_mod_linear	29	0.0004	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	29	0.0006	0.0027	0.0027	0.8732	0.8732	0.8732
nsga2	29	0.0006	0.0023	0.0023	0.8739	0.8739	0.8739
moead_linear	30	0.0017	0.0022	0.0022	1.0341	0.8739	0.8739
moead_cheby	30	0.0006	0.0023	0.0023	0.8738	0.8738	0.8738
moead_mod_linear	30	0.0004	0.0022	0.0022	0.8738	0.8738	0.8738
moead_mod_cheby	30	0.0007	0.0024	0.0024	0.8736	0.8736	0.8736
nsga2	30	0.0006	0.0022	0.0022	0.8739	0.8739	0.8739

ZDT2 Results:

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	1	0.0005	0.0025	0.0025	0.5402	0.5402	0.5402
moead_cheby	1	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_mod_linear	1	0.0006	0.0125	0.0125	0.5257	0.5257	0.5257
moead_mod_cheby	1	0.0004	0.0027	0.0027	0.5403	0.5403	0.5403
nsga2	1	0.0004	0.0027	0.0027	0.5402	0.5402	0.5402
moead_linear	2	0.0005	0.0028	0.0028	0.5397	0.5397	0.5397
moead_cheby	2	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_mod_linear	2	0.0006	0.0032	0.0032	0.5391	0.5391	0.5391
moead_mod_cheby	2	0.0004	0.0027	0.0027	0.5403	0.5403	0.5403
nsga2	2	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_linear	3	0.0006	0.0115	0.0115	0.5285	0.5285	0.5285

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_cheby	3	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_mod_linear	3	0.0006	0.0187	0.0187	0.5174	0.5174	0.5174
moead_mod_cheby	3	0.0004	0.0026	0.0026	0.5404	0.5404	0.5404
nsga2	3	0.0004	0.0025	0.0025	0.5405	0.5405	0.5405
moead_linear	4	0.0004	0.0033	0.0033	0.5393	0.5393	0.5393
moead_cheby	4	0.0004	0.0024	0.0024	0.5406	0.5406	0.5406
moead_mod_linear	4	0.0008	0.0071	0.0071	0.5342	0.5342	0.5342
moead_mod_cheby	4	0.0004	0.0025	0.0025	0.5403	0.5403	0.5403
nsga2	4	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_linear	5	0.0005	0.0041	0.0041	0.5378	0.5378	0.5378
moead_cheby	5	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_mod_linear	5	0.0006	0.0028	0.0028	0.5397	0.5397	0.5397
moead_mod_cheby	5	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	5	0.0004	0.0024	0.0024	0.5403	0.5403	0.5403
moead_linear	6	0.0004	0.0092	0.0092	0.5313	0.5313	0.5313
moead_cheby	6	0.0028	0.0023	0.0028	0.7992	0.5406	0.5406
moead_mod_linear	6	0.0008	0.0185	0.0185	0.5132	0.5132	0.5132
moead_mod_cheby	6	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
nsga2	6	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	7	0.0005	0.0092	0.0092	0.5309	0.5309	0.5309
moead_cheby	7	0.0004	0.0025	0.0025	0.5404	0.5404	0.5404
moead_mod_linear	7	0.0009	0.0080	0.0080	0.5317	0.5317	0.5317
moead_mod_cheby	7	0.0004	0.0028	0.0028	0.5403	0.5403	0.5403
nsga2	7	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_linear	8	0.0005	0.0032	0.0032	0.5393	0.5393	0.5393
moead_cheby	8	0.0017	0.0023	0.0023	0.7221	0.5405	0.5405
moead_mod_linear	8	0.0004	0.0503	0.0503	0.4785	0.4785	0.4785
moead_mod_cheby	8	0.0004	0.0025	0.0025	0.5404	0.5404	0.5404
nsga2	8	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_linear	9	0.0004	0.0033	0.0033	0.5398	0.5398	0.5398
moead_cheby	9	0.0014	0.0023	0.0023	0.6456	0.5406	0.5406
moead_mod_linear	9	0.0004	0.0027	0.0027	0.5400	0.5400	0.5400
moead_mod_cheby	9	0.0004	0.0025	0.0025	0.5405	0.5405	0.5405
nsga2	9	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_linear	10	0.0005	0.0027	0.0027	0.5399	0.5399	0.5399
moead_cheby	10	0.0004	0.0024	0.0024	0.5406	0.5406	0.5406
moead_mod_linear	10	0.0011	0.0082	0.0082	0.5307	0.5307	0.5307
moead_mod_cheby	10	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	10	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_linear	11	0.0005	0.0033	0.0033	0.5393	0.5393	0.5393
moead_cheby	11	0.0004	0.0024	0.0024	0.5403	0.5403	0.5403
moead_mod_linear	11	0.0004	0.0221	0.0221	0.5140	0.5140	0.5140
moead_mod_cheby	11	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	11	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	12	0.0005	0.0042	0.0042	0.5380	0.5380	0.5380
moead_cheby	12	0.0017	0.0024	0.0024	0.5739	0.5405	0.5405
moead_mod_linear	12	0.0008	0.0048	0.0048	0.5367	0.5367	0.5367
moead_mod_cheby	12	0.0004	0.0028	0.0028	0.5402	0.5402	0.5402
nsga2	12	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_linear	13	0.0005	0.0029	0.0029	0.5396	0.5396	0.5396
moead_cheby	13	0.0035	0.0023	0.0035	1.1186	0.5404	0.5404
moead_mod_linear	13	0.0012	0.0048	0.0048	0.5367	0.5367	0.5367
moead_mod_cheby	13	0.0039	0.0028	0.0039	0.8890	0.5400	0.5400
nsga2	13	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	14	0.0004	0.0027	0.0027	0.5401	0.5401	0.5401
moead_cheby	14	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_mod_linear	14	0.0007	0.0034	0.0034	0.5385	0.5385	0.5385
moead_mod_cheby	14	0.0004	0.0027	0.0027	0.5403	0.5403	0.5403
nsga2	14	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_linear	15	0.0004	0.0034	0.0034	0.5391	0.5391	0.5391
moead_cheby	15	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_mod_linear	15	0.0008	0.0242	0.0242	0.5059	0.5059	0.5059
moead_mod_cheby	15	0.0004	0.0027	0.0027	0.5400	0.5400	0.5400
nsga2	15	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	16	0.0008	0.0093	0.0093	0.5290	0.5290	0.5290
moead_cheby	16	0.0024	0.0022	0.0024	0.8707	0.5406	0.5406
moead_mod_linear	16	0.0014	0.0295	0.0295	0.5014	0.5014	0.5014
moead_mod_cheby	16	0.0004	0.0027	0.0027	0.5401	0.5401	0.5401
nsga2	16	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_linear	17	0.0004	0.0045	0.0045	0.5373	0.5373	0.5373
moead_cheby	17	0.0009	0.0023	0.0023	0.5405	0.5405	0.5405
moead_mod_linear	17	0.0006	0.0148	0.0148	0.5222	0.5222	0.5222
moead_mod_cheby	17	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	17	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	18	0.0004	0.0108	0.0108	0.5295	0.5295	0.5295
moead_cheby	18	0.0032	0.0023	0.0032	1.0405	0.5405	0.5405
moead_mod_linear	18	0.0004	0.0042	0.0042	0.5379	0.5379	0.5379
moead_mod_cheby	18	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
nsga2	18	0.0004	0.0024	0.0024	0.5404	0.5404	0.5404
moead_linear	19	0.0005	0.0034	0.0034	0.5393	0.5393	0.5393
moead_cheby	19	0.0018	0.0025	0.0025	0.7445	0.5403	0.5403
moead_mod_linear	19	0.0013	0.0105	0.0105	0.5292	0.5292	0.5292
moead_mod_cheby	19	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
nsga2	19	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
moead_linear	20	0.0004	0.0167	0.0167	0.5212	0.5212	0.5212
moead_cheby	20	0.0059	0.0024	0.0059	1.3617	0.5405	0.5405
moead_mod_linear	20	0.0004	0.0251	0.0251	0.5103	0.5103	0.5103
moead_mod_cheby	20	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
nsga2	20	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_linear	21	0.0007	0.0066	0.0066	0.5343	0.5343	0.5343
moead_cheby	21	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_mod_linear	21	0.0011	0.0052	0.0052	0.5360	0.5360	0.5360
moead_mod_cheby	21	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	21	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	22	0.0004	0.0042	0.0042	0.5379	0.5379	0.5379
moead_cheby	22	0.0074	0.0023	0.0074	1.3972	0.5404	0.5404
moead_mod_linear	22	0.0004	0.0462	0.0462	0.4848	0.4848	0.4848
moead_mod_cheby	22	0.0004	0.0025	0.0025	0.5403	0.5403	0.5403
nsga2	22	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	23	0.0004	0.0036	0.0036	0.5386	0.5386	0.5386
moead_cheby	23	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_mod_linear	23	0.0010	0.0315	0.0315	0.4958	0.4958	0.4958
moead_mod_cheby	23	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	23	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_linear	24	0.0008	0.0081	0.0081	0.5315	0.5315	0.5315
moead_cheby	24	0.0044	0.0025	0.0044	0.9481	0.5404	0.5404
moead_mod_linear	24	0.0005	0.0140	0.0140	0.5254	0.5254	0.5254
moead_mod_cheby	24	0.0004	0.0027	0.0027	0.5402	0.5402	0.5402
nsga2	24	0.0004	0.0023	0.0023	0.5404	0.5404	0.5404
moead_linear	25	0.0006	0.0033	0.0033	0.5390	0.5390	0.5390
moead_cheby	25	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_mod_linear	25	0.0009	0.0263	0.0263	0.5029	0.5029	0.5029
moead_mod_cheby	25	0.0004	0.0027	0.0027	0.5402	0.5402	0.5402
nsga2	25	0.0004	0.0025	0.0025	0.5402	0.5402	0.5402
moead_linear	26	0.0008	0.0061	0.0061	0.5347	0.5347	0.5347
moead_cheby	26	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_mod_linear	26	0.0008	0.0102	0.0102	0.5285	0.5285	0.5285
moead_mod_cheby	26	0.0004	0.0026	0.0026	0.5403	0.5403	0.5403
nsga2	26	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	27	0.0005	0.0034	0.0034	0.5397	0.5397	0.5397
moead_cheby	27	0.0004	0.0024	0.0024	0.5406	0.5406	0.5406
moead_mod_linear	27	0.0005	0.0260	0.0260	0.5048	0.5048	0.5048
moead_mod_cheby	27	0.0004	0.0028	0.0028	0.5400	0.5400	0.5400
nsga2	27	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_linear	28	0.0004	0.0130	0.0130	0.5264	0.5264	0.5264
moead_cheby	28	0.0046	0.0023	0.0046	1.3641	0.5405	0.5405
moead_mod_linear	28	0.0006	0.0170	0.0170	0.5204	0.5204	0.5204
moead_mod_cheby	28	0.0004	0.0026	0.0026	0.5404	0.5404	0.5404
nsga2	28	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	29	0.0004	0.0153	0.0153	0.5221	0.5221	0.5221
moead_cheby	29	0.0004	0.0023	0.0023	0.5406	0.5406	0.5406
moead_mod_linear	29	0.0013	0.0111	0.0111	0.5263	0.5263	0.5263

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_mod_cheby	29	0.0004	0.0026	0.0026	0.5402	0.5402	0.5402
nsga2	29	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405
moead_linear	30	0.0005	0.0034	0.0034	0.5388	0.5388	0.5388
moead_cheby	30	0.0004	0.0023	0.0023	0.5405	0.5405	0.5405
moead_mod_linear	30	0.0004	0.0213	0.0213	0.5151	0.5151	0.5151
moead_mod_cheby	30	0.0004	0.0025	0.0025	0.5404	0.5404	0.5404
nsga2	30	0.0004	0.0024	0.0024	0.5405	0.5405	0.5405

ZDT3 Results:

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	1	0.0033	0.1908	0.1908	1.3977	1.3304	0.7812
moead_cheby	1	0.0025	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	1	0.0024	0.1915	0.1915	1.3304	1.3304	0.7812
moead_mod_cheby	1	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	1	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	2	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
moead_cheby	2	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	2	0.0022	0.1911	0.1911	1.3302	1.3302	0.7811
moead_mod_cheby	2	0.0021	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	2	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	3	0.0101	0.1907	0.1907	2.1204	1.3305	0.7812
moead_cheby	3	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	3	0.0022	0.1921	0.1921	1.3303	1.3303	0.7811
moead_mod_cheby	3	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	3	0.0023	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	4	0.0040	0.1908	0.1908	1.6243	1.3304	0.7812
moead_cheby	4	0.0025	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	4	0.0023	0.1911	0.1911	1.3303	1.3303	0.7811
moead_mod_cheby	4	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	4	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	5	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_cheby	5	0.0039	0.1908	0.1908	1.4701	1.3304	0.7812
moead_mod_linear	5	0.0022	0.1912	0.1912	1.3303	1.3303	0.7811
moead_mod_cheby	5	0.0023	0.1908	0.1908	1.3302	1.3302	0.7811
nsga2	5	0.0021	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	6	0.0035	0.1908	0.1908	1.4291	1.3304	0.7812
moead_cheby	6	0.0048	0.1907	0.1907	1.5857	1.3304	0.7812
moead_mod_linear	6	0.0024	0.1916	0.1916	1.3303	1.3303	0.7811
moead_mod_cheby	6	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	6	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	7	0.0025	0.1908	0.1908	1.3304	1.3304	0.7812
moead_cheby	7	0.0023	0.1906	0.1906	1.3305	1.3305	0.7812
moead_mod_linear	7	0.0023	0.1916	0.1916	1.3303	1.3303	0.7811
moead_mod_cheby	7	0.0022	0.1906	0.1906	1.3304	1.3304	0.7812
nsga2	7	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	8	0.0065	0.1907	0.1907	2.0727	1.3304	0.7812
moead_cheby	8	0.0037	0.1906	0.1906	1.5085	1.3304	0.7812
moead_mod_linear	8	0.0024	0.1918	0.1918	1.3304	1.3304	0.7812
moead_mod_cheby	8	0.0021	0.1906	0.1906	1.3304	1.3304	0.7812
nsga2	8	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	9	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_cheby	9	0.0032	0.1906	0.1906	1.4322	1.3304	0.7812
moead_mod_linear	9	0.0022	0.1915	0.1915	1.3303	1.3303	0.7811
moead_mod_cheby	9	0.0021	0.1908	0.1908	1.3303	1.3303	0.7812
nsga2	9	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	10	0.0021	0.1906	0.1906	1.3305	1.3305	0.7812
moead_cheby	10	0.0023	0.1907	0.1907	1.3305	1.3305	0.7812
moead_mod_linear	10	0.0021	0.1910	0.1910	1.3303	1.3303	0.7811
moead_mod_cheby	10	0.0024	0.1908	0.1908	1.3303	1.3303	0.7811
nsga2	10	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	11	0.0022	0.1908	0.1908	1.3303	1.3303	0.7811
moead_cheby	11	0.0023	0.1906	0.1906	1.3304	1.3304	0.7812
moead_mod_linear	11	0.0022	0.1917	0.1917	1.3303	1.3303	0.7811
moead_mod_cheby	11	0.0030	0.1907	0.1907	1.3805	1.3303	0.7811
nsga2	11	0.0024	0.1908	0.1908	1.3302	1.3302	0.7811
moead_linear	12	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_cheby	12	0.0036	0.1907	0.1907	1.3594	1.3304	0.7812
moead_mod_linear	12	0.0023	0.1908	0.1908	1.3304	1.3304	0.7812
moead_mod_cheby	12	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	12	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	13	0.0025	0.1907	0.1907	1.3304	1.3304	0.7812
moead_cheby	13	0.0037	0.1907	0.1907	1.5205	1.3304	0.7812
moead_mod_linear	13	0.0022	0.1916	0.1916	1.3303	1.3303	0.7811
moead_mod_cheby	13	0.0055	0.1908	0.1908	1.6695	1.3302	0.7811
nsga2	13	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	14	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_cheby	14	0.0043	0.1906	0.1906	1.7279	1.3305	0.7812
moead_mod_linear	14	0.0022	0.1916	0.1916	1.3304	1.3304	0.7812
moead_mod_cheby	14	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	14	0.0025	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	15	0.0022	0.1908	0.1908	1.3303	1.3303	0.7811
moead_cheby	15	0.0022	0.1906	0.1906	1.3305	1.3305	0.7812
moead_mod_linear	15	0.0022	0.1919	0.1919	1.3304	1.3304	0.7812
moead_mod_cheby	15	0.0026	0.1908	0.1908	1.3302	1.3302	0.7810

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
nsga2	15	0.0024	0.1908	0.1908	1.3302	1.3302	0.7811
moead_linear	16	0.0023	0.1908	0.1908	1.3304	1.3304	0.7812
moead_cheby	16	0.0042	0.1907	0.1907	1.6595	1.3304	0.7812
moead_mod_linear	16	0.0023	0.1952	0.1952	1.3292	1.3292	0.7805
moead_mod_cheby	16	0.0024	0.1908	0.1908	1.3304	1.3304	0.7812
nsga2	16	0.0023	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	17	0.0022	0.1908	0.1908	1.3304	1.3304	0.7812
moead_cheby	17	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	17	0.0022	0.1910	0.1910	1.3303	1.3303	0.7811
moead_mod_cheby	17	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	17	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	18	0.0023	0.1908	0.1908	1.3304	1.3304	0.7812
moead_cheby	18	0.0052	0.1907	0.1907	1.8264	1.3303	0.7811
moead_mod_linear	18	0.0022	0.1914	0.1914	1.3302	1.3302	0.7811
moead_mod_cheby	18	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	18	0.0022	0.1906	0.1906	1.3304	1.3304	0.7812
moead_linear	19	0.0022	0.1906	0.1906	1.3304	1.3304	0.7812
moead_cheby	19	0.0037	0.1906	0.1906	1.5346	1.3305	0.7812
moead_mod_linear	19	0.0023	0.1909	0.1909	1.3304	1.3304	0.7812
moead_mod_cheby	19	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811
nsga2	19	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	20	0.0085	0.1908	0.1908	2.3130	1.3304	0.7812
moead_cheby	20	0.0064	0.1907	0.1907	2.1475	1.3304	0.7812
moead_mod_linear	20	0.0023	0.1911	0.1911	1.3302	1.3302	0.7811
moead_mod_cheby	20	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811
nsga2	20	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	21	0.0070	0.1908	0.1908	1.6513	1.3304	0.7812
moead_cheby	21	0.0024	0.1906	0.1906	1.3304	1.3304	0.7812
moead_mod_linear	21	0.0021	0.1913	0.1913	1.3305	1.3305	0.7812
moead_mod_cheby	21	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811
nsga2	21	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	22	0.0025	0.1908	0.1908	1.3304	1.3304	0.7812
moead_cheby	22	0.0067	0.1907	0.1907	2.1860	1.3304	0.7812
moead_mod_linear	22	0.0021	0.1910	0.1910	1.3305	1.3305	0.7812
moead_mod_cheby	22	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	22	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	23	0.0038	0.1908	0.1908	1.5530	1.3304	0.7812
moead_cheby	23	0.0078	0.1907	0.1907	2.4448	1.3305	0.7812
moead_mod_linear	23	0.0023	0.1920	0.1920	1.3303	1.3303	0.7811
moead_mod_cheby	23	0.0023	0.1908	0.1908	1.3301	1.3301	0.7810
nsga2	23	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	24	0.0024	0.1907	0.1907	1.3305	1.3305	0.7812
moead_cheby	24	0.0063	0.1907	0.1907	1.7368	1.3304	0.7812
moead_mod_linear	24	0.0022	0.1927	0.1927	1.3266	1.3266	0.7790

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_mod_cheby	24	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	24	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	25	0.0022	0.1909	0.1909	1.3303	1.3303	0.7811
moead_cheby	25	0.0021	0.1906	0.1906	1.3304	1.3304	0.7812
moead_mod_linear	25	0.0022	0.1916	0.1916	1.3304	1.3304	0.7812
moead_mod_cheby	25	0.0022	0.1907	0.1907	1.3303	1.3303	0.7811
nsga2	25	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	26	0.0055	0.1909	0.1909	1.7025	1.3304	0.7812
moead_cheby	26	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	26	0.0022	0.1912	0.1912	1.3302	1.3302	0.7811
moead_mod_cheby	26	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	26	0.0025	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	27	0.0023	0.1908	0.1908	1.3304	1.3304	0.7812
moead_cheby	27	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	27	0.0023	0.1910	0.1910	1.3304	1.3304	0.7812
moead_mod_cheby	27	0.0023	0.1907	0.1907	1.3303	1.3303	0.7811
nsga2	27	0.0024	0.1907	0.1907	1.3303	1.3303	0.7811
moead_linear	28	0.0023	0.1907	0.1907	1.3305	1.3305	0.7812
moead_cheby	28	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	28	0.0024	0.1937	0.1937	1.3302	1.3302	0.7811
moead_mod_cheby	28	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	28	0.0024	0.1907	0.1907	1.3304	1.3304	0.7812
moead_linear	29	0.0023	0.1907	0.1907	1.3305	1.3305	0.7812
moead_cheby	29	0.0023	0.1907	0.1907	1.3305	1.3305	0.7812
moead_mod_linear	29	0.0024	0.1917	0.1917	1.3303	1.3303	0.7811
moead_mod_cheby	29	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	29	0.0024	0.1907	0.1907	1.3303	1.3303	0.7812
moead_linear	30	0.0036	0.1907	0.1907	1.4906	1.3305	0.7812
moead_cheby	30	0.0022	0.1907	0.1907	1.3304	1.3304	0.7812
moead_mod_linear	30	0.0021	0.1924	0.1924	1.3305	1.3305	0.7812
moead_mod_cheby	30	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812
nsga2	30	0.0023	0.1907	0.1907	1.3304	1.3304	0.7812

DTLZ1 Results:

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	1	0.0077	0.0184	0.0184	0.1903	nan	0.1901
moead_cheby	1	0.0078	0.0185	0.0185	0.1872	nan	0.1882
moead_mod_linear	1	0.0079	0.0176	0.0176	0.1907	nan	0.1902
moead_mod_cheby	1	0.0079	0.0187	0.0187	0.1877	nan	0.1881
nsga2	1	0.0076	0.0291	0.0291	0.1860	nan	0.1854

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_linear	2	0.0076	0.0182	0.0182	0.1912	nan	0.1903
moead_cheby	2	0.0079	0.0187	0.0187	0.1885	nan	0.1883
moead_mod_linear	2	0.0078	0.0192	0.0192	0.1899	nan	0.1898
moead_mod_cheby	2	0.0079	0.0205	0.0205	0.1868	nan	0.1869
nsga2	2	0.0079	0.0266	0.0266	0.1862	nan	0.1859
moead_linear	3	0.0077	0.0179	0.0179	0.1906	nan	0.1904
moead_cheby	3	0.0078	0.0188	0.0188	0.1870	nan	0.1876
moead_mod_linear	3	0.0077	0.0192	0.0192	0.1895	nan	0.1896
moead_mod_cheby	3	0.0080	0.0187	0.0187	0.1871	nan	0.1871
nsga2	3	0.0076	0.0300	0.0300	0.1834	nan	0.1853
moead_linear	4	0.0079	0.0199	0.0199	0.1887	nan	0.1898
moead_cheby	4	0.0079	0.0180	0.0180	0.1871	nan	0.1882
moead_mod_linear	4	0.0077	0.0194	0.0194	0.1911	nan	0.1901
moead_mod_cheby	4	0.0078	0.0193	0.0193	0.1868	nan	0.1871
nsga2	4	0.0081	0.0306	0.0306	0.1857	nan	0.1865
moead_linear	5	0.0077	0.0182	0.0182	0.1904	nan	0.1902
moead_cheby	5	0.0080	0.0188	0.0188	0.1893	nan	0.1888
moead_mod_linear	5	0.0075	0.0191	0.0191	0.1895	nan	0.1900
moead_mod_cheby	5	0.0078	0.0190	0.0190	0.1871	nan	0.1876
nsga2	5	0.0076	0.0290	0.0290	0.1857	nan	0.1861
moead_linear	6	0.0077	0.0180	0.0180	0.1901	nan	0.1902
moead_cheby	6	0.0077	0.0176	0.0176	0.1884	nan	0.1889
moead_mod_linear	6	0.0075	0.0196	0.0196	0.1906	nan	0.1898
moead_mod_cheby	6	0.0075	0.0191	0.0191	0.1868	nan	0.1869
nsga2	6	0.0077	0.0266	0.0266	0.1863	nan	0.1864
moead_linear	7	0.0076	0.0188	0.0188	0.1898	nan	0.1901
moead_cheby	7	0.0076	0.0180	0.0180	0.1878	nan	0.1884
moead_mod_linear	7	0.0076	0.0194	0.0194	0.1881	nan	0.1895
moead_mod_cheby	7	0.0080	0.0198	0.0198	0.1856	nan	0.1866
nsga2	7	0.0080	0.0297	0.0297	0.1863	nan	0.1867
moead_linear	8	0.0077	0.0183	0.0183	0.1900	nan	0.1902
moead_cheby	8	0.0081	0.0193	0.0193	0.1869	nan	0.1879
moead_mod_linear	8	0.0077	0.0192	0.0192	0.1896	nan	0.1897
moead_mod_cheby	8	0.0081	0.0189	0.0189	0.1876	nan	0.1873
nsga2	8	0.0080	0.0242	0.0242	0.1861	nan	0.1863
moead_linear	9	0.0079	0.0191	0.0191	0.1891	nan	0.1898
moead_cheby	9	0.0075	0.0184	0.0184	0.1882	nan	0.1885
moead_mod_linear	9	0.0072	0.0195	0.0195	0.1911	nan	0.1900
moead_mod_cheby	9	0.0081	0.0194	0.0194	0.1868	nan	0.1868
nsga2	9	0.0083	0.0286	0.0286	0.1845	nan	0.1851
moead_linear	10	0.0073	0.0182	0.0182	0.1896	nan	0.1903
moead_cheby	10	0.0078	0.0189	0.0189	0.1878	nan	0.1878
moead_mod_linear	10	0.0078	0.0197	0.0197	0.1893	nan	0.1894
moead_mod_cheby	10	0.0079	0.0200	0.0200	0.1871	nan	0.1879

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
nsga2	10	0.0081	0.0255	0.0255	0.1869	nan	0.1864
moead_linear	11	0.0078	0.0177	0.0177	0.1898	nan	0.1902
moead_cheby	11	0.0080	0.0196	0.0196	0.1882	nan	0.1872
moead_mod_linear	11	0.0073	0.0187	0.0187	0.1905	nan	0.1900
moead_mod_cheby	11	0.0079	0.0201	0.0201	0.1864	nan	0.1874
nsga2	11	0.0080	0.0261	0.0261	0.1850	nan	0.1860
moead_linear	12	0.0075	0.0181	0.0181	0.1904	nan	0.1902
moead_cheby	12	0.0078	0.0186	0.0186	0.1881	nan	0.1880
moead_mod_linear	12	0.0075	0.0196	0.0196	0.1901	nan	0.1898
moead_mod_cheby	12	0.0080	0.0185	0.0185	0.1866	nan	0.1880
nsga2	12	0.0079	0.0307	0.0307	0.1845	nan	0.1843
moead_linear	13	0.0078	0.0198	0.0198	0.1910	nan	0.1899
moead_cheby	13	0.0077	0.0180	0.0180	0.1880	nan	0.1887
moead_mod_linear	13	0.0077	0.0198	0.0198	0.1896	nan	0.1894
moead_mod_cheby	13	0.0078	0.0189	0.0189	0.1879	nan	0.1879
nsga2	13	0.0078	0.0270	0.0270	0.1866	nan	0.1863
moead_linear	14	0.0076	0.0185	0.0185	0.1899	nan	0.1901
moead_cheby	14	0.0080	0.0178	0.0178	0.1883	nan	0.1883
moead_mod_linear	14	0.0077	0.0183	0.0183	0.1913	nan	0.1901
moead_mod_cheby	14	0.0076	0.0194	0.0194	0.1875	nan	0.1877
nsga2	14	0.0074	0.0246	0.0246	0.1871	nan	0.1871
moead_linear	15	0.0077	0.0187	0.0187	0.1885	nan	0.1902
moead_cheby	15	0.0080	0.0184	0.0184	0.1874	nan	0.1884
moead_mod_linear	15	0.0077	0.0192	0.0192	0.1900	nan	0.1898
moead_mod_cheby	15	0.0080	0.0193	0.0193	0.1887	nan	0.1884
nsga2	15	0.0076	0.0250	0.0250	0.1865	nan	0.1870
moead_linear	16	0.0078	0.0171	0.0171	0.1909	nan	0.1903
moead_cheby	16	0.0079	0.0194	0.0194	0.1881	nan	0.1881
moead_mod_linear	16	0.0074	0.0198	0.0198	0.1884	nan	0.1898
moead_mod_cheby	16	0.0077	0.0183	0.0183	0.1886	nan	0.1880
nsga2	16	0.0076	0.0270	0.0270	0.1855	nan	0.1862
moead_linear	17	0.0078	0.0192	0.0192	0.1890	nan	0.1902
moead_cheby	17	0.0079	0.0181	0.0181	0.1880	nan	0.1881
moead_mod_linear	17	0.0076	0.0198	0.0198	0.1904	nan	0.1898
moead_mod_cheby	17	0.0078	0.0193	0.0193	0.1864	nan	0.1866
nsga2	17	0.0077	0.0296	0.0296	0.1854	nan	0.1857
moead_linear	18	0.0074	0.0172	0.0172	0.1903	nan	0.1904
moead_cheby	18	0.0075	0.0182	0.0182	0.1892	nan	0.1886
moead_mod_linear	18	0.0077	0.0194	0.0194	0.1903	nan	0.1897
moead_mod_cheby	18	0.0080	0.0198	0.0198	0.1861	nan	0.1870
nsga2	18	0.0076	0.0293	0.0293	0.1845	nan	0.1844
moead_linear	19	0.0077	0.0184	0.0184	0.1907	nan	0.1900
moead_cheby	19	0.0079	0.0185	0.0185	0.1888	nan	0.1877
moead_mod_linear	19	0.0073	0.0189	0.0189	0.1900	nan	0.1901

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_mod_cheby	19	0.0079	0.0190	0.0190	0.1880	nan	0.1878
nsga2	19	0.0079	0.0285	0.0285	0.1843	nan	0.1855
moead_linear	20	0.0076	0.0188	0.0188	0.1898	nan	0.1900
moead_cheby	20	0.0080	0.0186	0.0186	0.1865	nan	0.1881
moead_mod_linear	20	0.0077	0.0185	0.0185	0.1899	nan	0.1900
moead_mod_cheby	20	0.0078	0.0186	0.0186	0.1882	nan	0.1880
nsga2	20	0.0076	0.0267	0.0267	0.1852	nan	0.1845
moead_linear	21	0.0077	0.0193	0.0193	0.1902	nan	0.1899
moead_cheby	21	0.0079	0.0183	0.0183	0.1881	nan	0.1887
moead_mod_linear	21	0.0076	0.0184	0.0184	0.1913	nan	0.1900
moead_mod_cheby	21	0.0081	0.0185	0.0185	0.1869	nan	0.1879
nsga2	21	0.0079	0.0298	0.0298	0.1860	nan	0.1862
moead_linear	22	0.0075	0.0184	0.0184	0.1904	nan	0.1901
moead_cheby	22	0.0077	0.0186	0.0186	0.1893	nan	0.1889
moead_mod_linear	22	0.0077	0.0190	0.0190	0.1895	nan	0.1900
moead_mod_cheby	22	0.0078	0.0190	0.0190	0.1861	nan	0.1873
nsga2	22	0.0081	0.0271	0.0271	0.1865	nan	0.1861
moead_linear	23	0.0078	0.0180	0.0180	0.1902	nan	0.1903
moead_cheby	23	0.0080	0.0186	0.0186	0.1880	nan	0.1877
moead_mod_linear	23	0.0077	0.0189	0.0189	0.1916	nan	0.1899
moead_mod_cheby	23	0.0081	0.0200	0.0200	0.1866	nan	0.1875
nsga2	23	0.0080	0.0282	0.0282	0.1876	nan	0.1868
moead_linear	24	0.0074	0.0182	0.0182	0.1896	nan	0.1901
moead_cheby	24	0.0078	0.0187	0.0187	0.1871	nan	0.1880
moead_mod_linear	24	0.0075	0.0194	0.0194	0.1891	nan	0.1899
moead_mod_cheby	24	0.0078	0.0202	0.0202	0.1873	nan	0.1869
nsga2	24	0.0079	0.0293	0.0293	0.1852	nan	0.1853
moead_linear	25	0.0076	0.0177	0.0177	0.1885	nan	0.1902
moead_cheby	25	0.0078	0.0183	0.0183	0.1886	nan	0.1878
moead_mod_linear	25	0.0075	0.0195	0.0195	0.1901	nan	0.1892
moead_mod_cheby	25	0.0077	0.0187	0.0187	0.1869	nan	0.1869
nsga2	25	0.0078	0.0240	0.0240	0.1883	nan	0.1875
moead_linear	26	0.0077	0.0180	0.0180	0.1896	nan	0.1904
moead_cheby	26	0.0081	0.0185	0.0185	0.1891	nan	0.1886
moead_mod_linear	26	0.0078	0.0184	0.0184	0.1886	nan	0.1898
moead_mod_cheby	26	0.0079	0.0199	0.0199	0.1887	nan	0.1877
nsga2	26	0.0079	0.0252	0.0252	0.1873	nan	0.1863
moead_linear	27	0.0076	0.0177	0.0177	0.1896	nan	0.1903
moead_cheby	27	0.0077	0.0184	0.0184	0.1884	nan	0.1880
moead_mod_linear	27	0.0077	0.0205	0.0205	0.1904	nan	0.1897
moead_mod_cheby	27	0.0078	0.0188	0.0188	0.1866	nan	0.1873
nsga2	27	0.0077	0.0310	0.0310	0.1856	nan	0.1854
moead_linear	28	0.0080	0.0184	0.0184	0.1906	nan	0.1901
moead_cheby	28	0.0081	0.0185	0.0185	0.1878	nan	0.1882

Algorithm	Run	GD	IGD	Δ	HV Platemo	HV Rect	HV HSO
moead_mod_linear	28	0.0076	0.0188	0.0188	0.1898	nan	0.1900
moead_mod_cheby	28	0.0078	0.0187	0.0187	0.1882	nan	0.1873
nsga2	28	0.0077	0.0260	0.0260	0.1870	nan	0.1873
moead_linear	29	0.0079	0.0176	0.0176	0.1911	nan	0.1905
moead_cheby	29	0.0075	0.0187	0.0187	0.1874	nan	0.1874
moead_mod_linear	29	0.0076	0.0187	0.0187	0.1906	nan	0.1899
moead_mod_cheby	29	0.0077	0.0198	0.0198	0.1871	nan	0.1871
nsga2	29	0.0079	0.0276	0.0276	0.1866	nan	0.1859
moead_linear	30	0.0075	0.0175	0.0175	0.1897	nan	0.1904
moead_cheby	30	0.0078	0.0186	0.0186	0.1877	nan	0.1885
moead_mod_linear	30	0.0078	0.0193	0.0193	0.1907	nan	0.1896
moead_mod_cheby	30	0.0077	0.0205	0.0205	0.1854	nan	0.1869
nsga2	30	0.0080	0.0279	0.0279	0.1845	nan	0.1846

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002, doi: 10.1109/4235.996017.

[2] Q. Zhang and H. Li, “MOEA/d: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007, doi: 10.1109/TEVC.2007.892759.

[3] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, “PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum],” *IEEE Computational Intelligence Magazine*, vol. 12, no. 4, pp. 73–87, 2017, doi: 10.1109/MCI.2017.2742868.

[4] L. While, P. Hingston, L. Barone, and S. Huband, “A faster algorithm for calculating hypervolume,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 29–38, 2006, doi: 10.1109/TEVC.2005.851275.