Imperial College London

MSc INDIVIDUAL RESEARCH PROJECT 2021

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Derivative-free Multi-objective Optimization in Julia

Author: Yuanyuan Zhang (CID: 01888715)

Course: MSc Control and Optimisation

Supervisor: Prof. Eric C. Kerrigan

Co-supervisor: Dr. Marta Zagorowska

Second Marker: Prof. David Angeli

Date: September 1, 2021

Abstract

Bi-objective mesh adaptive direct search (BiMADS) is a derivative-free optimization algorithm to solve optimization problems with two objectives. BiMADS approximates the Pareto set of the bi-objective problem to find the best trade-off. This project implements the BiMADS algorithm in an existing single-objective optimization package `DirectSearch.jl` in Julia as an extension. In addition, a hypervolume indicator is developed to evaluate the quality of the obtained Pareto set. Based on this hypervolume indicator, a new stopping condition is designed and implemented for terminating the optimization once the hypervolume achieves the expected value. Following this, two more stopping conditions are also added to the `DirectSearch.jl` for the first time, which allows the users to limit the running time for the optimization or terminate the optimization by key interruption. At last, a unit test file is designed for all the new implementations to ensure all the elements perform as expected.

The developed package was first tested on a building temperature control case study formulated as a bi-objective problem, which aims to find a trade-off between the error in following the setpoint and sensitivity to disturbances. Then, 20 bi-objective optimization function sets are selected to test `DirectSearch.jl` and compare with a state-of-art bi-objective optimization solver `NOMAD`. The comparison between these two implementations shows that `DirectSearch.jl` is outstanding in terms of running time. Also, it is more flexible to customize the optimization using the Julia implementations because it provides more user-defined arguments and multiple stopping conditions.

Keywords: Derivative-free optimization, Multi-objective optimization, Direct search, BiMADS, Julia.

Acknowledgements

I would first like to express my sincere gratitude to my project supervisor Prof. Eric Kerrigan for his invaluable advice throughout the project. Even some short comments are also crucial to my project because they help me avoid working in the wrong direction, which also shows his immense knowledge in Control and Optimization.

I would also like to thank my project co-supervisor Dr. Marta Zagorowska, who provides numerous useful materials and detailed instructions for my programming, presentation and final thesis, which helps me start up the project and carry it on smoothly.

Finally, I would like to thank my parent for their unconditional support, especially when I was infected with Covid-19 virus in January. Those days will be tougher without their comfort.

Contents

1 Introduction

1.1 Motivation and Background

Derivative-free Optimization (DFO) Efficient optimization usually relies on extracting the information from derivative functions and constraints in terms of various variables. However, the growth in sophistication for scientific and engineering problems sometimes causes the derivatives of cost/constraints functions unavailable or impractically expensive to be calculated. For instance, the objective functions are noisy, or the variables are within an absolute value operator. Therefore, DFO methods are introduced, which do not rely on the derivatives of the objective functions but only require the comparison of evaluations of the objective functions in each iteration [1, 2].

Multi-objective Optimization (MOO) Most of the popular and reliable optimization algorithms are for a single-objective optimization (SOO), while most real-life problems involve decisions that depend on multiple and conflicting criteria. In this case, there is usually no single solution that simultaneously minimizes all the objective functions [3]. Therefore the multi-objective optimizer is designed to find the best trade-offs among these criteria [4].

Julia Julia, a modern language unveiled in 2012 [5], will be used in this project due to some of its outstanding features. Firstly, Julia owns straightforward syntax and supports Unicode characters, hence the users could directly write or read the variables as appeared in papers, even for greek letters. Meanwhile, the compiler for Julia is carefully designed so that Julia owns fast running performance as performance languages such as C/C++ [6]. Lastly, the project involves many mathematical algorithms that need to be implemented in code and Julia optimizes the syntax for scientific computing.

1.2 Literature Review

DFO methods for single-objective optimization usually compare between the values of the cost/constraints functions iteratively to evaluate the search progress. There are two main categories of DFO algorithms: Heuristic Methods and non-heuristic Methods [2, 7]. Heuristic Methods, such as genetic algorithms and Nelder-Mead Algorithm, mostly rely on simple concepts and structures. They are not very rigorous and do not include a guarantee of success. As for Non-heuristic methods, Direct Multi-search (DMS) Algorithm is the most representative and contains many powerful methods such as Exhaustive Search (ES), Grid Search (GS) and Generalized Pattern Search (GPS) [7]. These algorithms will converge to a local optimum by updating the incumbent point with a better candidate found by adjusting the search step length and polling around the current poll point iteratively. However, these methods could only be implemented in unconstrained optimization and generate trial points in a finite set of directions. Therefore, Mesh Adaptive Direct

Search (MADS) [8], which evolves from GPS, has been proposed to optimize the non-smooth function under general non-smooth constraints. This improvement is achieved by introducing the notions of frame, which will be explained in Section 2.2.

MOO problems typically could be reformulated into a series of single-objective problems to give an approximation of the Pareto front [9]. For example, the Weighted Sum (WS) method converts the MOO problem into SOO using a convex combination of objectives [10]. However, the WS method is incapable of generating Pareto points for non-convex Pareto front and different weights combinations might generate duplicate solutions. Furthermore, it can be complicated to choose a priori selection of weights for each objective function in reality. For black-box simulation optimization problems without explicit forms of objective functions, Pareto set pursuing (PSP) method approximates the objective functions by sampling near the Pareto optimal frontier and adding more samples to the set of non-dominated points iteratively [11]. Mesh Adaptive Direct Multi-search (DMulti-MADS) [12] is a new convergence-proved method that emerged recently. Instead of aggregating MOO in SOO, DMulti-MADS is inspired by MADS and introduces single-objective direct search algorithms to MOO. It can generate a population of Pareto solutions that converges to the Pareto front with a uniform coverage measure [7].

In this project, bi-objective mesh adaptive direct search (BiMADS) proposed by [4] will be used to solve a bi-objective optimization (BOO) problem. This method provides an approximation for Pareto front of a BOO problem by first formulating the MOO into a series of SOO structures and then solves a series of bound-constrained SOO using MADS with increasingly stringent stopping criteria.

1.3 Project Specification

This project will focus on the BOO problem and aims to implement BiMADS optimization algorithm based on the existing MADS optimization package `DirectSearch.jl` in Julia [13], to develop a derivative-free solver for bi-objective optimization. Several new stopping criteria will be added to the package, which provides more flexibility for the users to solve complex optimization problems. The solver will be tested in a building temperature control problem and the performance will be compared with `NOMAD-3.9 C++` using bi-objective problems with known Pareto fronts.

Report Structure Section 2 introduces the notions and necessary background theories for this project. Section 3 first analyzes the required features for the BiMADS in `DirectSearch.jl`. Then, it describes the structures of the codes and the designs of some key functions. Section 4 details the building case and the test functions mathematically. In Section 5, BiMADS in `DirectSearch.jl` is tested on the building case with the numerical results presented. Also, the performance of the BiMADS in `DirectSearch.jl` is compared with that in `NOMAD-3.9 C++`.

2 Theoretical Background

This section introduces the notions and theories required for this project. Firstly, the definitions of bi-objective optimization and Pareto dominance are given. Secondly, the principles of MADS and BiMADS are described along with their software implementations. Lastly, the evaluation methods used to assess the solver are provided as well.

2.1 Bi-objective Optimization

The Bi-objective problem (BOP) to be solved in this project can be generally formed as

$$\text{BOP}: \quad \min_{x \in \Omega} F(x) = (f_1(x), f_2(x)), \tag{2.1}$$

where n is the dimension of the variables, F represents the projection from the variable space $\mathbb{R}^m$ to the objective space $\mathbb{R}^2$ and $\Omega \in \mathbb{R}^m$ is the feasible set. However, there is usually no unique solution that minimizes all the objective functions at the same time [2]. Therefore, to achieve the best trade-offs between these functions, the notion of Pareto optimum is introduced to generate a set of solutions for the BOP.

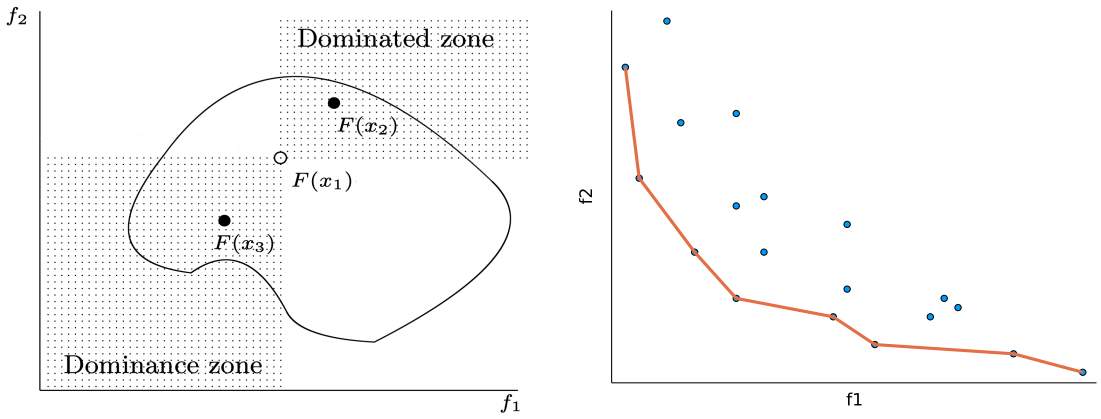
Pareto Optimum

In bi-objective problems, a solution is defined as a Pareto optimum solution if it is impossible to find another variable that improves one objective without degrading the other [14]. A series of Pareto optimum solutions could form the Pareto set, from which the users could select the solutions in terms of the preferred criterion. Mathematically, the Pareto optimum is defined by means of Pareto dominance [4] shown below, where $u, v \in \Omega$ are the decision variables:

- $u \preceq v$ (u weakly dominates v) $\iff f_i(u) \leq f_i(v)$ for $i = 1, 2$,
- $u \prec v$ (u dominates v) $\iff u \preceq v$ and $f_i(u) < f_i(v)$ for $i = 1$ or $i = 2$,
- $u \sim v$ (u is indifferent to v) otherwise.

Thus, a solution is a Pareto optimum solution if and only if there is no variable that dominates the current point.

Figure 2.1 illustrates these notions with three points. In Figure 2.1(a), x_1 dominates all the feasible points in the dominated zone (i.e., $x_1 \prec x_2$), while it is dominated by all the points in the dominance zone (i.e., $x_3 \prec x_1$). By this definition, the Pareto set can be obtained as shown in Figure 2.1(b) on the orange line.



(a) Pareto Dominance [4]

(b) Pareto Set

Fig. 2.1: Illustration for Pareto dominance and set in the objective space

2.2 Derivative-free Optimization

BiMADS is a derivative-free optimization algorithm that approximates the Pareto set of a bi-objective problem. It first formulates the problem into a series of single-objective problems and then solves a series of bound-constrained single-objective problems using MADS with increasingly stringent stopping criteria.

Mesh Adaptive Direct Search (MADS)

MADS is originally proposed in [8], which is a direct search method used for minimizing a non-smooth function $f : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ within the constraint set $\Omega \subseteq \mathbb{R}^m$. The MADS algorithm generally has three phases for iteration k:

- Search Phase: Generating a set of search points using a specified search algorithm, which aims to search around the entire objective space to find the best point currently (incumbent point).
- Poll Phase: Searching along with all possible directions for the given initial point or the incumbent point, to find a new dominating point for the next iteration.
- Update Phase: Updating the parameters such as Mesh/Frame size to enlarge or refine the search space.

The concepts of Mesh and Frame are essential to MADS, where the mesh size vector $\Delta_k^m \in \mathbb{R}^n$ determine the set of points that may be evaluated as trial points. The frame size vector $\Delta_k^p \in \mathbb{R}^n$ defines the search space from where the new incumbent point can be selected. Figure 2.2 illustrates the mesh size, frame size and the MADS will evaluate the trail points (i.e., $p^{1,2,3}$) around the current incumbent point x_k to find the new dominating point.

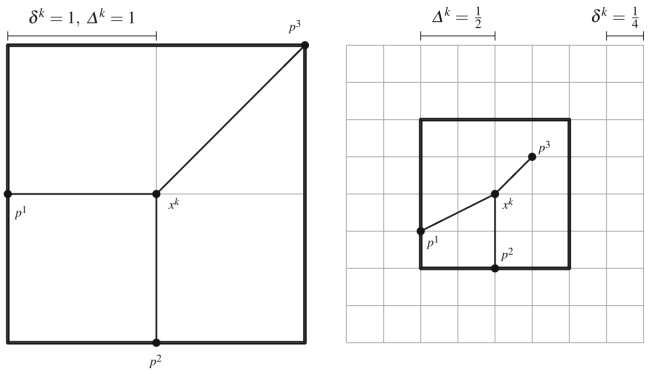


Fig. 2.2: Examples of meshes and frames in $\mathbb{R}^2$ for different Δ_k^m and Δ_k^p [8]

In addition, the MADS implemented in `DirectSearch.jl` has been improved according to [15] and [16], where the first one introduces the `OrthoMADS` to minimize the size of the cones of unexplored directions. The latter one introduces Granular variables to solve the problems with constraints on the granularity of variables.

2.3 Bi-objective mesh adaptive direct search (BiMADS)

The original BiMADS algorithm is shown in Algorithm 1 [4] and the detailed explanation for each part is also shown in this section.

2.3.1 Initialization

The BiMADS first runs the Initialization stage with a given point x_0. This stage minimizes two single-objective problem

$$\min_{x\in\Omega} f_1(x) \quad \text{and} \quad \min_{x\in\Omega} f_2(x) \tag{2.2}$$

using MADS individually. During the optimization, all the dominating points in Search or Poll phase have to be stored and evaluated by Pareto dominance criterion to obtain the very first Pareto set $X_{\mathcal{L}}$ with cardinality J. Also, the weight for each points $w(x)$ and the weighted distance $\delta \in \mathbb{R}^J$ need to be set as above to determine the reference point.

2.3.2 Main Iterations

After obtaining the first Pareto set, the main part of the BiMADS will be run iteratively until hitting the stopping condition. The updated Pareto set will gradually approach the Pareto front and attempt a uniform coverage of the Pareto front.

Algorithm 1 BiMADS Algorithm [4]

INITIALIZATION:

- Apply MADS from initial point x_0 to solve $\min_{x\in X} f_1(x)$ and $\min_{x\in X} f_2(x)$, where $X \in \mathbb{R}^n$ is the feasible set.
- Let $X_{\mathcal{L}} = \left\{x^1, x^2, \ldots, x^J\right\}$ be an ordered list of pairwise non-dominated points such that $f_1\left(x^1\right) < f_1\left(x^2\right) < \cdots < f_1\left(x^J\right)$ and $f_2\left(x^1\right) > f_2\left(x^2\right) > \cdots > f_2\left(x^J\right)$
- Initialize the weight $w(x) = 0$ for all $x \in X$ and let $\delta > 0$

MAIN ITERATION: Repeat

- REFERENCE POINT DETERMINATION:
  - If $J > 2$, let $\hat{\jmath} \in \operatorname{argmax}_{j=2,\ldots,J-1} \delta^j = \frac{\left\|F\left(x^j\right)-F\left(x^{j-1}\right)\right\|^2+\left\|F\left(x^j\right)-F\left(x^{j+1}\right)\right\|^2}{w\left(x^j\right)+1}$ and determine the reference point $r = \left(f_1\left(x^{\hat{\jmath}+1}\right), f_2\left(x^{\hat{\jmath}-1}\right)\right)$.
  - If $J = 2$, let $x^{\hat{\jmath}} = x^2$, determine the reference point $r = \left(f_1\left(x^2\right), f_2\left(x^1\right)\right)$ and set $\delta^{\hat{\jmath}} = \frac{\left\|F\left(x^2\right)-F\left(x^1\right)\right\|^2}{w\left(x^2\right)+1}$.
  - If $J = 1$, let $x^{\hat{\jmath}} = x^1, \delta^{\hat{\jmath}} = \frac{\delta}{w\left(x^{\hat{\jmath}}\right)+1}$ and apply the MADS algorithm from $x^{\hat{\jmath}}$ to solve $\min_{x\in X} f_1(x)$ and $\min_{x\in X} f_2(x)$. Terminate MADS using the same condition as in INITIALIZATION and continue to the step UPDATE $X_{\mathcal{L}}$.
- SINGLE-OBJECTIVE FORMULATION MINIMIZATION:

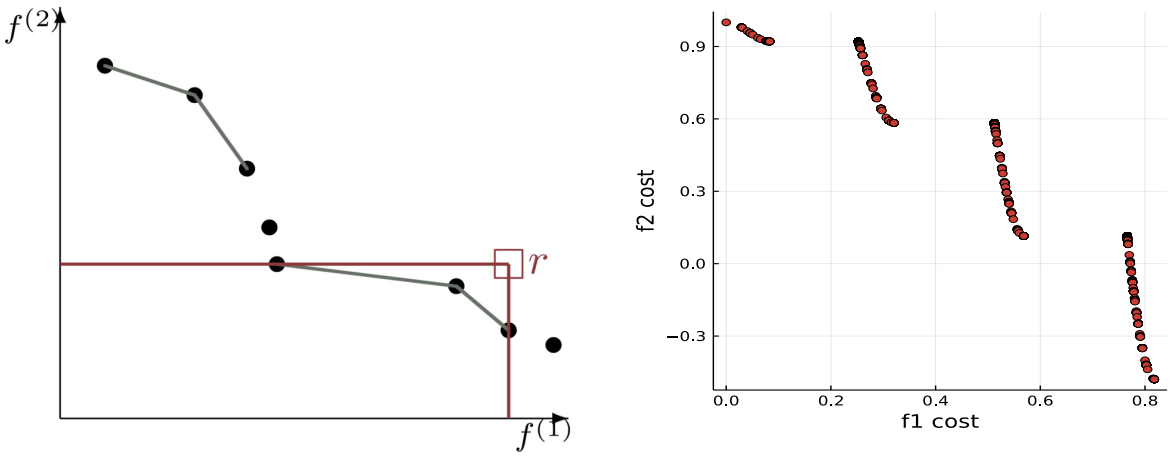
  Solve a single-objective formulation ϕ_r by MADS from the starting point $x^{\hat{\jmath}}$. Terminate MADS when the mesh size parameter Δ^m drops below $\Delta\left(\delta^{\hat{\jmath}}\right) = O\left(\delta^{\hat{\jmath}}\right)$ or if a maximal number of objective evaluations is attained.
- UPDATE $X_{\mathcal{L}}$

  Add all non-dominated points found in the current iteration to $X_{\mathcal{L}}$, remove dominated points from $X_{\mathcal{L}}$ and order the resulting list of points.

  Increase weights: $w\left(x^{\hat{\jmath}}\right) \leftarrow w\left(x^{\hat{\jmath}}\right) + 1$ for each $x \in X_{\mathcal{L}}$.

Reference Point Determination

Firstly, the reference point $r \in \mathbb{R}^2$ should be found for the single-objective reformulation and optimization. If $J > 2$, the euclidean distance between 3 successive points in the Pareto set are calculated as shown in Figure 2.3(a). The points with the largest distance will be potentially selected as the reference point. The BiMADS algorithm will search around the reference point and try to fill the gap with other Pareto solutions to approach the actual Pareto front. Figure 7.21 shows the progress for each iteration. However, some bi-objective problems have discontinuous Pareto fronts naturally as shown in Figure 2.3(b). Hence the BiMADS will stall near these break-points and waste time on searching for nothing. Therefore, a weight should be associated with each point to count the number of being the reference point and the point j with the largest weighted distance δ^j will be selected to generate the reference point. If $J = 2$, then the second point is selected. If $J = 1$, then we repeat the Initialization with the new initial point x^1.



(a) Construction of Reference point [4]

(b) Discontinuous Pareto front

Fig. 2.3: Illustration for reference point determination and natural gaps in Pareto front

Single-objective formulation minimization

After obtaining the reference point $r \in \mathbb{R}^2$ associated with Pareto solution x^j, the then bi-objective problem could be reformulated as single-objective problem

$$\min_{x \in \Omega} \phi_r(F(x)), \tag{2.3}$$

where $\phi_r : \{\mathbb{R} \cup \{+\infty\}\}^2 \to \{\mathbb{R} \cup \{+\infty\}\}$ is

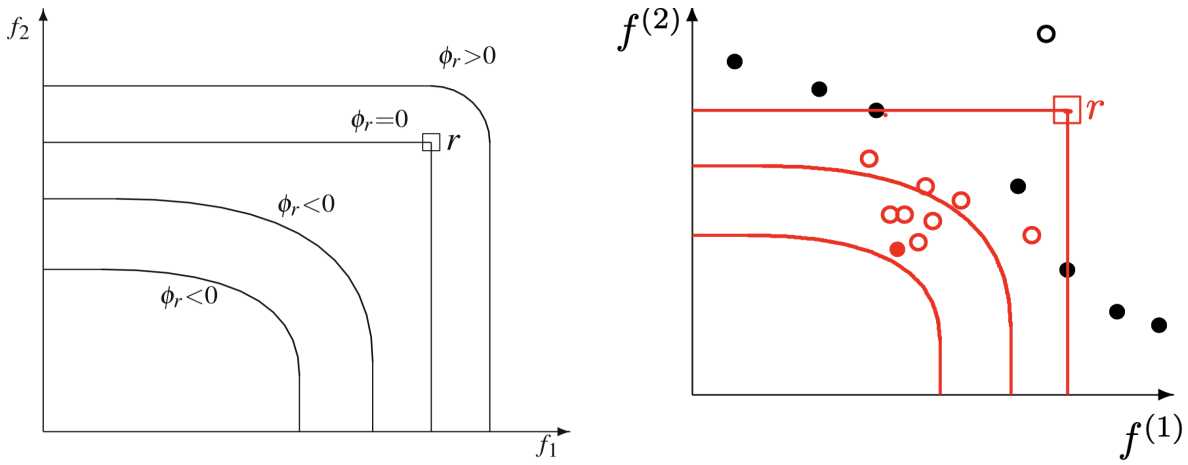
$$\phi_r(s) := \begin{cases} -\left(r_1 - f_1(x)\right)^2 \left(r_2 - f_2(x)\right)^2 & \text{if } f_1(x) \leq r_1 \text{ and } f_2(x) \leq r_2 \\ \left(\max\left\{f_1(x) - r_1, 0\right\}\right)^2 + \left(\max\left\{f_2(x) - r_2, 0\right\}\right)^2 & \text{otherwise.} \end{cases} \tag{2.4}$$

Under this reformulation, the level set of ϕ_r is shown in Figure 2.4(a). The values of ϕ_r are negative if and only if the trial points are in the dominance zone of r. Otherwise, $\phi_r \geq 0$. Then in each iteration, the MADS will be applied to the problem in Equation (2.3), which will try to evaluate trial points lying in the dominance zone of x^j and attempt to fill the gap as shown in Figure 2.4(b). Similarly, all the dominating points for this MADS optimization should be added to $X_{\mathcal{L}}$. The convergence analysis is detailed in [4].

Update $X_{\mathcal{L}}$

At the end of each iteration, the BiMADS will remove all the dominated points in $X_{\mathcal{L}}$ with only Pareto solutions left. The weight associated with the current selected point x^j should also be increased by 1 to presenting stalling for some problems with discontinuous Pareto fronts.

The original BiMADS algorithm only supports two stopping conditions. The optimization will be terminated either after a fixed number of BiMADS iterations or when the δ^j is lower than a pre-set threshold.



(a) Level set of ϕ_r (b) Trail points in dominance zone

Fig. 2.4: Illustration for problem reformulation [7]

However, more stopping conditions is required to gain more flexibility for solving some complex problems, which will be explained in Section 3.2.

2.4 Software Packages for BiMADS

NOMAD in C++

The original BiMADS algorithm and some improvements [16, 15] have been implemented in `NOMAD-3.9` in C++ [17]. This package is developed by the group who proposed MADS/BiMADS but it does not provide enough flexibility on modifying the parameters to adapt to different problems. In this project, `NOMAD-3.9 C++` will be used as a benchmark to evaluate the performance of BiMADS in `DirectSearch.jl`. The aim is to achieve a similar or better performance while providing more flexibility on customizing the optimizations.

DirectSearch.jl

The current version of `DirectSearch.jl` was developed by Edward Stables [13] and Lukas Baliunas [18], and only support single-objective optimization using MADS. This package leverages Julia's features such as multiple dispatch and abstract type to speed the optimization and simplify the code structure. Here are some important types with their hierarchy relations and functionalities.

- `DSProblem` inherit from `AbstractProblem`, which is used to define the problems to be optimized. The members of this type includes objective functions, constraint functions, initial points, stopping conditions, iteration/evaluation limitations, current feasible solution and cost.

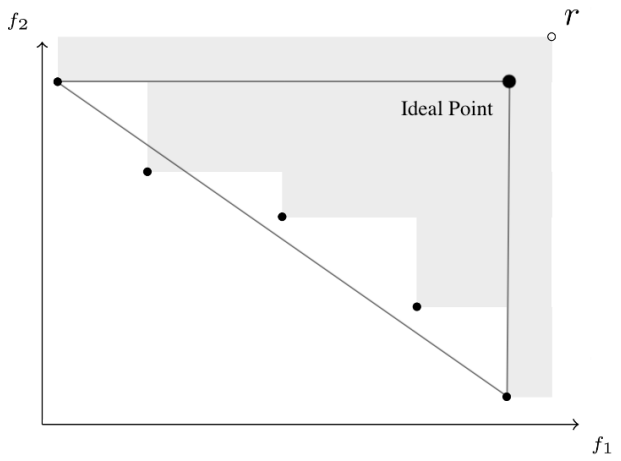


Fig. 2.5: Hypervolume Indicator for bi-objective problems [12]

- `Status`, `Config` are two concrete types and instantiated in `DSProblem`. They store the configuration options and status information for the solver, such as current mesh/poll size, direction, iteration number.

- `AbstractStoppingCondition` is an abstract type and cannot be instantiated. It contains all the concrete types for stopping condition such as `IterationStoppingCondition`, `MeshPrecisionStoppingCondition` etc. These different concrete types make it possible to dispatch the same function for evaluating different stopping conditions.

In this project, BiMADS will be added into `DirectSearch.jl` as an extension. All the types mentioned above will be used for the same purpose as in MADS. Several new types will also be introduced in Section 3.2.

2.5 Evaluation Method

Hypervolume Indicator

Hypervolume indicator [19] is a common way to assess the quality of Pareto sets obtained from multi-objective optimizations. It calculates the volume of the objective space dominated by a Pareto set and delimited above by a reference point $r \in \mathbb{R}^p$, where p is the number of the objective functions. Figure 2.5 illustrates the hypervolume for bi-objective space, which is the area enclosed by all the Pareto solutions and the reference point. Furthermore, to normalize the numerical value of hypervolume, the actual area is divided by the triangle area enclosed by the ideal point and two endpoints as shown in Figure 2.5. According to [20], the selection of the reference point affects the performance of the indicator, hence the performance of the optimization algorithms. Some of the points may have low hypervolume contributions, which means the

hypervolume will not be decreased much without them. These points might be neglected, hence the Pareto set will be non-uniformly distributed. [20] suggest selecting the reference point k times away from the ideal point (in terms of each dimension) to ensure each Pareto solution has the same hypervolume contributions. Here, $k = 1 + 1/\mu$, where μ is the number of the points in the Pareto set.

Performance Profile

To compare the performance between `NOMAD` and `DirectSearch.jl`, the Julia package `BenchmarkProfiles.jl` will be used [21]. It is based on the performance profile, which describes the (cumulative) distribution function for a performance metric. The package will take the cost (i.e., run time, iteration number) of all the solvers for different optimization problems. Then, for each problem, the best cost will be selected as the baseline to carry out the comparison. Finally, the package will output a figure which describes the proportion of the problems that can be solved by the solver within a factor τ of the best solver.

Unit Test

Unit test is necessary for any software development. It aims to ensure that each unit in the software could provide the proper result as expected. Each unit test function will test a small unit (i.e., functions, objects) that can be logically isolated in a package or system and verify its correctness [22]. The main advantages of implementing the unit test can be concluded as

1. Unit test helps the developers to refactor codes or work on some unfamiliar codes because they can identify the bugs easily and quicker.

2. Unit test could serve as documentation. It helps new developers to learn the functionality provided by each unit. Also, it provides an appropriate or inappropriate usage of a function or module.

Julia provides a package called `Test.jl` to implement basic unit test using the following macros:

- `@testset` groups tests into a set.
- `@test` tests that if the expression evaluates to `true`.
- `@test_throw` tests that if the expression throws an exception/error.

Figure 2.6 shows an example of a failed unit test, where two failures occur in the test set named `Optimization for p1 and p2`. Thus, the users can quickly locate the bugs. As for this project, a test file will be design to test all the new implementations and all the tests should be passed in the final deliverable package.

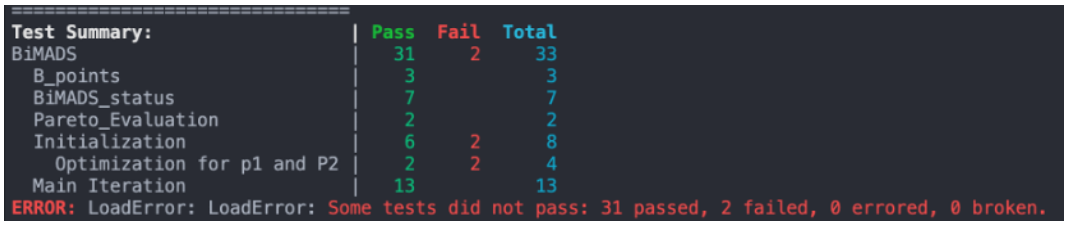


Fig. 2.6: Result of a failed unit test

3 Design and Implementation

This section provides the short requirements analysis, code structure design and code implementation for extending `DirectSearch.jl` with BiMADS algorithm, which is also the most important requirement of this project. The design will be explained in three parts: problem formulation, initialization and main BiMADS iteration. In addition, the explanations for different stopping conditions and unit test will also be detailed. All the new implementations can be found on GitHub.

3.1 BiMADS

3.1.1 Problem Formulation

In `DirectSearch.jl`, an optimization problem is defined by an instance of `DSProblem`, where users could set the parameters such as numbers of variables, iteration limits, initial points, granularity, and most importantly the objective function(s). As for the original code, the expression for the function is written in a `function` with `Float64` as the return value.

However, BiMADS problems will have two objective functions. Meanwhile, the mathematical expressions are required for both functions instead of two `Float64` numbers because these functions will be optimized by MADS separately in the initialization stage. Therefore, a new formulation structure is used for BiMADS as

```
function obj(x)
    f1(x) = ...
    f2(x) = ...
    return [f1,f2]
end
```

The users need to define a `function obj(x)` which contains two functions with `Float64` returned. The `obj(x)` returns a `Vector{function}` and will be passed to the instance of `DSProblem`. Then, this instance will be passed to `Optimize!`, which is the only public API for the optimization. To distinguish between single and bi-objective problems, the dimension of `obj(x)` needs to be assessed by the function

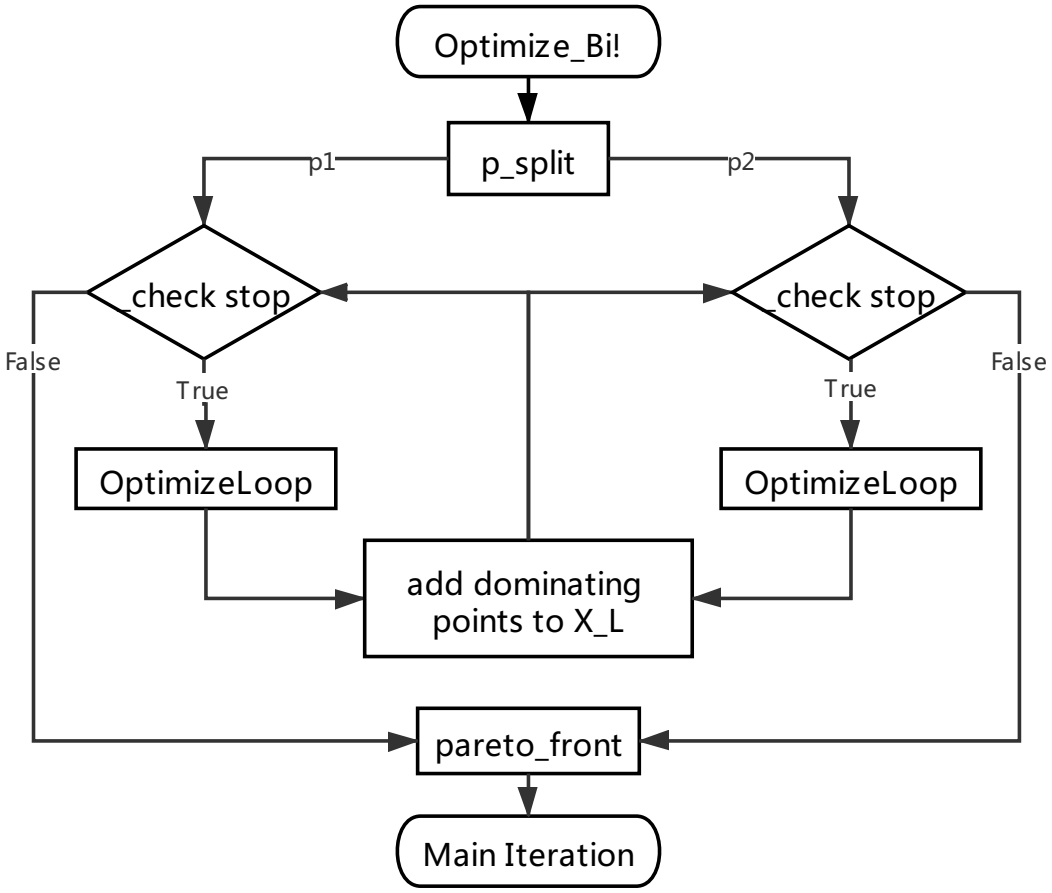


Fig. 3.1: Flow chart for Initialization

`p_dim`. Finally, the bi-objective problems will be solved by `Optimize_Bi!`, where the actual BiMADS algorithm implements.

3.1.2 Initialization

The flow chart for initialization stage is shown in Figure 3.1 which follows the INITIALIZATION part in Algorithm 1. To explain:

1. The bi-objective `DSProblem` instance `p` is firstly split into `p1` and `p2` by function `p_split`, where both `p1`, `p2` are instances with single-objective function `f1` and `f2`.

2. `p1` and `p2` will be optimized using MADS algorithm individually to find their optimum solutions. During the optimizations, if the current MADS iteration is labeled as `dominating iteration`, which means a new better point with smaller cost is found as the new incumbent point.

3. A new type named `B_points` was designed as shown in List 1 to store the following information. Firstly, the current solution x should be stored along with its costs in terms of `f1` and `f2` in line 1 and

2. These two costs will be used as a coordinate to identify a point in 2D objective space. In addition, as mentioned in Section 2.3.2, all the points should be associated with a weight for the reference point determination stage as in line 3. Also, the new type contains a inner constructor to create an instance for this type.

Listing 1 A new type named `B_points`

```
1    """
2    points for BiMADS Algo in objective space
3    """
4    mutable struct B_points
5        cost::Vector{Float64} #f1 and f2 value in objective space
6        x_now::Vector{Float64}# current variable
7        weight::Int           #weight for each undominated point to determine
         ↪ δ
8        function B_points(cost::Vector{Float64},x::Vector{Float64}, w::Int=0)
         ↪ #inner constructor
9            p = new()
10           p.cost = cost
11           p.x_now=x
12           p.weight = w
13           return p
14       end
15   end
```

Then, all the points in objective space can be stored in a `Vector{B_points}` set $X_{\mathcal{L}}$ for later usage.

4. The last step is to evaluate all the points using function `pareto_front` to find the Pareto set. The general design for this function is to filter out all the dominated points and sort the rest of them for the reference point determination.

In addition, if the iteration or function evaluation limit have already been run out in the initialization stage, an error will be thrown out to warn the users to provide more budgets for iteration or function evaluation limits so that the initialization stage could be finished.

3.1.3 Main Iteration

The flow chart for main iteration stage is shown in Figure 3.2 which follows the MAIN ITERATION part in Algorithm 1.

To explain, before entering the BiMADS loop, a new instance of `DSProblem` called `p_reform` should be initialized for the reformulated single-objective problem for ϕ_r. `p_reform` is a copy of the very first bi-objective problem instance `p` because most of their parameters are the same, such as stopping conditions.

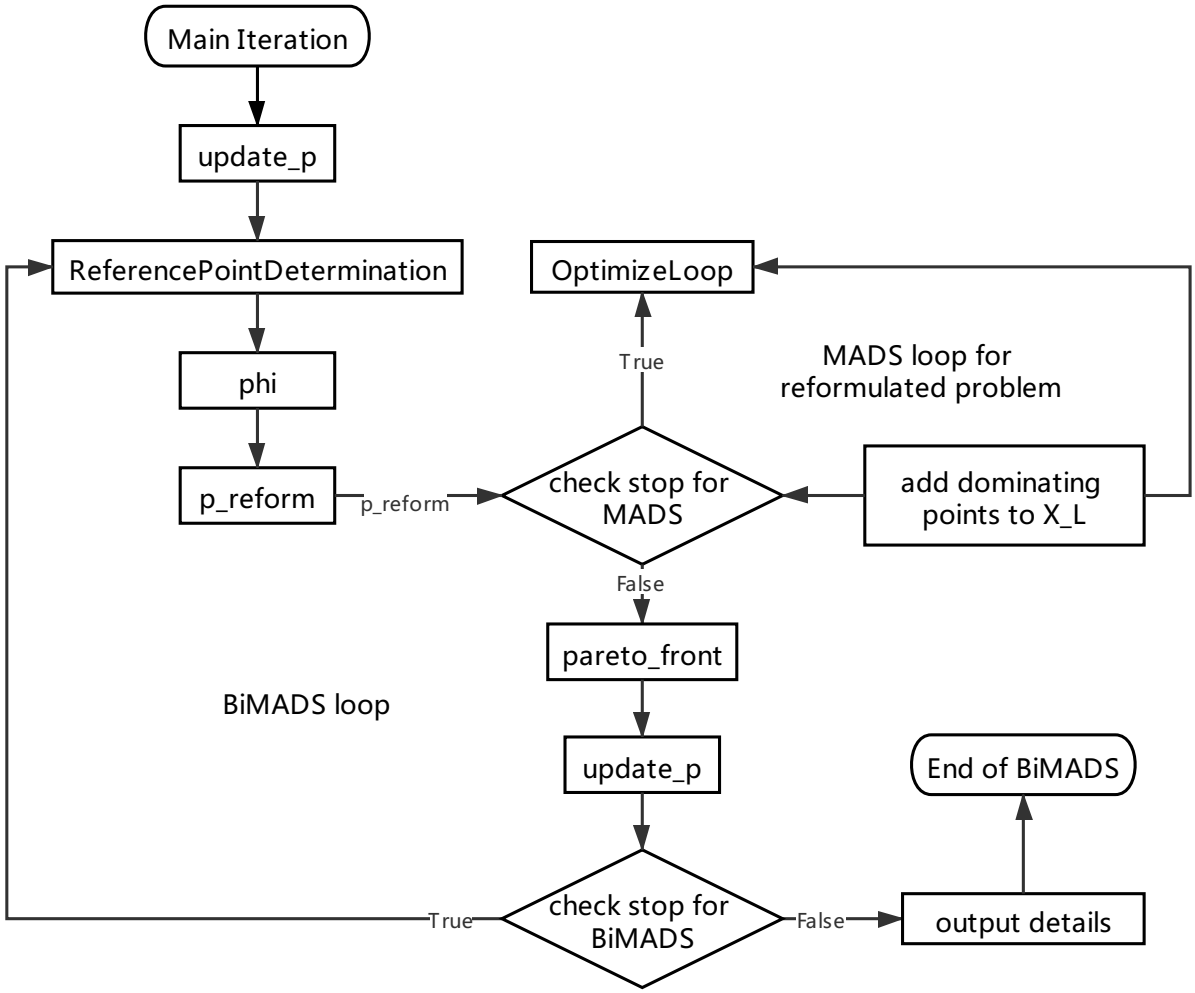


Fig. 3.2: Flow chart for Main Iteration

The only differences are the objective function and initial point which will be set later. The initialization stage also has already spent some iteration and function evaluation budgets. Therefore, the value of these two parameters should be subtracted from the total budgets in `p_reform` by function `update_p`.

In addition, the MADS for `p_reform` are nested in the BiMADS loop. However, their stopping conditions are almost the same. For instance, both of these algorithms will be terminated once reaching the iteration limit. Therefore, a new type is designed and implemented to store the BiMADS optimization status as shown in List 2, with comments for each member in the type. The status for MADS optimization will still be stored in the `p.status`, which is a member with type `status` in `DSProblem`.

Then, the following steps all need to be run repeatedly.

1. The function `ReferencePointDetermination` will be called to determine the reference point for the current iteration. If the cardinality of $X_{\mathcal{L}}$ is larger than 2, the function calculates the weighted

Listing 2 A new type named `BiMADS_status`

```
"""
Status for BiMADS main iteration, also contain stopping conditions

"""
mutable struct BiMADS_status
  iteration::Int64                #Total iteration spent
  func_evaluation::Int64          #Total function evaluation spent
  total_time::Float64             #Total running time
  hypervolume::Float64            #Hypervolume for current Pareto set
  opt_status::OptimizationStatus  #one of the stopping condition
  opt_string::String              #string for the condition above
  start_time::Float64             #start time for the optimization
  function BiMADS_status()        #inner constructor
      s=new()
      s.iteration=0
      s.func_evaluation=0
      s.total_time=0.0
      s.hypervolume=0.0
      s.opt_status=Unoptimized
      s.opt_string="Unoptimized"
      s.start_time=time()
      return s
  end
end
```

euclidean distance between three successive points and selects the group with the largest value to determine the reference point. Otherwise, the first point is selected. Also, the initial point for the reformulated problem can be obtained as the middle point (x^j) from the selected group.

2. The next step is to construct the ϕ_r by the function `phi`. The cost for each function should be calculated and stored in advance because the calculations for some objective functions might be extremely time-consuming. Then, the ϕ_r and x^j can be set as the objective function and initial point for the pre-constructed instance `p_reform`, using the provided API `SetObjective` and `SetInitialPoint`.

3. Afterwards, the `p_reform` is optimized by MADS with the normal search/poll step and stopping conditions. Similarly, the `x` value of all the `dominating iteration` should be stored along with their `f1`, `f2` cost in the set $X_{\mathcal{L}}$. This set needs to be evaluated by the function `pareto_front` to obtain the Pareto set.

4. After finishing the MADS for ϕ_r, the function `update_p` should be called again to update iteration and function evaluation limits during the main iteration loop. However, different from the `update_p` before the main iteration loop, the BiMADS optimization status also needs to be updated. Therefore,

multiply dispatch, one of Julia's unique feature, is applied to share the same namespace for different but similar functionalities. The compiler will distinguish them in terms of different combination of arguments. For example, the first function in List 3 has four arguments as inputs while the second one has three. The Julia compiler will call the corresponding function in terms of different numbers or types of inputs.

Listing 3 Multiple dispatch in Julia

```
function update_p(p, p1::DSProblem, p2::DSProblem, status::BiMADS_status)
#before Main Iteration Loop
end
function update_p(p_init::DSProblem, p_reform::DSProblem,
↪  status::BiMADS_status)
#during Main Iteration Loop
end
```

5. The design and implementations of the stopping conditions will be detailed in Section 3.2.

Once terminating the BiMADS, several useful information will be presented, such as numbers of non-dominated points (Pareto solutions), total iterations and function evaluations, total running time, hypervolume of the Pareto set and the termination reason. All the points in the Pareto set along with their `f1`, `f2` cost can be obtained as the returned value of function `Optimize!`.

3.2 Stopping Condition

Five stopping conditions have been implemented in BiMADS in `DirectSearch.jl` as:

- Iteration limit for total MADS iteration (iterations spent for $p1$, $p2$ and ϕ_r)
- Function evaluation limit for total MADS function evaluations (same as above)
- Running time limit
- Hypervolume limit
- Key interrupt

The last three conditions are unique for `DirectSearch.jl` and not implemented in `NOMAD`. The designs for the first four BiMADS stopping conditions follow the same structure as in old `DirectSearch.jl`, where three functions are required for each stopping condition:

1. Function `init_stoppingcondition` is called for the current condition to check that if the users defined limit number is legal (i.e., running time limit should be positive).

2. Function `CheckStoppingCondition` checks if the current status violates the corresponding limit.

3. Function `StoppingConditionStatus` returns a string that is the name of the current stopping condition.

The multiple dispatch is used again because although these functions have the same goal, but they will have different internal structures for different conditions. Therefore, to implement multiple dispatch, a series of types are required as the concrete types of `AbstractStoppingCondition`. `DirectSearch.jl` already has the types for the first two conditions. Thus, the new types for BiMADS are `RuntimeStoppingCondition` and `HypervolumeStoppingCondition`.

As for the internal design of the function `CheckStoppingCondition`, the values for current running time and hypervolume are stored in the type `BiMADS_status`. Hence, these functions only compare the values between current status and limits and return `false` once the optimization reaches the maximum time or expected hypervolume. In addition, here are two clarifications for the hypervolume stopping condition:

1. Two criteria could be used to terminate the optimization: 1) achieve the expected hypervolume, 2) the improvement for the hypervolume is lower than a threshold in terms of one or more iterations. However, during the tests, the author found that BiMADS algorithm will naturally stall in some iterations, which is also found in [12]. For example, in Figure 7.21, the first two iterations do not contribute much to the hypervolume. In contrast, the iteration 3-5 generate much more points. In this case, the optimization will be terminated at some stage due to the stalling phenomenon. Therefore, in the current implementation, the first criterion is applied. However, it requires the users to have at least a rough guess about the expected hypervolume.

2. The user-defined hypervolume limit cannot be achieved precisely with the same value. The reason is that the hypervolume gained in each iteration is out of control. Therefore, the final hypervolume that users get might be slightly larger than the pre-set value.

The default stopping conditions for BiMADS are only key interrupt, iteration and function evaluation limits, while the other two should be added manually. These conditions could be added using `AddStoppingCondition`.

The key interrupt stopping condition is necessary because for some time-consuming optimization problems, the users normally have little knowledge of the required time/limits. It is also too expensive to take several trials as experiments. Therefore, the key interrupt function provides a manner to stop the optimization whenever the users want (i.e., when the hypervolume does not change for a long time). Meanwhile, comparing to `CTRL+c`, the codes after the optimization can be executed as well. This functionality is realized by

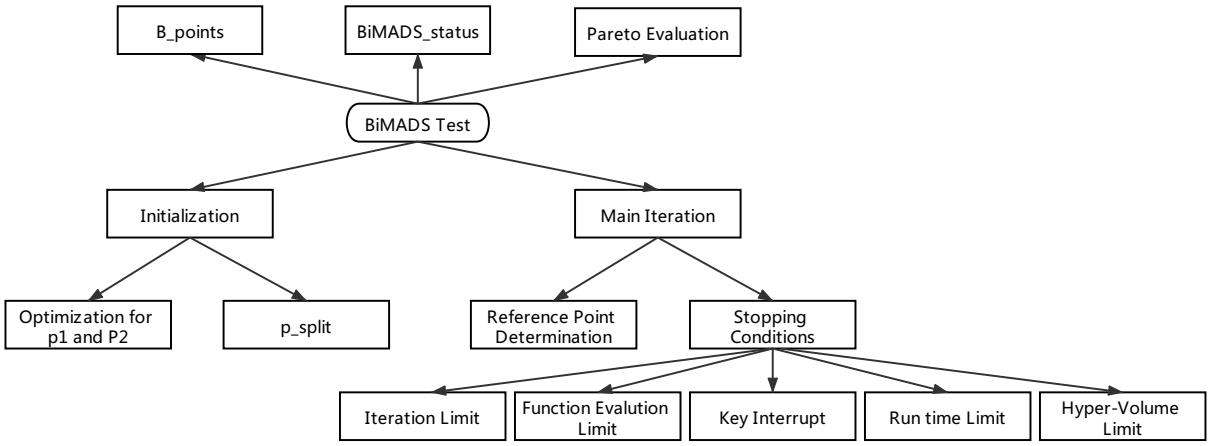


Fig. 3.3: Unit test tree

monitoring the read-eval-print loop (REPL, an interactive command-line in Julia) from the beginning of the optimization. The BiMADS will check the REPL in each iteration as below and terminates the optimization after receiving the letter "q" as an input.

3.3 Unit test

The general structure for the unit test design is shown in Figure 3.3. All the functions implemented for BiMADS have been considered and test directly or indirectly. The design of the test for each function is similar. The following part will describe the common ideal with three examples.

1. The function for checking iteration limit is tested directly by setting the limit to 1000 and checking if the optimization terminates after then. Meanwhile, function `update_p` are tested indirectly because the test will not pass if `update_p` makes a mistake.

2. The auxiliary function ϕ_r and the reference point determination function are tested by a set of giving numerical values. The outputs are then compared with the correct answers calculated by hand.

3. The error information also should be tested when something is wrong. For the initialization stage, the iteration limit is set to 10. Then, the error information is expected to be thrown to warn the users.

4 Test Problems

The developed package is tested on a building case study and a bi-objective problem set. This section will first introduce the modelling process for the building case. Then, the details for each optimization problem will also be provided. The codes in this section are also on GitHub.

4.1 Building Case Study

4.1.1 General description

The aim of the building case is to design a PID controller to ensure the internal building temperature follows the desired set-point and is robust to the external disturbance. The dynamic of the temperature $T(t)$ in the building is given by

$$\dot{T}(t) = AT(t) + c(t) + d(t), \tag{4.5}$$

where:

- A is a constant (see details in Appendix 7.3).
- $c(t)$ is the heat delivered to the flat, which is also the output of the PID controller.
- $d(t) = -AT_{ext}$ describes the disturbance function, where T_{ext} is the external temperature variation.

The objectives for the case are formulated as:

- Minimize the Integrated Absolute Error (IAE) between the current building internal temperature $T(t)$ and the pre-set temperature $T_r(t)$ as

$$\int_0^{Tf} |T(t) - T_r(t)| \, \mathrm{d}t. \tag{4.6}$$

- Minimize the maximum sensitivity

$$M_s = \max_{\omega} |S(j\omega)| \tag{4.7}$$

to external disturbance, where $S(s) = \frac{P(s)}{1+P(s)C(s)}$. M_s tells the worst-case amplification of the disturbances and $P(s),\ C(s)$ are the transfer function for the system and controller respectively. The controller is designed to minimize the maximum absolute value of $S(j\omega)$.

Both objective functions involved in the building case contain absolute operation, which causes non-differentiable points. Therefore, the derivative-free optimization is suitable in this case.

4.1.2 System Modelling

As for the system represented in Equation (4.5), $c(t)$ and $d(t)$ can be reckon as a single input, hence the transfer function in Laplace domain is obtained as

$$P(s) = \frac{T(s)}{C(s) + D(S)} \tag{4.8}$$

Also, the PID transfer function is

$$C(s) = \frac{K_d s^2 + K_p s + K_i}{s}, \tag{4.9}$$

where K_p, K_i, K_d are the proportional gain, integral gain and derivative gain, respectively.

Then, Figure 4.1 shows the block diagram for the system with a PID controller, where T_r is the pre-set temperature, $C(s)$ is transfer function of the PID controller, $P(s)$ is the transfer function for the system, $D(s)$ is the disturbance and $T(s)$ is the actual room temperature.

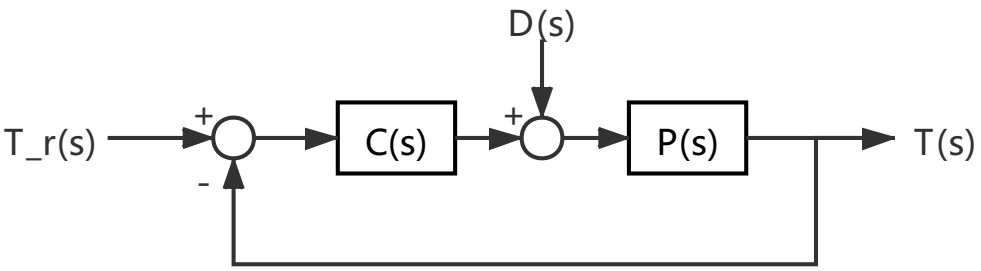


Fig. 4.1: Block diagram for the system with a PID controller

Finally, the transfer function for the system with the PID controller is formulated as

$$T(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} \cdot T_r(s) + \frac{P(s)}{1 + C(s)P(s)} \cdot D(s), \tag{4.10}$$

where $T_r(s)$ and $D(s)$ are the inputs for the system as a reference and disturbance. As for the numerical values for the other parameters in case study:

- Initial room temperature T_0 is 16 $^\circ C$
- The reference temperature T_r is changing between 16 and 20 $^\circ C$ every 5000 seconds
- The total simulation period T_f is $20,000$ seconds.
- The disturbance is a sinusoidal function as $2 + 0.5 \cdot \sin(\frac{t}{500})$.
- The initial values for the PID parameters are $K_p = K_i = K_d = 0.5$

4.1.3 System Simulation in Julia

The simulation for the system described in Equation (4.10) is realized using `ControlSystem.jl` package in Julia [23]. Firstly, the subsystem with input $T_r(s)$ in Equation (4.10) is implemented in function `Tref(p)` following the flow chart in Figure 4.2. The simulation will take the values for PID as an input and output the time response of the system within the required time period.

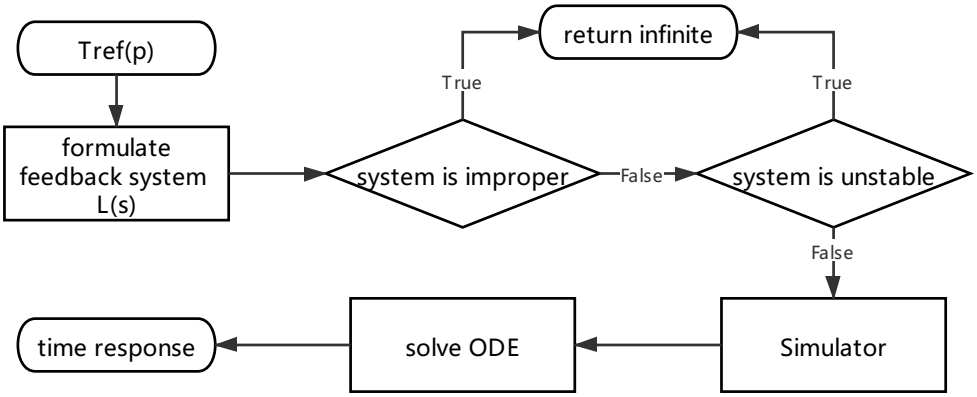


Fig. 4.2: System simulation in Julia for `Tref`

To explain, before the simulation, the transfer function for the feedback system should be proper (more poles than zeros) and stable so that the `ControlSystem.jl` will not waste time solving an unstable system using some inappropriate PID arguments. Then, a simulator is used to translate the system with given input signal to ordinary differential equations (ODEs), which is a manner that can be solved by an ODE solver. Then, the function `solve` will solve the ODEs with given initial condition x_0 within a time period `tspan`. For the simulation of the system with a long time scale, the function `solve` should be adapted to decrease the time for solving the ODEs [24]. The arguments combination with their values (selected by the author) are as followed

```
sol    = solve(s, x0, tspan,maxiters=1e7,dtmin=0.001,reltol =
↪  1e-1,force_dtmin=true)
```

- `maxiters` adjusts the maximum number of iterations before stopping. The value needs to be increased to allow a long time scale simulation.
- `dtmin` adjusts the minimum time step `dt` for adaptive timestepping (only works for adaptive method). This value needs to be large so that the solver will not waste time calculating an excessive precise solution. Meanwhile, the `force_dtmin` should be set as `true`, which allows the solver to continue when encountering the minimum time step. Otherwise, the solving process will be terminated early once `dt<dtmin`.
- `reltol` adjusts the relative tolerance in adaptive timestepping. This value needs to be large as well for the same reason as above.
- The proper method should be carefully selected for different problems (i.e., stiff or non-stiff). In this

project, the default solver of `ControlSystem.jl` - `Tsit5` (Tsitouras 5/4 Runge-Kutta method) are used to solve the ODEs.

Then, the same procedure is repeated for the subsystem with input $D(s)$ with function `Tdis(p)`. At last, the time responses obtained from two subsystems are added element-wise to get the time response for the entire system described in Equation (4.10), which is the dynamic of the actual room temperature within $20,000$ seconds.

Integrated absolute value

The time response obtained from the simulation above has the type of `Vector{Float64}`. Therefore the IAE can be calculated using

```
function IAE(p)
    f_DS(p)=mean(abs, ref .- (Tref(p) .+ Tdis(p)))  # IAE
    return [f_DS]
end
```

This function could also be fed into `DirectSearch.jl` as an objective function with the PID parameters as an input.

Maximum sensitivity

$S(s)$ in Equation (4.1.1) can be evaluated using function `bode` to get the magnitude for its frequency response. The output `mag` has the type of `Vector{Float64}`. Then, the largest value in this set is the maximum sensitivity.

Finally, the bi-objective problem for the building case is formulated as in List 4.

Listing 4 Bi-objective problem for building case

```
function Building_bimads(p)
    IAE(p)=mean(abs, ref .- (Tref(p) .+ Tdis(p)))  # IAE
    sen(p)=Disturbance(p) # Maximum sensitivity
    return [IAE, sen]
end
```

4.2 Test Functions Set

The Julia and C++ packages for BiMADS will be tested on 20 bi-objective optimization problems selected from [14, 25, 26, 27, 28]. The comparison will be made regarding the running time and number function

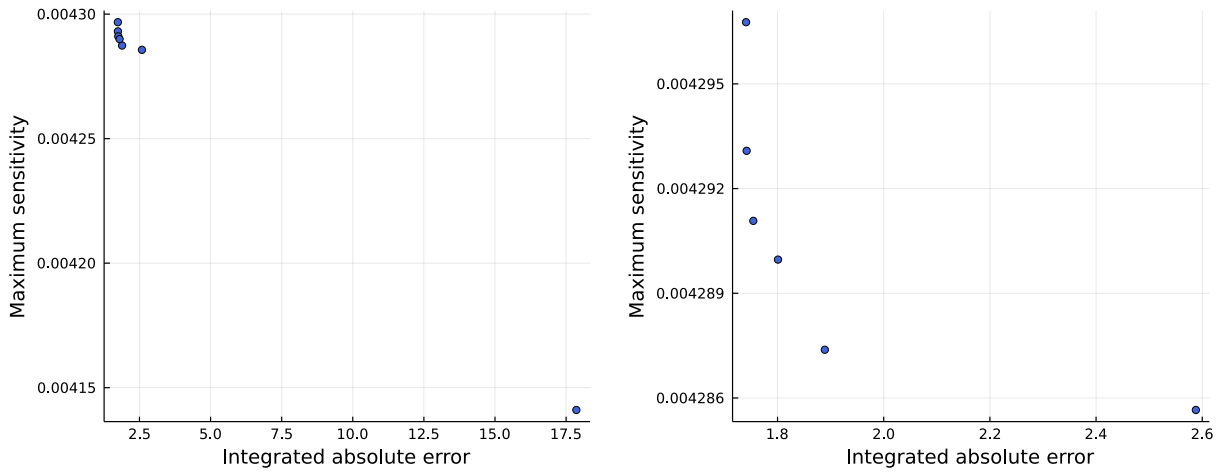
Table 4.1: Selected Problems with their dimensions and main features

Problems	**m**	**Mian Features**	**Problems**	**m**	**Main Features**
ZDT1 [26]	30	convex	ZDT2 [26]	30	concave
ZDT3 [26]	30	discontinuous	ZDT4 [26]	10	multimodality
ZDT6 [26]	10	bias	F5.1 [4]	2	convex
F5.2 [4]	2	concave	F5.3 [4]	2	discontinuous
F5.4 [4]	2	bias	F5.5 [4]	2	non-uniform
F1 [27]	5	convex	DTLZ1 [26]	2	multimodality
MOP2 [26]	4	concave	MOP3 [26]	2	convex
MOP4 [26]	3	discontinuous	MOP6 [26]	2	mixed front
FES1 [26]	10	mixed front	QV1 [26]	10	concave
OKA1 [25]	2	bias	FAR1 [26]	2	mixed front

evaluation used to achieve the maximum hypervolume (accurate to two decimal places). The selected problems all have certain difficulties for finding the Pareto optimum front, such as:

- The multimodality problems have multiple minima, where the optimizer might become stuck in.
- The bias problems have more solutions that exist in some areas of objective space than in others, which affects the convergence speed toward the Pareto front.
- The geometries of the Pareto fronts can be convex, concave, discontinuous or even some combination of former (mixed front) [26]. Also, the solutions along the Pareto front can be non-uniform distributed, which means the densities of the points are different in different areas. Thus, the optimizer might waste time searching in the gaps or fail to find all regions of the Pareto front.
- The dimension of the variable is large, so is the search space. The optimizer will need more function evaluations to find a better point.

All the selected problems are presented in Table 4.1 with dimension m and their main features. The mathematical expressions and the Pareto fonts for all the problems are detailed in Appendix 7.2, where $f_1(x)$ and $f_2(x)$ are the objective functions for all problems.



(a) Pareto front

(b) Pareto front (zoom in)

Fig. 5.1: Pareto front for the building case study

5 Result and Evaluation

This section provides the results of the building case study with evaluation. Then the test results of `DirectSearch.jl` and `NOMAD` on the same bi-objective problem set are presented and compared. At last, a short result for the unit test is shown with the measurement of the code coverage.

5.1 Building case study

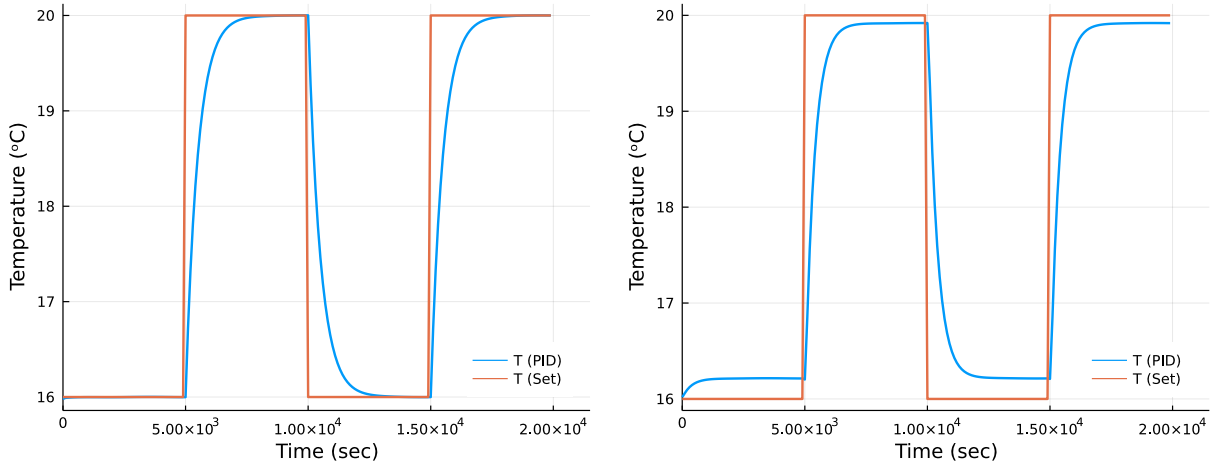
5.1.1 Result

The bi-objective problem is formulated as in List 4 with initial condition $p = [0.5, 0.5, 0.5]$. The iteration limit is set to 100 because the simulation of the long time period system is quite time-consuming. The Pareto front is finally obtained as shown in Figure 5.1(a), where the minimum value for IAE and maximum sensitivity is 1.74 and 0.00414, respectively. However, the corresponding IAE for the minimum sensitivity is 17.86, which is too large and ignored. Thus, the upper left part is zoomed in as in Figure 5.1(b). The numerical costs for these six points with their PID parameters are shown in Table 5.2. Also, the result of the initial condition and the ignored point are also point 0 and 7 in Table 5.2.

The Pareto font in Figure 5.1(b) shows that although the points at two ends have small values in terms of one criterion but large values for the other. Thus, the middle points are preferable for the building case. The time responses for point 4 and 5 are shown in Figure 5.2, which have comparatively small IAE and sensitivity at the same time.

Table 5.2: Numerical costs and PID parameters

point	IAE	Max sensitivity	PID (K_p, K_i, K_d)	point	IAE	Max sensitivity	PID (K_p, K_i, K_d)
0	6.071	0.005578	0.50, 0.50, 0.50	1	1.740	0.004297	-0.75, 0.62, 0.08
2	1.741	0.004293	-0.55, 0.62, 0.08	3	1.754	0.004291	-0.45, 0.67, 0.10
4	1.800	0.004289	-0.25, 0.47, 0.17	5	1.889	0.004287	-0.25, 0.47, 0.10
6	2.588	0.004285	-0.18, 0.47, 0.15	7	17.86	0.004142	7.96, 0.16, 10.99



(a) Time response for point 4

(b) Time response for point 5

Fig. 5.2: Time responses for point 4 and 5

5.1.2 Evaluation

The results show that the actual room temperature could follow the pre-set value and resist the external disturbance at the same time. The parameter for point 4 (Figure 5.2(a)) has a smaller IAE, hence the temperature follows the setpoint better but is more sensitive to the disturbance. In contrast, the parameter for point 4 (Figure 5.2(b)) enables the system to be more robust to the disturbance but has a large IAE as a trade-off. Due to the small maximum sensitivity for all the Pareto solutions, the effect from disturbance is not obvious in both cases.

Also, the points are non-uniformly distributed on the Pareto front, where most of the solutions have small IAE but large sensitivity. The reason is that for most PID parameters that provide small sensitivity, the corresponding IAE is extremely large. Therefore, most of these points will be dominated by the lower right point in Figure 5.1(a), which is the solution with the smallest sensitivity.

Lastly, learning from the experience of solving this bi-objective problem, several general suggestions

might be helpful to solve other problems:

1. If it takes a much longer time to get the cost for one of the objective functions, users are suggested to implement single-objective optimization first for this function to get the optimum solution. Then, the solution could be set as the initial point for the bi-objective optimization, which may save the time used for the BiMADS initialization stage.

2. If one of the objectives is more important than the other, the users are suggested to run the bi-objective optimization only for a few BiMADS iterations first to see the Pareto front. Then, the users could narrow the search space by setting the constraints for the objective functions so that the BiMADS will not waste time searching in a useless space.

   For example, IAE is more important than sensitivity because the sensitivity is already a small value for the building case. Without the constraints, BiMADS will waste time filling the large gap in Figure 5.1(a), which is unnecessary due to the very large IAE. After focusing on a narrow space such as in Figure 5.1(b), more valuable points with reasonable IAE and comparatively small sensitivity can be found.

5.2 Bi-objective problem set

5.2.1 Result

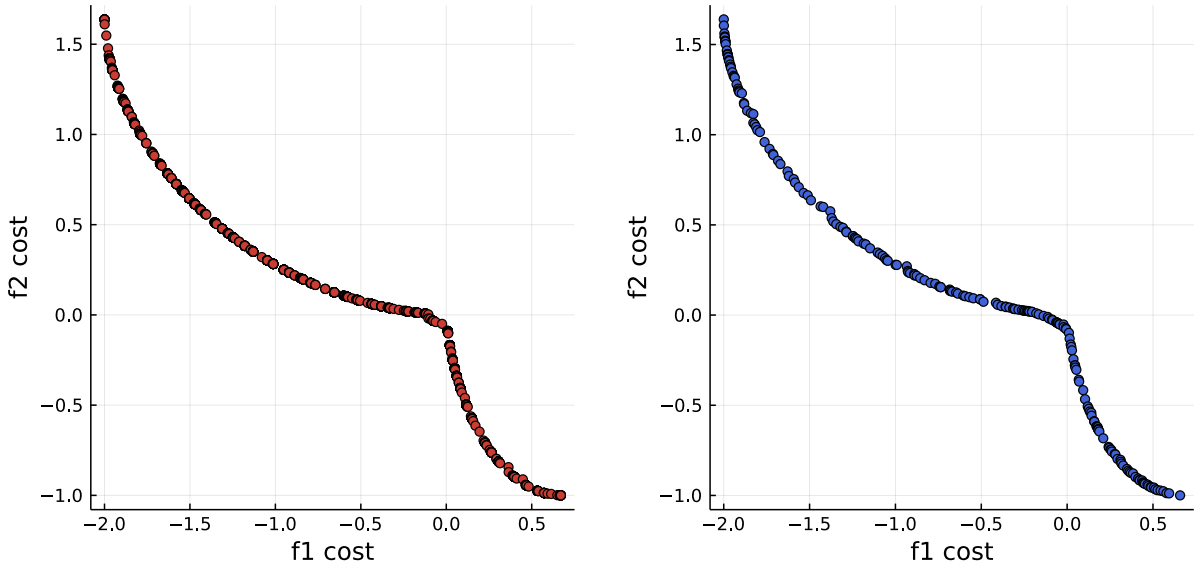
To ensure `NOMAD` and `DirectSearch.jl` run the optimization in a similar way, some of the settings must be the same. For example:

- The initial point, function evaluation budgets for each BiMADS iteration should be the same.
- Turn off the Nelder Mead Search and Speculative search in `NOMAD` because `DirectSearch.jl` does not have these two algorithms for search step.

As a result, all the Pareto fronts generated by `DirectSearch.jl` are shown in Appendix 7.2. Figure 5.3 shows the Pareto fronts of FAR1 generated by two packages as an example. Most of the results from `NOMAD` are similar to `DirectSearch.jl`, hence they will not be presented individually. The running time, the number of function evaluations and the number of total generated solutions for both packages are shown in Table 5.3. All the results are the mean value from 10 instances because BiMADS is a non-deterministic algorithm. In addition, `NOMAD` struggles to optimize several complex problems and cannot provide the Pareto fronts with enough hypervolume, even after a long time. These problems will be labelled with $*$ and the listed values are for their longest running time.

Table 5.3: Summary of the test for both packages

	Running time (sec)		**Function evaluations**		**Total points**	
Problem	DirectSearch	NOMAD	DirectSearch	NOMAD	DirectSearch	NOMAD
ZDT1* [26]	5.38	480	82099	4524	355	52
ZDT2* [26]	6.63	511	34795	5323	423	66
ZDT3* [26]	7.22	309	94106	3320	695	32
ZDT4 [26]	0.75	394	86819	4198	879	26
ZDT6 [26]	3.58	420	63502	3897	820	119
F5.1 [4]	1.82	24	18398	607	923	147
F5.2 [4]	1.54	23	18828	602	873	132
F5.3 [4]	1.92	48	19672	1207	809	201
F5.4 [4]	0.37	18	5880	632	164	110
F5.5 [4]	0.39	15	6392	617	373	107
F1 [27]	0.26	17	11291	609	287	113
DTLZ1 [26]	1.90	22	21694	1205	995	115
MOP2 [26]	0.21	32	5116	607	202	135
MOP3 [26]	7.02	92	79591	1728	1037	587
MOP4 [26]	1.19	123	28493	2391	302	211
MOP6 [26]	0.41	45	8913	870	387	175
FES1 [26]	0.85	102	23297	1654	302	105
QV1 [26]	0.50	43	5350	625	221	95
OKA1 [25]	4.8	89	42789	5168	151	44
FAR1 [26]	1.03	34	4902	626	401	214



(a) Result from `DirectSearch.jl`

(b) Result from `NOMAD`

Fig. 5.3: Pareto fronts for FAR1, which is the mixed front with both convex and concave elements

According to Table 5.3, the performance profiles are plotted in Figure 5.4 by `BenchmarkProfiles.jl` for the comparison of two packages, which illustrates the proportion of the problems that the solver can solve within a factor τ of the best solver.

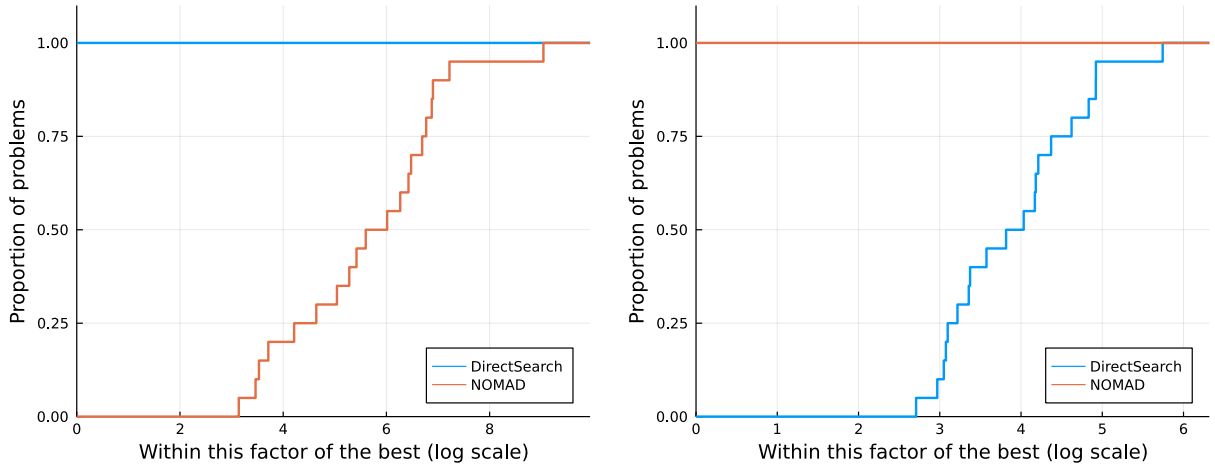
5.2.2 Evaluation

The package `DirectSearch.jl` could solve all the problems within a short time, while `NOMAD` requires an extremely long time to solve three of them. A common feature for these three problems is that they all have large dimensions of variable space ($m = 30$). Also, from the performance profile, in terms of the running time (Figure 5.4(a)), `NOMAD` always requires much more time than `DirectSearch.jl` to achieve the same hypervolume. In contrast, the number of function evaluations spent by `DirectSearch.jl` is always much more than `NOMAD` (Figure 5.4(b)). Due to a large number of function evaluations, the number of generated points on the Pareto front is also larger than in `NOMAD`.

From the evaluations above, several features for two packages can be concluded as:

1. Comparing with `NOMAD`, `DirectSearch.jl` tends to solve the problems by evaluating much more trial points in all directions to find a new Pareto solution. Benefiting from the fast speed of Julia, `DirectSearch.jl` could evaluate more points while maintaining a short total running time.

2. As for `NOMAD`, each function evaluation takes a much longer time, especially for the high dimensional problem, which needs the evaluations on more trial points to find a better solution. Hence, the `NOMAD`



(a) For running time

(b) For function evaluation

Fig. 5.4: Performance profiles

first utilizes some searching algorithms such as Nelder Mead Search for the search step in MADS (not in this project). This step will search around all the space to find the better solution instead of only focusing on the current incumbent solution. Furthermore, `NOMAD` chooses the trail points in a more clever way so that it can achieve the optimum solution by only evaluating a limited number of points.

3. The number of the points generated by `DirectSearch.jl` is much larger than in `NOMAD`, which means users have more options when selecting the Pareto solutions.

4. If the obtained Pareto front is not as expected (i.e., too few points), the function evaluation limit for each BiMADS iteration should be increased first instead of only increasing the BiMADS iterations. The reason is that for some difficult problems (i.e., non-uniform Pareto front), the evaluation budget for each iteration is crucial to evaluate enough trial points for reformulated function ϕ_r. Hence, more points could be generated in a single BiMADS iteration.

In summary, `DirectSearch.jl` relies on a large number of function evaluations and has a fast evaluating speed for a single trail point. Therefore, it is suitable for problems with any dimensions. However, some problems naturally take a long time to return their costs in each evaluation (i.e., simulation of long-period systems). These problems are more suitable for `NOMAD` because it could find the Pareto solutions with a limited number of function evaluations.

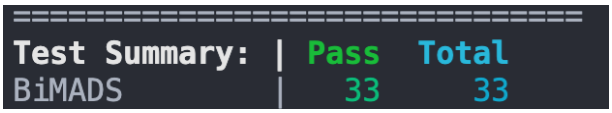


Fig. 5.5: Result of the unit test

5.3 Unit test

The result of the unit test is shown in Figure 5.5, which shows that all the test functions designed for BiMADS have passed the unit test. The Julia package `Coverage.jl` is used to track each line of the code and count how many times does it run. This is called a code coverage test, which reveals the parts of the code that is not tested yet and may have bugs. After applying this test, the code coverage for the designed unit test is 100% so that all the code implementations for BiMADS have been tested.

6 Conclusions and Future Work

6.1 Conclusions

This project extended the Julia optimization package `DirectSearch.jl` with the BiMADS algorithm, which enables the package to solve bi-objective problems and get the Pareto optimum set. Apart from the original algorithm, several new stopping conditions are added to improve the flexibility of the package. The developed package was first tested on a real-life PID design problem, which aims to control the room temperature and resist the effect from outside disturbance. Then, the performance of BiMADS in `DirectSearch.jl` was compared with a state-of-art bi-objective optimization solver `NOMAD` on a test set. The implementation of BiMADS in `DirectSearch.jl` is outstanding in terms of running time and provides more flexibility for the users.

6.2 Future work

1. The hypervolume indicator only support bi-objective optimization and there is no advanced algorithm applied for the calculation. In the future, the indicator should be extended to multi-dimension to support multi-objective optimization. Also, a new algorithm should be introduced to improve the speed because the calculations for high dimensional hypervolume is time-consuming.

2. One of the drawbacks for BiMADS is that the algorithm may stall in some iteration, which wastes the function evaluation budget searching in a narrow space. Inspired from DMulti-MADS [12], the optimization of the reformulated function ϕ_r needs to be adapted: 1) To avoid getting stuck in a narrow space, the search step for MADS should not be skipped. 2) Multi-polling may be used to implement

poll step on multiple trail points to increase the searching space.

3. As for the code in Julia, multiple dispatch can be used on formulating the bi-objective problem. A new type with the same namespace `DSProblem` can be designed, which inherits from `AbstractProblem`. To distinguish from old `DSProblem`, the new type will have several new input arguments such as total BiMADS iteration limit and hypervolume limit. The new type could simplify the code structure, improve efficiency and benefit future development.

References

[1] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," *arXiv preprint arXiv:1904.11585*, 2019.

[2] C. Dejemeppe, P. Schaus, and Y. Deville, "Derivative-free optimization: Lifting single-objective to multi-objective algorithm," in *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2015, pp. 124–140.

[3] J.-h. Ryu, S. Kim, and H. Wan, "Pareto front approximation with adaptive weighted sum method in multiobjective simulation optimization," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE, 2009, pp. 623–633.

[4] C. Audet, G. Savard, and W. Zghal, "Multiobjective optimization through a series of single-objective formulations," *SIAM Journal on Optimization*, vol. 19, no. 1, pp. 188–210, 2008.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[6] J. Bezanson, J. Chen, B. Chung, S. Karpinski, V. B. Shah, J. Vitek, and L. Zoubritzky, "Julia: Dynamism and performance reconciled by design," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–23, 2018.

[7] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization*, ser. Springer Series in Operations Research and Financial Engineering. Cham, Switzerland: Springer International Publishing, 2017.

[8] C. Audet and J. E. Dennis, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on optimization*, vol. 17, no. 1, pp. 188–217, 2006.

[9] A. Abraham and L. Jain, "Evolutionary multiobjective optimization," in *Evolutionary Multiobjective Optimization*. Springer, 2005, pp. 1–6.

[10] J. L. Cohon, *Multiobjective programming and planning*. Courier Corporation, 2004, vol. 140.

[11] S. Shan and G. G. Wang, "An efficient pareto set identification approach for multiobjective optimization on black-box functions," *J. Mech. Des.*, vol. 127, pp. 866–874, 2005.

[12] J. Bigeon, S. Le Digabel, and L. Salomon, "Dmulti-mads: mesh adaptive direct multisearch for bound-constrained blackbox multiobjective optimization," *Computational Optimization and Applications*, vol. 79, no. 2, pp. 301–338, 2021.

[13] E. P. Stables, "Static Optimization in the Julia Programming Language," 2020, Imperial College London. [Unpublished Final Year Project Report].

[14] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary computation*, vol. 8, no. 2, pp. 173–195, 2000.

[15] M. A. Abramson, C. Audet, J. E. Dennis Jr, and S. L. Digabel, "OrthoMADS: A deterministic MADS instance with orthogonal directions," *SIAM Journal on Optimization*, vol. 20, no. 2, pp. 948–966, 2009.

[16] C. Audet, S. Le Digabel, and C. Tribes, "The mesh adaptive direct search algorithm for granular and discrete variables," *SIAM Journal on Optimization*, vol. 29, no. 2, pp. 1164–1189, 2019.

[17] S. Le Digabel, "Algorithm 909: Nomad: Nonlinear optimization with the mads algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 37, no. 4, pp. 1–15, 2011.

[18] L. Baliumas, "Black-box dynamic optimisation," 2021, Imperial College London. [Unpublished Final Year Project Report].

[19] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms—a comparative case study," in *International conference on parallel problem solving from nature*. Springer, 1998, pp. 292–301.

[20] H. Ishibuchi, R. Imada, Y. Setoguchi, and Y. Nojima, "How to specify a reference point in hypervolume calculation for fair performance comparison," *Evolutionary computation*, vol. 26, no. 3, pp. 411–440, 2018.

[21] E. D. Dolan and J. J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical programming*, vol. 91, no. 2, pp. 201–213, 2002.

[22] H. Zhu, P. A. Hall, and J. H. May, "Software unit test coverage and adequacy," *Acm computing surveys (csur)*, vol. 29, no. 4, pp. 366–427, 1997.

[23] F. B. Carlson and M. Fält, "ControlSystems.jl : A Control Systems Toolbox for Julia," 2016.

[24] C. Rackauckas and Q. Nie, "DifferentialEquations. jl–a performant and feature-rich ecosystem for solving differential equations in Julia," *Journal of Open Research Software*, vol. 5, no. 1, 2017.

[25] Y. Davidor, H.-P. Schwefel, and R. Maenner, *Parallel Problem Solving from Nature-PPSN III: International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, Jerusalem, Israel, October 9-14, 1994. Proceedings*. Springer Science & Business Media, 1994, vol. 3.

[26] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.

[27] D. Brockhoff, T. Tusar, A. Auger, and N. Hansen, "Using well-understood single-objective functions in multiobjective black-box optimization test suites," *arXiv preprint arXiv:1604.00359*, 2016.

[28] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 1. IEEE, 2002, pp. 825–830.

7 Appendix

7.1 User Guide

The documentation for the current public version of `DirectSearch.jl` explains how to optimize a single-objective problem with MADS in detail. However, the code implemented in this project has not been released so far. Therefore, a simple instruction on how to optimize a bi-objective problem MOP4 (see Section 7.2.15) is shown below:

1. Objective functions should be first declared as a function:

```
function obj(x)
  f1(x) = sum(-10 * exp.(-0.2 * sqrt.(x[1:2].^2 + x[2:3].^2)))
  f2(x) = sum(abs.(x).^0.8 + 5 * sin.(x.^3))
    return [f1,f2]
end
```

2. The `DSProblem` is then formulated with initial condition as

```
p = DSProblem(3; objective=obj,initial_point=[0., 0., 0.])
```

3. The parameters such as iteration limit and function evaluation limit are set in the same way as shown in the public documentation.

4. The `key interrupt` stopping condition are activated by default. The other two new stopping conditions can be set as

```
AddStoppingCondition(p, HypervolumeStoppingCondition(1.3))#hypervolume
↪  limit
AddStoppingCondition(p, RuntimeStoppingCondition(2.0))#run time limit
```

5. Run the optimization using

```
result=Optimize!(p)
```

   `result` is of the type `Vector{B_points}` and stores all the Pareto solutions with their costs in terms of `f1` and `f2`. The costs can be used for plotting the Pareto front as

```
fig=plot_Bpoint(result)
```

   Based on the Pareto front, the users can select the solutions they prefer.

7.2 Bi-objective Problem set with Pareto fronts

7.2.1 ZDT1 [26]

Dimension of variables: $m = 30$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_m) &= 1 + 9 \cdot \sum_{i=2}^{m} x_i/(m-1) \\ h(f_1, g) &= 1 - \sqrt{f_1/g} \\ f_2(x_1, \ldots, x_m) &= g \cdot h \end{aligned} \tag{7.11}$$

Constraints: $x_1, \ldots, x_m \in [0, 1]$

Features: Convex, Large dimension

Hypervolume: 1.44

Pareto front

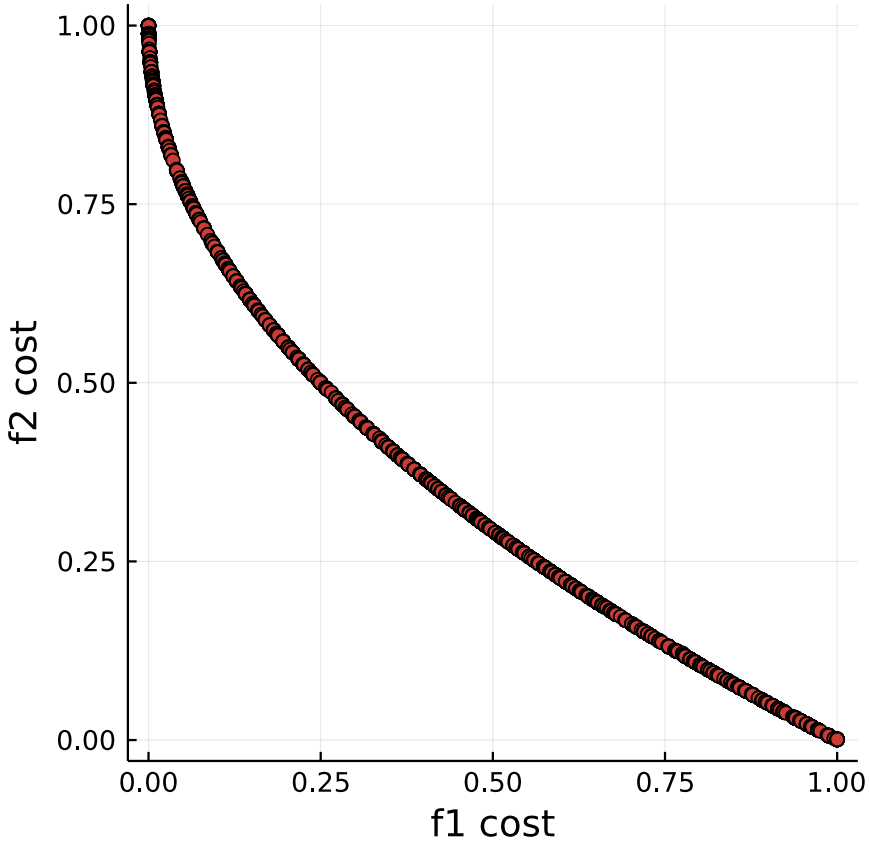


Fig. 7.1: Pareto front for ZDT1

7.2.2 ZDT2 [26]

Dimension of variables: $m = 30$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_m) &= 1 + 9 \cdot \sum_{i=2}^{m} x_i/(m-1) \\ h(f_1, g) &= 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1) \\ f_2(x_1, \ldots, x_m) &= g \cdot h \end{aligned} \tag{7.12}$$

Constraints: $x_1, \ldots, x_m \in [0, 1]$

Features: Concave, Large dimension

Hypervolume: 0.90

Pareto front

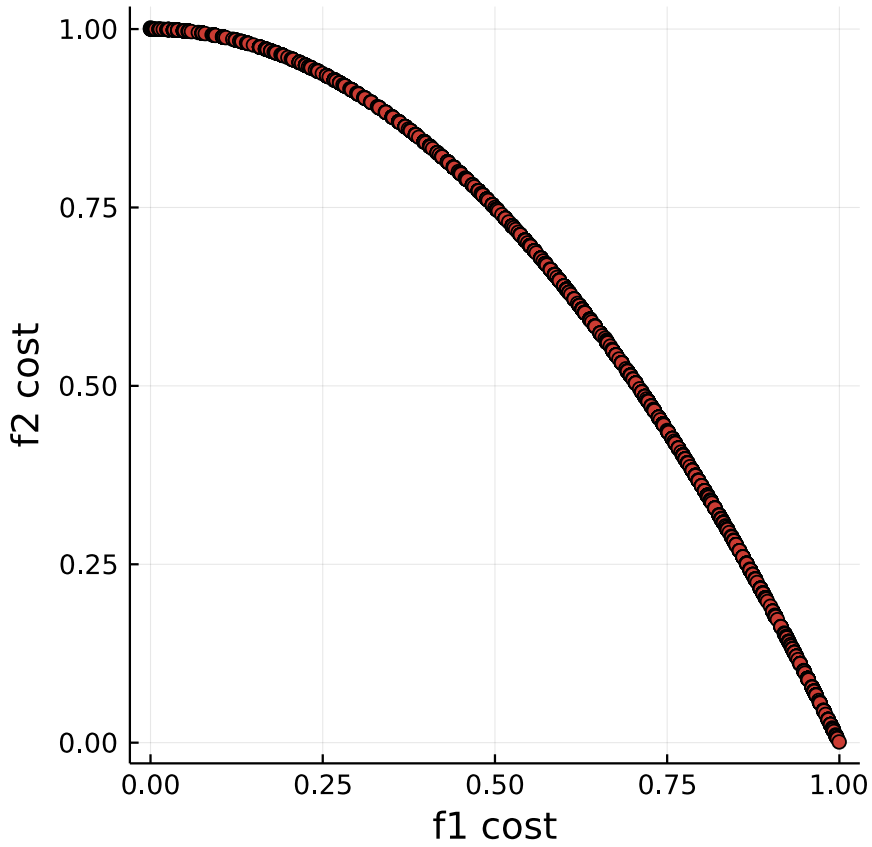


Fig. 7.2: Pareto front for ZDT2

7.2.3 ZDT3 [26]

Dimension of variables: $m = 30$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_m) &= 1 + 9 \cdot \sum_{i=2}^{m} x_i/(m-1) \\ h(f_1, g) &= 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1) \\ f_2(x_1, \ldots, x_m) &= g \cdot h \end{aligned} \tag{7.13}$$

Constraints: $x_1, \ldots, x_m \in [0, 1]$

Features: Discontinuous, Convex, Large dimension

Hypervolume: 1.17

Pareto front

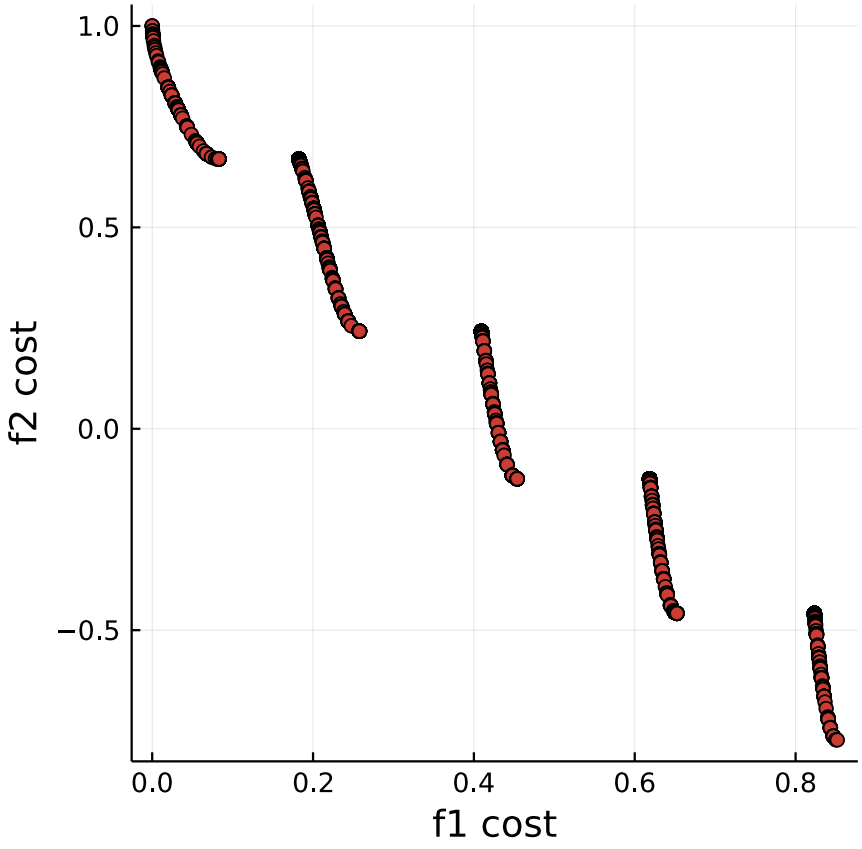


Fig. 7.3: Pareto front for ZDT3

7.2.4 ZDT4 [26]

Dimension of variables: $m = 10$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, \ldots, x_m) &= 1 + 10(m-1) + \sum_{i=2}^{m} \left(x_i^2 - 10\cos(4\pi x_i)\right) \\ h(f_1, g) &= 1 - \sqrt{f_1/g} \\ f_2(x_1, \ldots, x_m) &= g \cdot h \end{aligned} \tag{7.14}$$

Constraints: $x_1 \in [0, 1]$, and $x_2, \ldots, x_m \in [-5, 5]$

Features: Multimodality

Hypervolume: 1.91

Pareto front

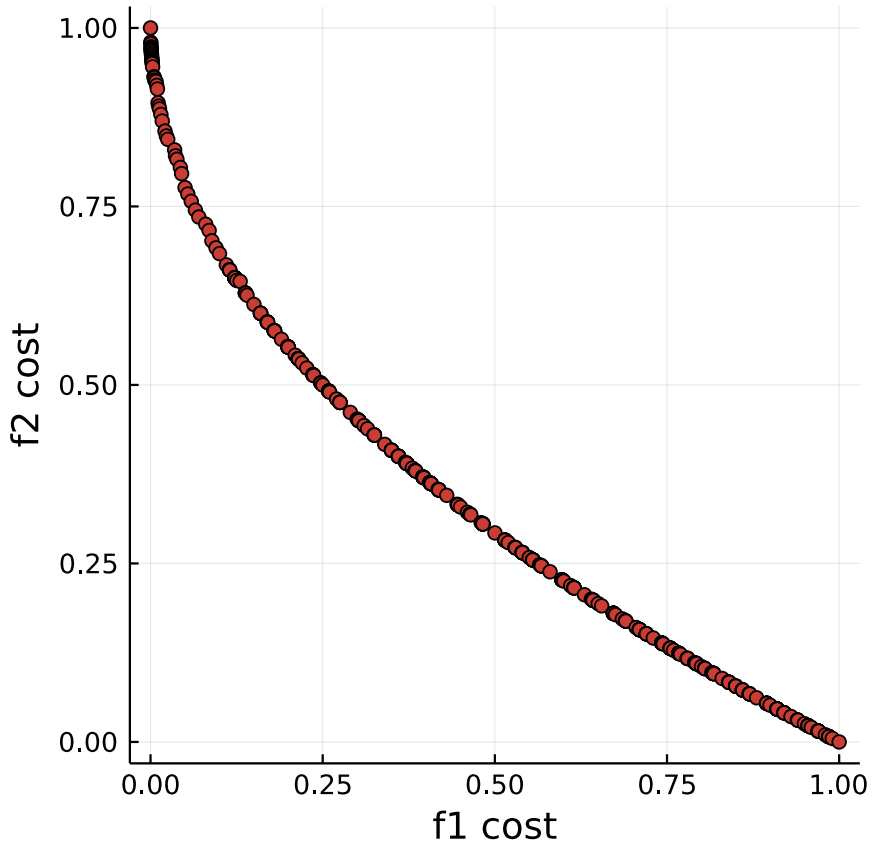


Fig. 7.4: Pareto front for ZDT4

7.2.5 ZDT6 [26]

Dimension of variables: $m = 10$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= 1 - \exp(-4x_1)\sin^6(6\pi x_1) \\ g(x_2,\ldots,x_m) &= 1 + 9\cdot\left(\left(\sum_{i=2}^{m} x_i\right)/(m-1)\right)^{0.25} \\ h(f_1,g) &= 1 - (f_1/g)^2 \\ f_2(x_1,\ldots,x_m) &= g\cdot h \end{aligned} \tag{7.15}$$

Constraints: $x_1,\ldots,x_m \in [0,1]$

Features: Bias: non-uniform search space.

Non-uniform: density of the solutions is lowest near the Pareto front.

Hypervolume: 1.17

Pareto front

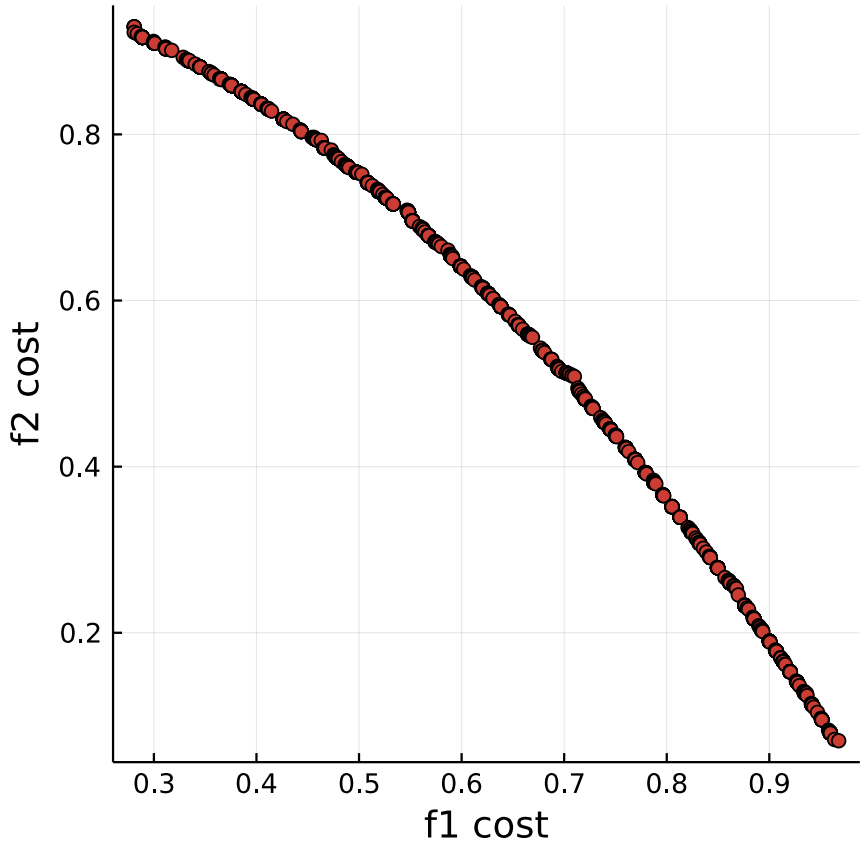


Fig. 7.5: Pareto front for ZDT6

7.2.6 F5.1 [4]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= 4x_1 \\ g\left(x_2\right) &= \begin{cases} 4 - 3\exp\left(-(\frac{x_2-0.2}{0.02})^2\right) & \text{if } 0 \leq x_2 \leq 0.4 \\ 4 - 2\exp\left(-(\frac{x_2-0.7}{0.2})^2\right) & \text{if } 0.4 \leq x_2 \leq 1 \end{cases} \\ h\left(f_1, g\right) &= \begin{cases} 1 - \left(\frac{f_1}{g}\right)^{0.25} & \text{if } f_1 \leq g \\ 0 & \text{otherwise} \end{cases} \\ f_2(x_1, x_2) &= g \cdot h \end{aligned} \tag{7.16}$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Convex

Hypervolume: 1.66

Pareto front

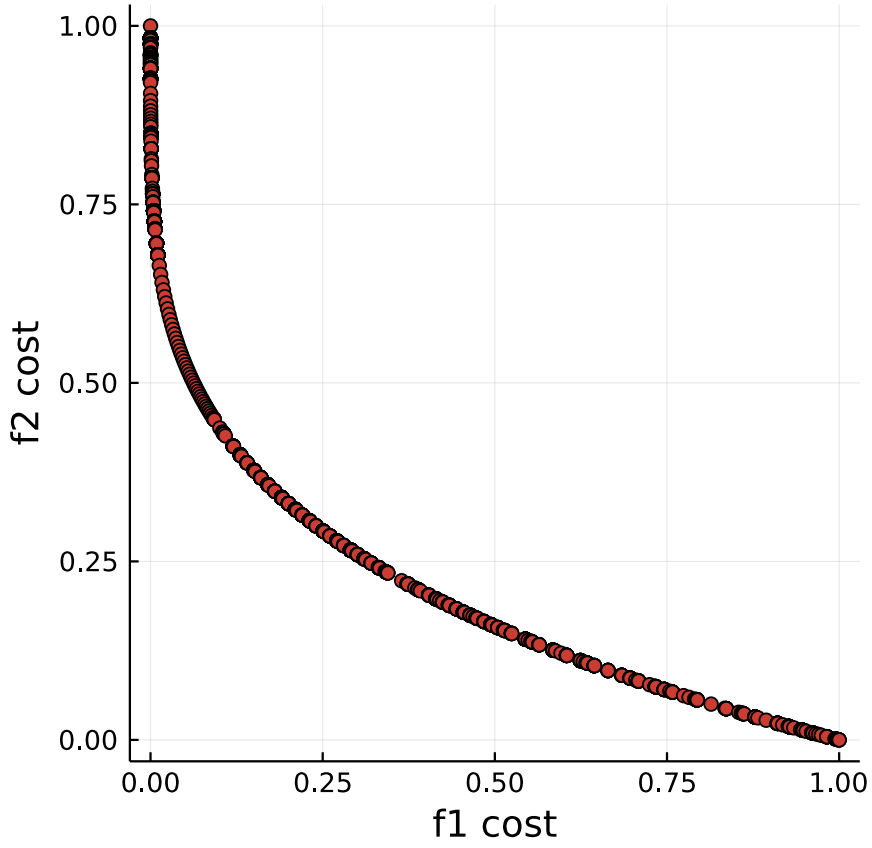


Fig. 7.6: Pareto front for F5.1

7.2.7 F5.2 [4]

Dimension of variables: $m = 2$

Problem Statement:

$$
\begin{aligned}
f_1(x_1) &= 4x_1 \\
g\left(x_2\right) &= \begin{cases} 4 - 3\exp\left(-(\frac{x_2-0.2}{0.02})^2\right) & \text{if } 0 \leq x_2 \leq 0.4 \\ 4 - 2\exp\left(-(\frac{x_2-0.7}{0.2})^2\right) & \text{if } 0.4 \leq x_2 \leq 1 \end{cases} \\
h\left(f_1, g\right) &= \begin{cases} 1 - \left(\frac{f_1}{g}\right)^4 & \text{if } f_1 \leq g \\ 0 & \text{otherwise} \end{cases} \\
f_2(x_1, x_2) &= g \cdot h
\end{aligned}
\tag{7.17}
$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Concave

Hypervolume: 0.68

Pareto front

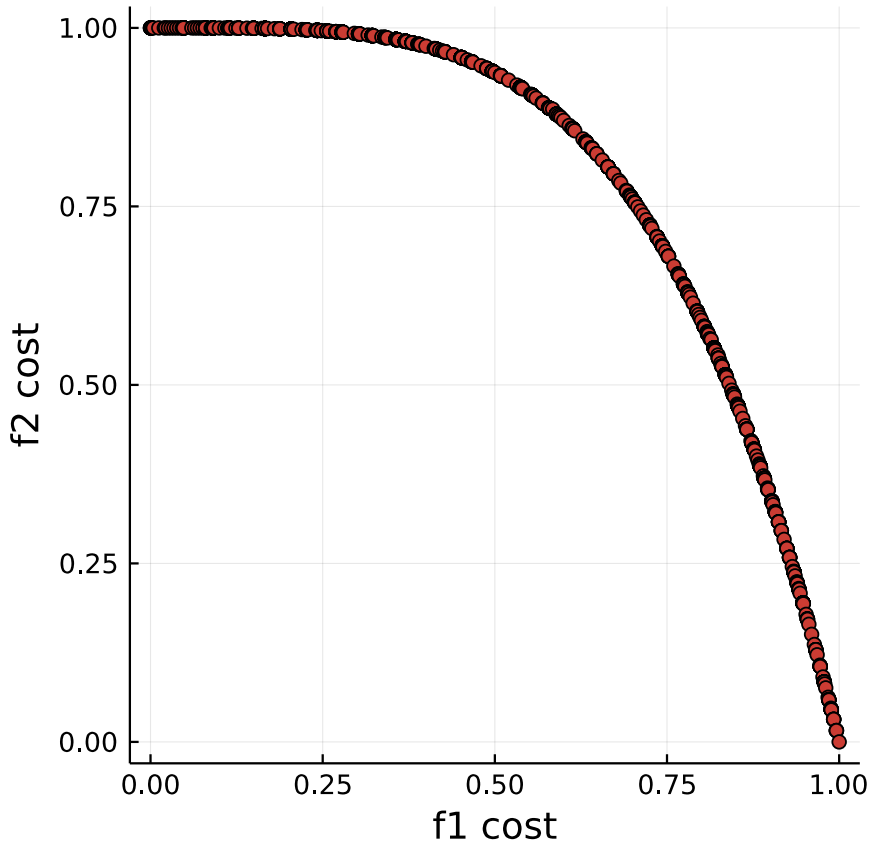


Fig. 7.7: Pareto front for F5.2

7.2.8 F5.3 [4]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2) &= 1 + 10x_2 \\ h\left(f_1, g\right) &= 1 - \left(\frac{f_1}{a}\right)^2 - \frac{f_1}{g}\sin\left(8\pi f_1\right) \\ f_2(x_1, x_2) &= g \cdot h \end{aligned} \tag{7.18}$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Discontinuous

Hypervolume: 0.85

Pareto front

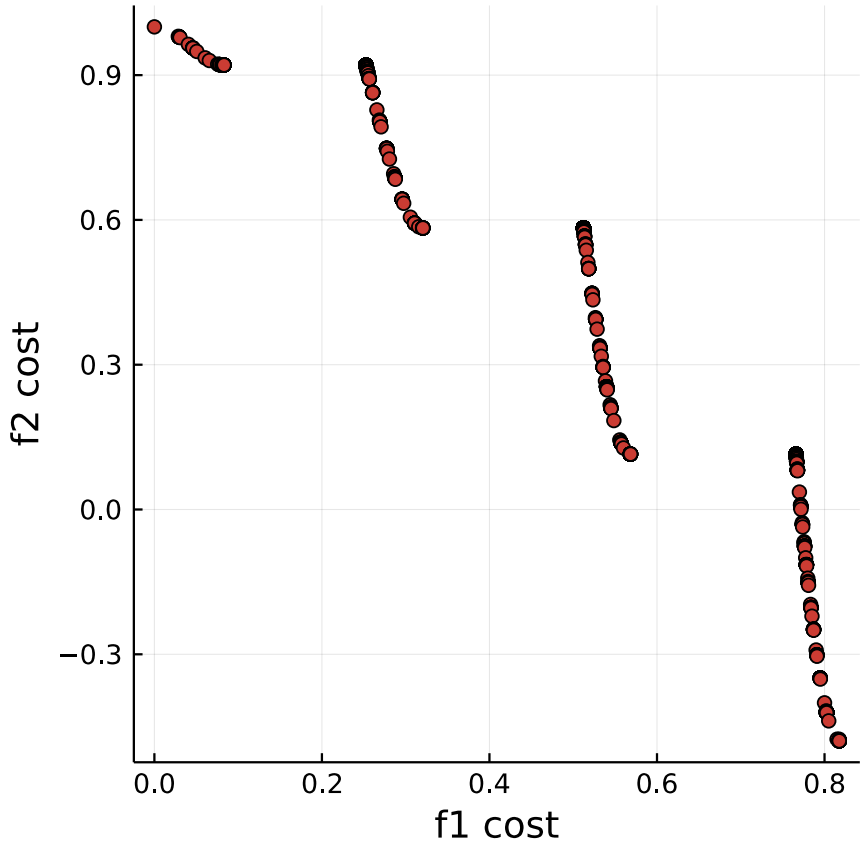


Fig. 7.8: Pareto front for F5.3

7.2.9 F5.4 [4]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g\left(x_2\right) &= 1 + x_2^{0.25} \\ h\left(f_1, g\right) &= 1 - \left(\frac{f_1}{g}\right)^2 \\ f_2(x_1, x_2) &= g \cdot h \end{aligned} \tag{7.19}$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Concave, Bias: non-uniform search space.

Hypervolume: 0.89

Pareto front

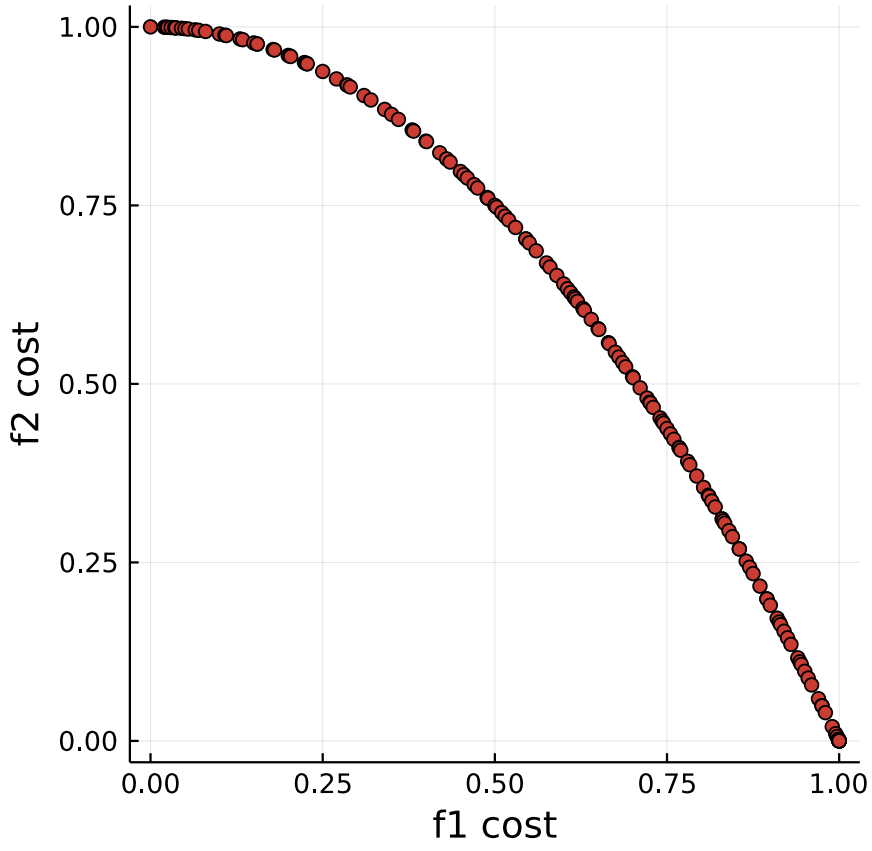


Fig. 7.9: Pareto front for F5.4

7.2.10 F5.5 [4]

Dimension of variables: $m = 2$

Problem Statement:

$$
\begin{aligned}
f_1(x_1) &= 1 - \exp(-4x_1)\sin^4(5\pi x_1) \\
h(f_1, g) &= \begin{cases} 1 - \left(\frac{f_1}{g}\right)^4 & \text{if } f_1 \le g, \\ 0 & \text{otherwise} \end{cases} \\
g(x_2) &= \begin{cases} 4 - 3\exp\left(-\frac{x_2 - 0.2}{0.02}\right)^2 & \text{if } 0 \le x_2 \le 0.4 \\ 4 - 2\exp\left(-\frac{x_2 - 0.7}{0.2}\right)^2 & \text{if } 0.4 \le x_2 \le 1 \end{cases} \\
f_2(x_1, x_2) &= g \cdot h
\end{aligned}
\tag{7.20}
$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Concave, Non-uniform: density of the solutions is lowest near the Pareto front.

Hypervolume: 1.861

Pareto front

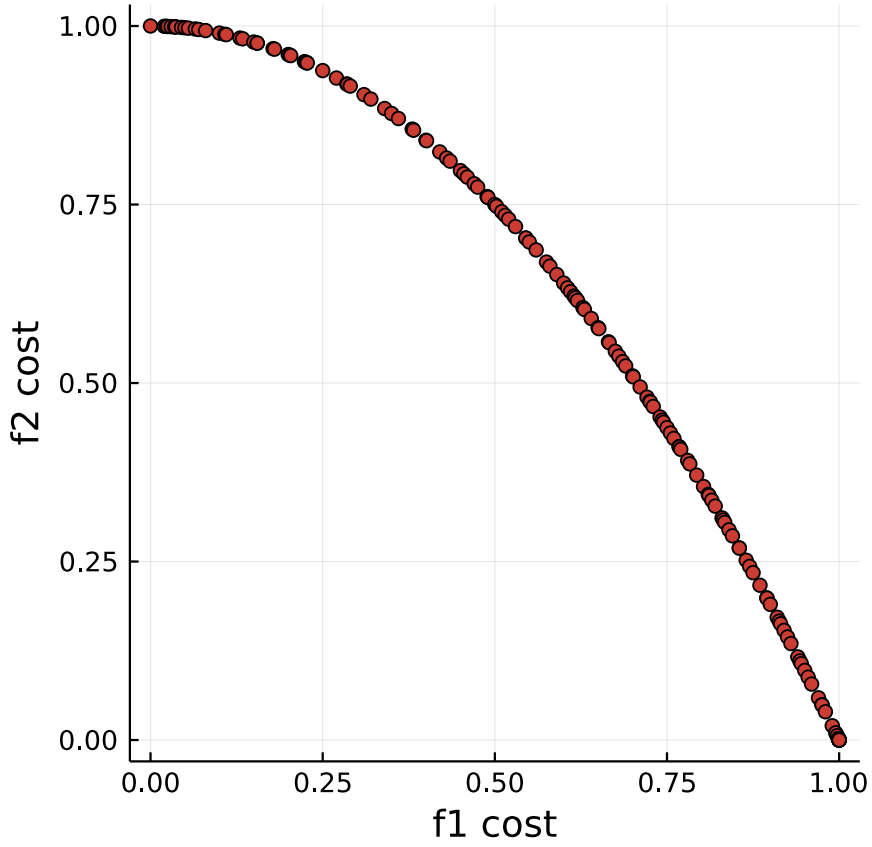


Fig. 7.10: Pareto front for F5.4

7.2.11 F1 [27]

Dimension of variables: $m = 5$

Problem Statement:

$$f_1(\mathbf{x}) = \left(\sum_i^m x_i^2\right)^2 - 20$$
$$f_2(\mathbf{x}) = \left(\sum_i^m x_i^2\right)^2 - 10 \tag{7.21}$$

Constraints: $x_1, \cdots, x_m \in [-5, 5]$

Features: Convex

Hypervolume: 1.20

Pareto front

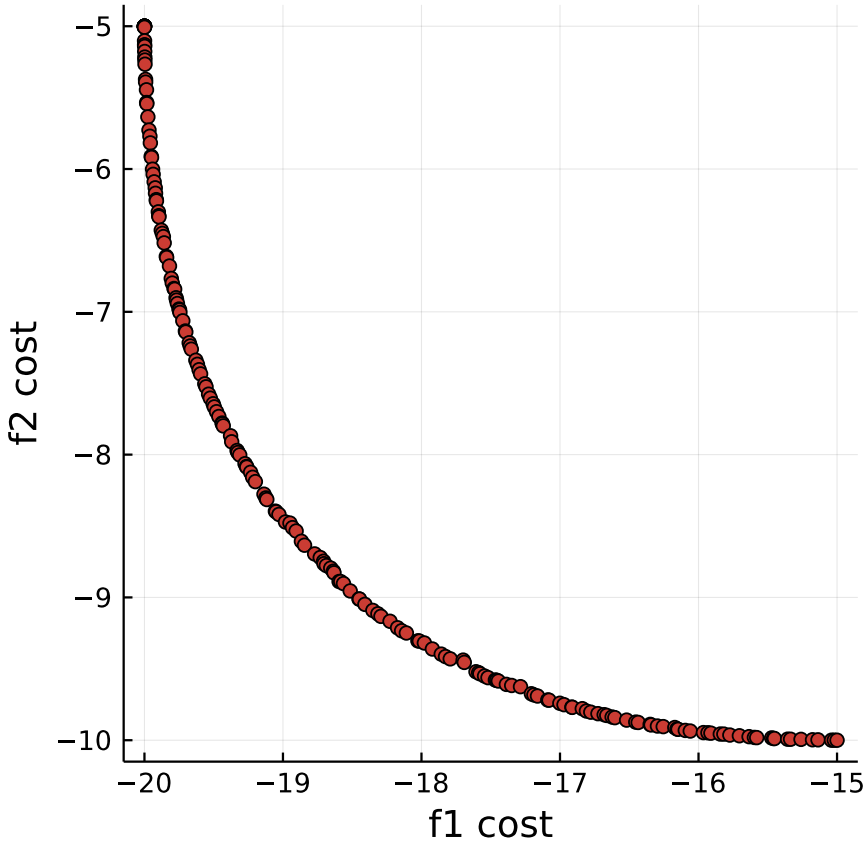


Fig. 7.11: Pareto front for F1

7.2.12 DTLZ1 [26]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} g(x_1) &= 100\left[1 + (x_1 - 0.5)^2 - \cos\left(20\pi\left(x_1 - 0.5\right)\right)\right] \\ f_1(x_1) &= (1+g)\cdot 0.5 \cdot x_1 \\ f_2(x_1, x_2) &= 0.5 \cdot (1+g) \cdot x_1 \cdot (1 - x_2) \end{aligned} \tag{7.22}$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Multimodality, linear

Hypervolume: 1.17

Pareto front

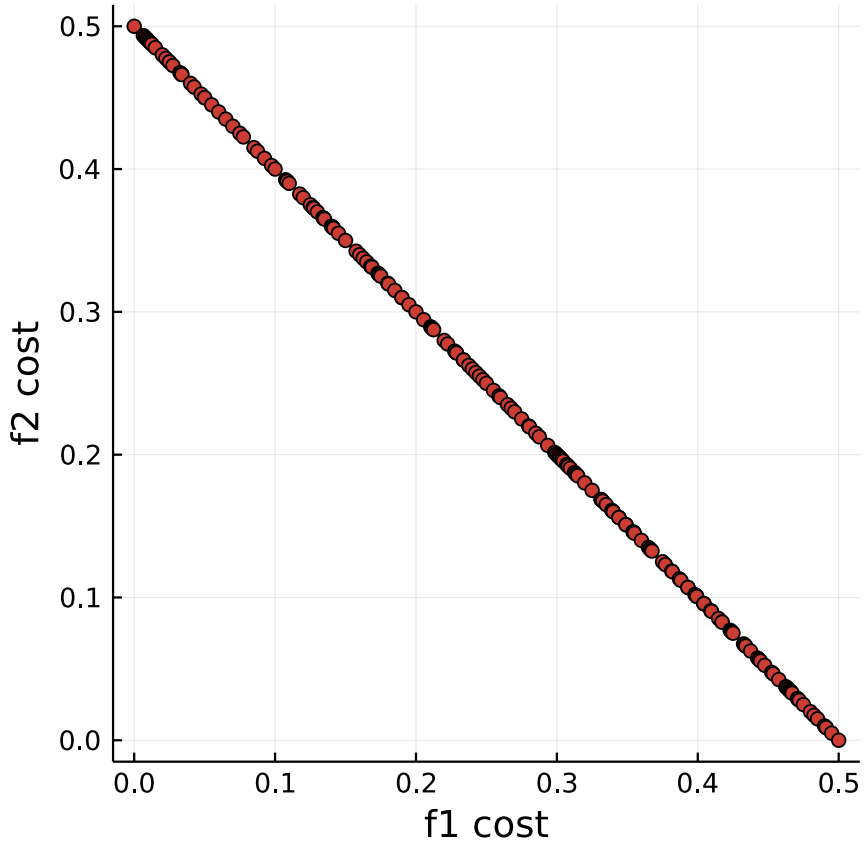


Fig. 7.12: Pareto front for DTLZ1

7.2.13 MOP2 [26]

Dimension of variables: $m = 3$

Problem Statement:

$$\begin{aligned} f_1(x_1, \ldots, x_m) &= 1 - \exp\left(-\sum_{i=1}^{m}\left(x_i - 1/\sqrt{m}\right)^2\right) \\ f_2(x_1, \ldots, x_m) &= 1 - \exp\left(-\sum_{i=1}^{m}\left(x_i + 1/\sqrt{m}\right)^2\right) \end{aligned} \tag{7.23}$$

Constraints: $x_1, \cdots, x_m \in [-4, 4]$

Features: Concave

Hypervolume: 0.86

Pareto front

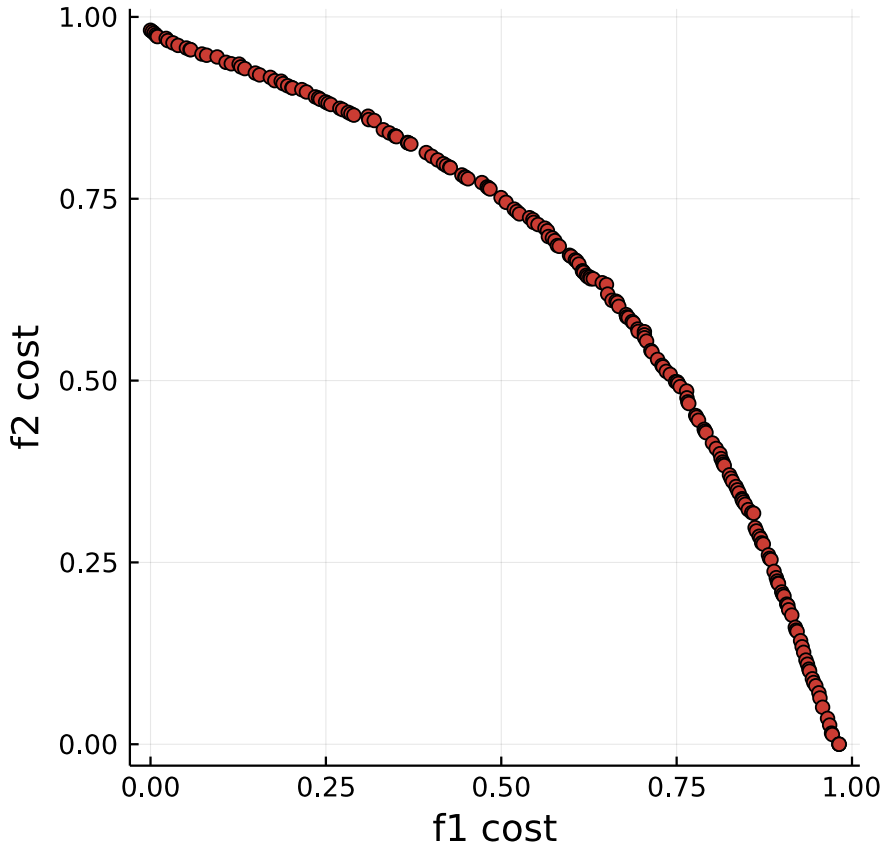


Fig. 7.13: Pareto front for MOP2

7.2.14 MOP3 [26]

Dimension of variables: $m = 2$

Problem Statement:

$$
\begin{aligned}
f_1(x_1, x_2) &= 1 + (A_1 - B_1)^2 + (A_2 - B_2)^2 \\
f_2(x_1, x_2) &= (x_1 + 3)^2 + (x_2 + 1)^2 \\
A_1 &= 0.5\sin(1) - 2\cos(1) + \sin(2) - 1.5\cos(2) \\
A_2 &= 1.5\sin(1) - \cos(1) + 2\sin(2) - 0.5\cos(2) \\
B_1 &= 0.5\sin(x_1) - 2\cos(x_1) + \sin(x_2) - 1.5\cos(x_2) \\
B_2 &= 1.5\sin(x_1) - 1\cos(x_1) + 2\sin(x_2) - 0.5\cos(x_2)
\end{aligned}
\tag{7.24}
$$

Constraints: $x_1, x_2 \in [-\pi, \pi]$

Features: Discontinuous, Convex

Hypervolume: 1.85

Pareto front

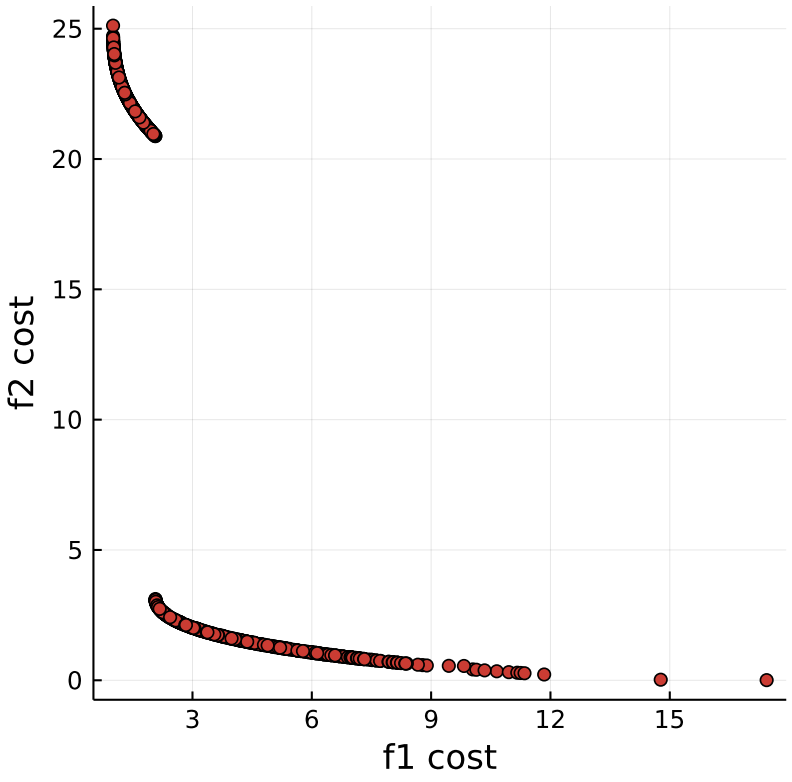


Fig. 7.14: Pareto front for MOP3

7.2.15 MOP4 [26]

Dimension of variables: $m = 3$

Problem Statement:

$$\begin{aligned} f_1(x_1, \cdots, x_m) &= \sum_{i=1}^{2} -10 \exp^{-0.2\sqrt{x_i^2 + x_{i+1}^2}} \\ f_2(x_1, \cdots, x_m) &= \sum_{i=1}^{3} |x_i|^{0.8} + 5 \sin\left(x_i^3\right) \end{aligned} \tag{7.25}$$

Constraints: $x_1, \cdots, x_m \in [-5, 5]$

Features: Degenerate points (single point on the Pareto front), Mixed front

Hypervolume: 0.642

Pareto front

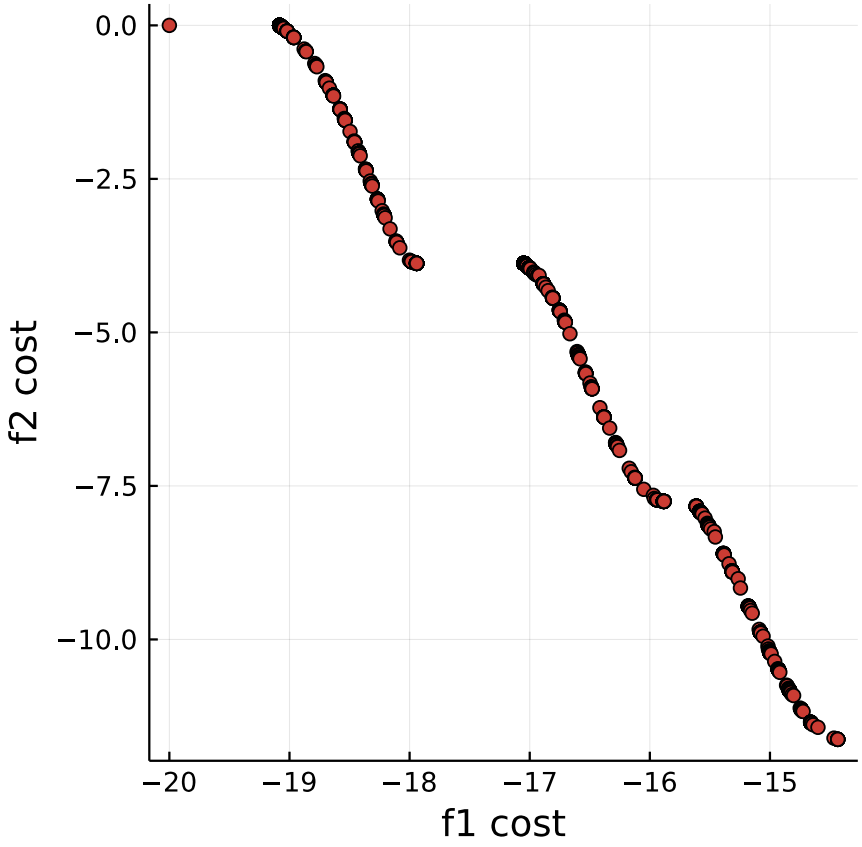


Fig. 7.15: Pareto front for MOP4

7.2.16 MOP6 [26]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} f_1(x_1) &= x_1 \\ f_2(x_1, x_2) &= (1 + 10x_2)\left(1 - \left(\frac{x_1}{1 + 10x_2}\right)^2 - \frac{x_1}{1 + 10x_2}\sin(8\pi x_1)\right) \end{aligned} \tag{7.26}$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Discontinuous, Multimodality, Mixed front

Hypervolume: 0.80

Pareto front

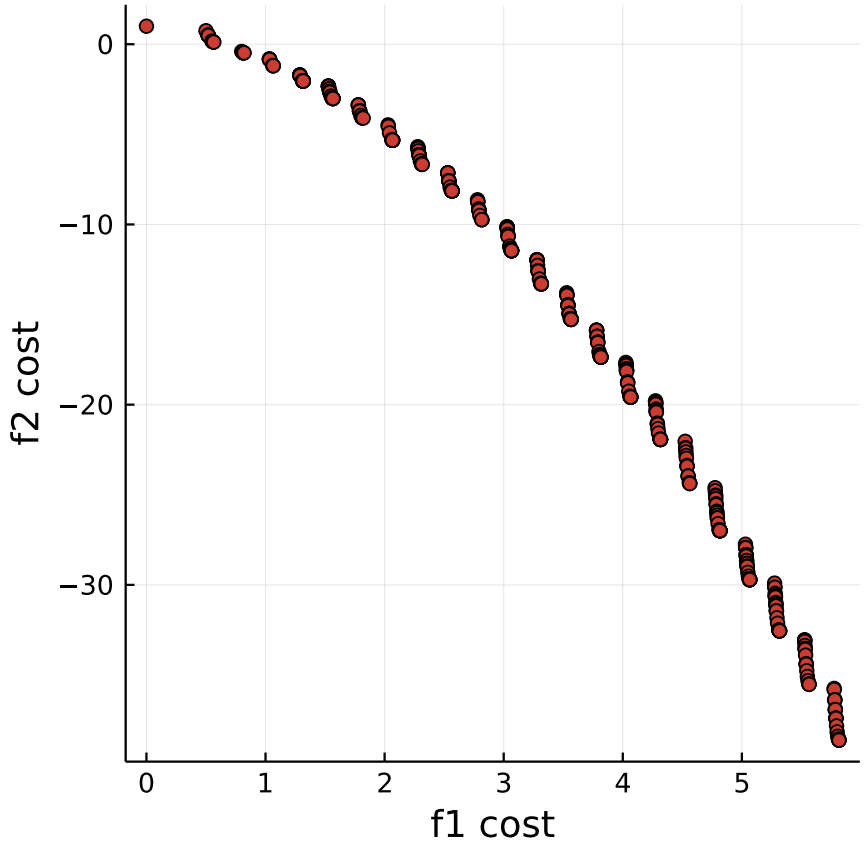


Fig. 7.16: Pareto front for MOP4

7.2.17 FES1 [26]

Dimension of variables: $m = 10$

Problem Statement:

$$\begin{aligned} f_1(x_1, \ldots, x_m) &= \sum_{i=1}^{m} \left| x_i - \exp\left((i/m)^2\right)/3 \right|^{0.5} \\ f_2(x_1, \ldots, x_m) &= \sum_{i=1}^{m} \left(x_i - 0.5\cos(10\pi i/m) - 0.5\right)^2 \end{aligned} \tag{7.27}$$

Constraints: $x_1, \cdots, x_m \in [0, 1]$

Features: Mixed front

Hypervolume: 0.80

Pareto front

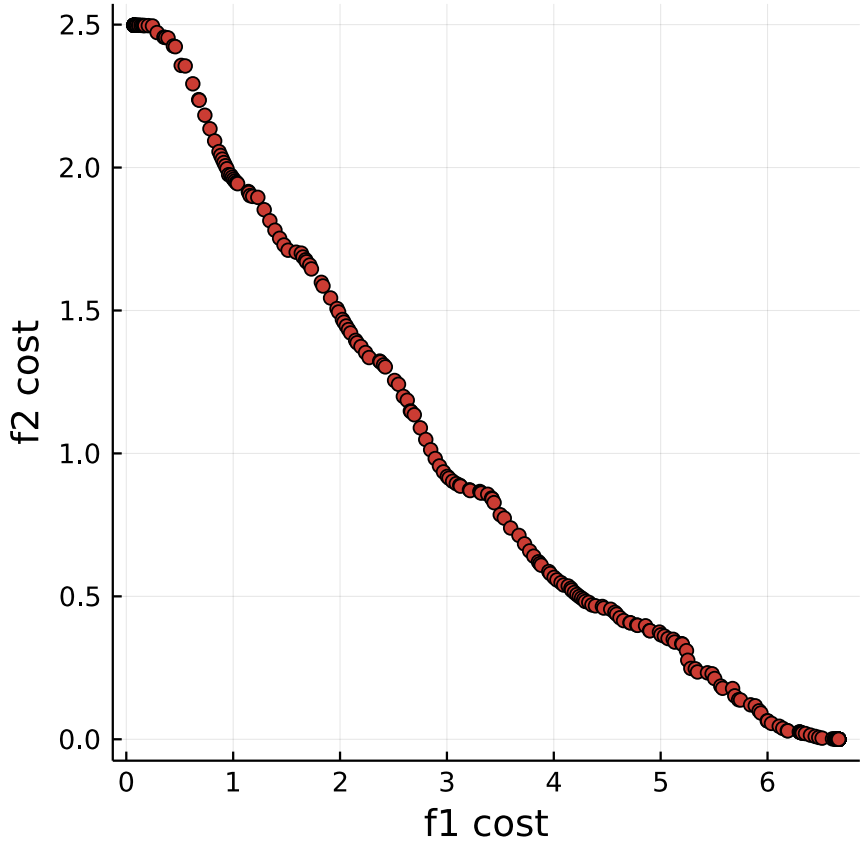


Fig. 7.17: Pareto front for FES1

7.2.18 QV1 [26]

Dimension of variables: $m = 10$

Problem Statement:

$$\begin{aligned} f_1(x_1, \ldots, x_m) &= \left[\sum_{i=1}^{m} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)/m\right]^{0.25} \\ f_2(x_1, \ldots, x_m) &= \left[\sum_{i=1}^{m} \left((x_i - 1.5)^2 - 10\cos(2\pi(x_i - 1.5)) + 10\right)/m\right]^{0.25} \end{aligned} \tag{7.28}$$

Constraints: $x_1, \cdots, x_m \in [0, 1]$

Features: Multimodality, Concave

Hypervolume: 0.87

Pareto front

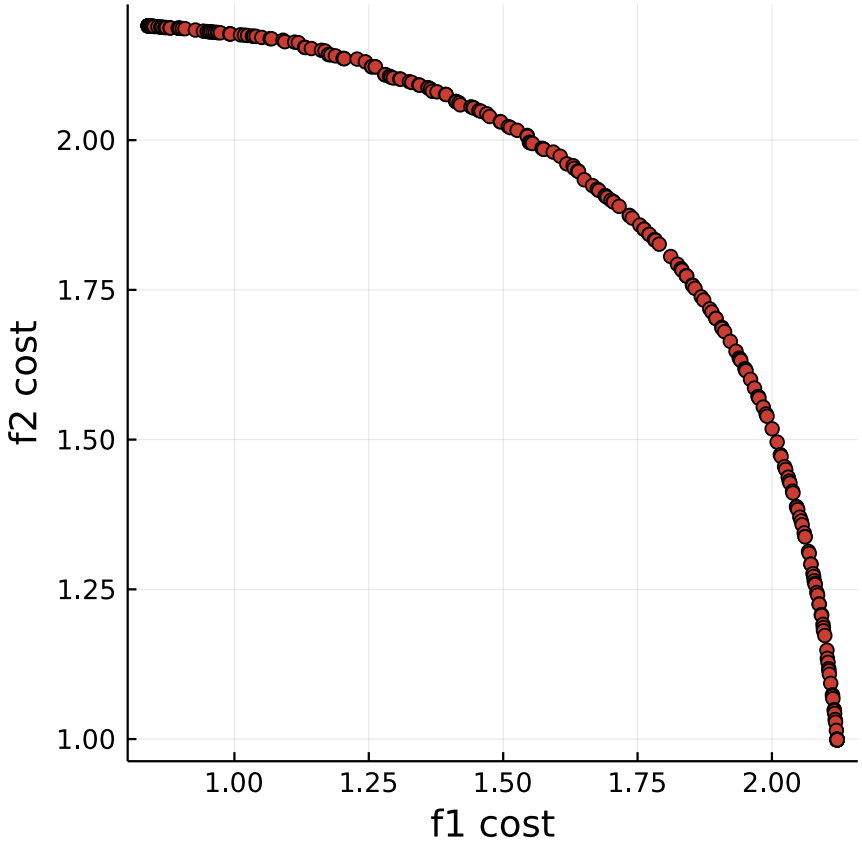


Fig. 7.18: Pareto front for QV1

7.2.19 OKA1 [25]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} y_1 &= x_1 \cdot \cos(\pi/12) - x_2 \cdot \sin(\pi/12) \\ y_2 &= x_1 \cdot \sin(\pi/12) + x_2 \cdot \cos(\pi/12) \\ f_1(x_1, x_2) &= y_1 \\ f_2(x_1, x_2) &= \sqrt{2\pi} - \sqrt{|y_1|} + 2|y_2 - 3\cos(y_1) - 3|^{1/3} \end{aligned} \tag{7.29}$$

Constraints: $x_1 \in [6\sin(\pi/12), 6\sin(\pi/12) + 2\pi\cos(\pi/12)]$, $x_2 \in [-2\pi\sin(\pi/12), 6\cos(\pi/12)]$

Features: Non-uniform

Hypervolume: 0.80

Pareto front

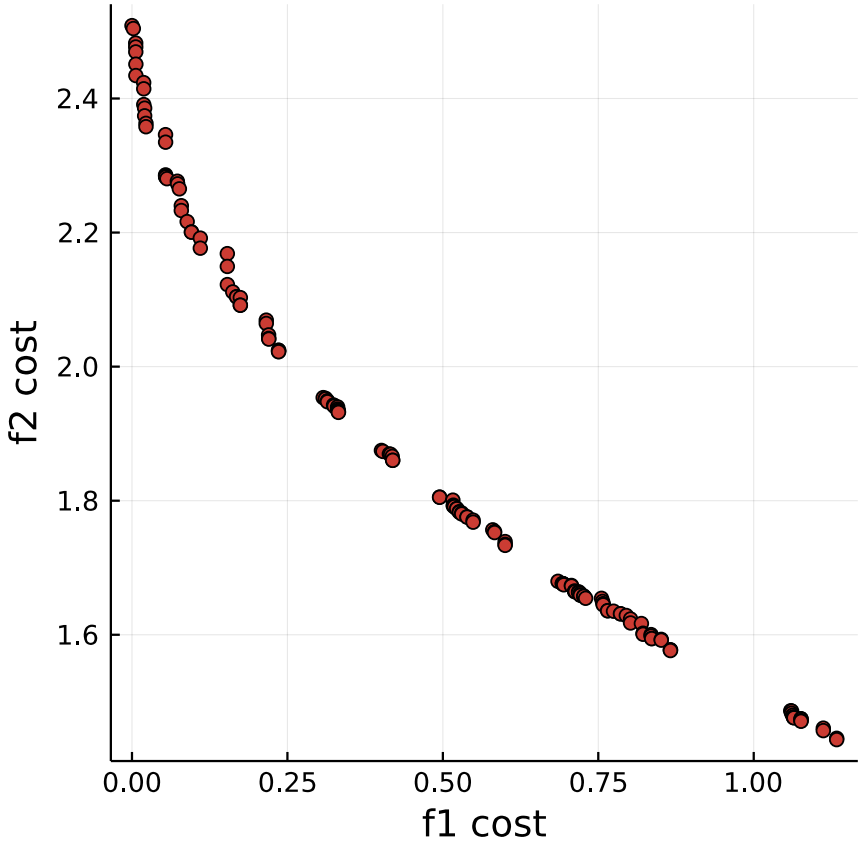


Fig. 7.19: Pareto front for OKA1

7.2.20 FAR1 [26]

Dimension of variables: $m = 2$

Problem Statement:

$$\begin{aligned} f_1(x_1, x_2) =& -2\exp\left(15\left(-(x_1-0.1)^2 - x_2^2\right)\right) - \exp\left(20\left(-(x_1-0.6)^2 - (x_2-0.6)^2\right)\right) \\ &+ \exp\left(20\left(-(x_1+0.6)^2 - (x_2-0.6)^2\right)\right) + \exp\left(20\left(-(x_1-0.6)^2 - (x_2+0.6)^2\right)\right) \\ &+ \exp\left(20\left(-(x_1+0.6)^2 - (x_2+0.6)^2\right)\right) \\ f_2(x_1, x_2) =& 2\exp\left(20\left(-x_1^2 - x_2^2\right)\right) + \exp\left(20\left(-(x_1-0.4)^2 - (x_2-0.6)^2\right)\right) \\ &- \exp\left(20\left(-(x_1+0.5)^2 - (x_2-0.7)^2\right)\right) - \exp\left(20\left(-(x_1-0.5)^2 - (x_2+0.7)^2\right)\right) \\ &+ \exp\left(20\left(-(x_1+0.4)^2 - (x_2+0.8)^2\right)\right) \end{aligned} \tag{7.30}$$

Constraints: $x_1, x_2 \in [0, 1]$

Features: Mixed front

Hypervolume: 0.27

Pareto front

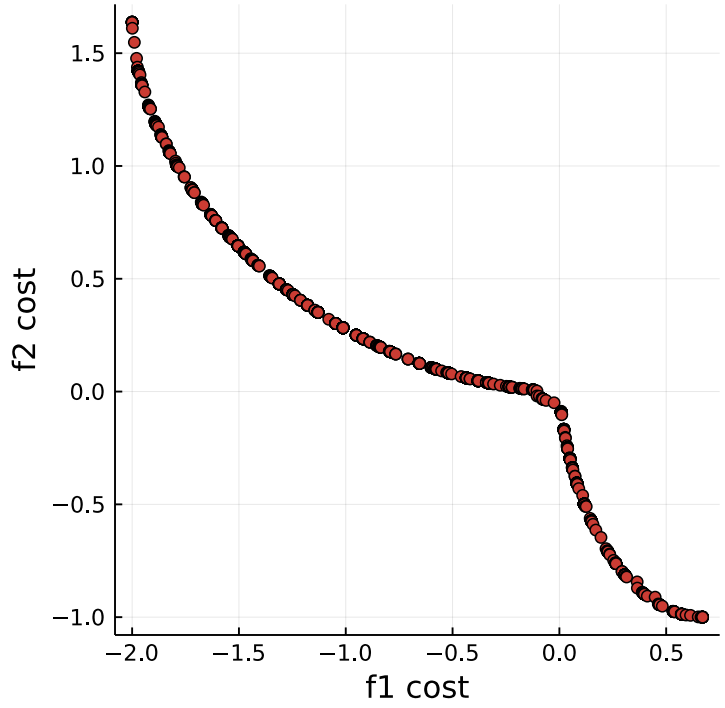


Fig. 7.20: Pareto front for FAR1

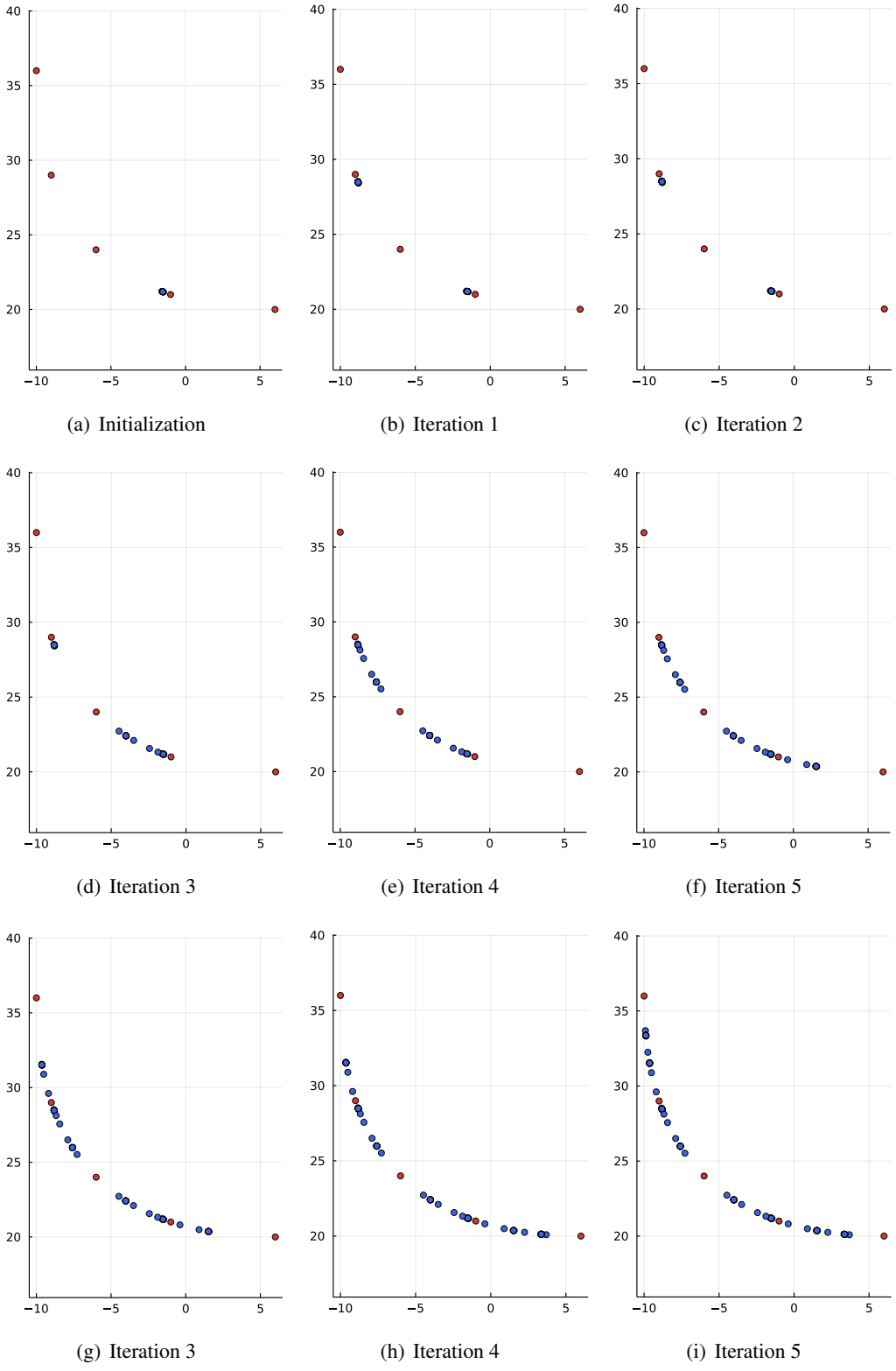
Table 7.4: Parameters for building case

Symbol	Global Data	Value	Unit	Symbol	Global Data	Value	Unit
a	Wall surface area	188.36	m^2	U	Average U value	0.65	W/m^2K
V	Building volume	225	m^3	n_{ac}	Air changes per hour	0.4	$°C$
ρ_{air}	Air density	1.225	kg/m^3	c_{air}	Air specific heat capacity	1.005	$kJ/kg\ K$

7.3 Parameters for Building Case

$$A = -\left(aU + V c_{\text{air}}\, \rho_{\text{air}}\, n_{\text{ac}}\right) \tag{7.31}$$

7.4 How does BiMADS fill the gaps gradually



(a) Initialization (b) Iteration 1 (c) Iteration 2

(d) Iteration 3 (e) Iteration 4 (f) Iteration 5

(g) Iteration 3 (h) Iteration 4 (i) Iteration 5

Fig. 7.21: BiMADS for 500 Iterations