FOPS Documentation

AToM Team

April 23, 2020

FAST OPTIMIZATION PROCEDURES
fops

Contents

List of Figures

Chapter 1

Introduction to FOPS

Fast Optimization Procedures (FOPS) is a toolbox for optimization in MATLAB. It covers local optimization methods as well as global ones. It is able to solve both single-objective and multi-objective optimization problems.

Newton (SONEW 14.1.6) and Nelder-Mead (SONEME 14.1.5) local optimization algorithms were implemented in FOPS. Global single-objective methods included are Genetic Algorithm (SOGA 14.1.1), Differential Evolution (SODE 14.1.3), Particle Swarm Optimization (SOPSO 14.1.2), Self-Organizing Migrating Algorithm (SOSOMA 14.1.4), and Covariance Matrix Adaptation Evolution Strategy (SOCMAES 14.1.7).

Multi-objective optimization problems are covered by Elitist Non-dominated Sorting Algorithm (NSGA-II 14.2.1), Generalized Differential Evolution 3 (GDE3 14.2.3) and Multi-Objective Particle Swarm Optimization (MOPSO 14.2.2).

Problems with variable number of dimensions are covered by single-objective Variable Number of Dimensions Particle Swarm Optimization (VNDPSO 14.3.1) algorithm, Variable Number of Dimensions Multi-Objective Particle Swarm Optimization (VNDMOPSO **??**), and multi-objective Variable Number of Dimensions Generalized Differential Evolution (VNDGDE3 14.3.3).

User can define easily its own problem in FOPS, but it also contains numerous benchmark problems. Additionally, most of the methods are able to solve constrained problems. All the optimization methods can also work with discrete decision space.

The FOPS enables user to use the chain of individual optimization algorithms, meaning that a single problem can be processed by the sequence of optimization methods. Therefore, advantages of individual algorithms can be exploited.

Various comparative studies can easily be performed in FOPS using a comparative study simulation type with prepared metrics for the evaluation of convergence properties.

The FOPS toolbox can be operated via command line or via Graphic User Interface (GUI).

Chapter 2

Optimization

Optimization is a process of finding optimal solutions. The definition of solution's quality is based on fitness values calculated from fitness functions. The fitness functions describe the behaviour of an optimized system with properties called decision variables. Therefore, the fitness values depend on the decision variables of the optimized system. The optimization process can be thereafter considered as a process of finding minima (or maxima) of the fitness values.

If the system is described by one fitness function, the optimization process is called a single-objective (SO) optimization. The optimization is called multi-objective (MO) in case of multiple fitness functions. Most real-world optimization problems are multi-objective, where the particular objectives are conflicting, which results in a set of optimal, trade-off, solutions called Pareto-front.

Optimization problems frequently involves constraints, that restrict some properties of the system to lie within pre-specified limits. Such restrictions are described by constraint functions. The solution that violates the constraint functions is called an infeasible solution. The infeasible solution can be taken as an optimal solution from the fitness values point of view, but a properly working optimization algorithm should penalize such solutions and present only feasible solutions as a result.

If a set of solutions is found in the single-objective optimization process, the optimal solution is the one with the smallest fitness value. This is more complicated in case of the multi-objective problem, because each solution is described with multiple values. If the first fitness value of one solution is lower than the first fitness value of the second solution, while the second fitness value is higher, it cannot be said which solution is better with respect to both objectives (fitness values). Neither solution dominates the other, therefore the solutions are non-dominated.

The aim of single objective optimization process is to find a solution as close to the true optimal solution as possible. But there are three main goals when solving the multi-objective problem:

- Maximize the number of solutions in non-dominated set.
- Minimize the distance of found non-dominated set produced by optimization algorithm to the true Pareto-front (optimal non-dominated set).
- Maximize the spread of found non-dominated solutions, so the solutions are distributed as uniformly over the Pareto-front as possible.

Note that the prefix *true* as in *true* optimal solution or *true* Pareto-front denotes the best possible solution or a set of solutions to be found for a given problem. The optimal solution or the Pareto-front denotes best solution or set of solutions found in particular optimization process. Therefore, the Pareto-front and the *true* Pareto-front are two different sets of solutions. Such a nomenuclature is used throughout the whole document.

Chapter 3

Simulations in FOPS

The FOPS toolbox offers three different types of simulation - tasks, chains and comparative studies. The comparative study can be seen as a set of several chains solving multiple problems. The chain is a set of optimization tasks connected together to optimize the single problem. An optimization task employs agents, changing their position in the decision space of optimization problem.

3.1 Agent

Agent is a fundamental particle of the optimization process. Agents are sometimes called individuals (frequently in Genetic Algorithms (GA)) or particles (often in Particle Swarm Optimization algorithms (PSO)). In PSO (see Subsection 14.1.2), the concept of the agent is often interpreted as a bee flying over a meadow searching for food. The optimization algorithm controls the position of the bee based on the quantity of food in a given location (expressed by the fitness value). Therefore, an agents has its `position` and a corresponding `fitness` value.

AGENT objects has the following properties:

`position`:	position of agent, double [1 x `nVars`]
`fitness`:	fitness values expresses the quality of the solution, double [1 x nObjs]
`constraints`:	constraint values determines the feasibility of the agent, double [1 x nCons]
`nVars`:	number of decision variables of the agent, double [1 x 1]

3.2 Task

The task is the simplest simulation type in FOPS. The task exploits a single optimization algorithm to solve an optimization problem. Tasks can be found in `tasks` container of the FOPS object. The task object OPTTASK (NSGAIITASK, MOPSOTASK, etc.) contains the following properties:

Task settings:	varies according to the optimization algorithm (e.g. `PC`, `PM`, `C1`, `C2`, `W`, `F`, etc.). See Section 7.1.
`name`:	unique identification of the task in the FOPS, char [1 x N]
`nAgents`:	number of agents, double [1 x 1]
`agents`:	container of objects of type AGENT (see Section 3.1), AGENT [1 x `nAgents`]
`currentIter`:	actual iteration - changes along the run of the optimization task, double [1 x 1]
`nIters`:	number of iterations, double [1 x 1]
`problem`:	optimization problem description (see Chapter 4), struct [1 x 1]
`problemName`:	name of the optimization problem, char [1 x M]
`problemConstrained`:	is set if the problem contains constraint functions, logical [1 x 1]

`isVectorized`:	is set if the fitness functions of the problem are vectorized, logical [1 x 1]
`nObjs`:	number of objectives of the problem, double [1 x 1]
`stopCondition`:	stop conditions of the optimization task (see Subsection 6.4.1), cell [S x 1]
`history`:	contains positions, fitness and constraints values from all iterations if `activeHistory` property is set, struct [1 x 1]
`results`:	results of the optimization run. Contains position, fitness values and constraints values of the found solutions, struct [1 x 1]
`finished`:	is set if the task was already run, logical [1 x 1]
`activeHistory`	if the property is set, `history` is filled during the optimization run, logical [1 x 1]
`showProgress`:	waitbar is shown during the optimization run if it is set, logical [1 x 1]
`userData`:	users arbitrary data, cell [1 x 1]

3.3 Chain

The chain is very similar to the task simulation, but instead of one optimization algorithm, it runs several optimization algorithms to solve one optimization problem. Tasks are chained, i.e. run sequentially one after another. Therefore, the user can exploit different advantages of different algorithms. Chains can be found in the `chains` container of the FOPS object. The CHAIN object has the following properties:

`name`:	unique identification of chain in FOPS, char [1 x M]
`tasks`:	container of tasks in chain (Section 3.2), OPTTASK [1 x N]
`problemName`:	name of optimization problem, char [1 x X]
`showProgress`:	waitbar is shown during the optimization run if it is set, logical [1 x 1]
`activeHistory`	if the property is set, `history` is filled during the optimization run, logical [1 x 1]
`computationalTime`:	time that the run of the chain took, double [1 x 1]
`history`:	contains positions, fitness and constraints values from all iterations of all tasks, struct [1 x 1]
`results`:	results of the optimization run. Contains position, fitness values and constraints values of the found solutions, struct [1 x 1]
`finished`:	is set if the chain was already performed, logical [1 x 1]

3.4 Comparative Study

The comparative study (CS) is the last and the most complex simulation type. The comparative study allows user to perform the optimization of numerous problems with multiple chains in order to compare the performance of different chains on a set of problems. User can enable the calculation of several predefined metrics which expresses the quality of found solutions.

Comparative studies can be found in `comparativeStudies` container of the FOPS object. The COMPSTUDY object has the following properties:

`name`:	unique identification of the comparative study in the FOPS, char [1 x M]
`chains`:	container of chains in comparative study (Section 3.3). Comparative study includes several unique chains and several problems. The overall number of columns of chains property is a product of number of unique chains and number of problems in comparative study. Number of rows of chains property is defined by `nRuns`

	property. See Figure 6.12 for better understanding of `chains` property dimensions, CHAIN [`nRuns` x N]
`nRuns`:	number of repetitions of all chains in the comparative study, double [1 x 1]
`problemNames`:	names of all problems in comparative study, char [1 x X]
`metricTypes`:	contains list of metrics that will be calculated during the simulation run (see Section **??**), METRICTYPE [1 x Y]
`showProgress`:	waitbar is shown during the optimization run if it is set, logical [1 x 1]
`activeHistory`	property `history` of individual chains is filled during the comparative study run, logical [1 x 1]
`results`:	fields named according to the enabled metrics, struct [1 x 1]
`finished`:	is set if the comparative study was already performed, logical [1 x 1]

The comparative study can be described as a useful tool for a statistical analysis in the optimization, but it does not do much by itself. It only combines functionality of the previous simulation types and calculates the metrics. User should also notice, that the overall number of simulations can very quickly increase and the comparative study computational time can dramatically grow, because overall number of simulations is a product of the number of problems in CS, the number of chains in CS and the number of runs of CS.

Chapter 4

User Problem

User defines optimization problems in the FOPS as a MATLAB function with one output argument - `problem` (see Listing 4.1). Location of problems functions is either current working directory or directories listed in `+install/problemFunctionPaths.txt` file. The FOPS package also contains numerous benchmark problems (see Chapter 15) located in `benchmark` folder in FOPS package.

Current directory and folders listed in `+install/problemFunctionPaths.txt` file (`benchmark` folder by default) are scanned for problem functions during the FOPS initialization. A MATLAB function is marked as a problem function if its header contains string `function problem = ...`. Problem function names are stored in `problemsList` property of the FOPS object along with the path. Therefore, if a new problem function is created, it is not possible to use it immediately, because such problem is not listed in the `problemList`. FOPS method `loadProblems` is used to refresh the `problemList` (see Subsection 6.4.4).

Name of problem function determines the string by which the problem will be called in FOPS. For example if problem function is named `MOZDT1`, user identifies such problem by the string `MOZDT1` (note that the problem name is case insensitive). Problem function returns the struct with fields that describe the problem. The problem is defined by at least two main fields `limits` (see Section 4.1) and `fitness` (see Section 4.2).

Listing 4.1 shows the `MODTLZ1` benchmark problem definition. The definition returns the problem struct with fields: `limits`, `fitness`, `isVectorized` (see Section 4.5), and `name` (see Section 4.3).

The function `MODTLZ1` has one input argument - `input`. Input argument `input` is further described in Subsection 6.1.1 (specifically Listing 6.6) or Subsection 6.2.1 (specifically Listing 6.23).

Listing 4.1: Definition of problem m-file.

```
function problem = MODTLZ1(input)
nVars = input.nVars;
nObjs = input.nObjs;
limits = [zeros(1, nVars); ones(1, nVars)];
fitness  = @(x) DTLZ1Fitness(x, nVars, nObjs);
problem = struct(...
   'limits', limits, ...
   'fitness', fitness, ...
   'isVectorized', true, ...
   'name', 'MODTLZ1');
end
```

The problem function can be also generated in graphic user interface (GUI) using the 💾 push button in Figure 6.2. Contrarily, user can load an existing problem m-file into GUI by using the 📂 push button in Figure 6.2.

4.1 limits

Property `limits` denotes ranges of decision variables. It is an array of doubles of size [2 x `nVars`], where `nVars` is number of decision variables. First row contains lower bounds of decision variables and upper bounds are in second row of `limits`.

Listing 4.2 shows an example of the limits definition. The first decision variable $x_1 \in \langle -2, 5 \rangle$. The second and the third decision variables range from 0 to 10.

Listing 4.2: Definition of problem limits.

```
limits = [-2, 0, 0; 5, 10, 10];
```

4.2 fitness

Property `fitness` denotes the fitness functions definition. It is defined as an anonymous function, array of anonymous functions or function handle. Listings 4.3, 4.4, and 4.5 shows the definition of the fitness functions, that are based on the `MO2D` benchmark problem. Variable `x` denotes decision variables.

Listing 4.3: Fitnes function as anonymous function.

```
fitness = @(x) (1 + x(:, 2)) / x(:, 1);
```

Listing 4.4: Fitnes function as an array of anonymous functions.

```
fitness = @(x) {...
   x(:, 1), ...
   (1 + x(:, 2)) ./ x(:, 1)};
```

Listing 4.5: Definition of fitness function by function handle.

```
fitness = @(x) MO2DFitness(x);
function fitness = MO2DFitness(x)
fitness(:, 1) = x(:, 1);
fitness(:, 2) = (1 + x(:, 2)) ./ x(:, 1);
end
```

The function `DTLZ1Fitness` that returns fitness values is mentioned in Listing 4.1. The `DTLZ1Fitness` function definition is shown in Listing 4.6. It has three input arguments (`x` suggesting agent's position, `nVars` is number of decision variables and `nObjs` is number of objectives) and one output argument (`fitness` values).

Listing 4.6: Fitness function of `MODTLZ1` problem.

```
function fitness = DTLZ1Fitness(x, nVars, nObjs)
k = nVars - nObjs + 1;
xm = x(:, nVars - k + 1:end);
g = 1*(k+sum((xm-0.5).^2 - cos(20*pi()*(xm-0.5)), 2));
fitness(:, 1) = 0.5*prod(x(:, 1:nObjs-1), 2) .* (1+g);
for iObj = 2:nObjs-1
   fitness(:, iObj) = 0.5*prod(x(:, 1:nObjs-iObj), 2).*(1-x(:, nObjs-iObj+1)) .* (1+g);
end
fitness(:, nObjs) = 0.5*(1-x(:, 1)) .* (1 + g);
end
```

Listing 4.7 shows the definition of minimal working example of problem definition.

Listing 4.7: Minimal working example of problem.

```
function problem = MySimpleProblem()
limits(:, 1) = [0.1, 1];
limits(:, 2) = [0,   1];
fitness = @(x) {...
```

```
   x(:, 1), ...                  % first fitness function
   (1 + x(:, 2)) ./ x(:, 1)};    % second fitness function
problem = struct('limits', limits, 'fitness', fitness);
end
```

4.3 name

Property name is used for the identification of the problem in FOPS. Therefore, it is possible to have problem defined by m-file named e.g. MySimpleProblem, but the field name in problems struct is e.g. problemName. If user wants to optimize such a problem, it will be called by MySimpleProblem, but in FOPS object the problem is later identified by the problemName.

For the sake of clarity, it is recommended to use the problem name identical to the name of the problem m-file or omit the problem's name property entirely. The name property of problem is by default set to the problem m-file name by FOPS class.

4.4 constraints

Property constraints denotes constraint functions definition. Constraint functions are defined in a similar way as fitness functions. The only difference is that constraint functions has two input arguments decision variables (x) and fitness values (f). Output argument is an array of logical values (g).

Listing 4.8 shows constraint function restricting the fitness value be lower than 25 by an anonymous function, while in the Listing 4.9, constraints functions are defined as a function handle. The second constraint function states that the product of decision variables has to be lower than 400.

Listing 4.8: Constraint function defined by anonymous function.

```
constraints = @(x, f) f(:, 1) > 25;
```

Listing 4.9: Constraint functions defined by function handle.

```
constraints = @(x, f) myConstraintFunc(x, f);
function g = myConstraintFunc(x, f)
   g(:, 1) = f(:, 1) > 25;
   g(:, 2) = x(:, 1) .* x(:, 2) < 400;
end
```

4.5 isVectorized

Default value of isVectorized property is false. True value of isVectorized means, that fitness and constraint functions are vectorized, i.e. fitness and constraint values of all agents in one iteration can be calculated at once.

4.6 discreteVariables

Property discreteVariables allows user to use the discrete decision space. The discreteVariables is a cell array containing vectors with decision variable samples. The number of cells is equal to the number of decision variables (nVars defined by the number of columns of limits). Empty cell denotes that particular decision variable will be real-coded.

Listing 4.10 shows the definition of the dicrete decision variables. The problem has 4 decision variables. Third cell of discreteVars variable is left empty, therefore the third decision variable will be real-coded. The range of all variable is $x_i \in \langle -4, 4\rangle$ for $i = 1, 2, 3, 4$, therefore all values in decisionVariables property has to be within the limits property.

Listing 4.10: Definition of discrete decision space.

```
limits = [-4, -4, -4, -4; 4, 4, 4, 4];
fitness = @(x) myFitnessFunction(x);
DV{1} = [-4, -3, -2, -1, 0, 1, 4];
DV{2} = linspace(-4, 4, 9);
DV{4} = [-4, -2, 0, 2, 4];
problem = struct('limits', limits, 'fitness', fitness, 'discreteVariables', {DV});
```

Note that the `discreteVars` variable is encapsulated to cell array to ensure that the dimension of `problem` struct is still [1 x 1]. More information about discrete variables can be found in Chapter 11.

4.7 surrogate

Property `surrogate` enables the surrogate optimization method. Property `surrogate` is a struct with mandatory field `type`. The `type` field specifies the type of surrogate optimization method to be used. Folder with corresponding name has to be located in `+surrogate` folder. The folder includes `main` function, defining the surrogate optimization method functionality. See more in Chapter 10.

4.8 initialPosition

The `initialPosition` contains initial position for optimization methods. It is a matrix of doubles with the number of columns equal to the number of decision variables and the number of rows equal to the number of agents. Listing 4.11 shows how to define initial position.

Listing 4.11: Definition of initial position for local optimization methods.

```
limits(:, 2) = [-4, 4];
fitness = @(x) myFitnessFunction(x);
initPosition = [0, 1];
problem = struct('limits', limits, 'fitness', fitness, ...
   'initialPosition', initPosition);
```

4.9 gradient

Property `gradient` is usable with the local methods only. The `gradient` property stands for a gradient of the fitness function and it can be defined by anonymous function or function handle. If `gradient` is not stated in local optimization, numerical gradient of fitness function is used. Listings 4.12 and 4.13 shows definition of gradient taken from `LOROS` problem:

Listing 4.12: Definition of gradient using anonymous function.

```
limits(:, 2) = [-5; 10];
fitness = @(x) {sum(100*(x(:, 2) - x(:,1).^2).^2 + (1-x(:,1)).^2, 2)};
gradient = @(x) {...
   -400*x(1,1)*(x(1,2)-x(1,1)^2)-2*(1-x(1,1)), 200*(x(1,2) - x(1,1)^2)};
problem = struct('limits', limits,  'fitness', fitness, ...
   'gradient', gradient);
```

Listing 4.13: Definition of gradient using function handle.

```
limits(:, 2) = [-5; 10];
fitness = @(x) {sum(100*(x(:, 2) - x(:,1).^2).^2 + (1-x(:,1)).^2, 2)};
gradient = @(x) gradientFunction(x);
problem = struct('limits', limits,  'fitness', fitness, ...
   'gradient', gradient);

function gradient = gradientFunction(x)
gradient = {-400*x(1,1)*(x(1,2)-x(1,1)^2)-2*(1-x(1,1)), 200*(x(1,2) - x(1,1)^2)};
```

```
end
```

4.10 hessian

Property `hessian` is usable with the local methods only. The `hessian` property stands for a hessian of the fitness function and it can be defined by anonymous function or function handle. If `hessian` is not stated in local optimization, numerical hessian of fitness function is used.

Listings 4.14 and 4.15 shows definition of hessian taken from `LOROS` problem:

Listing 4.14: Definition of hessian using anonymous function.

```
limits(:, 2) = [-5; 10];
fitness = @(x) {sum(100*(x(:, 2) - x(:,1).^2).^2 + (1-x(:,1)).^2, 2)};
hessian = @(x) {1200*x(1,1)^2 - 400*x(1,2) + 2, -400*x(1,1); -400*x(1,1), 200};
problem = struct('limits', limits,  'fitness', fitness, 'hessian', hessian);
```

Listing 4.15: Definition of hessian using function handle.

```
limits(:, 2) = [-5; 10];
fitness = @(x) {sum(100*(x(:, 2) - x(:,1).^2).^2 + (1-x(:,1)).^2, 2)};
hessian = @(x) hessianFunction(x);
problem = struct('limits', limits,  'fitness', fitness, 'hessian', hessian);

function hessian = hessianFunction(x)
hessian = {1200*x(1,1)^2 - 400*x(1,2) + 2, -400*x(1,1); -400*x(1,1), 200};
end
```

4.11 nVarsList

This property is usable with methods using variable number of dimensions (VND) only. It is a list of all the possible number of decision variables for a given problem. All values has to be positive integer values.

Listing 4.16 shows that agent's position can be constituted from 2, 3, 4, 5, 6, 7, 8, 9 or 10 decision variables. Note that `limits` has to be specified for all possible decision variables, therefore the `limits` has dimensions of [2 x 10].

Listing 4.16: Definition of `nVarsList`.

```
nVarsList = 2:10;
limit = [-10; 10];
limits = repmat(limit, 1, max(nVarsList));
fitness = @(x) myFitnessFunction(x);
problem = struct('limits', limits, 'fitness', fitness, 'nVarsList', nVarsList);
```

Listing 4.17 shows that agent's position can be constituted by even number of decision variables - 2, 4, 6, 8, 10, 12, 14, 16, 18 or 20. The property `limits` now has to have dimensions of [2 x 20].

Listing 4.17: Definition of `nVarsList` - even number of decision variables.

```
nVarsList = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20];
limit = [-10; 10];
limits = repmat(limit, 1, max(nVarsList));
fitness = @(x) myFitnessFunction(x);
problem = struct('limits', limits, 'fitness', fitness, 'nVarsList', nVarsList);
```

4.12 optimalPosition

Property `optimalPosition` specifies true optimal position for a given problem.

The optimalPosition is inserted directly as a numeric vector (see Listing 4.18) or as a name of mat-file, which contains field position (see Listing 4.19).

In case of multi-objective problem, optimalPosition is a set of the true optimal positions (matrix). If the problem is single-objective, the optimalPosition is only one true optimal position (vector). See Listing 4.18.

Listing 4.18 shows the direct inserting of optimalPosition.

Listing 4.18: Optimal position of single-objective problem.

```
limits(:, 1) = [-4, 4];
limits(:, 2) = [-4, 4];
fitness = @(x) myFitnessFunction(x);
optimalPosition = [0, 0];
problem = struct('limits', limits, 'fitness', fitness, ...
   'optimalPosition', optimalPosition);
```

In case of Listing 4.19, the file `MOPOL.mat` has to contain field position.

Listing 4.19: Optimal position are saved in file named `MOPOL.mat`.

```
limits(:, 1) = [-pi, pi];
limits(:, 2) = [-pi, pi];
fitness = @(x) MOPOLFitness(x);
optimalPosition = 'MOPOL';
problem = struct('limits', limits, 'fitness', fitness, ...
   'optimalPosition', optimalPosition);
```

4.13 optimalFitness

This property is similar to optimalPosition. It specifies true optimal fitness values of a given problem.

The optimalFitness is inserted directly as a numeric value (see Listing 4.20) or as a name of mat-file, which contains field fitness (see Listing 4.21).

In case of multi-objective problem, optimalFitness is a true Pareto-front (matrix). If the problem is single-objective, the optimalFitness is only one fitness value (scalar). See Listing 4.20.

In Listing 4.20, optimalFitness is directly inserted as a scalar value.

Listing 4.20: Optimal fitness of single-objective problem.

```
limits(:, 1) = [-4, 4];
limits(:, 2) = [-4, 4];
fitness = @(x) myFitnessFunction(x);
optimalFitness = 0;
problem = struct('limits', limits, 'fitness', fitness, ...
   'optimalFitness', optimalFitness);
```

In case of Listing 4.21, the file `MOPOL.mat` has to contain field fitness.

Listing 4.21: Optimal fitness values are saved in file named `MOPOL.mat`.

```
limits(:, 1) = [-pi, pi];
limits(:, 2) = [-pi, pi];
fitness = @(x) MOPOLFitness(x);
optimalFitness = 'MOPOL';
problem = struct('limits', limits, 'fitness', fitness, ...
   'optimalFitness', optimalFitness);
```

4.14 optimalDimension

This property specifies the optimal dimension of true optimal solutions. It can either be numeric or a function handle. In case of multi-objective problems with variable number of dimensions, the

Pareto-front can be build by solutions with different optimal dimensionality. Such dimensionality is obtained from the `optimalDimension` property.

4.15 reference

Property `reference` is used by hypervolume metric (see Subsection **??**). Reference point should be specified if hypervolume metric is used. Otherwise, the reference point is taken as a maximum of the fitness values in each objective, which can cause incomparable metric results (see Subsection **??**). If the number of elements of the reference vector does not match the number of objectives (`nObjs`), first element of reference vector is repeated `nObjs`-times.

The two-objective problem is assumed in Listing 4.22. Hypervolume will be calculated with the reference point [2, 2].

Listing 4.22: How to set hypervolume reference point.

```
limits(:, 1) = [-4, 4];
limits(:, 2) = [-4, 4];
limits(:, 3) = [-4, 4];
fitness = @(x) myFitnessFunction(x);
reference = [2, 2];
problem = struct('limits', limits, 'fitness', fitness, ...
   'reference', reference);
```

4.16 fullControl

This property allows skilled users to have reference to OptTask object during fitness functions and constraint functions evaluations. This is particularly useful when user wants to exploit `userData` property of OptTask object (see Subsections 6.1.1, 6.2.1 and 6.3.1 to see how to set `userData` property of OptTask).

In Listing 4.23, the `fullControl` property is enabled, which allows user to gain access to OptTask's `userData` property. But there are many more possibilities with the access to OptTask object. Note that fitness functions and constraint functions has to have one additional input argument (`task`) with the `fullControl` property set.

Listing 4.23: `fullControl` property of problem is set to true.

```
limits(:, 1) = [-4, 4];
limits(:, 2) = [-4, 4];
limits(:, 3) = [-4, 4];
fitness = @(x, task) advancedFitnessFunction(x, task);
constraints = @(x, f, task) advancedConstraintFunction(x, f, task);
problem = struct('limits', limits, 'fitness', fitness, ...
   'constraints', constraints, 'fullControl', true);

function fitness = advancedFitnessFunction(x, OptTask)
   UD = OptTask.userData;
   ... % some operations with user's data
   fitness = computeFitnessValues(x, UD);
end

function constraints = advancedConstraintFunction(x, f, OptTask)
   UD = OptTask.userData;
   ... % some operations with user's data
   constraints = computeConstraintsValues(x, f, UD);
end
```

4.17 systemResponse

Property `systemResponse` is usable, when a user wants to visualize the individual solution. Property `systemResponse` holds the function handle, which is performed upon selection of individual solution in an interactive plot of results in GUI. See more about the `systemResponse` feature in Chapter 9.

In Listing 4.24, the property `systemResponse` is being defined as a function handle. The function handle has two input argument - position and fitness values of a given solution. Function `plotCircles` is an abstract function here. It may contain arbitrary user's code.

Listing 4.24: `systemResponse` property example.

```
limits(:, 1) = [-4, 4];
limits(:, 2) = [-4, 4];
limits(:, 3) = [-4, 4];
fitness  = @(x, task) circlesFitness(x, task);
sysRes = @(x, f) plotCircles(x, f);
problem = struct('limits', limits, 'fitness', fitness, ...
   'systemResponse', sysRes);

function plotCircles(X, F)
% possibly some operation with X (position) and F (fitness values)
% possibly some plotting of X or F
end
```

Chapter 5

FOPS Class

FOPS class is the core class of the FOPS package. FOPS class operates all types of simulations via numerous access methods. The simulations are operated by the following public methods:

`addTask` (Subsection 6.1.1),

`changeTaskSettings` (Subsection 6.1.5),

`deleteTask` (Subsection 6.1.2),

`renameTask` (Subsection 6.1.3),

`runTask` (Subsection 6.1.4),

`addChain` (Subsection 6.2.1),

`changeChainSettings` (Subsection 6.2.5),

`deleteChain` (Subsection 6.2.2),

`renameChain` (Subsection 6.2.3),

`runChain` (Subsection 6.2.4),

`addCompStudy` (Subsection 6.3.1),

`changeCompStudySettings` (Subsection 6.3.5),

`deleteCompStudy` (Subsection 6.3.2),

`renameCompStudy` (Subsection 6.3.3),

`runCompStudy` (Subsection 6.3.4),

`addStopCondition` (Subsection 6.4.1),

`getProblem` (Subsection 6.4.2),

`loadProblems` (Subsection 6.4.4),

`open` (Subsection 6.4.5),

`quit` (Subsection 6.4.6),

`reset` (Subsection 6.4.7),

`resume` (Subsection 6.4.8),

`save` (Subsection 6.4.9),

`displayResults` (Subsection 6.4.3),

`startGUI` (Subsection 6.4.10).

The FOPS is initialized by command `fops = FOPS()`. Initialization of FOPS with graphic user interface (GUI) is done by `fops = FOPS(true)`. To display GUI of existing FOPS object, command `fops.startGUI()` has to be used.

FOPS object has the following properties:

`tasks`:	container of tasks in FOPS (Section 3.2), OptTask [1 x N]
`chains`:	container of chains in FOPS (Section 3.3), Chain [1 x M]
`comparativeStudies`:	container of comparative studies in FOPS (Section 3.4), CompStudy [1 x P]
`algorithmList`:	list of all algorithms implemented in FOPS (see Chapter 14), cell [1 x A]
`metricList`:	list of all metrics implemented in FOPS (see Section ??), cell [1 x B]
`problemList`:	list of all problems found in given locations (by default `benchmark` folder and current folder), struct [1 x C]
`taskSettingsList`:	list of all task settings found during FOPS initialization (`taskSettingsfiles` folder and current folder), struct [1 x D]
`GUI`:	reference to GUI object of FOPS, GUI [1 x 1]
`showProgress`:	if this property is set, waitbar is shown during the run of chain, logical [1 x 1]
`activeHistory`	if the property is set, positions and fitness value from all iterations will be saved to `history` property of all tasks, logical [1 x 1]
`algorithmProperties`:	holds the default values and whole names of properties for each algorithm in FOPS, struct [1 x 1]
`openedGUI`:	shows whether the GUI is opened, logical [1 x 1]

5.1 GUI

Figure 5.1 shows the basic layout of GUI of FOPS. The `MENU` (box 1) contains entries `Load Problems`, `Open FOPS`, `Reset FOPS`, `Save FOPS`, `Quit GUI` and `Quit FOPS` as shown in Figure 5.2. Preferences (box 1) serves to activate the history of simulations and to activate progress waitbars. Templates (box 1) enables user to switch GUI color templates.

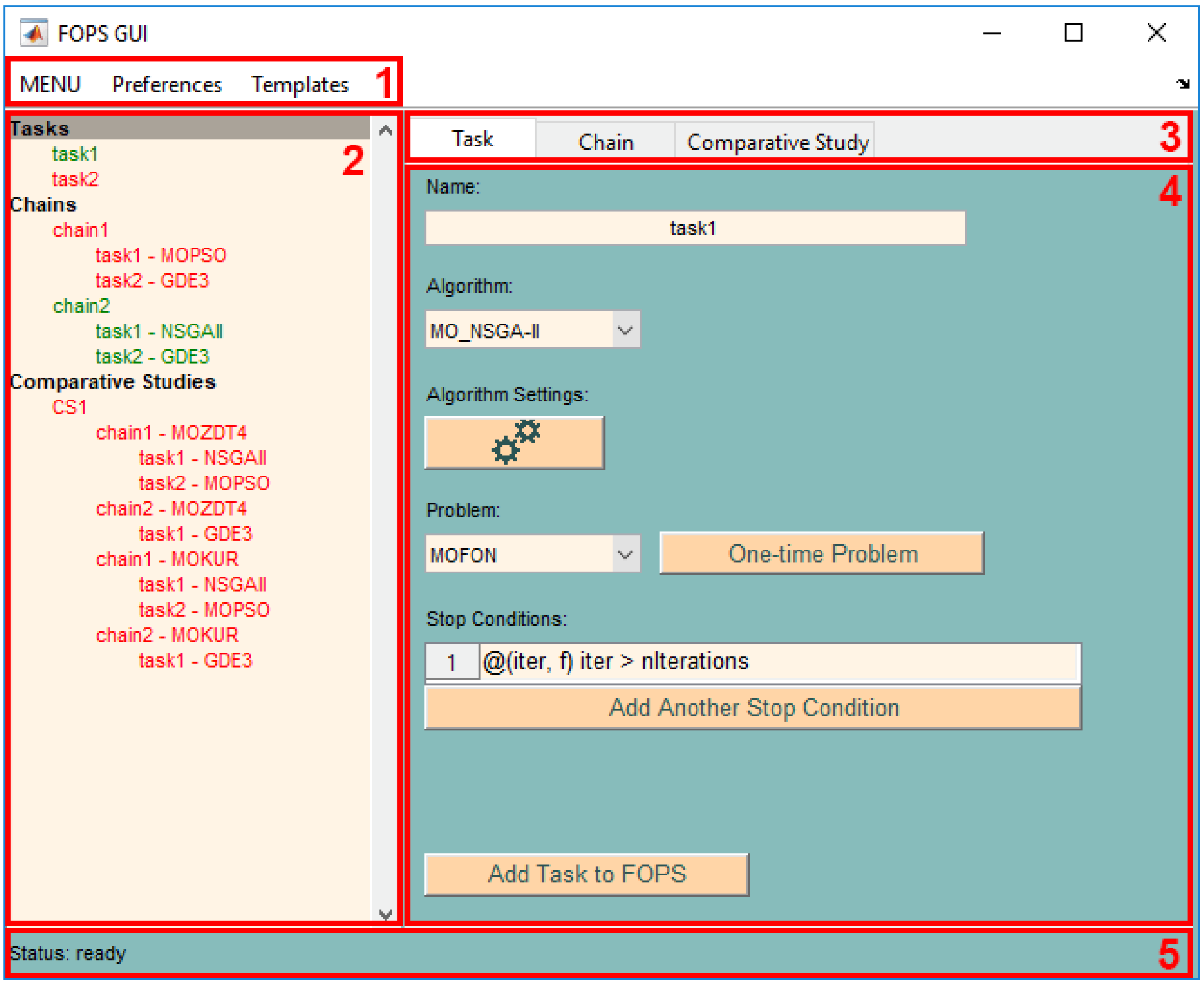
The `Shadow Green` template is the default one (set by the `defaultTemplate` property of GUI class). GUI can be launched with another color template by entering an integer value as an input argument of `FOPS` constructor, `GUI` constructor or `startGUI` method. There are four different templates prepared for GUI numbered from 1 to 4. Therefore, to start a GUI with e.g. third color template, user should use commands `fops = FOPS(3)` or `fops.startGUI(3)`. Color template settings is located in `setGUIColors` method in GUI class.

Box 2 borders `FOPS Tree`. The `FOPS Tree` depicts simulations present in FOPS object. The `FOPS Tree` is divided into three parts according to the three simulation types. Right mouse button click in `FOPS Tree` casts the context menu, which allows user to operate FOPS simulations. Most of `FOPS Tree` entries can be accessed via keyboard shortcuts - run simulation by `CTRL+R`, change simulation settings by `CTRL+H`, delete simulation by `CTRL+X`, and display simulation results by `CTRL+D`. These shortcuts can be used even without casting the context menu, which speeds up the work with FOPS GUI. Simulation name in green font denotes that simulation was already run, while the red font suggests that simulation was not run yet.

Box 3 marks tab group where user selects between task, chain and comparative study simulation type.

In box 4 are controls needed to add task (note that task tab is currently selected). Chain and comparative study controls are different and will be shown in Sections 6.2 and 6.3, respectively.

Box 5 contains status bar. Status bar contains simple progress bar and a rough estimation of simulation time when simulation is in progress.



Tasks
task1
task2
Chains
chain1
task1 - MOPSO
task2 - GDE3
chain2
task1 - NSGAII
task2 - GDE3
Comparative Studies
CS1
chain1 - MOZDT4
task1 - NSGAII
task2 - MOPSO
chain2 - MOZDT4
task1 - GDE3
chain1 - MOKUR
task1 - NSGAII
task2 - MOPSO
chain2 - MOKUR
task1 - GDE3

Name: task1

Algorithm: MO_NSGA-II

Algorithm Settings:

Problem: MOFON

Stop Conditions:
1 @(iter, f) iter > nIterations

Status: ready

Figure 5.1: Graphic user interface of FOPS.

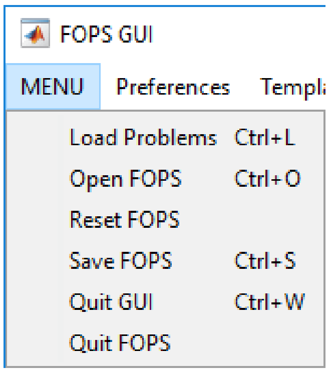


Figure 5.2: MENU of GUI.

Chapter 6

FOPS Methods

6.1 Task Methods

The box 4 in Figure 5.1 shows the tab for adding task to FOPS.

6.1.1 addTask

`fops.addTask(problem, algorithm, settings, name, userData)`

Inputs:

`problem`	cell [N x 1] or cell [N x 2]
`algorithm`	cell [N x 1]
`settings`	optional, cell array of structs, cell [N x 1]
`name`	optional, cell [N x 1]
`userData`	optional, cell [1 x 1]

Method `addTask` has 5 input arguments: `problem`, `algorithm`, `settings`, `name` and `userData`. First two arguments are mandatory while the remaining three are optional, but has to remain in a given order.

`problem` is defined as a cell with struct with fields presented in Chapter 4 or as a cell with the name of problem function (m-file) which returns such struct.

`algorithm` defines optimization algorithm that will be used to optimize the given problem.

`settings` argument defines parameters of optimization algorithm. It involves number of agents (`nAgents`), number of iterations (`nIters`) and algorithm dependent parameters such as probability of mutation (`PC`), inertia weight (`W`) etc. It is struct with field names suggesting type of parameter (see Listing 6.1). For more info about the task settings see Section 7.1.

Listing 6.1: Definition of settings struct.

```
settings = struct('nAgents', 25, 'nIters', 200, 'W', 0.5, 'BP', 10);
```

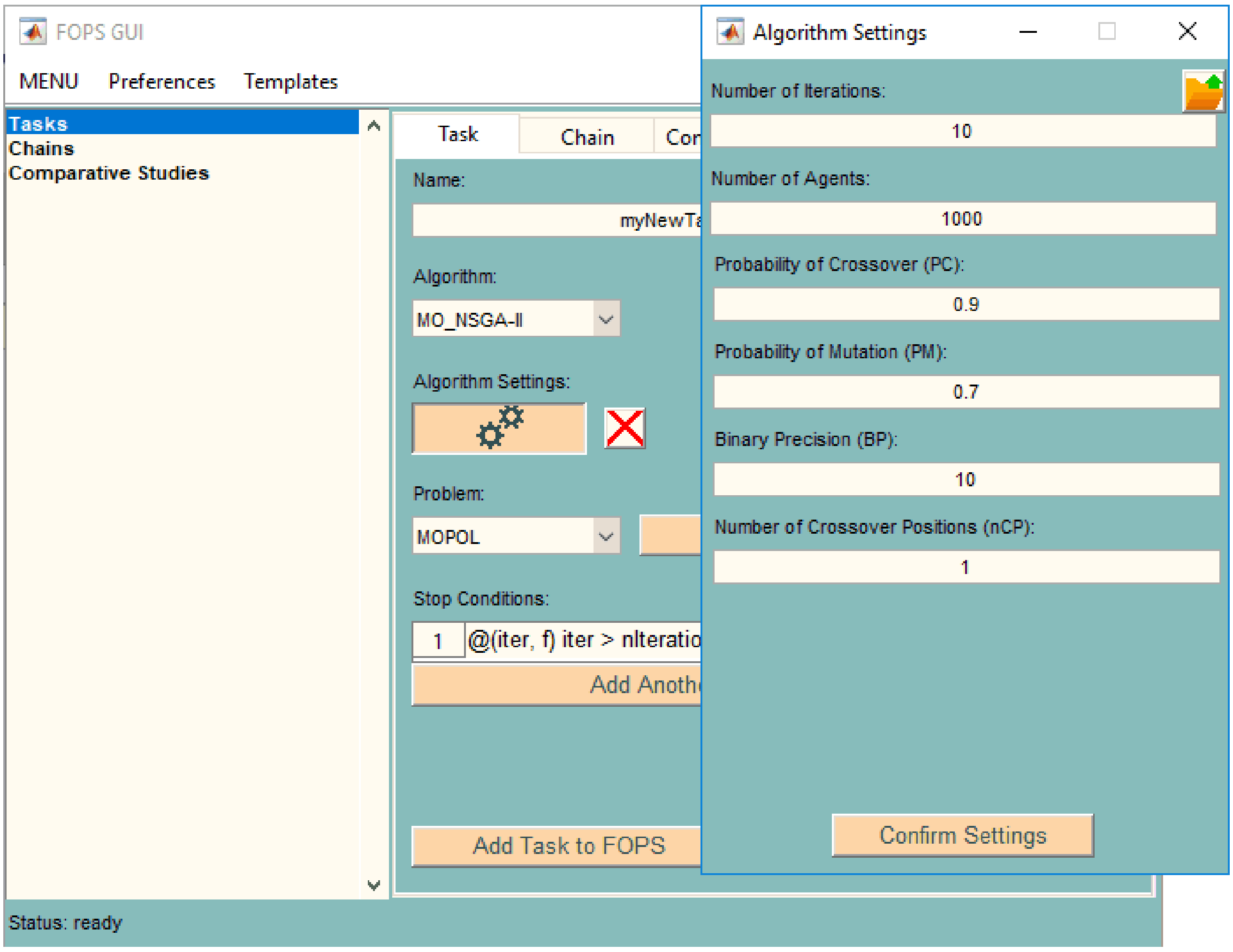
`name` is a unique task name. Such names are shown in `FOPS Tree` (see Figure 5.1) and are used to identify tasks.

Last input argument is `userData`, which holds user's entries. It can be employed by user in fitness function if `fullControl` property of problem is set to true (See Section 4.16).

Listing 6.2 shows a minimal working example of adding an optimization task with predefined problem. At first, FOPS object has to be initialized. Afterwards, optimization task is added by `addTask` method using only mandaroty inputs. Default task settings (See Section 7.1) and default name (in this case `task1`) is used.

Listing 6.2: Minimal working example of `addTask` method.

```
fops = FOPS(); % initialization of FOPS
```



Name: myNewTa

Algorithm: MO_NSGA-II

Algorithm Settings:

Problem: MOPOL

Stop Conditions:

1 @(iter, f) iter > nIteratio

Number of Iterations: 10

Number of Agents: 1000

Probability of Crossover (PC): 0.9

Probability of Mutation (PM): 0.7

Binary Precision (BP): 10

Number of Crossover Positions (nCP): 1

Figure 6.1: Add task to FOPS and define its settings in GUI.

```
fops.addTask({'MOPOL'}, {'NSGA-II'})
```

All input arguments should be cell arrays, but if given input argument is e.g. simple string, it will be automatically converted to cell array. Therefore, Listings 6.2 and 6.3 has equal effect.

Listing 6.3: Minimal working example of `addTask` method without cell array brackets.

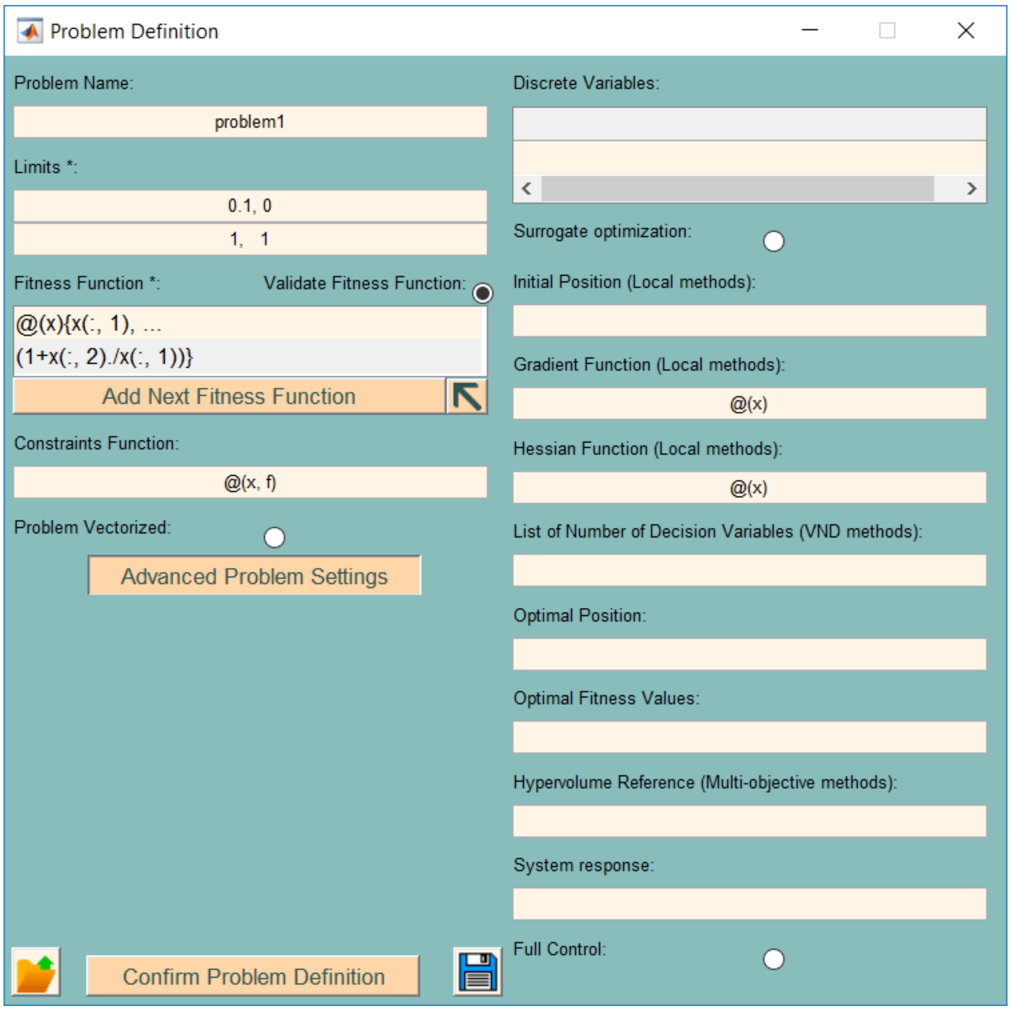
```
fops = FOPS(); % initialization of FOPS
fops.addTask('MOPOL', 'NSGA-II')
```

In Listing 6.4 the same task is added, but some of the algorithm's parameters are changed and task's name and `userData` are specified. Optimization task will have `1000` agents (`nAgents`) and will perform `10` iterations (`nIters`). Binary precision (`BP`) of all decision variables in NSGA-II task is set to `10`. All other algorithm properties are set as default. See Section 7.1 to learn more about task's properties.

Listing 6.4: Example of `addTask` method with user defined task's settings and name.

```
fops = FOPS(); % initialization of FOPS
settings = struct('nIterations', 10, 'BP', 10);
settings.nAgents = 1000;
taskName = 'myNewTask';
userData = {[1, 2, 3], [4, 5, 6]};
fops.addTask('MOPOL', 'NSGA-II', settings, taskName, userData)
```

The same operation (except for `userData`) can be done in GUI as shown in Figure 6.1. Algorithm NSGA-II is selected by `Algorithm` popup menu. Problem `MOPOL` is selected in `Problem` popup menu


Problem Definition

Problem Name: problem1

Limits *: 0.1, 0 / 1, 1

Fitness Function *: @(x){x(:, 1), ... (1+x(:, 2)./x(:, 1))}

Validate Fitness Function: ●

Constraints Function: @(x, f)

Problem Vectorized: ○

Discrete Variables:

Surrogate optimization: ○

Initial Position (Local methods):

Gradient Function (Local methods): @(x)

Hessian Function (Local methods): @(x)

List of Number of Decision Variables (VND methods):

Optimal Position:

Optimal Fitness Values:

Hypervolume Reference (Multi-objective methods):

System response:

Full Control: ○

Add Next Fitness Function
Advanced Problem Settings
Confirm Problem Definition

Figure 6.2: Definition of optimization problem in GUI of FOPS.

and `settings` can be adjusted in `Algorithm Settings` figure after clicking on toggle button. Note that `Algorithm Settings` figure differs with the used algorithm. The ✗ push button serves to discard the changes of algorithm's settings.

The push button serves to load algorithm's settings from the text file.

Listing 6.5 shows how to add task with one-time problem. Such problem is inserted to `addTask` method as a struct. `fitness` and `limits` are taken from `MO2D` problem. Problem `name` is optional. For more info about problem properties see Chapter 4.

Listing 6.5: Example of `addTask` method with one-time problem.

```
fops = FOPS(); % initialization of FOPS
fitness = @(x) {x(:, 1), (1 + x(:, 2)) ./ x(:, 1)};
oneTimeProblem = struct('limits', [0.1, 0; 1, 1], 'fitness', fitness);
oneTimeProblem.name = 'oneTimeProblem';
fops.addTask(oneTimeProblem, 'NSGA-II')
```

The GUI version can be seen in Figure 6.2. The `Problem Definition` figure is displayed by `One-time Problem` toggle button shown in Figure 6.1. Properties `limits` and `fitness` are the only two mandatory properties of problem. All other properties have either default values or can be left empty. Right half of the Figure 6.2 is invisible after casting the `Problem Definition` figure and the right half is shown by clicking the `Advanced Problem Settings` toggle button. The problem properties are described in Chapter 4.

Some optimization problems can be scalable, which enables user to specify e.g. how many decision variables or fitness functions the problem has. Listing 6.6 shows how to define number of decision variables and number of objective functions for problem `MODTLZ1` (Definition of `MODTLZ1` problem can be seen in Listing 4.1). The problem will have 20 decision variables (nVars) and 4 objective functions (nObjs) and will be optimized by algorithm NSGA-II. The Listing 6.6 shows how to exploit problem function input argument by FOPS methods. Problem input argument can be either scalar number or struct. Input argument of problem cannot be defined in FOPS GUI.

Listing 6.6: Definition of input argument of problem when initializing task.

```
fops = FOPS(); % initialization of FOPS
fops.addTask({'MODTLZ1', struct('nVars', 20, 'nObjs', 4)}, 'NSGA-II')
```

The reason for cell arrays as an input argument type is that the user can add multiple tasks at once. The Listing 6.7 shows how to add 4 tasks at once. Each input argument of addTask method has four solitary entries. In problem variable, each row belongs to one task. The MyTask1 and MyTask4 (NSGA-II) and MyTask2 (MOPSO) tasks will have modified settings, while MyTask3 (GDE3) will have all properties set as default. All tasks will have the same userData property containing cell array with 2 strings.

Note that algorithms variable is row cell array, but it is more convenient to write it as column array, because addChain method (see Subsection 6.2.1) uses the similar structure, but rows of cell array are reserved for individual tasks of chain. The same recommendation applies to the settings and name input arguments.

The number of rows of problem and the number of elements of algorithms, settings and taskNames must be the same. Problems has to be strictly written in rows as shown in Listing 6.7, otherwise the string input in the second row of the problem cell array would cause an error.

To create multiple tasks with different userData properties, user needs to add tasks separately by using multiple-times the addTask method.

Listing 6.7: Creation of multiple tasks with one use of addTask method.

```
fops = FOPS(); % initialization of FOPS
myTempProblem = struct('limits', [0, 0; 1, 1], 'fitness', @(x)myFitnessF(x));
problem = {...
   'MOPOL',        []; ...
   myTempProblem, []; ...
   'MOZDT1',       25; ...
   'MODTLZ5',      struct('nVars', 8, 'nObjs', 6)};
algorithms = {'NSGAII', 'MOPSO', 'GDE3', 'NSGAII'};
NSGASettings = struct('nIterations', 1000, 'PC', 0.2, 'PM', 0.2);
settings = {...
   NSGASettings;                            ...  % this belongs to first task
   struct('nAgents', 10, 'W', [0.9, 0.5]); ...  % this belongs to second task
   [];                                      ...  % this belongs to third task
   NSGASettings};                                % this belongs to fourth task
taskNames = {'MyTask1'; 'MyTask2'; 'MyTask3'; 'MyTask4'};
userData = {'This userData will', 'be the same for all tasks'};
fops.addTask(problem, algorithms, settings, taskNames, userData)
```

Control elements which modifies stop conditions of task can also be seen in Figure 5.1. See Subsection 6.4.1 to learn more about stop conditions.

6.1.2 deleteTask

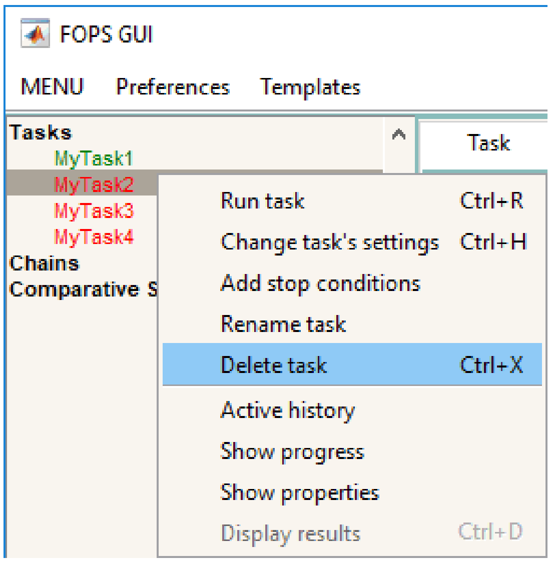
```
fops.deleteTask(id)
```

Inputs:

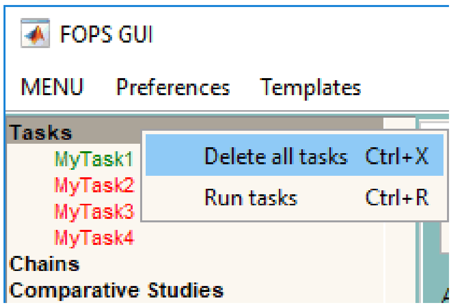
id	cell array of strings or doubles, cell [1 x N]

The method deleteTask has only one input argument - identification (id) of the task to delete.

The Listing 6.8 shows how to delete the task from FOPS. At the beginning, four tasks are added to FOPS. Task to delete can be identified by its name or by sequence number. Only MyTask1 remains in FOPS at the end of Listing 6.8 example.



(a) One task at a time.



(b) All tasks at once.

Figure 6.3: Delete task from FOPS in GUI.

Similar action is possible in GUI. Figure 6.3a shows how to delete task from FOPS in GUI. The `Context Menu` is displayed by right mouse click in `FOPS Tree` after selection of task to delete by left mouse click.

Listing 6.8: Deletion of task by `deleteTask` method.

```
fops = FOPS(); % initialization of FOPS
problem =      {'MOPOL';   'MOFON';    'MOKUR';    'MOSCH'};
algorithms =   {'NSGA-II'; 'MOPSO';    'GDE3';     'NSGA-II'};
taskNames =    {'MyTask1'; 'MyTask2';  'MyTask3';  'MyTask4'};
fops.addTask(problem, algorithms, [], taskNames)
fops.deleteTask(2)
fops.deleteTask('MyTask3')
fops.deleteTask({'MyTask4'})
```

Listing 6.9 shows how to delete multiple tasks with one command. At the end of an example, only `MyTask3` remains in FOPS.

Listing 6.9: Deletion of multiple tasks by `deleteTask` method at once.

```
fops = FOPS(); % initialization of FOPS
problem =      {'MOPOL';   'MOFON';    'MOKUR';    'MOSCH'};
algorithms =   {'NSGA-II'; 'MOPSO';    'GDE3';     'NSGA-II'};
taskNames =    {'MyTask1'; 'MyTask2';  'MyTask3';  'MyTask4'};
fops.addTask(problem, algorithms, [], taskNames)
fops.deleteTask({'MyTask1', 2, 4})
```

Note that an effect of three partial calls of `deleteTask` (See Listing 6.10) instead of the one used in Listing 6.9 is completely different. By the first call, the first task `MyTask1` is deleted. By the second call of `deleteTask`, the task `MyTask3` is deleted, because it now has sequence number of 2. The third call of `deleteTask` would cause an error, because only two tasks remains in the FOPS object.

Listing 6.10: Deletion of multiple tasks by multiple `deleteTask` method calls.

```
...
fops.deleteTask('MyTask1')
fops.deleteTask(2)
fops.deleteTask(4)
```

Figure 6.3b shows how to delete all tasks at once from FOPS in GUI. This `Context Menu` is called by selecting `Tasks` group and right mouse button click.

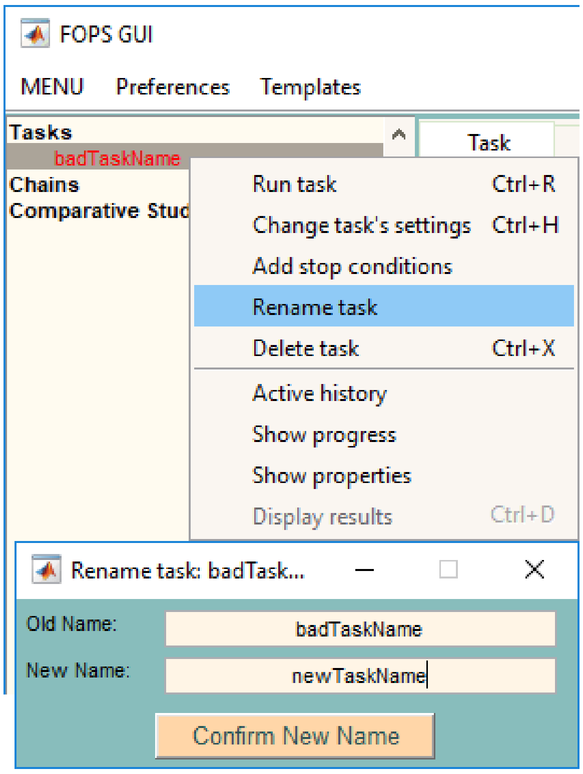


Figure 6.4: Rename task in FOPS in GUI.

6.1.3 renameTask

`fops.renameTask(oldName, newName)`

Inputs:

`oldName`	existing task id, cell of strings or doubles, cell [1 x N]
`newName`	new task name, cell of strings, cell [1 x N]

Tasks already added to FOPS can also be renamed. The method `renameTask` has two input arguments. The first argument is a current name of task and the second argument is a new name of task.

In Listing 6.11, the task named `badTaskName` is added to FOPS and afterwards it is renamed to `newTaskName`. Note that `oldName` is simple string, while `newName` is cell array. Both options are possible to both input arguments. The corresponding action done in GUI is shown in Figure 6.4.

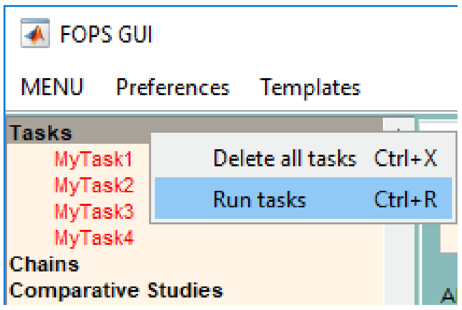
Listing 6.11: Rename task by `renameTask` method.

```
fops = FOPS(); % initialization of FOPS
fops.addTask('MOFON', 'GDE3', [], 'badTaskName')
fops.renameTask('badTaskName', {'newTaskName'})
```

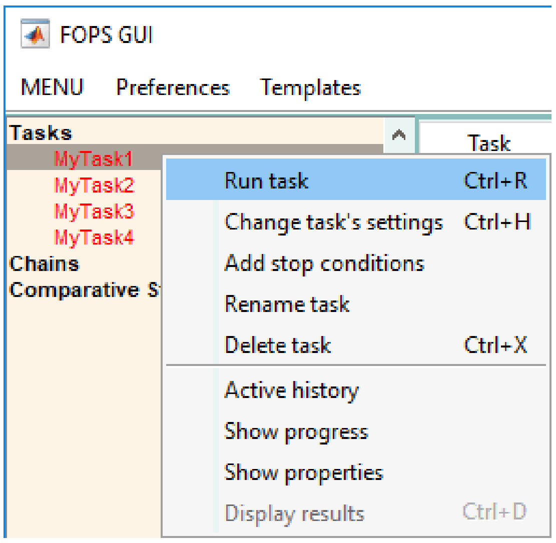
Input arguments are cell arrays due to a possibility of renaming multiple tasks at once as shown in Listing 6.12. Both input arguments are cell arrays with the same number of cells.

Listing 6.12: Rename multiple tasks at once by `renameTask` method.

```
fops = FOPS(); % initialization of FOPS
problem =       {'MOPOL';    'MOFON';     'MOKUR';     'MOSCH'};
algorithms =    {'NSGA-II';  'MOPSO';     'GDE3';      'NSGA-II'};
taskNames =     {'MyTask1';  'MyTask2';   'MyTask3';   'MyTask4'};
fops.addTask(problem, algorithms, [], taskNames)
fops.renameTask({'MyTask1', 'MyTask3', 'MyTask4'}, ...
   {'newNameOfTask1', 'newNameOfTask3', 'newNameOfTask4'})
```



(a) All tasks at once.



(b) One task at a time.

Figure 6.5: Run task in FOPS in GUI.

6.1.4 runTask

`fops.runTask(id)`

Inputs:

`id` optional, cell array of strings or doubles, cell [1 x N]

Previously created tasks are run by the method `runTask`. The method `runTask` has one optional input argument, which defines the task to be run. If no input argument is given, all tasks in FOPS are run (see Listing 6.13). Figure 6.5a shows how this can be done in FOPS GUI.

Listing 6.13: Run all tasks by `runTask` method.

```
fops = FOPS(); % initialization of FOPS
problem =       {'MOPOL';   'MOFON';    'MOKUR';    'MOSCH'};
algorithms =    {'NSGA-II'; 'MOPSO';    'GDE3';     'NSGA-II'};
taskNames =     {'MyTask1'; 'MyTask2';  'MyTask3';  'MyTask4'};
fops.addTask(problem, algorithms, [], taskNames)
fops.runTask()
```

Listing 6.14 shows how to run chosen tasks. Four tasks were added to the FOPS. First call of `runTask` method runs the third task identified by its name. Plain string (without cell brackets) as an input is again possible. The second call of `runTask` method runs first task identified by the sequence number. The last call of the `runTask` method shows how to call multiple tasks at once. Figure 6.5b shows how to run an individual task.

Listing 6.14: Run individual tasks by `runTask` method.

```
fops = FOPS(); % initialization of FOPS
problem =       {'MOPOL';   'MOFON';    'MOKUR';    'MOSCH'};
algorithms =    {'NSGA-II'; 'MOPSO';    'GDE3';     'NSGA-II'};
taskNames =     {'MyTask1'; 'MyTask2';  'MyTask3';  'MyTask4'};
fops.addTask(problem, algorithms, [], taskNames)
fops.runTask('MyTask3')          % will run third task
fops.runTask({1})                % will run first task
fops.runTask({'MyTask2', 4})     % will run remaining tasks
```

6.1.5 changeTaskSettings

`fops.changeTaskSettings(id, varargin)`

Inputs:

`id`	cell array of strings or doubles, cell [1 x N]
`varargin`	task settings. File name with path, settings struct or property-values pairs

This method changes properties of task that is already in FOPS object (see Section 7.1 to learn more about task properties). This method has variable number of input arguments. The first input argument always identifies the task to be changed (`id`). Following input arguments define properties to be changed. There are three ways to define task properties:

- Struct: Method `changeTaskSettings` has two input arguments in this case, where the second input argument is of type struct. Listing 6.15 shows how the settings struct is created.
- File: task settings is defined by content of a text file. The text file is identified by name. Therefore, the `changeTaskSettings` method has also two input arguments, where the second input argument is of type char. Setting files has to be listed in `taskSettingsList` property of FOPS object. See Listings 6.16 and 6.17 to see how the settings file is defined.
- Property-value pairs: In this case, property entries are strings similar to the field names of struct (see Section 7.1). Method `changeTaskSettings` has odd number of input arguments (including `id` argument). See Listing 6.18 to see how to use the property-value pairs input type.

Listing 6.15 shows how to change task's settings using settings struct. In an example, default settings (assigned in `addTask` method) of two NSGA-II tasks are altered. The cell array brackets are used in first `changeTaskSettings` call. It is NOT possible to change settings of multiple tasks at once, as the identification of task (`id`) in cell array might suggest. Cell array is needed for identification of task in chain or comparative study (See later in subsections 6.1.7 and **??**).

Listing 6.15: Change task's settings using settings struct.

```
fops = FOPS(); % initialization of FOPS
problem =      {'MOPOL';   'MOFON';    'MOKUR';    'MOSCH'};
algorithms =   {'NSGA-II'; 'MOPSO';    'GDE3';     'NSGA-II'};
taskNames =    {'MyTask1'; 'MyTask2';  'MyTask3';  'MyTask4'};
settings = struct('nIterations', 10, 'BP', 10);
settings.nAgents = 1000;
fops.addTask(problem, algorithms, [], taskNames)
fops.changeTaskSettings({1}, settings)
fops.changeTaskSettings('MyTask4', struct('PC', 0.5, 'nAgents', 200))
```

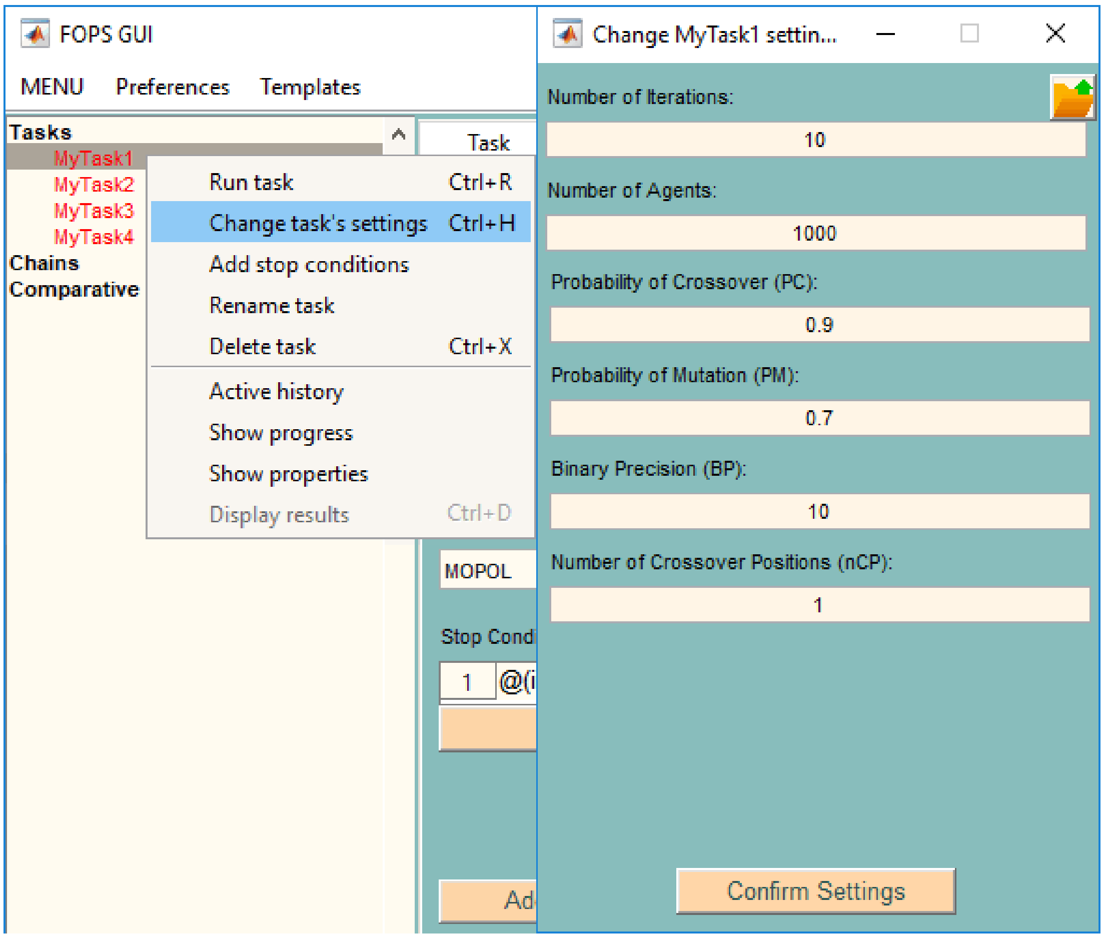
Listing 6.16 shows the usage of text file with algorithm settings. File `changeGDE3settings.txt` has to be listed in `taskSettingsList` property of FOPS (current path and `taskSettingsFiles` folders are scanned during FOPS object initialization by default). The content of the text file is shown in Listing 6.17. Note that strings of algorithm parameters (`nIters`, `nAgents`, etc.) has to be a single string (without whitespaces), otherwise the second string is taken as a parameter value and an error might occur. An 📂 icon in Figure 6.6 can be used to load the algorithm settings from the text file to `Settings Figure` of GUI.

Listing 6.16: Change task's settings using setting file.

```
fops = FOPS(); % initialization of FOPS
fops.addTask('MOFON', 'GDE3', [], 'MyTask')
filename = 'changeGDE3settings.txt';
fops.changeTaskSettings('MyTask', filename);
```

Listing 6.17: Content of settings file `changeGDE3settings.txt`.

```
% changeGDE3Settings file content
algorithm:  GDE3
nIters:     10
nAgents:    20
```



Number of Iterations: 10
Number of Agents: 1000
Probability of Crossover (PC): 0.9
Probability of Mutation (PM): 0.7
Binary Precision (BP): 10
Number of Crossover Positions (nCP): 1

Figure 6.6: Change task's settings in GUI.

```
F:          1.5
PC:         0.1
```

Listing 6.18 shows how to use the property-value input type when calling `changeTaskSettings` method. Number of agents (`nAgents`), probability of crossover (`PC`) and binary precision (`BP`) of NSGA-II task is altered. Note that binary precision (`BP`) value is a vector of three elements. This allows to set different `BP` for each decision variable of a problem. The number of elements in `BP` vector has to match the number of decision variables of problem (in this case `MOFON` problem).

Listing 6.18: Change task's settings using property - value pairs.

```
fops = FOPS(); % initialization of FOPS
fops.addTask('MOFON', 'NSGA-II')
fops.changeTaskSettings(1, 'nAgents', 150, 'PC', 0.3, 'BP', [15, 5, 10]);
```

Figure 6.6 shows how to change task's settings in GUI. After clicking on `Change tasks settings` entry of selected task from `Context Menu`, figure `Change task settings` is displayed.

6.1.6 changeTaskSettings of task in chain

Listing 6.19 shows how to change the settings of an algorithm in chain (see Section 6.2 to learn more about chain methods). The chain `MyChain` contains two tasks, NSGA-II and GDE3. Method `changeTaskSettings` changes number of agents (`nAgents`) and probability of crossover (`PC`) of the first task in the first chain. All elements of identification (`id`) can be either strings with simulation names or sequence number of individual simulations. Each call of `changeTaskSettings` method does identical operation (second and third call is only for the demonstration purpose, otherwise are useless/duplicate). The name `task1` is a default name of the first task in chain (note that FOPS automatically concatenates the default task name (`task1`) with algorithm type (`NSGA-II`) for better orientation in FOPS Tree of GUI, but it is still possible to identify the task of chain by simple task1 name). MyChain is the first chain in FOPS.

Listing 6.19: Change settings of task in chain.

```
fops = FOPS(); % initialization of FOPS
fops.addChain({'MODTLZ1'}, {'NSGAII', 'GDE3'}, [], 'MyChain')
fops.changeTaskSettings({1, 1}, 'nAgents', 150, 'PC', 0.3);
fops.changeTaskSettings({'MyChain', 1}, 'nAgents', 150, 'PC', 0.3);
fops.changeTaskSettings({'MyChain', 'task1'}, 'nAgents', 150, 'PC', 0.3);
```

6.1.7 changeTaskSettings of task in chain in comparative study

Listing 6.20 shows how to change settings of task in chain in comparative study (see Section 6.3 to learn more about comparative study methods). Comparative study MyCS contains three chains. Method changeTaskSettings changes number of agents (nAgents) and probability of crossover (PC) of first task in second chain of MyCS comparative study. All elements of identification (id) can be either strings with simulation names or sequence number of individual simulations. Note that in it is necessary to differentiate between chains and unique chains in comparative study. If changeTaskSettings is called once, it changes all the occurrences of corresponding unique chain. See what the unique chain means in the Section 6.3. Also note, that the FOPS automatically concatenates the default chain name (chain1) with the problem name for better orientation in the FOPS Tree of GUI, but it is still possible to identify the chain in CS by simple chain1 name)

Listing 6.20: Change settings of task in chain in comparative study.

```
fops = FOPS(); % initialization of FOPS
chains = {'NSGAII', 'GDE3'; 'MOPSO', []; 'MOPSO', 'NSGAII'};
nRuns = 2; requests = []; settings = [];
fops.addCompStudy('MODTLZ1', chains, nRuns, requests, settings, 'MyCS')
fops.changeTaskSettings({'MyCS', 'chain2', 'task1'}, 'nAgents', 150, 'C1', 0.3);
```

6.2 Chain Methods

FOPS methods that works with chains are very similar to the methods related to task.

6.2.1 addChain

`fops.addChain(problem, algorithms, settings, name, userData)`

Inputs:

`problem`	cell [N x 1] or cell [N x 2]
`algorithms`	cell [N x M]
`settings`	optional, cell array of structs, cell [N x M]
`name`	optional, cell [N x 1]
`userData`	optional, cell [1 x 1]

Method `addChain` has 5 input arguments: `problem`, `algorithms`, `settings`, `name` and `userData`. First two arguments are obligatory while the remaining three are optional, but has to remain in a given order.

`problem` is defined as a cell with struct with fields presented in Chapter 4 or as a cell with name of problem function (m-file) which returns such struct.

`algorithms` argument defines optimization methods that will be exploited on a single optimization problem one after another.

`settings` argument specifies parameters of optimization algorithms. It involves number of agents (`nAgents`), number of iterations (`nIters`) and algorithm dependent parameters such as probability of mutation (`PM`), inertia weight (`W`) etc. It is a cell array of structs with field names suggesting type of parameter (see Section 7.1).

`name` is a unique chain name. Such names are shown in the `FOPS Tree` (see Figure 5.1) and are used to identify chains in FOPS.

Last input argument is `userData`, which holds user's entries. It can be employed by user in fitness function if `fullControl` property of problem is set to true (see Section 4.16). Argument `userData` will be passed to elementary OptTasks in Chain object. Property `userData` of all tasks in chain will be identical.

A minimal working example of chain creation is shown in Listing 6.21. Note that a chain with only one task is possible, but it has no particular meaning to use such a chain. All input arguments are supposed to be cell arrays, but if cell array brackets are omitted, input is automatically converted to a cell array. Therefore, both variants in Listing 6.21 are actually identical. Default task's settings and chain name is assigned if no `settings` and chain `name` is entered.

Listing 6.21: Minimal working example of `addChain` method.

```
fops = FOPS(); % initialization of FOPS
fops.addChain('MOFON', 'NSGA-II')
fops.addChain('MOFON', {'NSGA-II'})
```

In Listing 6.22, a chain with three algorithms is added and some algorithm's parameters and chain name are specified. Property `userData` is also entered. First and second tasks will have 1000 agents and will perform 10 iterations. Both, NSGA-II and GDE3, algorithms have probability of crossover (`PC`) set to 0.3. Third algorithm, the MOPSO one, will be set as default along with unspecified NSGA-II and GDE3 properties. See Section 7.1 to learn more about task's properties.

Listing 6.22: Add completely specified chain with `addChain` method.

```
fops = FOPS(); % initialization of FOPS
mySet = struct('nIterations', 10, 'PC', 0.3);
mySet.nAgents = 1000;
chainName = 'myNewChain';
userData = {[1, 2, 3], [4, 5, 6]};
```

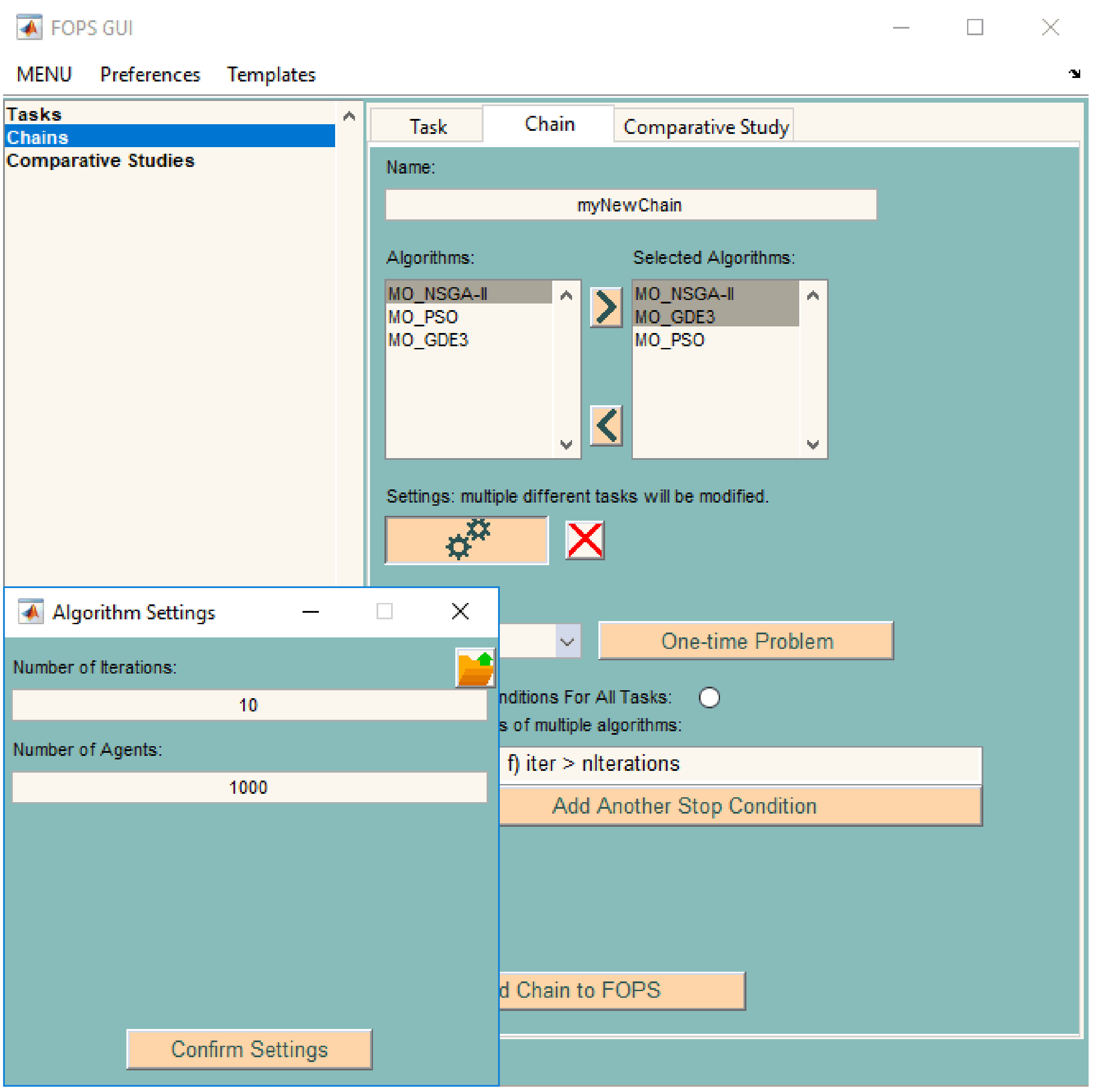


Figure 6.7: Add chain to FOPS and define algorithms' settings in GUI.

```
fops.addChain(...
   {'MODTLZ3'}, ...  % problem
   {'NSGA-II', 'GDE3',  'MOPSO'}, ...  % algorithms
   {mySet,     mySet,   []}, ...  % algoritmh settings
   chainName, ...
   userData)
```

The same operation (except for `userData`) can be done in GUI as shown in Figure 6.7. Algorithms NSGA-II, GDE3 and MOPSO are selected in `Algorithms` listbox and added to `Selected Algorithms` listbox by ❯ push button. They can be removed by ❮ push button. Problem `MOPOL` is selected in `Problem` popup menu. To alter algorithms settings select one or more algorithms in `Selected Algorithms` listbox and click on ⚙ toggle button. When two or more different algorithms are present in `Selected Algorithms` listbox, `Algorithm Settings` figure shows only `Number of Iterations` and `Number of Agents` fields. Modifications of other parameters has to be done separately.

It is possible to use one-time problem similarly as shown in Listing 6.5 and also define one-time problem in GUI by clicking on `One-time Problem` toggle button (see Figure 6.2).

Some optimization problems can be scalable, which enables user to specify e.g. how many decision variables or fitness function the problem has. Listing 6.23 shows how to define number of decision variables and number of objective functions for problem MODTLZ1 (Definition of MODTLZ1 problem can be seen Listing 4.1). The problem will have 20 decision variables (nVars) and 4 objective functions (nObjs) and will be optimized by chain of algorithms consisting of MOPSO and GDE3.

The Listing 6.23 shows how to exploit problem function input argument by FOPS methods. Problem input argument can be either scalar number or struct. Input argument of problem cannot be defined from FOPS GUI.

Listing 6.23: Define number of decision variables and number of objective of problem in chain.

```
fops = FOPS(); % initialization of FOPS
fops.addChain({'MODTLZ1', struct('nVars', 20, 'nObjs', 4)}, {'MOPSO', 'GDE3'})
```

User can add multiple chains using addChain method just once. The Listing 6.24 shows how to add 4 chains at once. In problem variable, each row belongs to one chain. Empty cell of settings cell array stands for default settings of corresponding task. All tasks in chain will have identical userData property containing cell array with 2 strings.

When adding 4 chains at once, problem cell has to have 4 rows and 1 or two columns, algorithms cell has to have again 4 rows and arbitrary number of columns, settings cell has to be of same size as algorithms cell and name cell has to have 4 elements. Problems has to be strictly written in rows as shown in Listing 6.24, otherwise string input in second row of problem cell array would cause an error.

To create multiple chains with different userData properties user needs to add chains separately by using multiple-times addChain method.

Listing 6.24: Add 4 chain at once with method addChain.

```
fops = FOPS(); % initialization of FOPS
myTempProblem = struct('limits', [0, 0; 1, 1], 'fitness', @(x)myFitnessF(x));
problem = {...
   'MOPOL',        [];...
   myTempProblem, [];...
   'MOZDT1',       25; ...
   'MODTLZ5',      struct('nVars', 8, 'nObjs', 6)};
algorithms = {...
   'NSGAII',    'MOPSO', 'GDE3';  ...  % chain 1
   'MOPSO',     'GDE3',  [];      ...  % chain 2
   'NSGAII',    [],      [];      ...  % chain 3
   'MOPSO',     [],      []};          % chain 4
NSGAIIset = struct('nIterations', 1000, 'PC', 0.2, 'PM', 0.2);
MOPSOset = struct('nAgents', 1000, 'C1', 0.2, 'C2', 0.2);
GDE3set = struct('F', 1.5);
settings = {...
   NSGAIIset,  [],      GDE3set; ...   % this belongs to first chain
   MOPSOset,   [],      []     ; ...   % this belongs to second chain
   [],         [],      []     ; ...   % this belongs to third chain
   MOPSOset,   [],      []};           % this belongs to fourth chain
name = {'MyChain1'; 'MyChain2'; 'MyChain3'; 'MyChain4'};
userData = {'This userData will', 'be the same for all chains'};
fops.addChain(problem, algorithms, settings, name, userData)
```

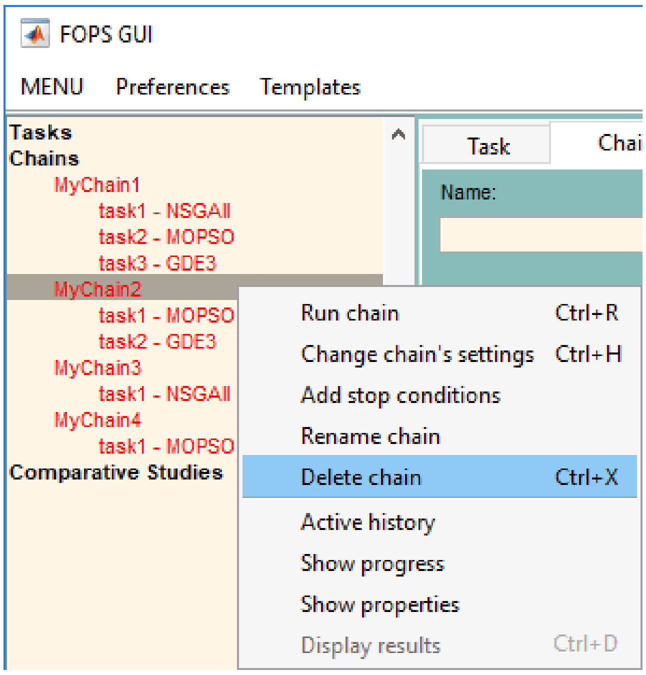
In Figure 6.7 can be seen control elements which modifies stop conditions of tasks in chain. See Subsection 6.4.1 to learn more about stop conditions.

6.2.2 deleteChain

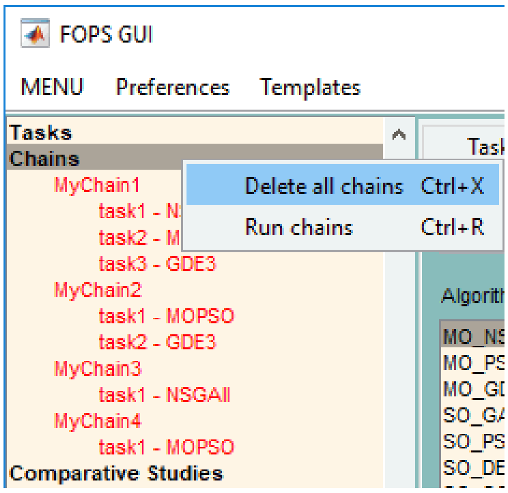
`fops.deleteChain(id)`

Inputs:

id	cell array of strings or doubles, cell [1 x N]



(a) One chain at a time.



(b) All chains at once.

Figure 6.8: Delete chain from FOPS in GUI.

Method deleteChain has only one input argument - identification (id) of chain to delete.

Listing 6.25 shows how to delete chain from FOPS. At the beginning, four chains are added to FOPS. Chains to delete can be identified by its name or by sequence number in FOPS. Only MyChain1 remains in FOPS at the end of Listing 6.25.

Listing 6.25: Delete chain from FOPS with method deleteChain.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'; 'MOKUR'; 'MOSCH'};
algorithms = {...
    'NSGAII',    'MOPSO', 'GDE3';   ...  % chain 1
    'MOPSO',     'GDE3',  [];       ...  % chain 2
    'NSGAII',    [],      [];       ...  % chain 3
    'MOPSO',     [],      []};           % chain 4
chainNames = {'MyChain1'; 'MyChain2'; 'MyChain3'; 'MyChain4'};
fops.addChain(problem, algorithms, [], chainNames)
fops.deleteChain(2)
fops.deleteChain('MyChain3')
fops.deleteChain({'MyChain4'})
```

Similar action is possible using GUI. Figure 6.8a shows how to delete chain from FOPS in GUI. The Context Menu is displayed by right mouse click in the FOPS Tree after selection of chain to delete by left mouse click.

Listing 6.26 shows how to delete multiple chains with one command. At the end of Listing 6.26 only MyChain3 is present in FOPS. Figure 6.8b shows how to delete all chains at once from FOPS in GUI. This Context Menu is called by selecting Chains group and right mouse button click.

Listing 6.26: Delete multiple chains from FOPS at once with method deleteChain.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'; 'MOKUR'; 'MOSCH'};
algorithms = {...
   'NSGAII',    'MOPSO', 'GDE3';   ...  % chain 1
   'MOPSO',     'GDE3',  [];       ...  % chain 2
   'NSGAII',    [],      [];       ...  % chain 3
   'MOPSO',     [],      []};           % chain 4
chainNames = {'MyChain1'; 'MyChain2'; 'MyChain3'; 'MyChain4'};
fops.addChain(problem, algorithms, [], chainNames)
fops.deleteChain({'MyChain1', 2, 4})
```

Note that an effect of three partial calls of deleteChain (See Listing 6.27) instead of one call in 6.26 is completely different. By first call, first chain MyChain1 is deleted. By second call of deleteChain, the chain MyChain3 is deleted, because it now has sequence number of 2. Third call of deleteChain would cause an error, because only two chains remains in FOPS object.

Listing 6.27: Three partial use of deleteChain leads to different output.

```
...
fops.deleteChain('MyChain1')
fops.deleteChain(2)
fops.deleteChain(4)
```

6.2.3 renameChain

`fops.renameChain(oldName, newName)`

Inputs:

oldName	existing chain name, cell of strings or doubles, cell [1 x N]
newName	new chain name, cell of strings, cell [1 x N]

Chains already added to FOPS can also be renamed. Method renameChain has two input arguments. The first argument (oldName) is a current name of chain and second argument (newName) contains new name of chain.

In Listing 6.28, chain named badChainName is added to FOPS and afterwards it is renamed to newChainName. Note that oldName is simple string, while newName is cell array. Both options are possible to both input arguments. The corresponding action done in GUI is shown in Figure 6.9.

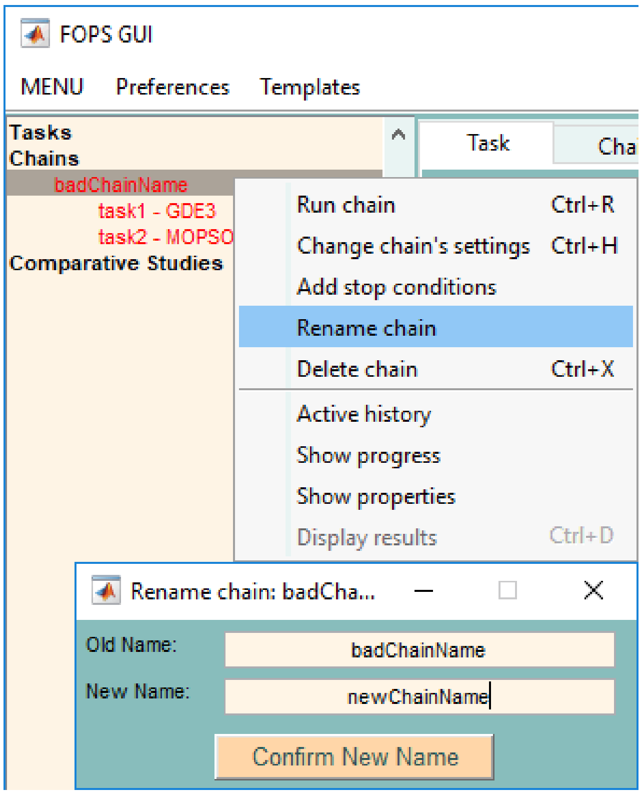
Listing 6.28: Rename chain named badChainName to newChainName.

```
fops = FOPS(); % initialization of FOPS
fops.addChain('MOFON', {'GDE3', 'MOPSO'}, [], 'badChainName')
fops.renameChain('badChainName', {'newChainName'})
```

Input arguments are originally cell arrays due to possibility of renaming multiple chains at once as shown in Listing 6.29. Both input arguments (oldName and newName) are cell arrays with the same number of cells.

Listing 6.29: Rename multiple chains at once.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'; 'MOKUR'; 'MOSCH'};
algorithms = {...
   'NSGAII',    'MOPSO', 'GDE3';   ...  % chain 1
   'MOPSO',     'GDE3',  [];       ...  % chain 2
   'NSGAII',    [],      [];       ...  % chain 3
   'MOPSO',     [],      []};           % chain 4
chainNames = {'MyChain1'; 'MyChain2'; 'MyChain3'; 'MyChain4'};
fops.addChain(problem, algorithms, [], chainNames)
fops.renameChain({'MyChain1', 'MyChain3', 'MyChain4'}, ...
   {'newNameOfChain1', 'newNameOfChain3', 'newNameOfChain4'})
```



Old Name: badChainName
New Name: newChainName

Figure 6.9: Rename chain in FOPS in GUI.

6.2.4 runChain

`fops.runChain(id)`

Inputs:

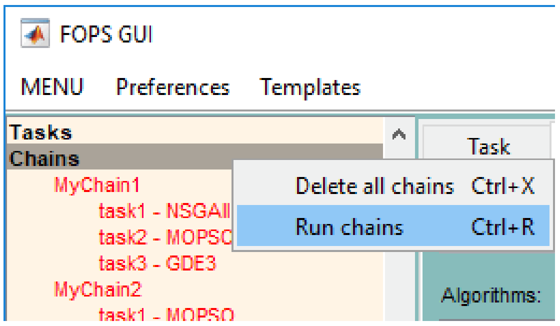
`id`	optional, cell array of strings or doubles, cell [1 x N]

Previously created chains are run by method `runChain`. Method `runChain` has one optional input argument (`id`) which defines chain to be run. If no input argument is given, all chains in FOPS will be run (see Listing 6.30). Figure 6.10a shows how all chains can be run in FOPS GUI.

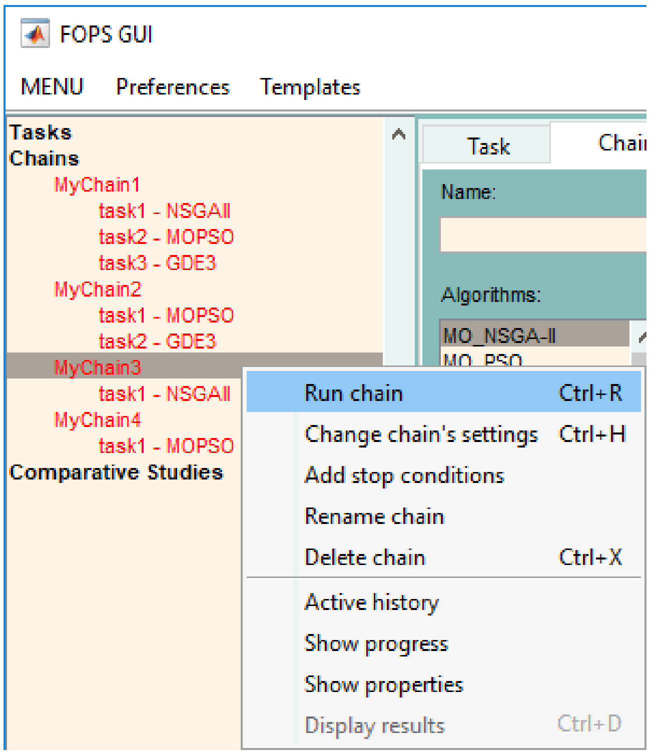
Listing 6.30: Run all chains in FOPS.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'; 'MOKUR'; 'MOSCH'};
algorithms = {...
   'NSGAII',    'MOPSO', 'GDE3';   ...  % chain 1
   'MOPSO',     'GDE3',  [];       ...  % chain 2
   'NSGAII',    [],      [];       ...  % chain 3
   'MOPSO',     [],      []};           % chain 4
chainNames = {'MyChain1'; 'MyChain2'; 'MyChain3'; 'MyChain4'};
fops.addChain(problem, algorithms, [], chainNames)
fops.runChain()
```

Listing 6.31 shows how to run chosen chains. Four chains were added to FOPS. First call of `runChain` method runs third chain identified by name `MyChain3`. A plain string as an input is possible. Second call of `runChain` method runs first chain identified by sequence number. Last call of `runChain` method shows how to call multiple chains at once. Figure 6.10b shows how to run individual chain.



(a) All chains at once.



(b) One chain at a time.

Figure 6.10: Run chain in FOPS in GUI.

Listing 6.31: Run individual chain in FOPS.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'; 'MOKUR'; 'MOSCH'};
algorithms = {...
    'NSGAII',    'MOPSO', 'GDE3';   ... % chain 1
    'MOPSO',     'GDE3',  [];       ... % chain 2
    'NSGAII',    [],      [];       ... % chain 3
    'MOPSO',     [],      []};          % chain 4
chainNames = {'MyChain1'; 'MyChain2'; 'MyChain3'; 'MyChain4'};
fops.addChain(problem, algorithms, [], chainNames)
fops.runChain('MyChain3')  % will run third chain
fops.runChain({1})  % will run first chain
fops.runChain({'MyChain2', 4})  % will run remaining chains
```

6.2.5 changeChainSettings

`fops.changeChainSettings(id, varargin)`

Inputs:

`id`	cell array of strings or doubles, cell [1 x N]
`varargin`	chain settings. Settings struct or property-values pairs

This method changes properties of chain that is already in FOPS object. Basically, only problem and algorithms in chain can be changed. This method has variable number of input arguments. The first input argument always identifies chain (`id`) of which settings will be changed. Other input arguments define properties to be changed (`varargin`). There are two ways to define chain settings:

- Struct: struct with fields `problem` or `algorithms`. Method `changeChainSettings` has two input arguments in this case, where the second input argument is of type struct.

- Property-value pairs: In this case, property entries are strings `problem` or `algorithms`. Method `changeChainSettings` has odd number of input arguments (including chain `id` argument).

Listing 6.32 shows how to change problem of chain using struct. Initially, a problem of the chain is `MOZDT1`, but it is changed to `MOZDT6` problem. Chain identification (`id`) is a cell array, but it is also possible to omit the cell array brackets. It is NOT possible to change settings of multiple chains at once, as the identification of chain (`id`) in cell array might suggest. Cell array is needed for identification of chain in comparative study (See later in this section in Listing 6.37).

Listing 6.32: Change problem of chain from `MOZDT1` to `MOZDT6`.

```
fops = FOPS(); % initialization of FOPS
fops.addChain({'MOZDT1'}, {'NSGAII', 'MOPSO', 'GDE3'}, [], {'chainToModify'});
fops.changeChainSettings({'chainToModify'}, struct('problem', 'MOZDT6'));
```

Listing 6.33 shows how to change a problem of chain and specify its input arguments (Definition of `MODTLZ1` problem can be seen in Listing 4.1). Note that there are double cell array brackets. It is necessary to ensure that the input struct has only one element, because the struct `problem` created in Listing 6.34 is of size [1 x 2] which is unsuitable for `changeChainSettings` method.

Listing 6.33: Change problem of chain and specify problem input arguments.

```
fops = FOPS(); % initialization of FOPS
fops.addChain({'MOZDT2'}, {'NSGAII', 'MOPSO', 'GDE3'}, [], {'chainToModify'});
problem = {{'MODTLZ1', struct('nVars', 25, 'nObjs', 5)}};
fops.changeChainSettings({'chainToModify'}, struct('problem', problem));
```

Listing 6.34: Invalid problem struct if cell brackets are omitted.

```
problem = {'MODTLZ1', struct('nVars', 25, 'nObjs', 5)}
```

Listing 6.35 shows how to change algorithms in chain using property-value pairs.

Listing 6.35: Change tasks in chain using property-value pair.

```
fops = FOPS(); % initialization of FOPS
fops.addChain({'MOZDT1'}, {'NSGAII', 'MOPSO', 'GDE3'}, [], {'chainToModify'});
fops.changeChainSettings({'chainToModify'}, 'algorithms', {'GDE3', 'MOPSO'});
```

Listing 6.36 shows how to change a problem of chain and specify the number of decision variables. Note that in this case only one set of cell array brackets is required.

Listing 6.36: Change problem of chain and specify input arguments by property - value pair.

```
fops = FOPS(); % initialization of FOPS
fops.addChain({'MOZDT1'}, {'NSGAII', 'MOPSO', 'GDE3'}, [], {'chainToModify'});
fops.changeChainSettings({'chainToModify'}, 'problem', {'MOZDT4', 20});
```

Listing 6.37 shows how to change algorithms of unique chain in comparative study (see Section 6.3 to learn more about comparative study methods). Comparative study has 2 chains: first chain contains MOPSO and NSGA-II tasks and second chain contains NSGA-II and GDE3 tasks. There are three problems in comparative study and number of runs is four, therefore the `chains` property of comparative study will be of size [4 x 6] (4 - `nRuns`, 6 - 2 unique chains times 3 problems). Method `changeChainSettings` changes algorithms of first unique chain. If there are multiple problems and/or multiple runs in comparative study, the changes of unique chain are executed for all occurences of unique chain in comparative study. It is not possible to change problem of individual chain in comparative study, because it would cause inconsistency in CompStudy object. Use `changeCompStudySettings` method instead (see Subsection 6.3.5).

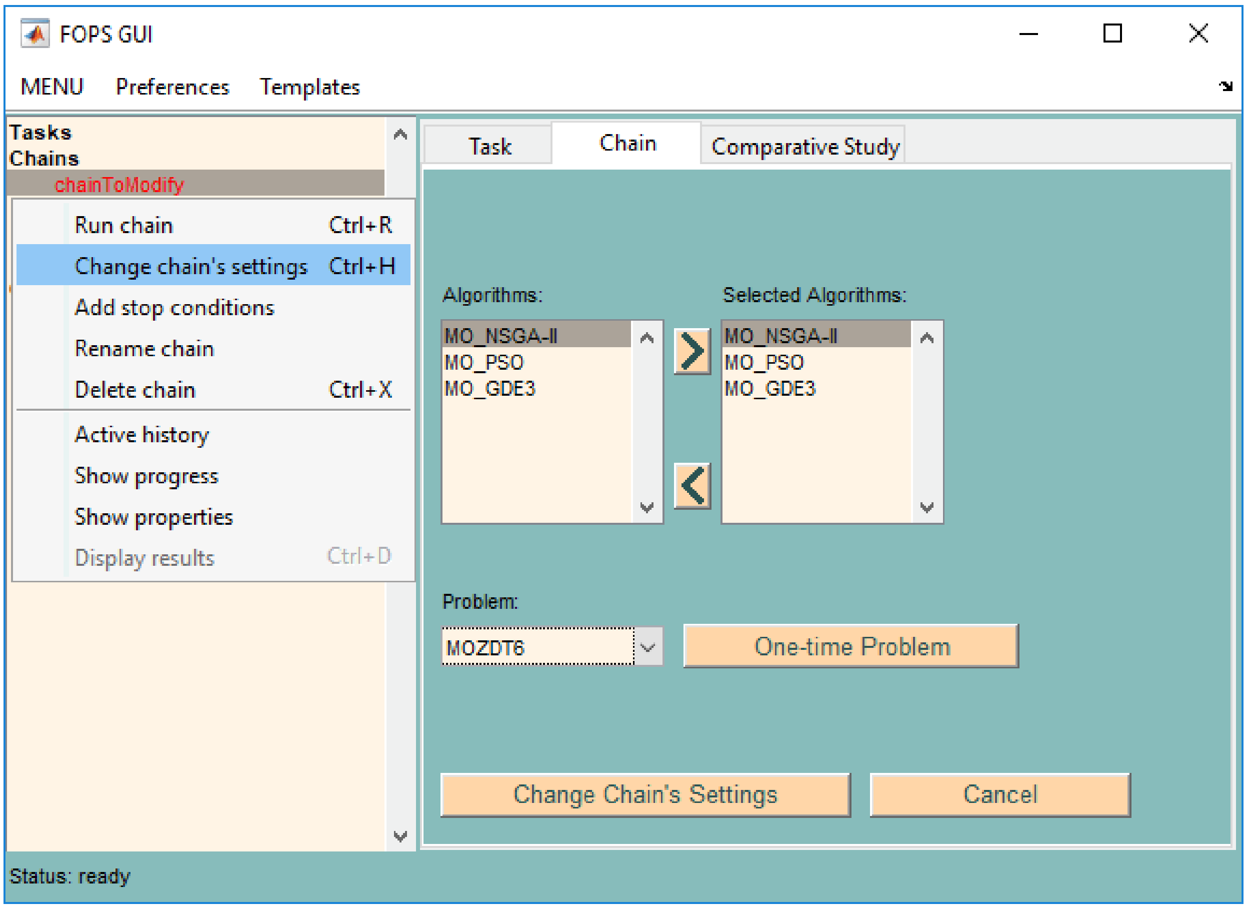


Figure 6.11: Change chain's settings in GUI.

Listing 6.37: Change problem and algorithms of chain in comparative study.

```
fops = FOPS(); % initialization of FOPS
chains = {'MOPSO', 'NSGAII'; 'NSGAII', 'GDE3'};
problems = {'MOZDT1'; 'MOPOL'; 'MOSCH'};
nRuns = 4; requests = []; settings = [];
fops.addCompStudy(problems, chains , nRuns, requests, settings, 'CSToModify');
fops.changeChainSettings({'CSToModify', 1}, 'algorithms', {'GDE3', 'NSGA-II', 'MOPSO'})
```

Note that identification of chain (`id`) in `changeChainSettings` method in Listing 6.37 is cell array of two elements, where the first element is an identification of a comparative study, while the second element is an identification of a chain in comparative study `CSToModify`. All the elements of identification (`id`) can be either strings with simulation names or sequence number of individual simulations. Listing 6.38 shows all the possibilities how to identify an identical chain. Name `chain1` is a default name of the first chain in the comparative study. `CSToModify` is the first comparative study in FOPS.

Also note that the FOPS automatically concatenates default chain name (`chain1`) with the problem (e.g. `MOPOL`) for better orientation in `FOPS Tree` of GUI, but it is still possible to identify chain in comparative study by simple `chain1` name.

Listing 6.38: Possible ways how to identify first chain of comparative study in FOPS.

```
...
fops.changeChainSettings({'CSToModify', 'chain1'}, ...
...
fops.changeChainSettings({1, 1}, ...
...
fops.changeChainSettings({1, 'chain1'}, ...
...
```

Figure 6.11 shows how to change chain settings in FOPS GUI. `Context Menu` is induced by right mouse click on previously selected chain. After selecting `Change chain's settings` in the `Context Menu`, tab `Chain`, originally for adding chains, is transformed to change settings mode (control elements of name, algorithm settings and stop conditions are hidden). This suggest the drawback of `changeChainSettings` method: if algorithms are changed, they are initialized with default settings and default stop condition (stop when `nIterations` is reached).

To change settings of tasks in chain, `changeTaskSettings` method has to be used (see Subsection 6.1.5, particularly Listing 6.19). Stop conditions can be added by `addStopCondition` method (see Subsection 6.4.1).

6.3 Comparative Study Methods

6.3.1 addCompStudy

`fops.addCompStudy(problem, chain, nRuns, metrics, settings, name, userData)`

Inputs:

problem	cell [N x 1] or cell [N x 2]
chain	cell [X x Y]
nRuns	optional, number of runs (repetitions), double [1 x 1]
metrics	optional, cell array of string, cell [M x 1]
settings	optional, cell array of structs, cell [X x Y]
name	optional, unique name of comparative study, cell [1 x 1]
userData	optional, cell [1 x 1]

Method addCompStudy has 7 input arguments: problem, algorithm, nRuns, metrics, settings, name and userData. First two input arguments are obligatory while the remaining five are optional, but has to remain in a given order.

The problem is defined as cell array with names of problem functions (m-files) or problem struct (see Chapter 4 for more info about problem definition).

The chain defines unique chains that will be processed by comparative study ("unique chain" suggests, that the chain will be repeated in chains property of CompStudy object according to the number of problems. Each repetition will have different problem assigned).

The nRuns defines the number of repetitions of all the elementary simulations in comparative study.

The metrics is a cell array of metric names (see Chapter 8).

The settings argument defines parameters of optimization algorithms in unique chains. It has to have the same dimensions as the chains property. It is a cell array of structs with field names suggesting types of parameters (see Section 7.1 for more info about task settings).

The name is a unique comparative study name. Such names are shown in FOPS Tree (see Figure 5.1) and are used to identify comparative studies.

Last input argument is userData, which can hold user's entries. It can be employed by user in fitness function if fullControl property of problem is set to true (see Section 4.16).

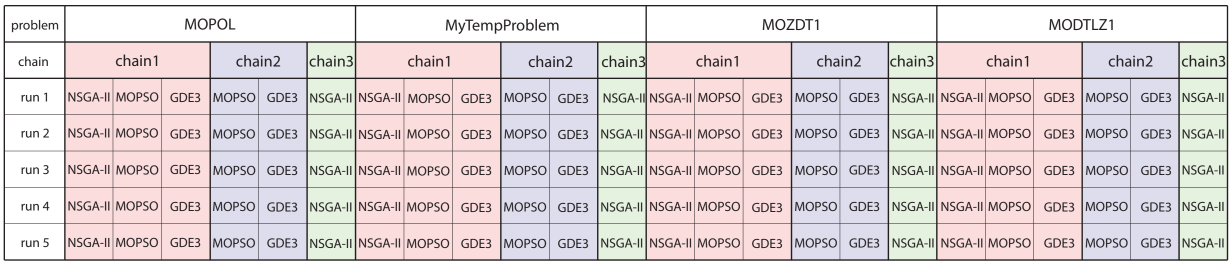
A minimal working example of adding comparative study can be seen in Listing 6.39. Only problem and chain input arguments are specified. First and third problems were predefined (benchmark problems) while the second one is a one-time problem. The first chain has three tasks and the second chain has only one task. In the chains property of CompStudy object will be created 6 different chains. They are all possible combinations of the two unique chains and three problems inserted to the addCompStudy method. The nRuns will be 1 by default, algorithm settings (see Section 7.1) and comparative study name will be also set as default. No metrics will be calculated.

Listing 6.39: Minimal working example of addCompStudy method.

```
fops = FOPS(); % initialization of FOPS
problemStruct = struct('limits', [0, 0;1, 1], 'fitness', @(x)Myfitness(x));
fops.addCompStudy(...
   {'MOZDT1'; problemStruct; 'MOZDT3'}, ... problems
   {'NSGAII',  'MOPSO', 'GDE3'; ... algorithms: chain1
   'NSGAII',   [],      []}) ...  % algorithms: chain2
```

Listing 6.40 shows all the features of comparative study.

The variable problem is a cell array, with arbitrary number of rows. The second column of problem serves for input arguments of problem functions (see Chapter 4 or Listing 6.6 for more



problem	MOPOL						MyTempProblem						MOZDT1						MODTLZ1					
chain	chain1			chain2		chain3	chain1			chain2		chain3	chain1			chain2		chain3	chain1			chain2		chain3
run 1	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II
run 2	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II
run 3	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II
run 4	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II
run 5	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II	NSGA-II	MOPSO	GDE3	MOPSO	GDE3	NSGA-II

Figure 6.12: Structure of `chains` property of comparative study in FOPS from Listing 6.40.

info about problem function input arguments). Therefore, fourth problem `MODTLZ5` will have `8` decision variables and `6` objective functions.

Variable `chain` is a cell array with names of optimization algorithms, that will be employed by comparative study. Each row of cell array stand for one unique chain.

Variable `nRuns` denotes that each chain in comparative study will be run `2` times. Argument `nRuns` is for statistical purposes.

After the comparative study is run, property `results` of CompStudy object, will in this case contains fields `GD` (generational distance) and `T` (computational time).

The dimensions of `settings` variable follows the dimensions of `chain` variable. Empty cell in `settings` cell array results in default algorithm settings (see Section 7.1 to learn more about default task properties values).

`MyCS` string stands for comparative study's unique `name`.

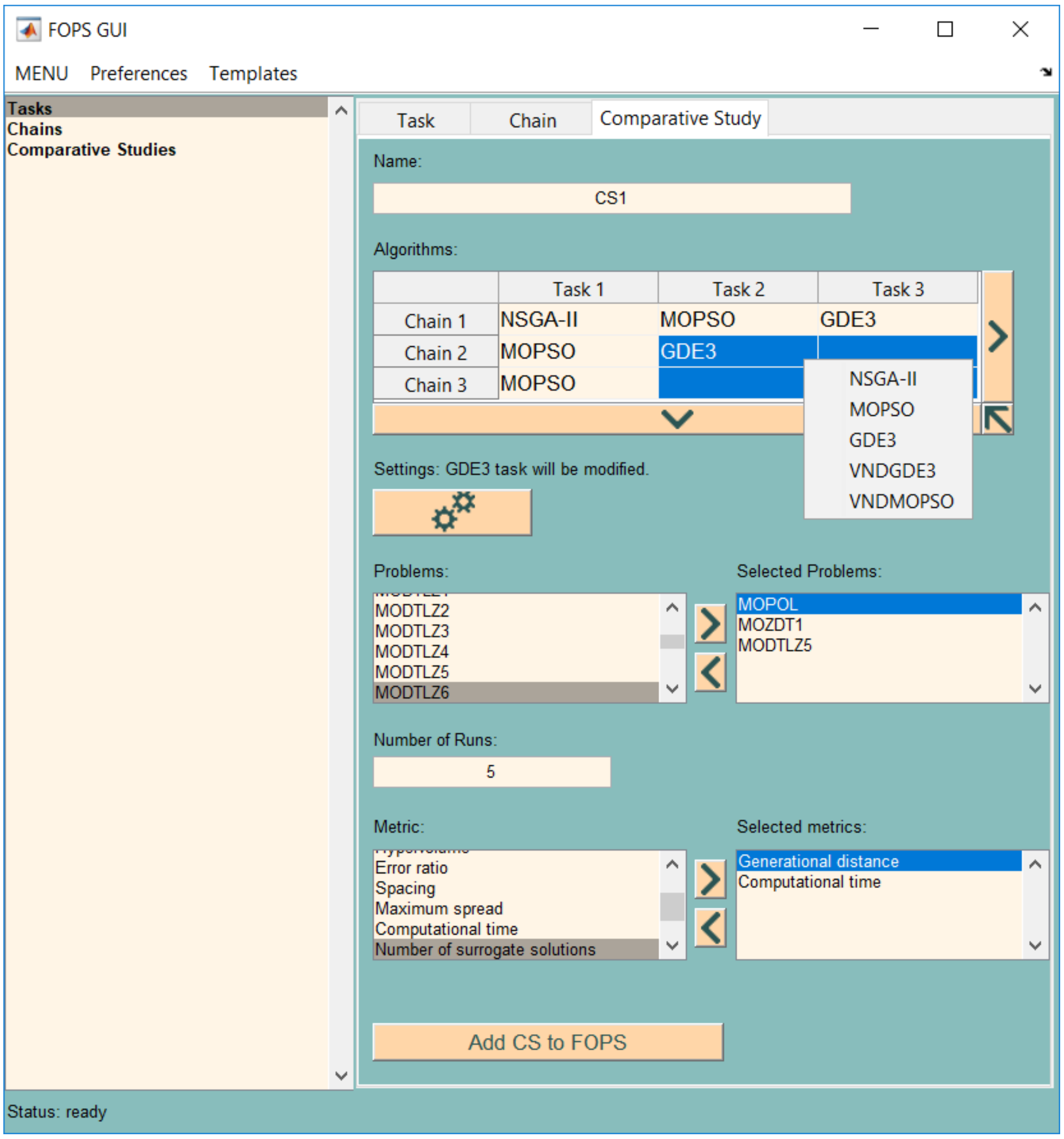
Input argument `userData` is copied to every task in every chain of comparative study.

The `chains` property of comparative study in FOPS object will contain array of Chain objects of size [12 x 5]. Number 12 is a product of number of problems (4 - `problem` variable) and number of unique chains (3 - `chain` variable). Number 5 indicates number of repetitions of every chain in comparative study (`nRuns` = 5). Figure 6.12 shows the `chains` property of comparative study created in Listing 6.40.

Listing 6.40: Add completely defined comparative study.

```
fops = FOPS(); % initialization of FOPS
myTempProblem = struct('limits', [0, 0; 1, 1], 'fitness', @(x)MyFitness(x));
problem = {...
   'MOPOL',        [];...
   myTempProblem, [];...
   'MOZDT1',       25; ...
   'MODTLZ5',      struct('nVars', 8, 'nObjs', 6)};
chain = {...
   'NSGAII',    'MOPSO', 'GDE3';   ...  % chain 1
   'MOPSO',     'GDE3',  [];       ...  % chain 2
   'NSGAII',    [],      []};      ...  % chain 3
nRuns = 5;
metrics = {'GD', 'computationalTime'};
NSGAIIset = struct('nIterations', 1000, 'PC', 0.2, 'PM', 0.2);
MOPSOset = struct('nAgents', 1000, 'C1', 0.2, 'C2', 0.2);
GDE3set = struct('F', 1.5);
settings = {...
   NSGAIIset,   [],      GDE3set; ...   % this belongs to first chain
   MOPSOset,    [],      []     ; ...   % this belongs to second chain
   [],          [],      []};     ...   % this belongs to third chain
userData = {'This is userData', 'that will be same for all simulations in CS'};
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS', userData);
```

Figure 6.13 shows how to set and add comparative study in FOPS GUI. Algorithms are added using `Algorithm Context Menu`, which is cast by selecting a cell in the `Algorithms` table. As suggested in Figure 6.13, it is also possible to select multiple cells at once and assign all of them the same algorithm (use right mouse button to cast `Algorithm Context Menu`. `Algorithms` table



Name: CS1

Algorithms:

	Task 1	Task 2	Task 3
Chain 1	NSGA-II	MOPSO	GDE3
Chain 2	MOPSO	GDE3	
Chain 3	MOPSO		

NSGA-II
MOPSO
GDE3
VNDGDE3
VNDMOPSO

Settings: GDE3 task will be modified.

Problems:
MODTLZ2
MODTLZ3
MODTLZ4
MODTLZ5
MODTLZ6

Selected Problems:
MOPOL
MOZDT1
MODTLZ5

Number of Runs: 5

Metric:
Error ratio
Spacing
Maximum spread
Computational time
Number of surrogate solutions

Selected metrics:
Generational distance
Computational time

Status: ready

Figure 6.13: Add comparative study to FOPS in GUI.

is expanded by > and ∨ push buttons.

To alter algorithms settings select one or more algorithms in `Algorithms` table and click on ⚙ toggle button. When two or more different algorithms are selected in `Algorithms` table, the `Algorithm Settings` figure will show only `Number of Iterations` and `Number of Agents` fields. Modifications of other task parameters has to be done separately.

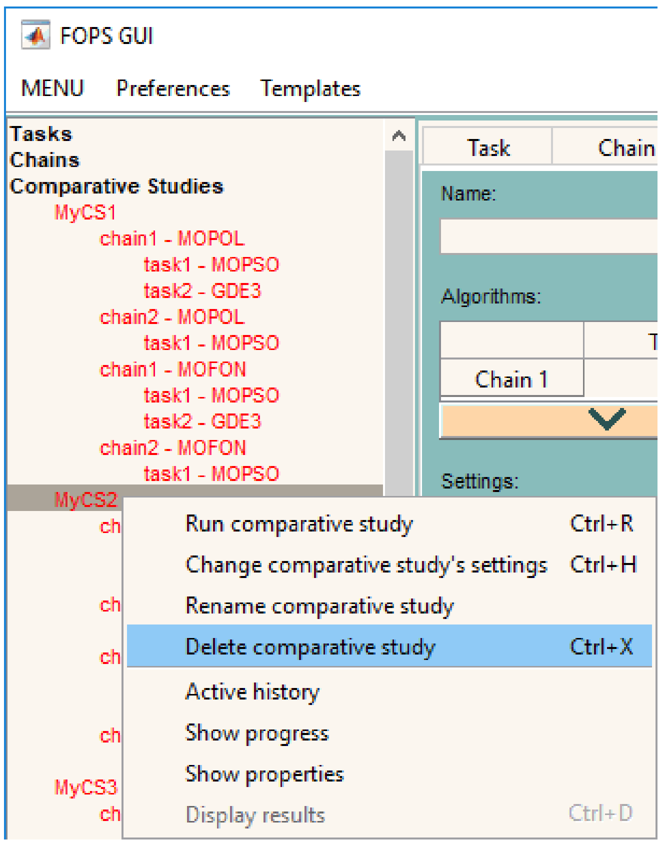
Problems are selected from the `Problems` listbox and added to `Selected Problems` listbox by > pushbutton. Problems can be removed by < push button. Note that it is not possible to add one-time problem (see Figure 6.2 to learn what is the one-time problem) in case of comparative study in GUI mode. Users are encouraged to generate the problem m-file by the problem figure accessible from task or chain tabs (see Figure 6.2 in Subsection 6.1.1 or Chapter 4 to see how to create such a function).

Selection of metrics works identically as selection of problems.

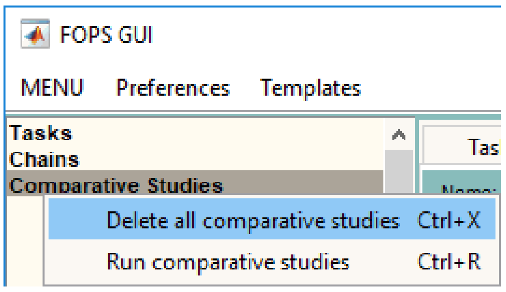
6.3.2 deleteCompStudy

```
fops.deleteCompStudy(id)
```

Inputs:



(a) One comparative study at a time.


FOPS GUI
MENU
Preferences
Templates
Tasks
Chains
Comparative Studies
Delete all comparative studies Ctrl+X
Run comparative studies Ctrl+R

(b) All comparative studies at once.

Figure 6.14: Delete comparative study from FOPS in GUI.

id	cell array of strings or doubles, cell [1 x N]

Method deleteCompStudy has only one input argument - identification of comparative study to delete (id).

Listing 6.41 shows how to delete comparative study from FOPS. At the beginning, four comparative studies are added to FOPS. Comparative studies to delete can be identified by its name or by its sequence number in FOPS object. Only MyCS1 remains in FOPS at the end of Listing 6.41.

Listing 6.41: Delete comparative study from FOPS.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'};
chain = {...
    'MOPSO',     'GDE3';       ...  % chain 1
    'MOPSO',     []};            % chain 2
nRuns = []; metrics = []; settings = [];
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS1')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS2')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS3')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS4')
fops.deleteCompStudy(2)
fops.deleteCompStudy('MyCS3')
fops.deleteCompStudy({'MyCS4'})
```

Similar action is possible using GUI. Figure 6.14a shows how to delete comparative study from FOPS in GUI. The context menu is displayed by right mouse click in FOPS Tree after selection of comparative study to delete by left mouse click.

Listing 6.42 shows how to delete multiple comparative studies with one command. At the end of Listing 6.42 only MyCS3 remains in FOPS.

Listing 6.42: Delete multiple comparative studies from FOPS at once.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'};
chain = {...
   'MOPSO',    'GDE3';      ...  % chain 1
   'MOPSO',    []};             % chain 2
nRuns = []; metrics = []; settings = [];
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS1')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS2')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS3')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS4')
fops.deleteCompStudy({'MyCS1', 2, 4})
```

Note that an effect of three partial calls of `deleteCompStudy` (See Listing 6.43) instead of the one used in Listing 6.42 is completely different. By the first call, the first comparative study `MyCS1` is deleted. By the second call of `deleteCompStudy`, the comparative study `MyCS3` is deleted, because it now has sequence number 2. The third call of `deleteCompStudy` would cause an error, because only two comparative studies remains in FOPS object.

Listing 6.43: Three partial calls of `deleteCompStudy` method.

```
...
fops.deleteCompStudy('MyCS1')
fops.deleteCompStudy(2)
fops.deleteCompStudy(4)
```

Figure 6.14b shows how to delete all comparative studies at once from FOPS in GUI. This `Context Menu` is called by selecting `Comparative studies` group and right mouse button click.

6.3.3 renameCompStudy

`fops.renameCompStudy(oldName, newName)`

Inputs:

`oldName`	existing comparative study name, cell of strings or doubles, cell [1 x N]
`newName`	new comparative study name, cell of strings, cell [1 x N]

Comparative study already added to FOPS can also be renamed. Method `renameCompStudy` has two input arguments. The first argument is a current name of comparative study and second argument contains new name of comparative study.

In Listing 6.44, comparative study named `badCSName` is added to FOPS and afterwards it is renamed to `newCSName`. Note that `oldName` is a simple string, while `newName` is a cell array. Both options are possible to both input arguments. The corresponding action done in GUI is shown in Figure 6.15.

Listing 6.44: Rename comparative study in FOPS.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'};
chain = {'MOPSO',    'GDE3'};
nRuns = 1; metrics = []; settings = [];
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'BadCSName')
fops.renameCompStudy('BadCSName', {'newCSName'})
```

Input arguments are originally cell arrays due to possibility of renaming multiple comparative studies at once as shown in Listing 6.45. Both input arguments are cell arrays with the same number of cells.

Listing 6.45: Rename multiple comparative studies in FOPS at once.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'}; chain = {'MOPSO', 'GDE3'};
```

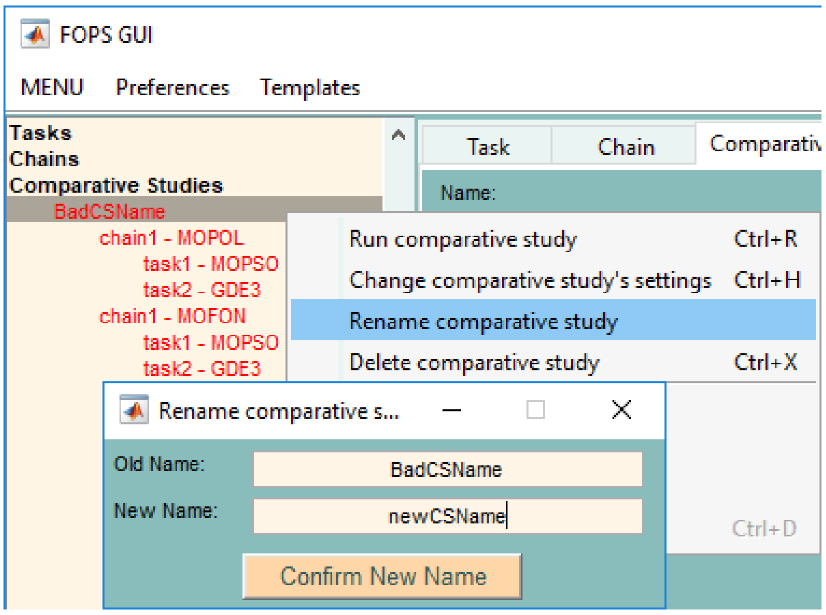


Figure 6.15: Rename comparative study in FOPS in GUI.

```
nRuns = 1; metrics = []; settings = [];
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS1')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS2')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS3')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS4')
fops.renameCompStudy({'MyCS1', 'MyCS3', 'MyCS4'}, ...
   {'newNameOfCS1', 'newNameOfCS3', 'newNameOfCS4'})
```

6.3.4 runCompStudy

`fops.runCompStudy(id)`

Inputs:

`id`	optional, cell array of strings or doubles, cell [1 x N]

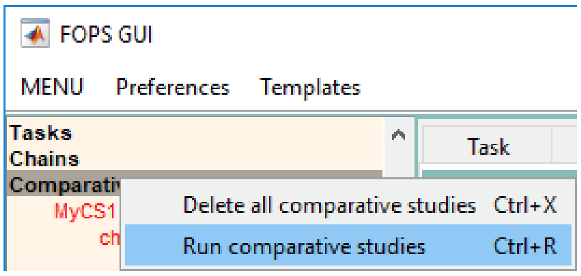
Previously created comparative studies are run by a `runCompStudy` method. The method `runCompStudy` has one optional input argument, which defines comparative studies to be run (`id`). If no input argument is given, all comparative studies are run (see Listing 6.46). Figure 6.16a shows how this can be done in FOPS GUI.

Listing 6.46: Run all comparative studies in FOPS.

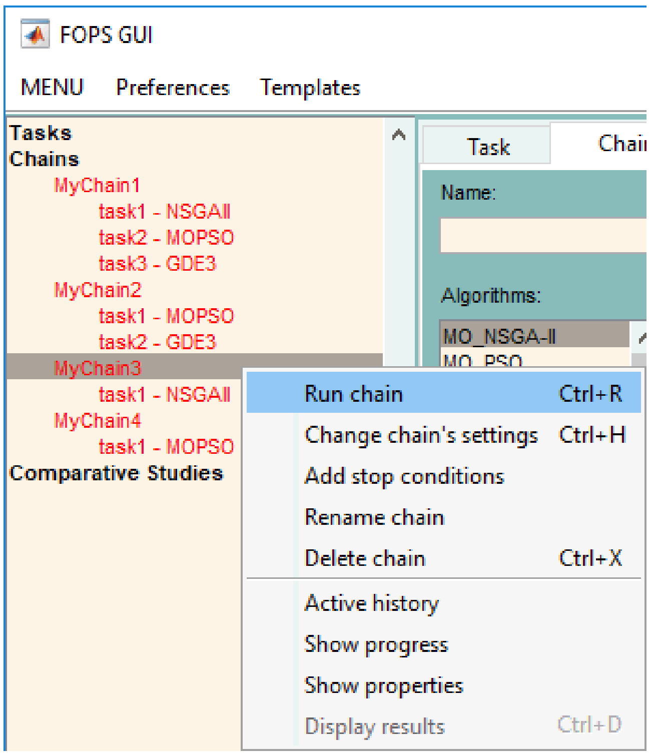
```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'}; chain = {'MOPSO', 'GDE3'};
nRuns = 1; metrics = []; settings = [];
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS1')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS2')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS3')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS4')
fops.runCompStudy()
```

Listing 6.47 shows how to run chosen comparative studies. Four comparative studies were added to FOPS. First call of `runCompStudy` method runs third comparative study identified by name (`MyCS3`). Note that `id` does not need to be encapsulated in cell array brackets. The second call of `runCompStudy` method runs the first comparative study identified by sequence number. Last call of `runCompStudy` method shows how to call multiple comparative studies at once.

Listing 6.47: Run comparative study in FOPS.


FOPS GUI
MENU
Preferences
Templates
Tasks
Chains
Comparati
MyCS1
ch
Task
Delete all comparative studies Ctrl+X
Run comparative studies Ctrl+R

(a) All comparative studies at once.



(b) One comparative study at a time.

Figure 6.16: Run comparative study in FOPS in GUI.

```
fops = FOPS(); % initialization of FOPS
problem = {'MOPOL'; 'MOFON'}; chain = {'MOPSO', 'GDE3'};
nRuns = 1; metrics = []; settings = [];
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS1')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS2')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS3')
fops.addCompStudy(problem, chain, nRuns, metrics, settings, 'MyCS4')
fops.runCompStudy('MyCS3')  % will run third CS
fops.runCompStudy({1})  % will run first CS
fops.runCompStudy({'MyCS2', 4})  % will run remaining CSs
```

Figure 6.16b shows how to run individual comparative study.

6.3.5 changeCompStudySettings

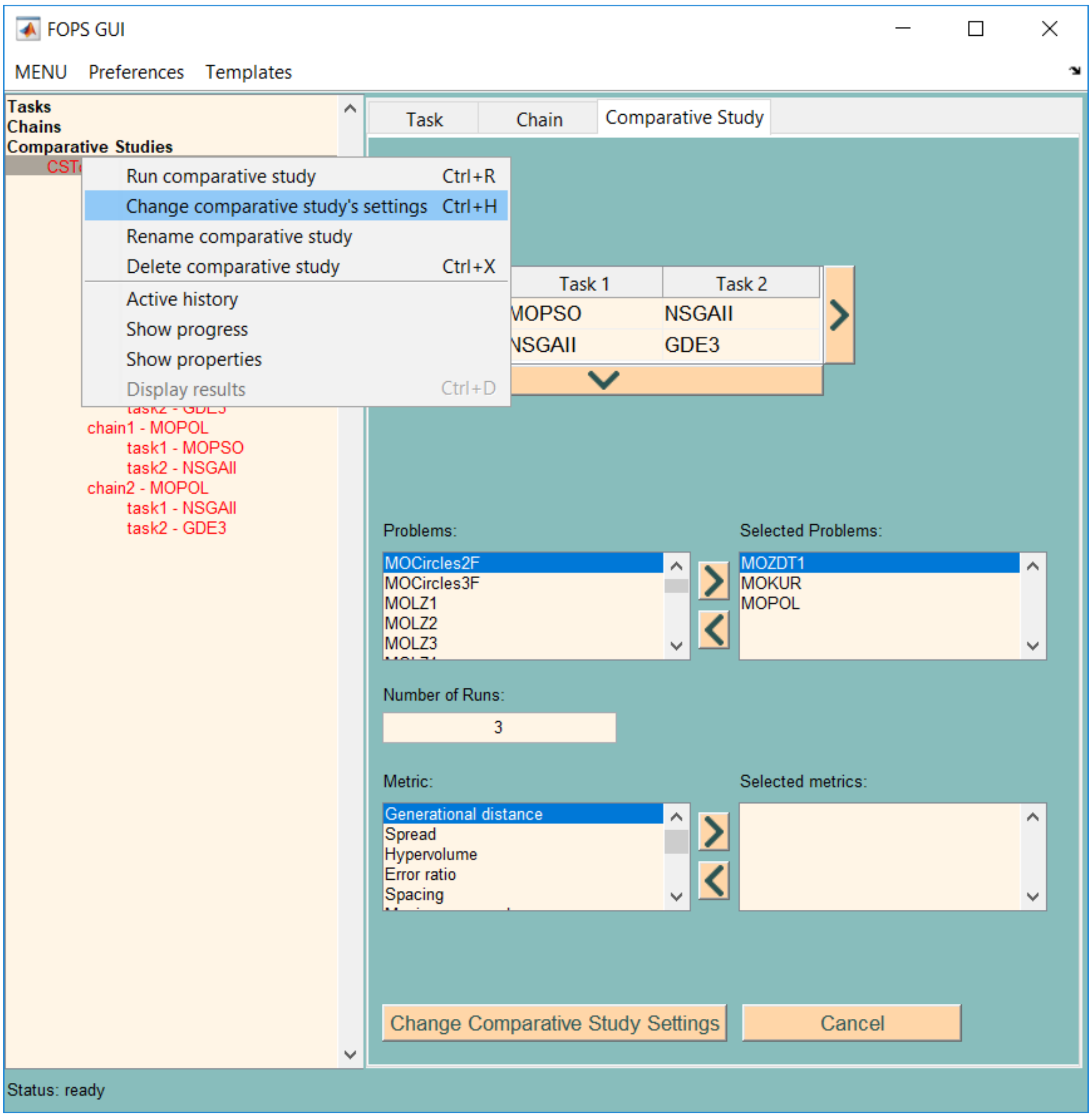
`fops.changeCompStudySettings(id, varargin)`

Inputs:

`id`	cell array of strings or doubles, cell [1 x N]
`varargin`	comparative study settings. Settings struct or property-values pairs

This method changes properties of comparative study that is already in FOPS object. Basically, it can change chains, problems, number of runs (`nRuns`) or metrics in comparative study (see Section 7.3 to learn more about comparative study settings). This method has variable number of input arguments. The first input argument (`id`) always identifies comparative study of which properties will be changed. Other input arguments define properties to be changed. There are two ways to define comparative study settings:

- Struct: struct with field name defining what property will be changed and corresponding content. Method `changeCompStudySettings` has two input arguments in this case, where



CST
task2 - GDE3
chain1 - MOPOL
task1 - MOPSO
task2 - NSGAII
chain2 - MOPOL
task1 - NSGAII
task2 - GDE3

Task 1	Task 2
MOPSO	NSGAII
NSGAII	GDE3

Problems:
MOCircles2F
MOCircles3F
MOLZ1
MOLZ2
MOLZ3

Selected Problems:
MOZDT1
MOKUR
MOPOL

Number of Runs: 3

Metric:
Generational distance
Spread
Hypervolume
Error ratio
Spacing

Selected metrics:

Status: ready

Figure 6.17: Change settings of comparative study in GUI.

the second input argument is of type struct.

- Property-value pairs: In this case, property entries are strings defining what property will be changed. Method changeCompStudySettings has odd number of input arguments (including comparative study id argument).

Listing 6.48 shows how to change problems in comparative study using struct. Initial problem in comparative study is MOZDT1. Method changeCompStudySettings replaces MOZDT1 problem with four problems, that have also additional input arguments (see Listing 6.6 in Subsection 6.1.1 or Chapter 4 for better understanding). The comparative study after the change contains 4 problems and 2 unique chains and nRuns property is 3. Therefore, the chains property in CompStudy object will be of size [3 x 8] - 8 chains will be repeated 3 times.

Comparative study identification (id) is a cell array, but it is also possible to omit the cell array brackets. It is NOT possible to change settings of multiple comparative studies at once, as the identification input argument in cell array might suggest. Cell array is preserved here only for the purposes of coherence with task and chain methods. Identification can also be made by the sequence number of comparative study.

Note that there are two pairs of cell array brackets in a problems field of struct in Listing 6.48 (the first pair in newProblems variable assignment and the second in settingStruct assignment). It is necessary to ensure that the settingStruct has only one element, because struct settingsStruct created in Listing 6.49 is of size [4 x 3] and it would cause an error.

Listing 6.48: Change problems in comparative study.

```
fops = FOPS(); % initialization of FOPS
chain = {'MOPSO', 'NSGAII'; 'NSGAII', 'GDE3'}; % two chains
fops.addCompStudy({'MOZDT1'; 'MOKUR'; 'MOPOL'}, chain, 3, [], [], 'CSToModify');
newProblems = {...
   'MOZDT2', ; ...
   'MOZDT4',  25; ...
   'MOZDT6',  []; ...
   'MODTLZ1',  struct('nObjs', 5)};
settingStruct = struct('problems', {newProblems});
fops.changeCompStudySettings({'CSToModify'}, settingStruct)
```

Listing 6.49: Invalid settingsStruct with omitted cell array brackets.

```
settingStruct = struct('problems', newProblems);
```

Listing 6.50 shows how to add problem to comparative study and change nRuns property. Initially, the comparative study contains 2 problems and 2 unique chains. Therefore, the chains property of CompStudy had size of [2 x 4]. Method changeCompStudySettings adds two problems and changes nRuns property, meaning that the chains property will now have size of [3 x 8].

Listing 6.50: Add problems and change nRuns property of comparative study.

```
fops = FOPS(); % initialization of FOPS
chain = {'MOPSO', 'NSGAII'; 'NSGAII', 'GDE3'}; % two chains
fops.addCompStudy({'MOZDT1'; 'MOKUR'}, chain, 2, [], [], 'CSToModify');
fops.changeCompStudySettings('CSToModify', ...
   struct('addProblems', {{'MOSCH'; 'MOZDT2'}}, 'nRuns', 3))
```

Listing 6.51 shows how to add new unique chains to comparative study and change metrics of comparative study. Initially, the comparative study contains 3 problems and 2 unique chains. Therefore, the chains property of CompStudy had size [2 x 6]. Method changeCompStudySettngs adds three unique chains and changes metrics (No metrics were present after addCompStudy but they would have been replaced by changeCompStudySettings method anyway). Property chains will now have size of [2 x 15].

Note that when there are changes of unique chains (chains or addChains properties) of comparative study (as in Listing 6.51), settings of every algorithm in comparative study is set as default, even if some chains remained presumably unchanged (occurs when addChains property is used), because each chain (and its tasks) in comparative study has to be initialized anew.

Listing 6.51: Add chains and change requests in comparative study.

```
fops = FOPS(); % initialization of FOPS
chain = {'MOPSO', 'NSGAII'; 'NSGAII', 'GDE3'}; % two chains
fops.addCompStudy({'MOZDT1'; 'MOKUR'; 'MOPOL'}, chain, 2, [], [], 'CSToModify');
fops.changeCompStudySettings({'CSToModify'}, ...
   'Add chains', {'GDE3', 'MOPSO'; 'NSGAII', []; 'GDE3', []}, ...
   'metrics', {'GD', 'HV', 'computationalTime'})
```

Figure 6.17 shows how to change comparative study settings in FOPS GUI. Context Menu is induced by the right mouse click on a selected comparative study. After selecting the Change comparative study's settings in Context Menu, Comparative Study tab, originally for adding comparative studies, is transformed to the change settings mode and control elements of name and algorithm settings are hidden. This suggests that if algorithms are changed, they are initialized with default settings and the default stop condition (stop when nIterations is reached).

To change settings of tasks in chains of comparative study, changeTaskSettings method has to be used (see Subsection 6.1.5, particularly Listings 6.20). Stop conditions can be added by addStopCondition method (see Subsection 6.4.1).

6.4 Other Methods

6.4.1 addStopCondition

`fops.addStopCondition(id, stopCond, isChain)`

Inputs:

`id`	cell array of strings or doubles, cell [1 x N]
`stopCond`	stop conditions, handle functions, cell [1 x M]
`isChain`	optional, `id` is ID of chain, logical [1 x 1]

This method allows the user to define an additional stop condition of task. The default stop condition for all tasks in FOPS is that a current iteration (`currentIter` property of OptTask object) has to be lower than the number of iterations (`nIters` property of OptTask object).

Stop conditions are inserted as anonymous functions or function handles with two input arguments - current iteration (`iter`) and fitness values (`f`).

Method `addStopCondition` has three input arguments. The first input argument identifies a task or chain (`id`), the second argument is a cell array containing the stop conditions (`stopCond`) and the third, optional, argument specifies, whether the `id` input arguments refers to the task or chain simulation type. If `isChain` is true, the `id` represents a path to a chain in FOPS. Otherwise, the `id` leads to a task in FOPS. Note that the third input argument is logical, but strings `task` or `chain` are also possible.

Listing 6.52 shows how to add a stop condition to a task. Variable `iter` denotes the current iteration and the `f` denotes the fitness values. Argument `iter` is a scalar integer, while the `f` is an array of doubles of size [`nAgents` x `nObjs`]. The stop condition in Listing 6.52 expresses that a task should be finished, when any second objective function value decreases under 10. Note that both input arguments (`id` and `stopCond`) should be cell arrays, but non-cell inputs are also acceptable.

Listing 6.52: Add stop condition to first task in FOPS.

```
fops = FOPS(); % initialization of FOPS
fops.addTask('MOZDT4', 'NSGA-II');
stopCond = @(iter, f) min(f(:, 2))<10;
fops.addStopCondition(1, {stopCond});
```

In Listing 6.53, the stop condition is assigned to the second task of the first chain in FOPS object. Note that it is also possible to use a string identification of simulations in `addStopCondition` method, therefore the command in Listing 6.54 is equivalent to the one in Listing 6.53. Also note that FOPS automatically concatenates default task or chain names (`task2`) with the algorithm type (e.g. `MOPSO`) or the problem name (e.g. `MOPOL`) for better orientation in `FOPS Tree` of GUI, but it is still possible to identify the task in chain or the chain in comparative study by simple names such as `task1` or `chain1`.

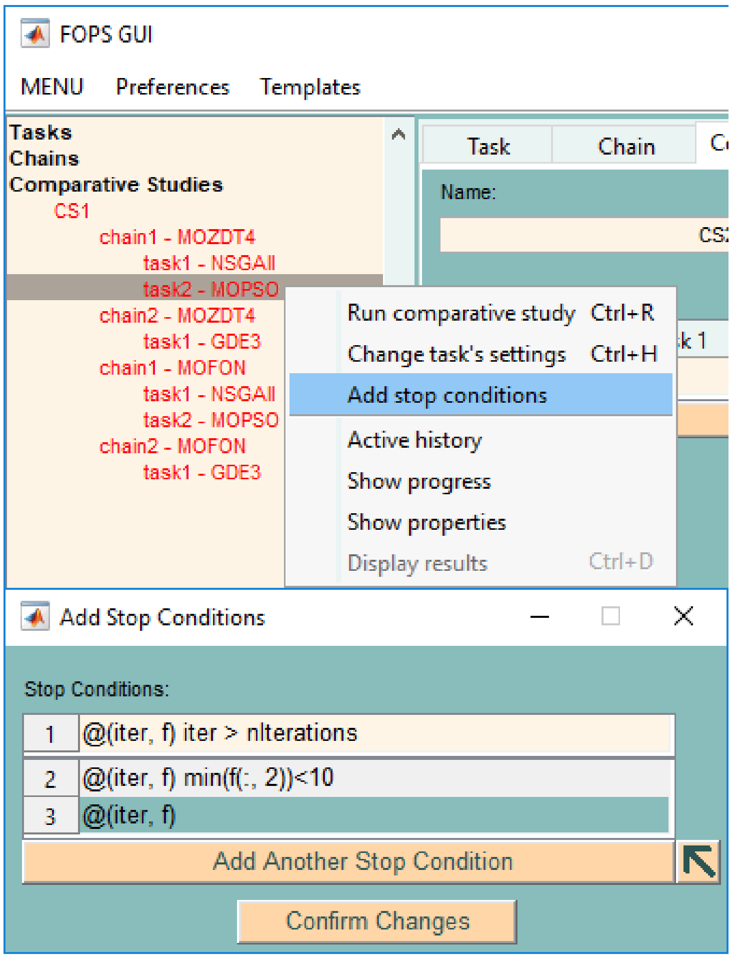
Listing 6.53: Add stop condition to second task of chain in FOPS.

```
fops = FOPS(); % initialization of FOPS
fops.addChain('MOZDT4', {'NSGA-II', 'MOPSO'});
stopCond = @(iter, f) min(f(:, 2))<10;
fops.addStopCondition({1, 2}, {stopCond});
```

Listing 6.54: Equivalent command of `addStopCondition` method.

```
fops.addStopCondition({'chain1', 'task2'}, {stopCond});
```

In Listing 6.55 the stop condition is assigned to both tasks of the first chain in FOPS object. The third input argument of `addStopCondition` method suggests, that the `1` as `id` is an identification of the chain. Omission of `isChain` input argument (or `isChain = false`) would cause an error, because no task is present in FOPS object (FOPS was initialized anew and only the chain was added).



```
1  @(iter, f) iter > nIterations
2  @(iter, f) min(f(:, 2))<10
3  @(iter, f)
```

Figure 6.18: Add stop condition to task in chain in comparative study in GUI.

Listing 6.55: Add stop condition to all tasks of chain in FOPS.

```
fops = FOPS(); % initialization of FOPS
fops.addChain('MOZDT4', {'NSGA-II', 'MOPSO'});
stopCond = @(iter, f) min(f(:, 2))<10;
isChain = true;
fops.addStopCondition(1, {stopCond}, isChain);
```

Listing 6.56 shows how to add a stop condition to a task of chain in comparative study. The second task of the first unique chain (the one consisted of `NSGA-II` and `MOPSO` algorithms) will have additional stop condition. Note that `chains` property in CompStudy object has the size of [3 x 4], because both unique chains are repeated for both problems in the comparative study. The stop condition will be assigned to the first task in the first and third column of `chains` (`chain1 - MOZDT4` and `chain1 - MOFON`).

Listing 6.56: Add stop condition to first task of chain in comparative study in FOPS.

```
fops = FOPS(); % initialization of FOPS
fops.addCompStudy({'MOZDT4'; 'MOFON'}, {'NSGA-II', 'MOPSO'; 'GDE3', []}, 3);
stopCond = @(iter, f) min(f(:, 2))<10;
fops.addStopCondition({1, 1, 2}, {stopCond});
```

Listing 6.57 shows how to assign a stop condition to all tasks of chain in comparative study. The stop condition is assigned to all tasks of all occurences of the first unique chain (`chain1 - MOZDT4` and `chain1 - MOFON`).

Listing 6.57: Add stop condition to all task of chain in comparative study in FOPS.

```
fops = FOPS(); % initialization of FOPS
fops.addCompStudy({'MOZDT4'; 'MOFON'}, {'NSGA-II', 'MOPSO'; 'GDE3', []}, 3);
```

```
stopCond = @(iter, f) min(f(:, 2))<10;
isChain = true;
fops.addStopCondition({1, 1}, {stopCond}, isChain);
```

Figure 6.18 shows how to add a stop conditions to a task in FOPS GUI. It works identically with tasks and chains.

As shown in Figures 5.1 or 6.7, stop conditions can be assigned to tasks or chains before adding it to FOPS by `Stop Conditions` control buttons. These control buttons are not present in the `Comparative Study` tab. It suggests, that the use of stop conditions in a comparative study is rather senseless, although it is possible with a little difficulty.

6.4.2 getProblem

`problemStruct = fops.getProblem(problemName)`

Inputs:

`problemName` problem name and optional input argument, cell [1 x 1] or cell [1 x 2]

Outputs:

`problemStruct` optimization problem definition, struct [1 x 1]

This method returns the problem struct of a problem already defined in FOPS as a problem function. Such method is helpful if the user wants to modify a problem already defined but without changes of the problem m-file.

The Listing 6.58 shows how to obtain definition of `MOFON` benchmark problem and make the first and third decision variable be discrete.

Listing 6.58: Define discrete variables to `MOFON` problem.

```
fops = FOPS();
MOFON = fops.getProblem('MOFON');
DV{1} = [-4, -3, -2, 2, 3, 4];
DV{3} = linspace(-3, 4, 101);
MOFON.discreteVars = DV;
fops.addTask(MOFON, 'MOPSO')
fops.runTask
```

The Listing 6.59 shows how to load the definition of `MODTLZ2` problem with `15` decision variables and `4` fitness functions.

Listing 6.59: Load customized `MODTLZ2` problem.

```
fops = FOPS();
problemSettings = struct('nVars', 15, 'nObjs', 4);
MODTLZ2 = fops.getProblem({'MODTLZ2', problemSettings});
fops.addTask(MODTLZ2, 'GDE3')
fops.runTask
```

6.4.3 displayResults

`fops.displayResults(id, type)`

Inputs:

`id`	cell array of strings or doubles, cell [1 x N]
`type`	optional, simulation type, string or double, char [1 x N] or double [1 x 1]

This method launches the `Results Figure` of GUI of the FOPS simulation identified by `id`. The variable `id` can be either the string name of the simulation or the sequence number of the simulation - double. The type of the simulation is specified by `type` input argument. It can be either string (`task`, `chain` or `CS`) or double (1 - task, 2 - chain, 3 - comparative study). If `type` input is omitted, the task type is assumed. Use `Display results` entry in `Context Menu` of selected object in `FOPS Tree` in GUI to cast `Results Figure` or type in `CTRL+D` shortcut if main GUI window is active.

The Listing 6.60 shows how to launch the `Results Figure` of the task `myTask`. The fourth and fifth lines in listing have an identical effect.

Listing 6.60: Display results of task `myTask`.

```
fops = FOPS(); % initialization of FOPS
fops.activeHistory = true;
fops.addTask({'MOPOL'}, {'NSGA-II'}, [], 'myTask')
fops.runTask()
fops.displayResults('myTask', 1)
fops.displayResults(1)
```

The Listing 6.61 shows how to launch the `Results Figure` of chain `myChain`. The fourth and fifth lines in listing have an identical effect.

Listing 6.61: Display results of chain `myChain`.

```
fops = FOPS(); % initialization of FOPS
fops.activeHistory = true;
fops.addChain({'MOPOL'}, {'NSGA-II', 'MOPSO'}, [], 'myChain')
fops.runChain()
fops.displayResults(1, 2)
fops.displayResults('myChain', 'chain')
```

The Listing 6.62 shows how to launch the `Results Figure` of the first comparative study (`CS1`) in FOPS object.

Listing 6.62: Display results of comparative study.

```
fops = FOPS(); % initialization of FOPS
fops.activeHistory = true;
problems = {'MOPOL'; 'MOFON'};
algorithms = {'NSGA-II', 'GDE3'; 'MOPSO', 'NSGA-II'};
fops.addCompStudy(problems, algorithms, 3, {'GD', 'Time'})
fops.runCompStudy()
fops.displayResults(1, 3)
```

The Listing 6.63 shows how to launch the `Results Figure` of the task that is a part of the comparative study. Note that the Listing 6.63 is a continuation of Listing 6.62. The first command shows results of the first task of the second chain (the one with `MOPOL` problem) in the first comparative study. The second command shows results of the second task of the fourth chain (the one with `MOFON` problem) in the first comparative study. Property `nRuns` of the comparative study was set to 3. The selection between results of different runs of a simulation is performed by `Choose Run of Simulation` popup menu in `Results Figure` (see in Chapter 9).

Listing 6.63: Display results of diffrent tasks in comparative study.

```
...
fops.displayResults({1, 2, 1}, 1) % chain2 - MOPOL, task1 - MOPSO
fops.displayResults({1, 4, 2}, 1) % chain2 - MOFON, task2 - NSGA-II
```

6.4.4 loadProblems

`fops.loadProblems(folders)`

Inputs:

`folders`	cell array of strings, cell [1 x N]

This method allows the user to add folders with problem functions. The method searches through all the m-files in `folders` location and each function which begins with string `function problem =` (whitespace insensitive) is marked as a problem function and is further accessible by FOPS methods by the problem function name (case insensitive).

Use `Load Problems` entry of GUI's `MENU` to load problem function in GUI of FOPS (see Figure 5.2).

6.4.5 open

`fops.open(filename)`

Inputs:

`filename`	name of `*.fops` file with FOPS instance, may include path, char [1 x N]

This method allows user to open previously saved FOPS instance. This method can be performed only with blank FOPS instance. Use `Open FOPS` entry of GUI's `MENU` to open FOPS instance in GUI of FOPS (see Figure 5.2).

The FOPS instance is saved by `save` method (see Subsection 6.4.9) or in `MENU` of FOPS GUI. Save method can be also called when a run of simulation is terminated by `Stop Simulation` push button in the `Status Bar` of GUI. In such case, an instance with unfinished simulation is stored in `*.fops` file and if such FOPS instance is opened using `open` method, it is possible to resume it by `resume` method or by clicking the `Resume` push button in the `Status Bar` of FOPS GUI.

6.4.6 quit

`fops.quit()` This method calls the destructor of FOPS object. Use `Quit FOPS` entry of GUI's `MENU` to quit FOPS instance in GUI of FOPS (see Figure 5.2).

6.4.7 reset

`fops.reset()`

This method serves to reset the FOPS object. It initializes the FOPS object anew. Use `Reset FOPS` entry of GUI's `MENU` to reset FOPS instance in GUI of FOPS (see Figure 5.2).

6.4.8 resume

`fops.resume()`

This method serves to resume the FOPS simulations. In GUI it is accessible via `Resume` push button in `Status Bar`. `Resume` push button appears only if an unfinished simulation is present in the FOPS instance. The FOPS simulation can be aborted by `Stop Simulation` push button in `Status Bar` or by pressing `CTRL+C` (Note that such action terminates all MATLAB activities).

If the FOPS instance with aborted simulation is saved by the `save` method, resume of an unfinished simulation is possible after opening the FOPS instance by the `open` method.

6.4.9 save

`fops.save(filename)`

Inputs:

`filename`	name of file where the FOPS instance will be stored, may include path, char [1 x N]

This method allows user to store the elaborated FOPS instance. Such instance is stored in `*.fops` file. Method `save` is accessible in bash mode or in `MENU` of FOPS GUI. Use `Save FOPS` entry of GUI's `MENU` to save FOPS instance in GUI of FOPS (see Figure 5.2).

Save method can be also called when a running simulation is terminated by the `Stop Simulation` push button in `Status Bar` of GUI and user selects the save FOPS instance option. Such FOPS instance can be opened using the `open` method and the simulation can be resumed by the `resume` method or by clicking on `Resume` push button in `Status Bar` of FOPS GUI.

6.4.10 startGUI

`fops.startGUI(color)`

Inputs:

`color`	index of color template, double [1 x 1]

This method opens GUI of existing FOPS object.

Chapter 7

Simulation Settings

7.1 Task Settings

Task settings defines the behavior of an optimization algorithm during the optimization process. This section contains list of all the task properties - abbreviation, whole name, default values in { } brackets, and restrictions of values. More info about the task properties can be found in Chapter 14 or adjacent references.

Default algorithm properties can be viewed and/or redefined in `setDefaultValues` method of FOPS class.

7.1.1 General Task Settings

`nAgents`

`Double:` {100}
Number of agents. Positive integer.

SOGA:	`nAgents` >= `TS` (see Subsection 7.1.2).
NSGA-II:	`nAgents` >= 2 (also uses tournament selection).
SODE:	`nAgents` > 3 (trial vector is combination of three agents).
GDE3:	`nAgents` > 3 (trial vector is combination of three agents).
VNDGDE3:	`nAgents` > 3 (trial vector is combination of three agents).
SONEW:	`nAgents` = 1 and it cannot be changed (Newton method is local method).

`nIters`

`Double:` {100}
Number of iterations. Positive integer.

7.1.2 SOGA Settings

`PC`

`Double:` {0.9}
Probability of crossover. `PC` $\in \langle 0, 1\rangle$.

`PM`

`Double:` {0.7}
Probability of mutation. `PM` $\in \langle 0, 1\rangle$.

`TS`

`Double:` {2}
Tournament size. Positive integer, `TS` <= `nAgents`.

`BP`

`Double:` {20}
Binary precision. Positive integer.

BP can be stated as a scalar value (all decision variables will have equal BP) or as a vector of integers, where the number of elements must correspond to the number of decision variables of problem.

BP is automatically set according to number of possible states of decision variable when discrete decision variable is used (see Chapter 11). User is informed by warning message about the fact.

nCP

Double: {1}
Number of crossover positions. Positive integer. nCP $< \sum$ BP.

CP

Double: {0}
List of allowed crossover positions. Non-negative integers. Zero value denotes that all crossover positions are allowed. CP $< \sum$ BP.

nMP

Double: {1}
Number of mutation positions. Positive integer. nMP $<= \sum$ BP.

MP

Double: {0}
List of allowed mutation positions. Non-negative integers. Zero value denotes that all mutation positions are allowed. MP $<= \sum$ BP.

More info in Subsection 14.1.1.

7.1.3 SOPSO Settings

W

Double: {[0.6, 0.4]}
Inertia wieght. Non-negative values.

W usually linearly changes its value throughout the optimization run. First value defines inertia weight in first iteration of algorithms, while the second value is an inertia weight in the last iteration. A scalar input is also possible.

C1

Double: {1.5}
Cognitive learning factor. Altough it is not limited, C1 < 2 is recommended.

C2

Double: {1.5}
Social learning factor. Altough it is not limited, C1 < 2 is recommended.

BT

String: {'Reflecting'} | 'Absorbing' | 'Invisible'
Boundary type.

More info in Subsection 14.1.2.

7.1.4 SODE Settings

F

Double: {0.2}
Scaling factor. Although it is not limited, F $\in \langle 0,\ 3\rangle$ is recommended.

PC

Double: {0.2}
Probability of crossover. PC $\in \langle 0, 1\rangle$.

More info in Subsection 14.1.3.

7.1.5 SOSOMA Settings

`PR`

: `Double:` {`0.1`}
  Probability of perturbation. `PR` $\in \langle 0, 1 \rangle$.

`ST`

: `Double:` {`4`}
  Number of steps. Non-negative integer.

`PL`

: `Double:` {`1.15`}
  Multiplicator of distance between steps.

`BT`

: `String:` {`'Reflecting'`} | `'Absorbing'` | `'Invisible'`
  Boundary type.

`MT`

: `String:` {`'AllToLeader'`} | `'AllToAll'`
  Migration type.

More info in Subsection 14.1.4.

7.1.6 SONEME Settings

`Alpha`

: `Double:` {`1.2`}
  Reflection coefficient. Non-negative value.

`Beta`

: `Double:` {`0.5`}
  Contraction coefficient. Non-negative value.

`Gama`

: `Double:` {`0.5`}
  Expansion coefficient. Non-negative value.

`BT`

: `String:` {`'Reflecting'`} | `'Absorbing'` | `'Invisible'`
  Boundary type.

More info in Subsection 14.1.5.

7.1.7 SONEW Settings

As was mentioned in general settings (Subsection 7.1.1), number of agents `nAgents` is set to 1 and cannot be changed.

`Diff`

: `Double:` {$\sqrt[3]{\text{eps}}$}
  Difference used for numerical gradient and hessian calculation. The variable eps is a floating-point relative accuracy. Non-negative value.

More info in Subsection 14.1.6.

7.1.8 SOCMAES Settings

`Mu`

: `Double:` {`0.5`}
  Fraction of agents used for update. `Mu` $\in \langle 0, 1 \rangle$.

More info in Subsection 14.1.7.

7.1.9 NSGA-II Settings

`PC`

`Double:` {0.9}
Probability of crossover. `PC` $\in \langle 0, 1 \rangle$.

`PM`

`Double:` {0.7}
Probability of mutation. `PM` $\in \langle 0, 1 \rangle$.

`TS`

`Double:` {2}
Tournament size. Positive integer, `TS` <= `nAgents`.

`BP`

`Double:` {20}
Binary precision. Positive integer

`BP` can be stated as a scalar value (all decision variables will have equal `BP`) or as a vector of integers, where the number of elements must correspond to the number of decision variables of problem.

`BP` is automatically set according to number of possible states of decision variable when discrete decision variable is used (see Chapter11). User is informed by warning message about the fact.

`nCP`

`Double:` {1}
Number of crossover positions. Positive integer. `nCP` $< \sum$ `BP`.

`CP`

`Double:` {0}
List of allowed crossover positions. Non-negative integers. Zero value denotes that all crossover positions are allowed. `CP` $< \sum$ `BP`.

`nMP`

`Double:` {1}
Number of mutation positions. Positive integer. `nMP` <= $\sum$ `BP`.

`MP`

`Double:` {0}
List of allowed mutation positions. Non-negative integers. Zero value denotes that all mutation positions are allowed. `MP` <= $\sum$ `BP`.

More info in Subsection 14.2.1.

7.1.10 MOPSO Settings

`W`

`Double:` {[0.8, 0.5]}
Inertia wieght. Non-negative values.

`W` usually linearly changes its value throughout the optimization run. First value defines inertia weight in first iteration of algorithms, while the second value is an inertia weight in the last iteration. A scalar input is also possible.

`C1`

`Double:` {1.5}
Cognitive learning factor. Altough it is not limited, `C1` < 2 is recommended.

`C2`

`Double:` {1.5}
Social learning factor. Altough it is not limited, `C1` < 2 is recommended.

`BT`

`String:` {`'Reflecting'`} | `'Absorbing'` | `'Invisible'`
Boundary type.

`RGB`

`Double:` {`1`}
Random global best. `RGB` $\in \langle 0, 1 \rangle$.

More info in Subsection 14.2.2.

7.1.11 GDE3 Settings

`F`

`Double:` {`0.2`}
Scaling factor. Although it is not limited, `F` $\in \langle 0,\ 3 \rangle$ is recommended.

`PC`

`Double:` {`0.2`}
Probability of crossover. `PC` $\in \langle 0, 1 \rangle$.

More info in Subsection 14.2.3.

7.1.12 VND-PSO Settings

`W`

`Double:` {`[0.9, 0.4]`}
Inertia wieght. Non-negative values.

`W` usually linearly changes its value throughout the optimization run. First value defines inertia weight in first iteration of algorithms, while the second value is an inertia weight in the last iteration. A scalar input is also possible.

`C1`

`Double:` {`1.49`}
Cognitive learning factor. Altough it is not limited, `C1` < 2 is recommended.

`C2`

`Double:` {`1.49`}
Social learning factor. Altough it is not limited, `C1` < 2 is recommended.

`BT`

`String:` {`'Reflecting'`} | `'Absorbing'` | `'Invisible'`
Boundary type.

`PTF`

`Double:`{`[0.07, 0.13, 0.8]`}
Probabilities to follow. `PTF`$_i \in \langle 0, 1 \rangle$ for $i = 1, 2, 3$. Sum of all three values must be lower than or equal to 1.

`DCT`

`String:` {`'Best DC'`} | `'Random DC'` | `'No DC'`
Dimension check type.

More info in Subsection 14.3.1.

7.1.13 VND-MOPSO Settings

`W`

`Double:` {`[0.9, 0.4]`}
Inertia wieght. Non-negative values.

`W` usually linearly changes its value throughout the optimization run. First value defines inertia weight in first iteration of algorithms, while the second value is an inertia weight in the last iteration. A scalar input is also possible.

`C1`

`Double:` {`1.49`}
Cognitive learning factor. Altough it is not limited, `C1` < 2 is recommended.

`C2`

`Double:` {`1.49`}
Social learning factor. Altough it is not limited, `C1` < 2 is recommended.

`BT`

`String:` {`'Reflecting'`} | `'Absorbing'` | `'Invisible'`
Boundary type.

`PTF`

`Double:`{`[0.02, 0.02, 0.96]`}
Probabilities to follow. `PTF`$_i \in \langle 0, 1\rangle$ for $i = 1, 2, 3$. Sum of all three values must be lower than or equal to 1.

`DCT`

`String:` {`'Best DC'`} | `'Random DC'` | `'No DC'`
Dimension check type.

More info in Subsection 14.3.2.

7.1.14 VND-GDE3 Settings

`F`

`Double:` {`0.2`}
Scaling factor. Although it is not limited, `F` $\in \langle 0,\ 3\rangle$ is recommended.

`PC`

`Double:` {`0.2`}
Probability of crossover. `PC` $\in \langle 0, 1\rangle$.

More info in Subsection 14.1.3.

`PTF`

`Double:`{`[0.33, 0.33, 0.34]`}
Probabilities to follow. `PTF`$_i \in \langle 0, 1\rangle$ for $i = 1, 2, 3$. Sum of all three values must be lower than or equal to 1.

`DCT`

`String:` {`'Best DC'`} | `'Random DC'` | `'No DC'`
Dimension check type.

7.2 Chain Settings

CHAIN object contains multiple OPTTASK objects which optimizes the same problem. The properties that can be changed in chain are `problem` and `algorithms`.

`Algorithms`

`Cell array of strings`
`Algorithms` is a cell array with optimization algorithm names as shown in Subsection 6.2.5.

`Problem`

`Cell array of strings`
`Problem` is a cell array, where the first cell contains problem name or a problem struct as shown in Subsection 6.2.5. Cell array can be of size [1 x 1] or [1 x 2]. Second column is reserved

for input arguments of the problem function (see Chapter 4 or Listing 6.6). Note that the change of the problem of a chain that is in comparative study is not allowed. Changes of problems in a comparative study has to be made by the `changeCompStudySettings` method.

7.3 Comparative Study Settings

User can change problems, unique chains, requests and number of runs of comparative study.

`Problems`

`Cell array of strings`
`Problems` replaces all current problems in a comparative study with the problems specified as a `Problems` input. Each unique chain in the comparative study will be repeated according to the number of problems (see Listing 6.48 or Figure 6.12).

`Problems` is a cell array, where the first column of cell represents problem names or a problem structs as shown in Subsection 6.3.5. Cell array can be of size [N x 1] or [N x 2]. Second column can contain additional input arguments of problem function (see Listing 6.48 or Chapter 4 to understand problem function input arguments).

`Add problems`

`Cell array of strings`
`Add problems` adds new problems alongside the current problems in a comparative study. Each unique chain in the comparative study will be repeated according to number of problems (see Listing 6.50).

`Add problems` is a cell array, where the first column of cell represents problem names or a problem structs as shown in Subsection 6.3.5. Cell array can be of size [N x 1] or [N x 2]. Second column can contain additional input arguments of problem function (see Listing 6.48 or Chapter 4 to understand problem function input arguments).

`Chains`

`Cell array of strings`
`Chains` changes unique chains of a comparative study. The problems in the comparative study remains the same.

`Chains` is a cell array with the names of optimization algorithms. Each line in cell defines one chain with multiple tasks (see Listing 6.51).

`Add chains`

`Cell array of strings`
`Add chains` adds unique chains to a comparative study. The problems in the comparative study remains the same. Properties of individual algorithms in all chains is set with defaults values, even if some chains in comparative study are supposed to be unchanged.

`Add chains` is a cell array with the names of optimization algorithms. Each line in cell defines one chain with multiple tasks (see Listing 6.51).

`Metrics`

`Cell array of strings`
`Metrics` replaces all current metrics in a comparative study with new ones. To learn more about metrics see Chapter 8.

`Metrics` is a cell array with names of metrics introduced in Section **??**. An example can be seen in Listing 6.51.

`Add metrics`

`Cell array of strings`
`Add metrics` adds new metrics to a comparative study. To learn more about metrics see Chapter 8.

`Add metrics` is a cell array with names of metrics introduced in Section **??**.

`nRuns`

`Double`

`nRuns` sets how many times all the individual simulations will be performed throughout the whole comparative study. `nRuns` is a positive integer scalar value.

Chapter 8

Metrics

Metrics in FOPS provides sort of post-processing in comparative studies. When a comparative study in FOPS is initialized, user can assign predefine metrics to be calculated. Each time some chain in the comparative study is run, metric values present in `metricTypes` property of CompStudy are calculated.

Metrics are used to qualify the results of an optimization run. There are several metrics implemented in FOPS. Single-objective optimization has only one solution as a result, therefore it is relatively easy to qualify it. Contrarily, multi-objective optimization results in a set of solutions and it is difficult to say whether the found set is good or bad. This is the reason why most of the metrics are related to the multi-objective optimization.

Metric objects has only one property:

`type` Identifies a type of metric, MetricType [1 x 1]

Calculated metric values are stored in `results` property of CompStudy object. Property `results` contains fields named after the metric types of the size identical to the size od `chains` property of CompStudy.

8.1 Generational Distance (GD)

This metric is related to multi-objective optimization. The generational distance finds average Euclidean distance of member of set Q from P^* according to:

$$GD = \frac{\left(\sum_{i=1}^{|Q|} d_i\right)}{|Q|}, \tag{8.1}$$

where the distance d_i is the Euclidean distance between solution $i \in Q$ and the nearest member of P^*.

$$d_i = min_{k=1}^{|P^*|} \sqrt{\sum_{m=1}^{M} \left(f_m^{(i)} - f_m^{*(k)}\right)^2}, \tag{8.2}$$

where $f_m^{*(k)}$ is the m-th objective function value of the k-th member of P^* and $f_m^{(i)}$ is the m-th objective function value of the i-th member of Q.

It is obvious that the lower the value of `GD` is, the better the found set is. This metric requires knowledge of true Pareto-front (Section 4.13 shows how to assign true Pareto-front of problem). More information about the metric can be found in [2].

8.2 Spread (Δ, D)

This metric is related to multi-objective optimization. Spread metric measures quality of distribution of non-dominated solutions, but it also takes in account the distance from true Pareto-front extremes. Therefore, it requires knowledge of true Pareto-front (Section 4.13 shows how to assign true Pareto-front of problem). The metric is described by:

$$\Delta = \frac{d^e + \sum_{i=1}^{|Q|} \left|d_i - \bar{d}\right|}{d^e + |Q| \cdot \bar{d}}, \tag{8.3}$$

where d_i is the average of two Euclidean distances from two neigboring solutions from non-dominated set of i-th solution and $\bar{d}$ is a mean value of d_i values. The parameter d^e is the sum of Euklidean distances between adjacent extrems of sets Q and P^*.

Δ can be zero only if extrems of found non-dominated set are identical to the extrems of true Pareto-front and simultanously all distances between neighbouring solutions are equal to their mean value (solutions are uniformly distributed). More information about the metric can be found in [2].

8.3 Hypervolume (HV)

This metric is related to multi-objective optimization. It calculates the volume in the objective space covered by members of Q set. `HV` is a sum of volumes of hypercubes v_i. Each hypercube v_i is constructed with a reference point W and the solution $i \in Q$ as a diagonal corners of the hypercube.

Reference point W has significant influence on the `HV` scale. Reference point for the hypervolume calculation can be defined in a problem struct (see Section 4.15). If a reference point is too far from found Pareto-front, the value of hypervolume will be large and changes of the Pareto-front will have small impact on the `HV` value. Contrarily, it can happen that the whole non-dominated set will be above reference point and the `HV` will be zero. If no reference point is stated, maximal fitness values in each objective from set Q are taken as a hypervolume reference point.

It is obvious that the scale of `HV` metric depends on the problem (scale of fitness values and reference point). The bigger the value of `HV` is, the better the non-dominated set was found. For two-objective problems, `HV` is calculated simply by summing up the surfaces of rectangles (depicted in [2]). For many-objective problems (three or more objectives), the calculation of `HV` becomes complicated. The WFG technique is used in such cases [3], but it can be a time consuming operation for large Q sets.

8.4 Error Ratio (ER)

This metric is related to multi-objective optimization. It simply counts the number of solutions from found set (will be called Q in this section) which are not members of the true Pareto set (will be called P^* throughout this section). This metric requires knowledge of true Pareto-front (Section 4.13 shows how to assign true Pareto-front of problem).

$$ER = \frac{\sum_{i=1}^{|Q|} e_i}{|Q|}, \tag{8.4}$$

where $e_i = 1$ if solution $i \notin P^*$ and $e_i = 0$ otherwise. The value of `ER` should be as low as possible. The `ER` $= 0$ means, that all members of set Q lies on true Pareto-front.

The drawback of this metric is, that set P^* has finite number of members and member of set Q can be marked as error, although it is true optimal solution, but lies between the two solutions from P^*. To avoid such difficulty, a tolerance was introduced to `ER` metric in FOPS. The tolerance was set to 1e$-$3, therefore if a member of Q is within tolerated distance from P^* it is still considered true optimal.

Another drawback is that if no member of Q is in the Pareto-optimal set, user has no knowledge about the closeness of the set Q from set P^*. More information about the metric can be found in [2].

8.5 Spacing (S)

This metric is related to multi-objective optimization. It measures relative distance between consecutive solutions in the obtained non-dominated set according to:

$$S = \sqrt{\frac{1}{|Q|} \sum_{i=1}^{|Q|} \left(d_i - \bar{d}\right)^2}, \tag{8.5}$$

where d_i is the distance of i-th solution from the closest solutions of Q set as follows:

$$d_i = min \sum_{m=1}^{M} \left| f_m^i - f_m^k \right|, \tag{8.6}$$

where $k \in Q \wedge k \neq i$. Variable $\bar{d}$ from Equation 8.5 is a mean value of distances calculated by Equation 8.6:

$$\bar{d} = \sum_{i=1}^{|Q|} \frac{d_i}{|Q|}. \tag{8.7}$$

The metric described by Equations 8.5, 8.6 and 8.7 measures the standard deviations of different d_i values. If an algorithm finds solutions that are near uniformly spaced, the corresponding `S` value will be small, therefore the smaller the spacing value is, the better the distribution of non-dominated set was. The advantage of this metric is that it does not requires true Pareto-front of problem. More information about the metric can be found in [2].

8.6 Maximum Spread (MD)

This metric is related to multi-objective optimization. It measures the length of diagonal of a hyperbox formed by extremes of non-dominated set according to:

$$MD = \sqrt{\sum_{m=1}^{M} \left(\min_{i=1}^{|Q|} f_m^i - \max_{i=1}^{|Q|} f_m^i\right)^2}. \tag{8.8}$$

Basically, it is Euklidean distance of extreme solutions of non-dominated set. Disadvantage of such metric is that for each problem it has different scale.

If true Pareto-front is stated in problem (see Section 4.13 how to define true Pareto-front of problem), `MD` can be normalized with extrems of true Pareto-front. Metric will be then calculated as follows:

$$MD = \sqrt{\frac{1}{M} \sum_{m=1}^{M} \left(\frac{\min_{i=1}^{|Q|} f_m^i - \max_{i=1}^{|Q|} f_m^i}{F_m^{\max} - F_m^{\min}}\right)^2}, \tag{8.9}$$

where $F_m^{\max}$ and $F_m^{\min}$ are the maximum and minimum value, respectively, of m-th objective value of extrems of true Pareto-front $P*$. In this case, `MD` $= 1$ if widely spread solutions are obtained. More information about the metric can be found in [2].

8.7 Fitness Error (FER)

This metric is related to single-objective optimization. `FER` metric expresses the absolute difference between true optimal fitness value (see Section 4.13 how to set optimal fitness value to problem) and fitness value found by optimization process.

$$FER = \left| f^P - f^Q \right|, \tag{8.10}$$

where f^P is the fitness value of true optimum and f^Q is the fitness value of found solution.

8.8 Position Error (DER)

This metric is related to single-objective optimization. `DER` denotes decision space error and it is an Euclidean distance between optimal position of problem (see Section 4.12 how to set optimal position to problem) and position of found solution.

$$DER = \sqrt{\sum_{i=1}^{N} \left(x_i^P - x_i^Q\right)^2}, \tag{8.11}$$

where N is the number of decision variables, x_i^P is the i-th decision variable of true optimal solution and x_i^Q is the i-th decision variable of found solution.

8.9 Computational Time (T)

Computational time is taken from CHAIN object after it is run. It is obtained by `tic toc` function of MATLAB.

8.10 Number of Surrogate Solutions (nS)

This metric is related to surrogate optimization technique (see Chapter 10). It denotes how many times was used the surrogate solution during a run of simulation. If the surrogate optimization is disabled, the `NaN` value will be present in `result` property of `nS` METRIC object.

8.11 Number of solutions (N)

This metric is related to multi-objective optimization. It shows how many solutions were present in the non-dominated set.

8.12 Number of Decision Variables Deviation (NVD)

This metric is related to variable number of dimensions optimization. It shows the average difference between the found dimensionality of a solution and the true optimal dimensionality of a solution.

Chapter 9

Results in GUI

This chapter presents a tool for a simple vizualization of optimization results. The `Results Figure` of FOPS GUI can be used to easily access results and vizualize data from simulations in FOPS. Results of optimization are located in `results` property of OptTask, Chain or CompStudy objects.

There are two modes of `Results Figure` - the mode where results of tasks or chains are visualized and the mode where results of comparative study are visualized.

9.1 Results of Task or Chain

When a task or chain is run, its name in the `FOPS Tree` turns green and `Display results` entry is allowed in the `Context Menu` (see Figure 9.1).

Figure 9.2 shows the layout of `Results Figure` of FOPS GUI. In the upper part of the figure are the control elements that controls what should be plotted. The lower part of figure consists of two tables - left table shows position of agents and right table shows corresponding fitness values.

The `Results Figure` contains following control elements:

`Choose What To Plot` allows user to choose whether he wants to visualize results of a simulation (`Results` option), history of a simulation run (`History` option) or convergence plot of some metric (`Generational distance`, `Spread`, `Hypervolume` or other options). Note that to visualize the history or metric convergence plots, the history of a simulation must be present in the corresponding simulation object (enabled `activeHistory` before a simulation run).

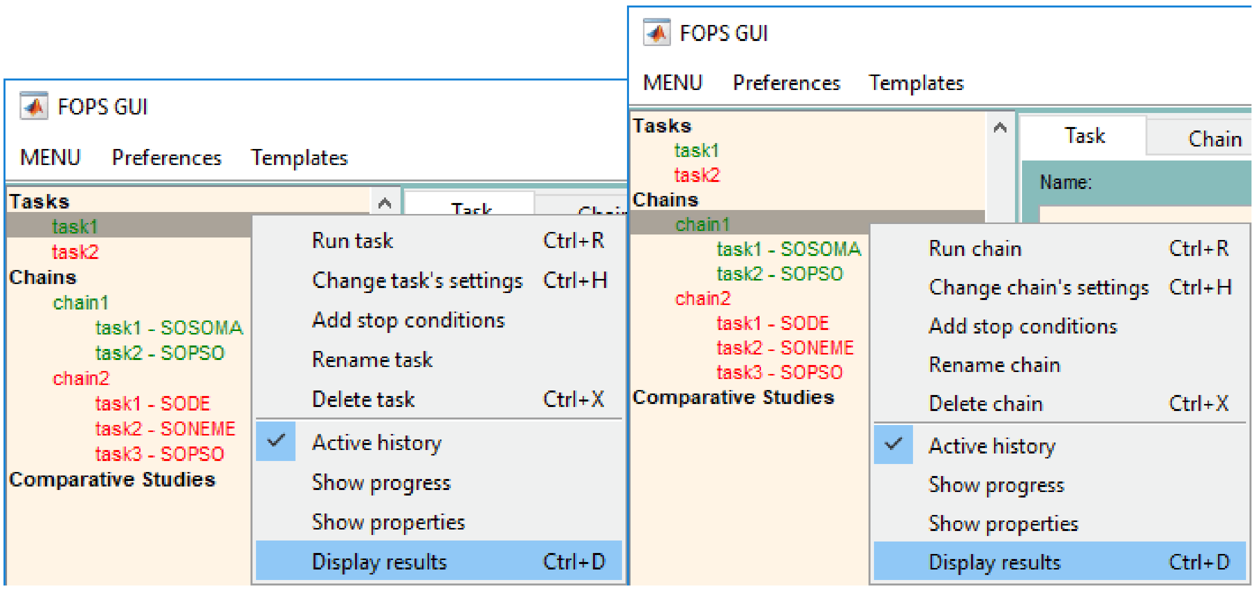
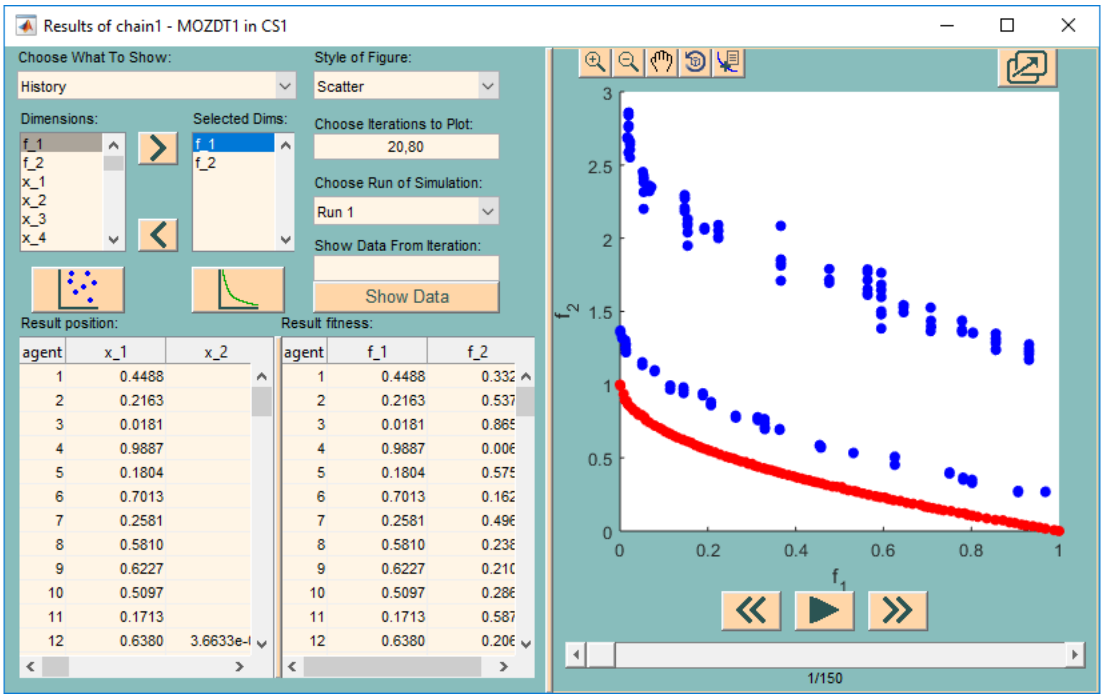


Figure 9.1: Cast `Results Figure` in GUI.



Result position:

agent	x_1	x_2
1	0.4488	
2	0.2163	
3	0.0181	
4	0.9887	
5	0.1804	
6	0.7013	
7	0.2581	
8	0.5810	
9	0.6227	
10	0.5097	
11	0.1713	
12	0.6380	3.6633e-

Result fitness:

agent	f_1	f_2
1	0.4488	0.332
2	0.2163	0.537
3	0.0181	0.865
4	0.9887	0.006
5	0.1804	0.575
6	0.7013	0.162
7	0.2581	0.496
8	0.5810	0.238
9	0.6227	0.210
10	0.5097	0.286
11	0.1713	0.587
12	0.6380	0.206

Figure 9.2: `Results Figure` of FOPS GUI.

`Dimensions` list box contains all dimensions of a problem, where the letter `f_`i denotes i-th fitness function and the letter `x_`i denotes i-th decision variable. User can select any item from the `Dimensions` list box and add it to the `Selected Dimensions` list box by the > push button. Dimensions can also be removed by the < push button. `Selected Dimensions` list box can contain no more than 3 items. It is possible to combine position and fitness values in a single plot.

`Style of Figure` popup menu allows user to choose among different types of plot types in MATLAB. The default one is the `Scatter`, which prints the solutions as points in axes. Another one is `Plot`, which connects the solutions with lines and `MST` denoting the Minimum Spanning Tree. If three dimensions are present in the `Selected Dimensions` list box, `Contour`, `Contour3`, `Surf` (interpolated), and `Mesh` entries are also available. These are advanced volume plot methods, which uses triangulation of solution to convert it into volume form (see later in this chapter).

`Choose Iterations To Plot` edit box allows user to define iterations from which the data should be plotted. The default value is `1:nIters`. The value `nIters` cannot be exceeded (if chain's results are being visualized `nIters` is a sum of `nIters` properties of all tasks in chain). This edit box is visible only if `History` item is selected in `Choose What To Plot` popup menu.

`Choose Run of Simulation` popup menu allows user to choose between results of task or chain from different run. This popup menu is visible only if current task or chain is part of comparative study with `nRuns` higher than 1.

`Plot` push button , plots optimization data into axes present in the right side of the Figure 9.2.

`Plot True Optimum` push button allows user to plot true optima of a problem if the true Optimum is known (see Sections 4.13 and 4.12 to learn how to define it). Otherwise, push button is invisible.

`Show Data From Iteration` edit box and `Show Data` push button allows user to change content of the two tables in the bottom of the `Results Figure` (see Figure 9.2). Both controls are visible only if the history of a simulation exists. Input has to be scalar integer and maximal value is `nIters`.

Results Position, Results Fitness tables show the results of the simulation under inspection. The ratio between the Results Position table area and the Fitness Position table area can be changed by the thin push button between them.

Previous, Play/Pause, Next push buttons «, ▶, ❚❚, » and slider below them allow to browse solutions from the individual iterations of a simulation run if history is present. Otherwise, push buttons and slider are disabled.

ZoomIn, ZoomOut, Pan, Rotate3D, DataCursor push buttons 🔍, 🔍, ✋, ↻, ⌖ has identical functions as corresponding tools in the standard MATLAB figure.

NewFigure push button ⧉ allows user to export current content of the axes to a new MATLAB figure.

Note that the ratio between controls area and axes area can be changed by the thin push button between the two areas.

Listing 9.1 show how to vizualize results of the single-objective task with SORAS problem. After the Results Figure is shown, the Selected Dimensions list box contains only f_1 and item Results in the Choose What To Plot popup menu is selected. Tables in the lower part of Results Figure contain the best solution from each iteration of a simulation run. The found optimal solution is the first entry (last iteration) in tables (note that the optimal solution in the results property of the task is the last entry).

Tables contain entries from all iterations due to the possibility to plot convergence curve of optimization run for a single-objective problems. Figure 9.3 shows a convergence curve of the task from Listing 9.1. The figure is created when single-objective task is being vizualized and the Selected Dimensions list box contains only one entry (f_1 in this case). Note that SORAS problem was set to have 5 decision variables, otherwise the fitness values would drop near 0 in only a few iterations.

Listing 9.1: Creation of task with SORAS problem.

```
fops = FOPS();
fops.activeHistory = true;
settings.nAgents = 200;
settings.nIters = 100;
fops.addTask({'SORAS', 5}, 'SODE', settings)
fops.runTask
fops.displayResults(1, 1)
```

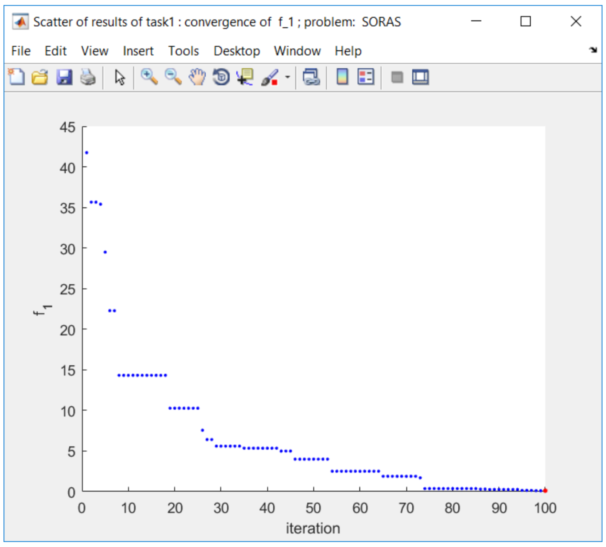


Figure 9.3: Convergence of fitness values of SORAS problem.

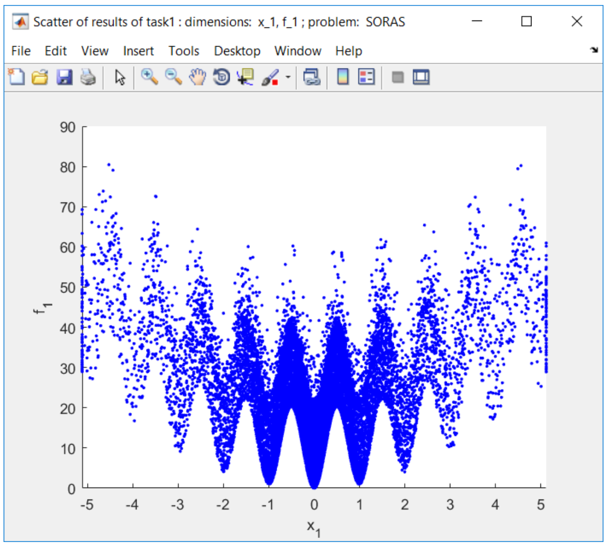


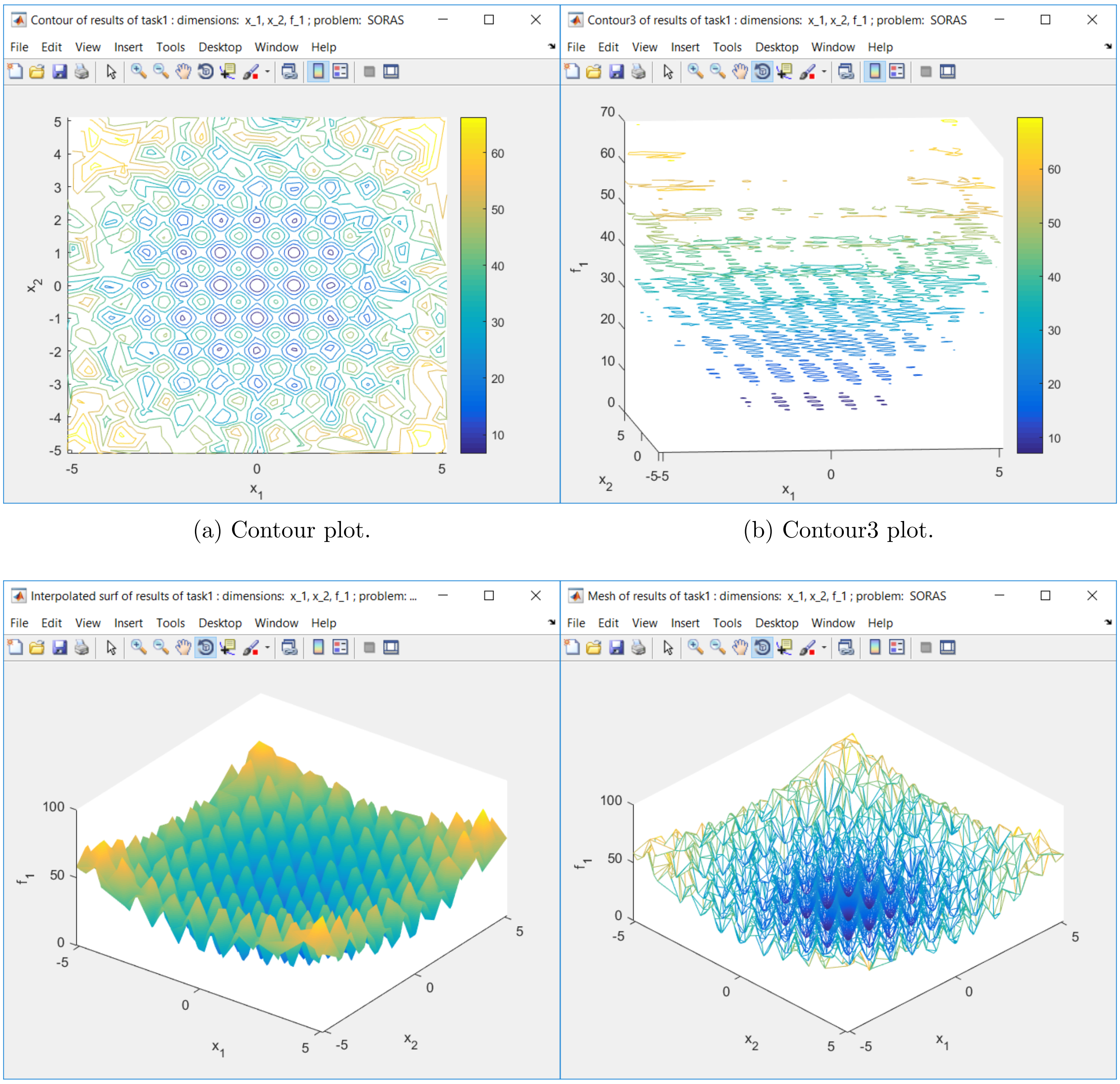
Figure 9.4: Dependence of fitness values on first decision variable of SORAS problem.

Figure 9.4 shows the dependence of fitness values on the first decision variable for `SORAS` problem. This time the problem had only two decision variables and algorithm had 1000 agents over 500 iterations. To generate such a figure, `Results Figure` is set as follows: the `Selected Dimensions` list box contain dimensions `x_1` and `f_1` below and the `Choose What To Plot` popup menu is set to `History`.

Figure 9.5 shows different plot types chosen in the `Style of Figure` popup menu. All figures show the dependence of fitness values on both decision variables of `SORAS` problem (the `Selected Dimensions` list box contains `x_1`, `x_2` and `f_1` and `Choose What To Plot` popup menu contains `History`).

Figure 9.5a shows the `Contour` style, which is 2D plot where the third dimension is substituted by the color of the curves (the color bar is shown). Note that only 10 curves are shown in each dimension and also the overall number of unique solutions to plot is limited to 10000 due the plotting complexity of Contour plot.

Figure 9.5b shows the `Contour3` style, which is 3D plot and the third dimension is also distinguished by the color of the curves (the color bar is shown). Note that only 10 curves are shown in



(a) Contour plot.

(b) Contour3 plot.

(c) Surf plot.

(d) Mesh plot.

Figure 9.5: Dependence of fitness values on both decision variables of `SORAS` problem.

each dimension and also the overall number of unique solutions to plot is limited to 10000 due the plotting complexity of Contour3 plot.

Figure 9.5c shows `Surf` style, which is 3D shaded surface plot. Note that the overall number of unique solutions to plot is limited to 100000 due the plotting complexity of Surf plot.

Figure 9.5d shows `Mesh` style, which is 3D mesh plot. Note that the overall number of unique solutions to plot is limited to 100000 due the plotting complexity of Mesh plot.

In Listing 9.2, comparative study with only one chain, one problem and 3 runs is created. History is enabled. Afterwards, it is run and a GUI is launched.

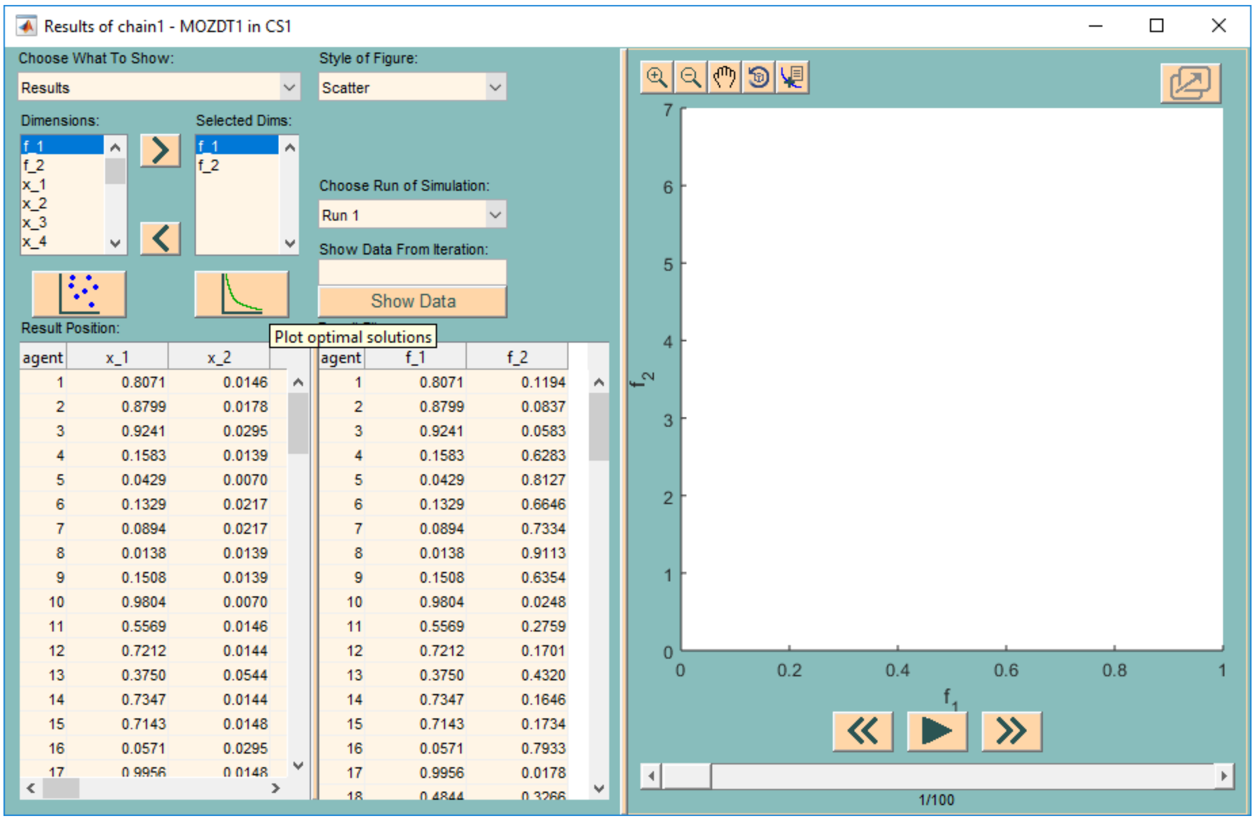
Listing 9.2: Creation of comparative study which results will be shown in GUI.

```
fops = FOPS();
fops.activeHistory = true;
set.nIters = 50;
requests = [];
settings = {[], set};
```

Figure 9.6 shows an animation of the results of the first run of the comparative study defined in Listing 9.2. At first, the true Pareto-front and the result Pareto-front are displayed while the `Choose What To Plot` is set to `Results`. Afterwards, the `Choose What To Plot` popup menu is changed to `History` and ▶ push button is pushed. The results from every second iterations are being plotted until `nIters` is reached.

9.2 Results of Comparative Study

It was mentioned in Chapter 8 that `Results figure` can display metric results of comparative study. Listing 9.3 shows creation of a comparative study with several two metrics and GUI is launched after running the comparative study.



agent	x_1	x_2
1	0.8071	0.0146
2	0.8799	0.0178
3	0.9241	0.0295
4	0.1583	0.0139
5	0.0429	0.0070
6	0.1329	0.0217
7	0.0894	0.0217
8	0.0138	0.0139
9	0.1508	0.0139
10	0.9804	0.0070
11	0.5569	0.0146
12	0.7212	0.0144
13	0.3750	0.0544
14	0.7347	0.0144
15	0.7143	0.0148
16	0.0571	0.0295
17	0.9956	0.0148

agent	f_1	f_2
1	0.8071	0.1194
2	0.8799	0.0837
3	0.9241	0.0583
4	0.1583	0.6283
5	0.0429	0.8127
6	0.1329	0.6646
7	0.0894	0.7334
8	0.0138	0.9113
9	0.1508	0.6354
10	0.9804	0.0248
11	0.5569	0.2759
12	0.7212	0.1701
13	0.3750	0.4320
14	0.7347	0.1646
15	0.7143	0.1734
16	0.0571	0.7933
17	0.9956	0.0178
18	0.4844	0.3266

Figure 9.6: Results figure of FOPS on the left side and an animation of the fitness values of the MOZDT1 problem on the right.

Listing 9.3: Creation of comparative study with requests.

```
fops = FOPS();
fops.showProgress = true;
problems = {'MOZDT1'; 'MOFON'};
chains = {'NSGAII', 'MOPSO', 'GDE3'; 'MOPSO', 'GDE3', []; 'MOPSO', [], []};
requests = {'GD', 'time'};
fops.addCompStudy(problems, chains, 10, requests)
fops.runCompStudy
```

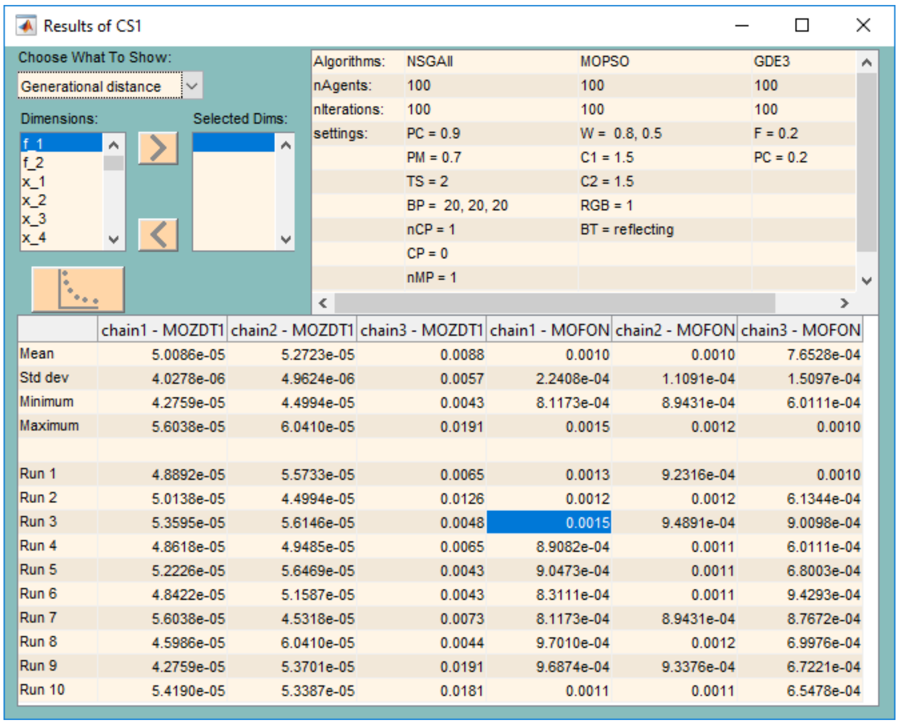
Figure 9.7 shows Results Figure of comparative study. In the top left corner of Results Figure is Choose What To Show popup menu, which allows user to select metric to be shown in Metrics table in the lower part of the Results Figure. Note that if values of selected metric was not calculated during the comparative study run (metrics input argument of CompStudy), they are calculated right away.

The Metrics table in the lower part of Figure 9.7 contains values from result properties of CompStudy object. Comparative study created in Listing 9.3 has property nRuns equal to 10, 3 unique chains and 2 problems. Therefore, the Metrics table has 6 columns (excluding the label column) and first 4 rows belong to statistical data - Mean denotes the mean value over all runs of a particular chain, Std dev denotes Standard deviation and Minimum and Maximum are minimal and maximal values over all runs, respectivelly. Under statistical data are 10 rows belonging to the ten repetitions of chains in the comparative study.

For better orientation in Metrics table, Properties table with chain's properties is shown in the upper part of Results Figure. Its content is based on selected cell in Metrics table. Column names of Metrics table suggests names of unique chains and an optimization problem.

The upper part of the Results Figure also shows controls for dimension selection identical to the controls from previous section. When the Results item is selected in the Choose What to Show, push buttons for adding and removing dimensions are enabled and the user can plot result Pareto-fronts of selected chains in Metrics table by clicking on push button. New MATLAB figure is opened. Afterwards, clicking on any chain in Metrics table highlights its results.

Metric convergence plots can also be plotted from Results Figure. In this case, history of all chains must be present, because the selected metric values are calculated for every iteration of each chain. Note that such an operation can be time demanding. Plotting is executed also by push



Algorithms:	NSGAII	MOPSO	GDE3
nAgents:	100	100	100
nIterations:	100	100	100
settings:	PC = 0.9	W = 0.8, 0.5	F = 0.2
	PM = 0.7	C1 = 1.5	PC = 0.2
	TS = 2	C2 = 1.5	
	BP = 20, 20, 20	RGB = 1	
	nCP = 1	BT = reflecting	
	CP = 0		
	nMP = 1		

	chain1 - MOZDT1	chain2 - MOZDT1	chain3 - MOZDT1	chain1 - MOFON	chain2 - MOFON	chain3 - MOFON
Mean	5.0086e-05	5.2723e-05	0.0088	0.0010	0.0010	7.6528e-04
Std dev	4.0278e-06	4.9624e-06	0.0057	2.2408e-04	1.1091e-04	1.5097e-04
Minimum	4.2759e-05	4.4994e-05	0.0043	8.1173e-04	8.9431e-04	6.0111e-04
Maximum	5.6038e-05	6.0410e-05	0.0191	0.0015	0.0012	0.0010
Run 1	4.8892e-05	5.5733e-05	0.0065	0.0013	9.2316e-04	0.0010
Run 2	5.0138e-05	4.4994e-05	0.0126	0.0012	0.0012	6.1344e-04
Run 3	5.3595e-05	5.6146e-05	0.0048	0.0015	9.4891e-04	9.0098e-04
Run 4	4.8618e-05	4.9485e-05	0.0065	8.9082e-04	0.0011	6.0111e-04
Run 5	5.2226e-05	5.6469e-05	0.0043	9.0473e-04	0.0011	6.8003e-04
Run 6	4.8422e-05	5.1587e-05	0.0043	8.3111e-04	0.0011	9.4293e-04
Run 7	5.6038e-05	4.5318e-05	0.0073	8.1173e-04	8.9431e-04	8.7672e-04
Run 8	4.5986e-05	6.0410e-05	0.0044	9.7010e-04	0.0012	6.9976e-04
Run 9	4.2759e-05	5.3701e-05	0.0191	9.6874e-04	9.3376e-04	6.7221e-04
Run 10	5.4190e-05	5.3387e-05	0.0181	0.0011	0.0011	6.5478e-04

Figure 9.7: Results of comparative study in Results Figure of FOPS GUI.

button and a new MATLAB figure is opened. Again, by clicking on any chain in `Metrics` table highlights the corresponding convergence plot.

Chapter 10

Tolerance-based Surrogate Method

Evolutionary algorithms usually exploits numerous agents (`nAgents`) over many iterations (`nIters`), which results in great number of fitness function evaluations. In real world, there are some optimization problems, where fitness function evaluation can be time consuming, therefore it is advantageous to keep the number of fitness function computations at minimum. However, evolutionary algorithm needs many agents and iterations to be able to find optimal solution.

Tolerance-based Surrogate Method allows optimization algorithm to skip some fitness function evaluations, if the position of agent is similar to some previously obtained solution. This can be helpful, particularly in real world problems, because some decision variables do not need to be overly precise (manufacturing limits and the like) and if some previously evaluated solution is similar to the newly generated one (they are close to each other in decision space), the fitness value of known solution is assigned to the new solution (solution is substituted). This is allowable, because an agent with similar position would most likely have also similar fitness values, therefore an error caused by such assumption is relatively small. Such methods are generally known as Surrogate optimization methods.

At the beginning of an algorithm run, no fitness values are known and all fitness function evaluations has to be performed and positions are stored in problem's surrogate field with corresponding fitness values (archive of known solutions). At some point in the optimization process the algorithm converges close to true Pareto-front and the generated positions (decision variables for which the fitness values have to be computed) begins to be similar (or equal) to members of the archive. Evaluation of fitness functions of such solutions has negligible contribution to the optimization process, therefore it is not performed and the fitness values of the closest solution are taken from the archive.

10.1 Tolerance Vector

How close has to be the generated position to a member of an archive is defines by the vector of tolerances, which has the number of elements equal to the number of decision variables (`nVars`). Each time a difference between position of a member of an archive and generated position is lower than the tolerance vector, the fitness functions evaluation is skipped.

Fig. 10.1 further clarifies the Tolerance-based Surrogate Method. It is the decision space of a `MO2D` problem (see Subsection 15.1.1). A light gray grid denotes the limits of decision variables, i.e. $x_1 \in \langle 0.1,\ 1\rangle$ and $x_2 \in \langle 0,\ 1\rangle$. Fitness functions are defined as follows:

$$f_1(\mathbf{x}) = x_1, \tag{10.1}$$

$$f_1(\mathbf{x}) = \frac{1 + x_2}{x_1}. \tag{10.2}$$

A red line marks the true Pareto-front of a problem in decision space. There are 15 solutions stored in the surrogate archive (their positions are marked with black thick crosses). Tolerance vector was set to $[0.05, 0.1]$ and areas within tolerance are marked by hatched boxes around solutions. Five of the solutions are marked with the index number from **1** to **5**. The solution **1**

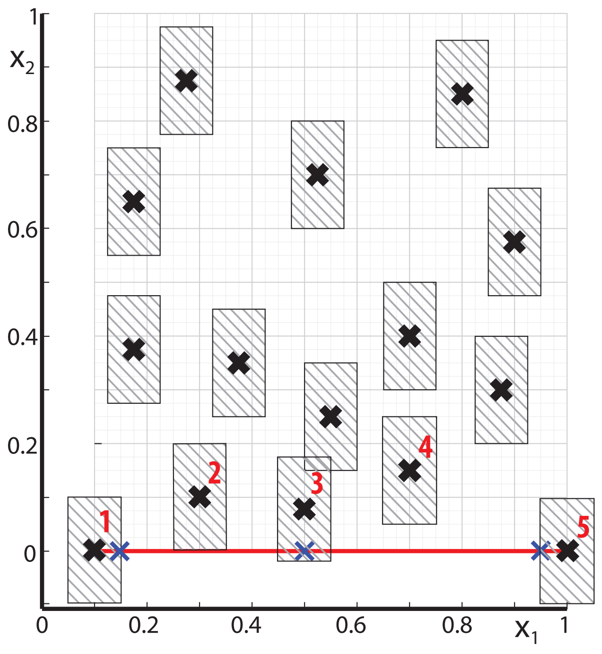


Figure 10.1: Decision space of MO2D problem with solutions stored in the interpolation archive.

is a true Pareto-optimal solution and its fitness values are [0.1, 10]. The solution **2** has fitness values [0.3, 3.67]. The solution **3** is not far from optimality (see that tolerance box covers true Pareto-front) and its fitness values are [0.5, 3.5]. The solution **4** has fitness values [0.7, 1.643] and the solution **5** has fitness values [1, 1]. All indexed solutions are non-dominated.

If newly generated solution has a position e.g. [0.94, 0] (marked with blue thin cross), it will fall in a tolerance area of solution **5** ([1, 0]) and even if its fitness values according to (10.1) and (10.2) would be [0.94, 1.064], the fitness values [1, 1] of the solution **5** will be assigned to it. A deviation in fitness values caused by Tolerance-based Surrogate Method is relatively small in this case.

Another generated solution has a position e.g. [0.14, 0] (marked with blue thin cross). Such solution will fall in a tolerance area of solution **1** ([0.1, 0]) and even if its fitness values according to (10.1) and (10.2) would be [0.1, 7.143], the fitness values [0.1, 10] of solution **1** will be assigned to it. The difference between true fitness values and the surrogate fitness values is rather large here, although the absolute distance between archive member and generated solution is identical as in the previous case with the solution **5**. This suggests that a setting of tolerance vector can be sometimes a difficult task.

The solution **3** in Fig. 10.1 indicates the drawback of Tolerance-based Surrogate Method. Border of hatched box of this solution lies on true Pareto-front, but the solution itself is rather far away ([0.5, 0.075]). Therefore, if new solution is generated within the hatched box eg. [0.5, 0] (marked with blue thin cross) then fitness values of known solution are assigned to it. But the fitness values of solution with position [0.5, 0] according to (10.1) and (10.2) are [0.5, 2], while the fitness values of solution **3** from surrogate archive are [0.5, 3.5]. Afterwards the solution [0.5, 0] (which is, as we know, better then surrogate archive member) will be supressed in an optimization process due to its worsened fitness values.

This drawback can be supressed with the use of a discrete decision space (see Chapter 11). Then, the tolerance vector can be set to almost zero values and the new solution can be either identical or differ by the step of discrete decision variable. If new solution generated by the optimization algorithm already exists in the surrogate archive, it is not calculated again.

Chapter 11

Discrete Variables

The definition of discrete decision variables in a problem struct was introduced in Section 4.6. The discrete decision variables are often required in real world optimization problems. For example, resistors made in sets E12 or E24 or diameter of some mechanical component can be made with limited precision. All the algorithms in FOPS supports discrete decision space although most algorithms (except the GA algorithms), are originally real-coded.

User inserts all samples of the decision variable to the property `discreteVariables` of a problem. Therefore, if user wants the decision variable take integer values between 0 and 10, the corresponding field of `discreteVariables` will contain the vector $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. An algorithm (except GAs) during its run generates real values, but they are afterwards adjusted to the closest member of the `discreteVariables` property (note that closeness is relative - it is ensured, that each member of `discreteVariables` property will be selected with the same probability). The adjusted values are inserted to a `position` properties of AGENTs and the fitness values are calculated from modified positions. The discretization of decision variables is more or less enforced to real-coded algorithms.

Discretization principle of GAs (SOGA and NSGA-II, Subsections 14.1.1 and 14.2.1) is completely inverse. If real-coded decision space is used, the binary sequence is generated as a position and it is afterwards mapped to the `limits` of a problem (see Section 4.1), but the decision space remains still discrete (density of discrete points depends on binary precision (`BP`) of a given decision variable).

If the `discreteVariables` property is defined in a problem optimized by the genetic algorithm, the binary precision `BP` of genetic algorithm is adjusted according to the number of elements of corresponding cell in `discreteVariables` property. If the default value of `BP` is `20` for each decision variable and the first cell of `discreteVariables` has 100 elements (100 discrete points of first decision variable), the binary precision of first discrete variable will be adjusted to `7` ($2^7 = 128$, $128 > 100$). The decision space will be shrinked, therefore it iwill be easier for the algorithm to find the optimal solutions.

Chapter 12

Constraints Handling

It was mentioned in Chapter 2, that some optimization problems are constrained, i.e. some properties of optimized system has additional restrictions. Solution that violates constraint function is called infeasible solution. Infeasible solution can be seen as an optimal one from the fitness values point of view, but an optimization algorithm should neglect such solutions during its run and present only feasible solutions as a result.

There are two ways of constraints handling in FOPS. The first one is specific to NSGA-II algorithm, while other algorithms uses the second approach.

1. Algorithm NSGA-II sorts the combined population (parent and offspring population) according to non-dominated levels in its each iteration(see Subsection 14.2.1). If the population is sorted into e.g. 10 non-dominated levels, then each solution has its rank from 1 to 10. An unfeasible solution has to be penalized in order to neglect such solution for following iterations. In NSGA-II algorithm, all unfeasible solutions are penalized by adding `penalization` to its rank, where the `penalization` is equal to the number of non-dominated levels in combined population. This suggests, that only if there are more than `nAgents` unfeasible solutions in combined population (it has 2·`nAgents` members), the unfeasible solution can be selected for next iterations.
2. Constraints in other algorithms are handled similarly. Since no ranks are used in other algorithms, fitness values of unfeasible solutions are altered. If a solution is unfeasible, the worst fitness values throughout the whole population are assigned to it. This ensures that even one feasible solution will dominate the unfeasible solutions. Therefore, the unfeasible solutions will not be further exploited.

Chapter 13

Input String Lexicon

When strings are inserted by user into FOPS, there are various alternatives that will be translated into the correct string. Therefore, user does not have to remember the exact form of input arguments. In this chapter will be discused possible string combinations for each input type.

Strings in this chapter are adjusted for good readability i.e. they contain whitespaces and/or capital letters, but the string translator in FOPS is case insensitive and removes all whitespaces and non-word characters from input strings. For example: user inserts `'NUmber of-Iterations?'`, which will be transformed into `'numberofiterations'` and the string `'numberofiterations'` is recognized as `nIters` property.

13.1 Problem Properties

`limits`
: `'limits'` | `'limit'` | `'lims'` | `'lim'`
: Limits of decision variables (see Section 4.1).

`fitness`
: `'fitness'` | `'fitnes'` | `'fintes'` | `'fintess'` | `'fit'` | `'f'`
: Fitness functions of problem (see Section 4.2).

`name`
: `'name'` | `'problem name'`
: Name of problem (see Section 4.3).

`constraints`
: `'constraints'` | `'constraint'` | `'constrains'` | `'constrain'` | `'cons'`
: Consitraint functions of problem (see Section 4.4).

`isVectorized`
: `'is vectorized'` | `'vectorized'` | `'isvector'` | `'vector'` | `'is vec'`
: Fitness and constraint functions of problem are vectorized (see Section 4.5).

`discreteVariables`
: `'discrete variables'` | `'discrete variable'` | `'discrete var'` | `'discrete vars'` | `'discrete'` | `'disc var'` | `'disc vars'` | `'DV'`
: Discrete decision variables of problem (see Chapter 11 or Section 4.6).

`surrogate`
: `'surrogate'` | `'surrogate method'` | `'surogate'` | `'surogate method'` | `'surro'` | `'suro'`
: Surrogate method (see Chapter 10 or Section 4.7).

`initialPosition`
: `'initial position'` | `'initial pos'` | `'initial X'` | `'init position'` | `'init pos'` | `'init X'` | `'start position'` | `'start pos'` | `'start X'`
: Initial position of local optimization (see Section 4.8).

`gradient`
: `'gradient'` | `'grad'`
Gradient of fitness functions (see Section 4.9).

`hessian`
: `'hessian'` | `'hesian'`
Hessian of fitness functions (see Section 4.10).

`nVarsList`
: `'nVars list'` | `'nVar list'` | `'nVar lst'` | `'nVars lst'` | `'Number of variables list'` | `'dimension list'` | `'dim list'` | `'nVars'` | `'nVar'`
Range of dimension (number of decision variables) of agent (see Section 4.11).

`optimalPosition`
: `'optimal position'` | `'opt X'` | `'optimal X'` | `'optimal pos'` | `'opt pos'` | `'opt position'`
Optimal positions of problem (see Section 4.12).

`optimalFitness`
: `'optimal fitness'` | `'opt F'` | `'optfitness'` | `'optimum'` | `'pareto front'` | `'true pareto front'`
Optimal fitness values of problem (see Section 4.13).

`optimalDimension`
: `'optimal dimension'` | `'optimal D'` | `'opt D'` | `'opt dim'` | `'opt dimension'` | `'optimal dim'`
Optimal dimension values of problem (see Section 4.14).

`reference`
: `'reference'` | `'reference point'` | `'hypervolume reference'` | `'hypervolume reference point'` | `'HV reference point'` | `'ref point'` | `'HV ref point'` | `'HV ref'` | `'HV reference'` | `'hypervolume ref'`
Reference point for hypervolume metric (see Section 8.3 and Section 4.15).

`fullControl`
: `'full control'` | `'ful control'`
Full control over OptTask object during fitness function evaluation (see Section 4.16).

`systemResponse`
: `'system response'` | `'response'` | `'sys res'` | `'system res'` | `'sys response'`
System response function handle (see Section 4.17).

13.2 General Task Properties

`nAgents`
: `'N agents'` | `'number of agents'` | `'number of agent'` | `'N agent'`
Number of agents (see Subsection 7.1.1).

`nIters`
: `'N iters'` | `'N iter'` | `'N iterations'` | `'N iteration'` | `'number of iterations'` | `'number of iteration'` | `'number of iters'` | `'number of iter'`
Number of iterations (see Subsection 7.1.1).

13.3 Genetic Algorithm Properties

The following properties are related to algoritms SOGA and NSGA-II (see Subsections 7.1.2 and 7.1.9).

`PC`
: `'PC'` | `'crossover probability'` | `'crossover prob'` | `'probability of crossover'`
: Probability of crossover.

`PM`
: `'PM'` | `'mutation probability'` | `'mutation prob'` | `'probability of mutation'`
: Probability of mutation.

`TS`
: `'TS'` | `'tournament size'` | `'tour size'`
: Tournament size (property of SOGA algorithm).

`BP`
: `'BP'` | `'binary precision'` | `'bin prec'` | `'binary prec'` | `'bin precision'`
: Probability of crossover.

`nCP`
: `'nCP'` | `'number of crossover positions'` | `'N cross positions'` |
`'number of cross positions'` | `'N cross position'` | `'number of cross pos'` |
`'N cross pos'`
: Number of crossover positions.

`CP`
: `'CP'` | `'crossover positions'` | `'crossover pos'` |
`'cross pos'` | `'C pos'` | `'crossing pos'`
: Crossover positions.

`nMP`
: `'nMP'` | `'number of mutation positions'` | `'N mut positions'` |
`'N mut position'` |
`'number of mut positions'` | `'N mut position'` | `'number of mut pos'` |
`'N mutation positions'` |
`'N mutation position'`
: Number of mutation positions.

`MP`
: `'MP'` | `'mutation positions'` | `'mutation points'` | `'mutation pos'` |
`'mut pos'` | `'mut P'`
: Mutation positions.

13.4 Particle Swarm Optimization Properties

The following properties are related to algoritms SOPSO, MOPSO and VND-PSO (see Subsections 7.1.3, 7.1.10 and 7.1.12).

`W`
: `'W'` | `'inertia weight'` | `'inertia weigth'` | `'weight'` | `'weigth'`
: Inertia weight.

`C1`
: `'C1'` | `'cognitive learning factor'`
: Cognitive learning factor.

`C2`
: `'C2'` | `'social learning factor'`
Social learning factor.

`BT`
: `'BT'` | `'boundary type'` | `'boundary'` | `'bound'` | `'bound type'`
Boundary type.

`RGB`
: `'RGB'` | `'random GBest'` | `'rand GBest'` | `'G best random'` | `'G best rand'` | `'global best rand'` | `'global best random'` | `'random global best'` | `'random global bests'` | `'rand global best'`
Random global best (property of MOPSO algorithm).

`DCT`
: `'DCT'` | `'DC type'` | `'dim C type'` | `'dim check type'` | `'dim check'` | `'dimension check'` | `'dimension check type'`
Dimension check type (property of VND-PSO and VND-MOPSO algorithms).

`PTF`
: `'PTF'` | `'probabilities to follow'` | `'prob to follow'` | `'probabilities'`
Probabilities to follow (property of VND-PSO and VND-MOPSO algorithms).

13.5 Differential Evolution Properties

The following properties are related to algoritms SODE and GDE3 (see Subsections 7.1.4 and 7.1.11).

`PC`
: `'PC'` | `'crossover probability'` | `'crossover prob'` | `'probability of crossover'`
Probability of crossover.

`F`
: `'F'` | `'scaling factor'` | `'scale factor'` | `'scaling'` | `'scale'`
Scaling factor.

`DCT`
: `'DCT'` | `'DC type'` | `'dim C type'` | `'dim check type'` | `'dim check'` | `'dimension check'` | `'dimension check type'`
Dimension check type (property of VND-GDE3 algorithm).

`PTF`
: `'PTF'` | `'probabilities to follow'` | `'prob to follow'` | `'probabilities'`
Probabilities to follow (property of VND-GDE3 algorithm).

13.6 Self-organizing Migrating Algorithm Properties

The following properties are related to algoritm SOSOMA (see Subsection 7.1.5).

`PL`
: `'PL'` | `'multiplier'` | `'step multiplier'`
Multiplicator of distance between steps.

`ST`
: `'ST'` | `'steps'` | `'stepsize'` | `'number of steps'` | `'N steps'`
Number of steps.

`PR`
: `'PR'` | `'probability'` | `'perturbance'` | `'perturb'` | `'probability of perturbance'` |

`'perturbance probability'` | `'perturbance prob'` | `'prob of perturbance'`
Probability of perturbance.

`BT`
`'BT'` | `'boundary type'` | `'boundary'` | `'bound'` | `'bound type'`
Boundary type.

`MT`
`'MT'` | `'migration type'` | `'migration'` | `'migrate'` | `'migrate type'` | `'mig type'`
Migration type.

13.7 Nelder-Mead Properties

The following properties are related to algoritm SONEME (see Subsection 7.1.6).

`aplha`
`'aplha'` | `'A'` | `'reflection coefficient'` | `'reflection'` | `'reflect'` | `'reflection coef'` | `'reflect coef'`
Reflection coefficient.

`beta`
`'beta'` | `'B'` | `'contraction coefficient'` | `'contraction'` | `'contract'` | `'contraction coef'` | `'contract coef'`
Reflection coefficient.

`gama`
`'gama'` | `'C'` | `'gamma'` | `'expansion'` | `'expansion coefficient'` | `'expand'` | `'expansion coef'` | `'expand coef'`
Expansion coefficient.

`BT`
`'BT'` | `'boundary type'` | `'boundary'` | `'bound'` | `'bound type'`
Boundary type.

13.8 Newton Algorithm Properties

The following properties are related to SONEW(see Subsection 7.1.7).

`diff`
`'diff'` | `'difference'` | `'diference'`
Difference.

13.9 Covariance Matrix Adaptation Evolutionary Strategies Properties

The following properties are related to SOCMAES(see Subsection 7.1.8).

`mu`
`'mu'` | `'mu portion'` | `'effective N'` | `'N eff'` | `'effective nAgents'`
Effective number of agents.

13.10 Chain Properties

The following properties are related to chain (see Section 7.2).

`algorithms`
: `'algorithms'` | `'algorithm'` | `'tasks'` | `'task'` | `'method'` | `'methods'` | `'algh'` | `'alghs'`
: Change algorithms in chain.

`problem`
: `'problem'` | `'problems'` | `'prob'` | `'probs'`
: Change problem of chain.

13.11 Comparative Study Properties

The following properties are related to comparative study (see Section 7.3).

`nRuns`
: `'N runs'` | `'N run'` | `'num of runs'` | `'num of run'` | `'number of runs'` | `'number of run'` | `'repeats'` | `'repetitons'` | `'repetitons'`
: Change number of runs of comparative study.

`chains`
: `'chains'` | `'chain'` | `'algorithms'` | `'algorithm'`
: Change chains in comparative study.

`addChains`
: `'add chains'` | `'add chain'` | `'add algorithms'` | `'add algorithm'` | `'ad chains'` | `'ad chain'`
: Add chains to comparative study.

`problems`
: `'problems'` | `'problem'` | `'probs'` | `'prob'`
: Change problems in comparative study.

`addProblems`
: `'add problems'` | `'add problem'` | `'add probs'` | `'add prob'` | `'ad problems'` | `'ad problem'`
: Add problems to comparative study.

`requests`
: `'requests'` | `'request'` | `'reqs'` | `'req'`
: Change requests in comparative study.

`addRequests`
: `'add requests'` | `'add request'` | `'add reqs'` | `'add req'` | `'ad requests'` | `'ad request'`
: Add requests to comparative study.

13.12 Algorithm Types

`SOGA`
: `'SOGA'` | `'SOGA Task'`
Algorithm SOGA (see Subsection 14.1.1).

`SOPSO`
: `'SOPSO'` | `'SOPSO Task'`
Algorithm SOPSO (see Subsection 14.1.2).

`SODE`
: `'SODE'` | `'SODE Task'`
Algorithm SODE (see Subsection 14.1.3).

`SOSOMA`
: `'SOSOMA'` | `'SOSOMA Task'`
Algorithm SOSOMA (see Subsection 14.1.4).

`SONEME`
: `'SONEME'` | `'SONEMET'` | `'NEME'` | `'Nelder Mead'` | `'SONEME Task'`
Algorithm SONEME (see Subsection 14.1.5).

`SONEW`
: `'Newton'` | `'SO Newton'` | `'New'` | `'SO New'` | `'Newton Task'`
Newton algorithm (see Subsection 14.1.6).

`SOCMAES`
: `'SOCMAES'` | `'CMAES'` | `'ES'` | `'CMAES task'` | `'SOCMAES Task'`
Algorithm SOCMAES (see Subsection 14.1.7).

`NSGAII`
: `'NSGA II'` | `'NSGA 2'` | `'MO NSGA II'` | `'MO NSGA 2'` | `'NSGA II Task'` | `'NSGA 2 Task'`
Algorithm NSGA-II (see Subsection 14.2.1).

`MOPSO`
: `'MOPSO'` | `'MOPSO Task'`
Algorithm MOPSO (see Subsection 14.2.2).

`GDE3`
: `'GDE 3'` | `'GDE'` | `'GDE III'` | `'MO GDE'` | `'MO GDE 3'` | `'MO GDE III'` | `'GDE 3 Task'` | `'GDE Task'` | `'GDE III Task'`
Algorithm GDE3 (see Subsection 14.2.3).

`VNDPSO`
: `'VND PSO'` | `'SO VND PSO'` | `'VND SOPSO'` | `'PSO VND'` | `'SOPSO VND'` | `'VND PSO Task'`
Algorithm VND-PSO (see Subsection 14.3.1).

`VNDMOPSO`
: `'VND MOPSO'` | `'VND PSO MO'` | `'MOVND PSO'` | `'PSO VND MO'` | `'PSO MO VND'` | `'MOPSO VND'` | `'VND MOPSO Task'` | `'VDN MOPSO'`
Algorithm VND-MOPSO (see Subsection 14.3.2).

`VNDGDE3`
: `'VND GDE3'` | `'VND GDE3 Task'` | `'GDE3 VND'` | `'VND GDE'` | `'GDE VND'` | `'VND GDE III'` | `'GDE III VND'` | `'VDN GDE3'`
Algorithm VND-GDE3 (see Subsection 14.3.3).

13.13 Metric Types

`GenerationalDistance`
: `'Generational Distance'` | `'GD'` | `'Generational Distance GD'` |

`'Generation Distance'` | `'Gen Dist'`
Generational distance metric (see Subsection 8.1).

`Spread`
`'Spread'` | `'Delta'` | `'Spread Delta'`
Spread metric (see Subsection 8.2).

`Hypervolume`
`'Hypervolume'` | `'HV'` | `'Hypervolume HV'`
Hypervolume metric (see Subsection 8.3).

`ErrorRatio`
`'Error Ratio'` | `'ER'` | `'Error Ratio ER'`
Error ratio metric (see Subsection 8.4).

`Spacing`
`'Spacing'` | `'S'` | `'Spacing S'`
Spacing metric (see Subsection 8.5).

`MaximumSpread`
`'Maximum Spread'` | `'D'` | `'Maximum Spread D'`
Maximum spread metric (see Subsection 8.6).

`FitnessError`
`'Fitness Error'` | `'FE'` | `'FER'` | `'F error'` | `'Fit E'` | `'Fit Error'` | `'Fitness E'` |
`'Fitness Error'` | `'Fitness Error FER'`
Fitness error metric (see Subsection 8.7).

`PositionError`
`'Position Error'` | `'DE'` | `'DER'` | `'PE'` | `'PER'` | `'D error'` | `'P error'` | `'Pos E'` |
`'Pos Error'` | `'Pos Er'` | `'XE'` | `'XER'` | `'X Error'` | `'Position E'` | `'Position Error DER'`
Position Error metric (see Subsection 8.8).

`ComputationalTime`
`'Computational Time'` | `'T'` | `'Time'` | `'CPU Time'` | `'Compute Time'` | `'Comp Time'` |
`'Computational Time T'`
Computational time metric (see Subsection 8.9).

`nS`
`'nS'` | `'Number of surrogate solutions'` | `'N surrogate'` | `'N surrogates'` | `'num surrogate'`
| `'num surrogates'` | `'N times used surrogate'` | `'N used surrogate'` |
`'Number of surrogate solutions'`
Number of surrogate solutions metric (see Subsection 8.10).

`N`
`'N'` | `'Number of solutions'` | `'N solutions'` | `'N sol'` | `'N sols'`
Number of solutions metric (see Subsection 8.11).

`NVD`
`'NVD'` | `'Number of variables deviation'` | `'N Vars'` | `'N Variables'` | `'N Variable'` | `'N
dim'` | `'N dims'` | `'N dims deviation'` |
`'N dims dev'` |
`'deviation'` |
`'N dev'` |
`'Number of variables'` |
`'N var dev'` |
`'N vars dev'`
Number of variables deviation metric (see Subsection 8.12).

13.14 Dimension Check Types

Dimension check type is related to VND algorithms (see Subsections 14.3.1, 14.3.2, and 14.3.3)

`Best`
: `'Best'` | `'Best DC'`
  Best dimension check type.

`Random`
: `'Random'` | `'RND'` | `'Rand'` | `'Random DC'` | `'Rand DC'` | `'Ran DC'`
  Random dimension check type.

`No`
: `'No'` | `'NA'` | `'None'` | `'No DC'` | `'None DC'`
  No dimension check.

13.15 Boundary Types

Boundary type is related to PSO, Nelder-Mead and SOMA algorithms (see Subsections 14.1.2, 14.2.2, 14.3.1, 14.3.2, 14.1.5 and 14.1.4)

`Reflecting`
: `'Reflecting'` | `'Ref'` | `'Reflect'`
  Reflecting boundary type.

`Absorbing`
: `'Absorbing'` | `'Abs'` | `'Absorb'`
  Absorbing boundary type.

`Invisible`
: `'Invisible'` | `'Inv'`
  Invisible boundary type.

13.16 Migration Types

Migration type is related to SOMA algorithm (see Subsection 14.1.4)

`AllToLeader`
: `'All To Leader'` | `'All To One'` | `'All Leader'` | `'All One'`
  All to leader migration type.

`AllToAll`
: `'All To All'` | `'All All'`
  All to all migration type.

Chapter 14

Algorithms

As was mentioned in Chapter 2, optimization methods can be divided into single-objective and multi-objective according to the number of fitness functions that describes the optimized system. Methods can also be either local or global. Local method is able to find only local optimal solution, while the global method should be able to find global optimal solution. Newton (see Subsection 14.1.6) and Nelder-Mead (see Subsection 14.1.5) methods are the only local methods implemented in FOPS.

FOPS also contains special single-objective method for solving problems with Variable Number of Dimension – VND-PSO (see Subsection 14.3.1).

14.1 Single-objective Algorithms

14.1.1 SOGA

As the name suggests, principle of genetic algorithm is based on natural genetics and natural selection. There exist numerous textbooks on GAs such as [4], [5], [6] or [2]. GA works with two populations (set of agents) – parent and offspring. Offspring population is derived from parent population in each iteration of genetic algorithm.

Formation of offspring population consists of selection, crossover and mutation. During selection operation, potentially strong individuals (agents) are selected among parent population and weak individuals are neglected. The objective of selection is to make duplicates of good solutions at the expense of bad solutions. Crossover is a pair-wise crossing of randomly selected individuals. Afterwards, mutation of offspring population is performed to ensure diversity in the population.

Tournament selection is used in FOPS. In tournament selection, tournaments are played between `TS` (Tournament Size, see Subsection 7.1.2) randomly selected solutions and the best solution according to fitness values is chosen to be further exploited. `nAgents` tournaments are played during each selection process. Therefore, parameter `TS` defines the depth of selection. If `TS` is set to one, solutions are picked completely at random. If `TS` is set to `nAgents` (maximal possible value), repetitions of the best solution in population forms the output of selection. The default value of `TS` in FOPS is `2`.

GA is a binary coded algorithm. It means that the agents position is described by sequence of binary digits. However, GAs are not restricted to use only integer values. The number of binary digits depends on `BP` (binary precision, see Subsection 7.1.2) parameter. `BP` can differ for each decision variable. Default value of `BP` in FOPS is `20` for each decision variable. Therefore, an agent with 3 decision variables and default `BP` value will be described by a sequence of 60 zeros and ones (note that the use of discrete variables (see Chapter 11) changes `BP` property).

In crossover, two solutions are picked at random from population and parts of binary coded positions are exchanged between the two picked solutions with probability of `PC` (Probability of Crossover, see Subsection 7.1.2). If `PC` is set to `1`, this operation creates `nAgents` new solutions. If `PC` is set to `0`, no crossover operation will be performed. Default value of `PC` in FOPS is `0.9`. `nCP` (number of Crossover Positions, see Subsection 7.1.2) parameter defines number of places, where

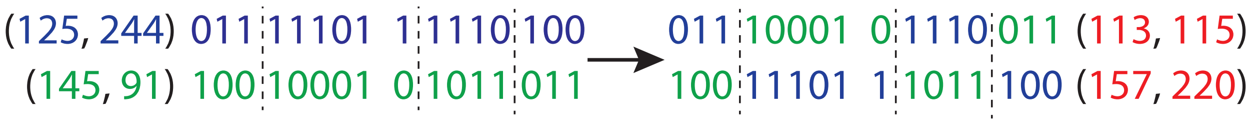


Figure 14.1: Crossover operation with nCP = 3.

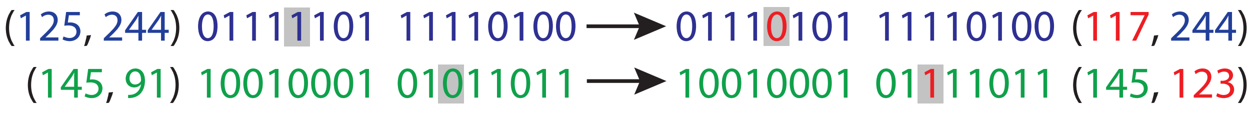


Figure 14.2: Mutation operation of two agents.

the binary positions will be crossed. Crossover positions are generated randomly. Default value of nCP in FOPS is 1. Maximal value of nCP is $\sum$ BP $- 1$. Such value results in swap of each even binary place.

Figure 14.1 depicts the crossover of two agents with two decision variables (BP $= 8$ for both decision variables). Brackets contain decimal values of binary coded positions.

Mutation operation changes randomly picked place in binary position with the probability of PM (Probability of Mutation, see Subsection 7.1.2). If PM is set to 1, each agents position is mutated. If PM is set to 0, no mutation operation will be performed. The default value of PM in FOPS is 0.7. Figure 14.2 depicts the mutation of two randomly picked agents. Positiion of mutated binary place is randomly picked.

14.1.2 SOPSO

Particle Swarm Optimization is evolutionary algorithm, which simulates movement of swarm of bees searching for food. It is, unlike GAs, real-coded algorithm. It was originally proposed for use in neural networks [7] and was later adopted as global optimizer.

The position of each agent is changed according to its own experience and that of its neighbors. The position x of i-th agent in subsequent iteration is changed by adding velocity v to a current position:

$$x_i(t) = x_i(t-1) + v_i(t), \tag{14.1}$$

where the velocity vector v is defined as follows:

$$v_i(t) = \mathtt{W} \cdot v_i(t-1) + \mathtt{C1} \cdot r_1 \left[x_{pbest} - x_i(t)\right] + \mathtt{C2} \cdot r_2 \left[x_{gbest} - x_i(t)\right], \tag{14.2}$$

where r_1, $r_2 \in \langle 0, 1 \rangle$ are random values, W is inertia weight, C1 and C2 are cognitive and social learning factors, respectively, x_{pbest} is personal best position and x_{gbest} is global best position.

Inertia weight introduces the impact of the previous velocities on current velocity. It is controlled by parameter W (see Subsection 7.1.3). The default value of W in FOPS is [0.8, 0.5]. The two values denotes, that W is linearly decreased from 0.8 to 0.5 during the simulation. Agents follow its inertia weight more in early iterations than in late iterations. The larger the W value the larger the impact of velocity from previous iteration.

Cognitive learning factor C1 represents attraction of agent to its own success, i.e. agent is attracted to the best position it has previously been at - x_{pbest}. The default value of parameter C1 in FOPS is 1.5. The larger the C1 the more attracted the particle is to the position x_{pbest} (see Subsection 7.1.3).

Social learning factor C2 represents attraction of agent to success of its neighbors, i.e. agent is attracted to the best position found so far - x_{gbest}. The default value of parameter C2 in FOPS is 1.5. The larger the C2 the more attracted the particle is to the position x_{gbest} (see Subsection 7.1.3).

When an agent's position is updated according to Equation 14.1, there is a chance, that the new position will be out of decision variables limits. In such situation, boundary conditions have to be exploited. Boundary conditions modify agents position and velocity vector. Decision variables that are out of its limits are set to a value of adjacent limit. Velocity vector is modified according to parameter BT (Boundary Type, see Subsection 7.1.3). There are three types of boundaries in FOPS:

- Reflecting (default): elements of velocity vector, which decision variables are out of limits are negated.
- Absorbing: elements of velocity vector, which decision variables are out of limits are zeroed.
- Invisible: no action is taken. Velocity vector is in few iterations pushed in a proper direction.

14.1.3 SODE

The Differential Evolution algorithm was introduced in 1997 [8]. Implemented version corresponds with DE/rand/1/bin version [9]. As a name suggests, it is also evolutionary algorithm. Initially, random population is generated and in each iteration of algorithm the trial vector u is created from agent's positions x (crossover operation) using the equation:

$$u_{i,j}\left(t\right) = x_{j,r_3} + \mathtt{F} \cdot \left(x_{j,r_2} - x_{j,r_1}\right), \tag{14.3}$$

where j denotes j-th decision variable, r_1, r_2, r_3 are randomly picked, mutually different and different from i agents indices and parameter F is scaling factor.

Equation 14.3 is used with the probability of PC (Probability of Crossover) or if j=j_{rand}, where $j_{rand} = 1, 2, \ldots \mathtt{nVars}$ is random value.

Both PC and F are control parameters that remains fixed during whole execution of algorithm. Lower value of PC (e.g. 0.1) are better for separable problems while larger values (e.g. 0.9) are better for non-separable problems. Parameter F controls speed and robustness of the search. Lower value the increases convergence speed with the risk of stacking into a local optimum. Default value of PC in FOPS is 0.2 and default value of F in FOPS is also 0.2.

14.1.4 SOSOMA

SOMA stands for Self=Organizing Migrating Algorithm first introduced by Ivan Zelinka in 2000. Its comprehensive review can be found in [10]. The algorithm works with set of agents that is generated randomly inside the decision space. At every iteration, the agent with the best value of objective function is assigned as so called leader. All other agents then visit temporal positions according to:

$$\mathbf{t}_{p,s}(i) = \mathbf{x}_p(i-1) + \left[\mathbf{x}_q(i-1) - \mathbf{x}_p(i-1)\right] \times \frac{s}{\mathtt{ST}} \times \mathtt{PL} * \mathbf{PV}_{p,s}, \tag{14.4}$$

where $\mathbf{x}_p(i)$ is the position of the leader in the i-th iteration, $\mathbf{x}_q$ is the q-th agent position. Index s goes from 1 to ST which is the number of steps used in one migration. Parameter PL defines the length of the path (if PL $= 1$ the migration ends exactly at leaders position). Symbol $*$ denotes the element-wise multiplication. Perturbation vector **PV** prevents the algorithm to get stuck in the local minimum. It is composed according to:

$$PV_n = \begin{cases} 1 & \text{if} rnd(n) > \mathtt{PR} \\ 0 & \text{if} rnd(n) \leq \mathtt{PR} \end{cases} \tag{14.5}$$

where **PV** denotes the probability of perturbation defined by a used. If any part of the **PV** is equal to zero, the migration does not respect corresponding dimension. If the objective function value of any agent's trial position gets better than the previous one, the agent changes its position. If the objective function value is not decreased, the agent remains in the same position.

There exist three variants of the SOMA algorithm. In 'AllToOne' variant, all agents from the set migrate towards the leader. In 'AllToAll', all combination of agents positions are used for the migration. In 'AllToRand' every agent migrates towards randomly chosen one.

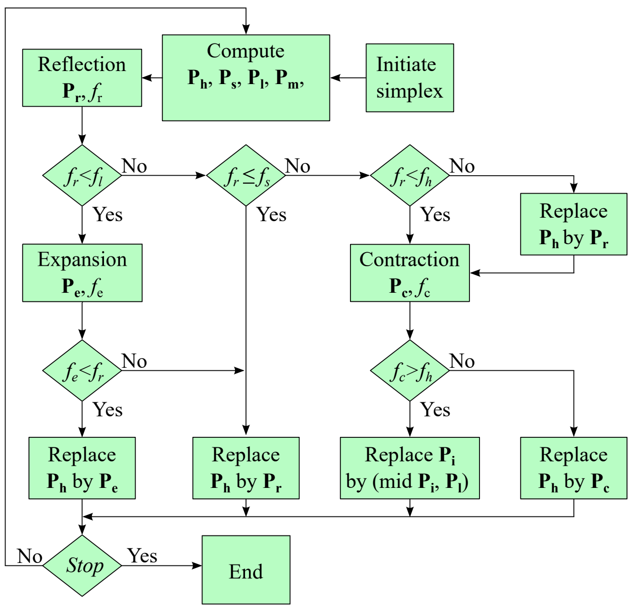


Figure 14.3: Pseudocode of the Nelder–Mead optimization algorithm.

14.1.5 SONEME

Nelder–Mead or simplex algorithm is fully described e.g. in [11]. If we want to solve a problem of dimension $N-1$, it works with a simplex of dimension N (line segment for 1D, triangle for 2D, tetrahedral for 4D, and so on). The i-th member of the initial simplex is created:

$$\mathbf{x}_i = \mathbf{x}_0 + p\mathbf{e}_i + \sum_{k\neq i}^{N} q\mathbf{e}_k, \tag{14.6}$$

where $\mathbf{x}_0$ is the initial guess set by a user, $\mathbf{e}_i$ denotes the i-th column of the identity matrix of size $N+1$. Coefficients p and q are computed:

$$p = \frac{a}{N\sqrt{2}}\left(\sqrt{N+1}+N-1\right), \tag{14.7}$$

$$q = \frac{a}{N\sqrt{2}}\left(\sqrt{N+1}-1\right) \tag{14.8}$$

with a being the initial simplex size chosen by a user.

Three methods for transformation of the simplex are defined, namely 'reflection':

$$\mathbf{p}_r = \mathbf{p}_m + \alpha(\mathbf{p}_m - \mathbf{p}_h), \tag{14.9}$$

'expansion':

$$\mathbf{p}_e = \mathbf{p}_m + \beta(\mathbf{p}_h - \mathbf{p}_m), \tag{14.10}$$

and 'contraction':

$$\mathbf{p}_c = \mathbf{p}_m + \gamma(\mathbf{p}_s - \mathbf{p}_m). \tag{14.11}$$

Here $\mathbf{p}_m$ is the centroid position computed as a mean of $\mathbf{p}_s$ (simplex member with second best objective function value) and $\mathbf{p}_l$ (the one with the worst value). Symbol $\mathbf{p}_h$ denotes the point with the best objective function value. The recommended values of the reflection, expansion, and contraction coefficients are $\alpha = 1$, $\beta = 0.5$, and $\alpha = 2$, respectively. The pseudocode of the algorithm is shown in Fig. **??**.

14.1.6 SONEW

The Newton method works with one solution at a time. The initial guess $\mathbf{x}_0$ defined by user. Then, it is updated at every iteration i according to:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{H}(\mathbf{x}_{i-1})^{-1} \cdot \mathbf{g}(\mathbf{x}_{i-1}) \tag{14.12}$$

where $\mathbf{H}(\mathbf{x}_{i-1})$ is the Hessian matrix computed for position $\mathbf{x}_{i-1}$ while $\mathbf{g}(\mathbf{x}_{i-1}$ is the gradient vector for the same point.

14.1.7 SOCMAES

To be added

14.2 Multi-objective Algorithms

14.2.1 NSGA-II

Elitist Non-dominated Sorting Genetic Algorithm is an improved version of NSGA algorithm [12]. It was proposed in 2002 [1]. As a name suggests it is elitist algorithm and selection operation is substituted by non-dominated sorting.

In initial iteration of the algorithm, random population is generated and selection, crossover and mutation is performed similarly as in SOGA algorithm (see Subsection 14.1.1). Afterwards, parent and offspring populations are sorted according to its non-dominated levels by fast non-dominated sorting algorithm [1]. Exactly `nAgents` individuals are selected according to their non-dominated level to form new parent population for next iteration (see Figure 14.4).

If some non-dominated level has to be trimmed to fit into precisely `nAgents` individuals, operation that maintains diversity of non-dominated set has to be used. Such method in FOPS uses crowded distance as a metric for the pruning [1]. Unfortunately, the crowding distance selection works satisfactorily only with 2-dimensional space, i.e. 2 objective functions. To make it work also for many-objective problems (3 or more), concept of Equal-average Nearest Neighbor Search (ENNS) [13] has to be adopted, which basically converts N-dimensional space into 2-dimensional space.

Settings parameters of NSGA-II algorithms are `PC`, `PM`, `BP` and `nCP` (see Subsection 14.1.1) and their default values in FOPS are the same as for SOGA algorithm (see Subsection 7.1.9. Parameter tournament selection `TS` is used only in an initial iteration. Therefore, its impact on the overall behavior of algorithm is negligible.

14.2.2 MOPSO

There are many modifications of Particle Swarm Optimization algorithm for solving Multi-objective problems. The first extension came in 1999 [14]. The implemented algorithm in FOPS is based on version presented in [15].

The main difference from the single-objective version is the selection of global bests. In single-objective version only one solution is said to be the global best, but a multi-objective problem presents multiple trade-off solutions that are said to be the good ones.

Non-dominated (trade-off) solutions, in other words Pareto-front, are stored in an external archive. External archive holds all the non-dominated solutions found so far and its members are used as global bests in velocity update (see Equation 14.2).

At the beginning of the algorithm, random population is generated and an external archive is initialized with non-dominated members of random population. Personal bests are found in a same way as in SOPSO algorithm (see Subsection 14.1.2). Global bests are selected among external archive members. The selection is based on two principles:

- Random assignment.

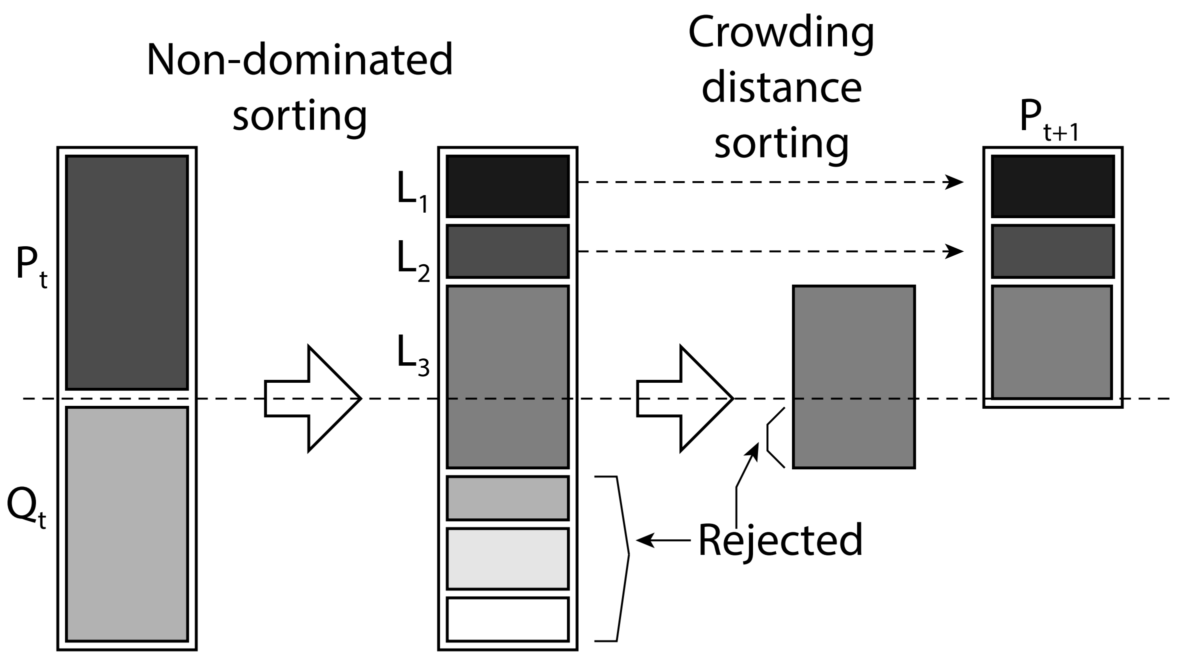


Figure 14.4: Non-dominated sorting based selection in NSGA-II algorithm [1].

- Minimal Euclidean distance between agents fitness values and external archive members fitness values.

The ratio between global bests assigned according to Euclidean distance and randomly assigned is controlled by parameter `RGB` (Random Global Best). It expresses the percentage of randomly assigned global bests. Therefore, `RGB`=`1` expresses that all global bests are selected randomly and algorithm focuses on exploration. `RGB`=`0` enforces the fast convergence to optima. The default value of `RGB` in FOPS is `1`.

Some MOPSO implementations uses mutation operation to promote diversity in an optimization process. The PSO algorithm is known for its fast convergence, which might lead to premature convergence to local optima on multi-objective version of PSO. Diversity in FOPS's MOPSO is promoted by random assignment of global best solution, therefore the `RGB` property is set to `1` as default.

The overall number of agents in external archive has to be limited, due to algorithm performance. External archive size is equal to the population size - `nAgents`. If there are more than `nAgents` non-dominated candidates for an external archive, solutions have to be pruned with regard to the diversity of non-dominated set. It is ensured similarly as in NSGA-II (see Subsection 14.2.1) [1], [13].

Other setting parameters are analogous to parameters of SOPSO (see Subsection 14.1.2) - `W`, `C1`, `C2` and `BT` . Their default values in FOPS are identical to SOPSO's (see Subsection 7.1.10).

14.2.3 GDE3

Third version of Generalized Differential Evolution is an extension of presented SODE algorithm (see Subsection 14.1.3). It was proposed in 2005 in [16].

The multi-objective version extends the single-objective one by the selection rule between decision and trial vector if both solutions are non-dominated. In such cases, both solutions are accepted and the population is extended.

Nonetheless, the overall number of agents is fixed throughout the simulation. If the population size exceeds the `nAgents` size at the end of an iteration, non-dominated solutions has to be pruned with regard to diversity of non-dominated set. It is ensured similarly as in NSGA-II (see Subsection 14.2.1) [1],[13].

Setting parameters of GDE3 are also `PC` and `F`. Default values of both parameters in FOPS are `0.2` (see Subsection 7.1.11).

14.3 Algorithms with Variable Number of Dimensions

14.3.1 VNDPSO

To be added

14.3.2 VNDMOPSO

To be added

14.3.3 VNDGDE3

To be added

Chapter 15

Benchmark Problems

15.1 Multi-objective

15.1.1 MO2D

This is a simple example of two-objective optimization problem. Problem has two decision variables. Problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_1 \in \langle 0.1, 1 \rangle, \tag{15.1}$$
$$x_2 \in \langle 0, 1 \rangle. \tag{15.2}$$

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.3}$$
$$f_2(\mathbf{x}) = \frac{1 + x_2}{x_1}. \tag{15.4}$$

Optimal position:

$$x_1 \in \langle 0.1, 1 \rangle, \tag{15.5}$$
$$x_2 = 0. \tag{15.6}$$

Optimal positions are reached if $x_1 \in \langle 0.1, 1 \rangle$ and $x_2 = \mathbf{0}$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.2 MO3D

This is a simple example of three-objective optimization problem taken from [17]. Problem has two decision variables. Problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_1 \in \langle -2, 2 \rangle, \tag{15.7}$$
$$x_2 \in \langle -2, 2 \rangle. \tag{15.8}$$

Fitness functions:

$$f_1(\mathbf{x}) = x_1^2 + (x_2 - 1)^2, \tag{15.9}$$
$$f_2(\mathbf{x}) = x_1^2 + (x_2 + 1)^2 + 1, \tag{15.10}$$
$$f_3(\mathbf{x}) = (x_1 - 1)^2 + x_2^2 + 2. \tag{15.11}$$

Optimal positions are inside the triangular area defined by corner points [0, -1], [1, 0] and [0, 1] and they are also stored in `position` property of mat-file `MO3D`.

Optimal fitness values are stored in `fitness` property of mat-file `MO3D`.

15.1.3 MOCantilaver

This problem is taken from [2]. It is a simple constrained problem with two decision variables and two objectives. The problem function serves as manual for problem m-file definition. Problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_1 \in \langle 10, 50 \rangle \text{ mm}, \tag{15.12}$$
$$x_2 \in \langle 200, 1000 \rangle \text{ mm}, \tag{15.13}$$

where x_1 denotes the diameter of the cantilaver and x_2 denotes the length of the cantilaver.

Fitness functions:

$$f_1(\mathbf{x}) = \rho \frac{\pi \cdot x_1^2}{4} x_2, \tag{15.14}$$
$$f_2(\mathbf{x}) = \delta(\mathbf{x}) = \frac{64P \cdot x_2^3}{3E \cdot \pi \cdot x_1^4}, \tag{15.15}$$

where $\rho = 7800\,\text{kg/m}^3$, $P = 1\,\text{kN}$ and $E = 207\,\text{GPa}$. First objectives denotes the weight of cantilaver and second objective - δ - denotes the end deflection of cantilaver.

Constraint functions:

$$g_1(\mathbf{x}) = S_y \geq \frac{32P \cdot x_2}{\pi \cdot x_1^3}, \tag{15.16}$$
$$g_2(\mathbf{x}) = \delta_{\max} \geq \delta(\mathbf{x}), \tag{15.17}$$

where $S_y = 300\,\text{MPa}$ and $\delta_{\max} = 5\,\text{mm}$.

Optimal positions are stored in `position` property of mat-file `MOCantilaver`. Optimal solutions are obtained if $x_2 = \mathbf{200}$ and $x_1 \in \langle 15.14, 50 \rangle$. Solutions with $x_1 \in \langle 10, 15.14 \rangle$ are infeasible.

Optimal fitness values are stored in `fitness` property of mat-file `MOCantilaver`.

15.1.4 MOCones

This is a simple two-objective constrained optimization problem. The optimization algorithm tries to find cone with minimal possible lateral surface area and total surface area, where both objectives are conficting and the constraint function defines minimal volume of the cone. Problem has two decision variables and it is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_1 \in \langle 0, 10 \rangle \text{ cm}, \tag{15.18}$$
$$x_2 \in \langle 0, 20 \rangle \text{ cm}, \tag{15.19}$$

where x_1 denotes radius of base of cone and x_2 denotes height of cone.

Fitness functions:

$$f_1(\mathbf{x}) = S = \pi \cdot x_1 \cdot s, \tag{15.20}$$
$$f_2(\mathbf{x}) = T = \pi \cdot x_1^2 + \pi \cdot x_1 \cdot s, \tag{15.21}$$

where f_1 denotes lateral surface area, f_2 denotes total surface area and $s = \sqrt{x_1^2 + x_2^2}$ is the slant length.

Constraint function:

$$g_1(\mathbf{x}) = V_{\min} < \frac{\pi \cdot x_1^2 \cdot x_2}{3}, \tag{15.22}$$

where $V_{\min} = 200\,\text{cm}^3$ is the minimal volume of cone.

It is difficult to explicitly state the optimal positions. Optimal positions are stored in `position` property of mat-file `MOCones` and they were obtained thank to very dense sampling of decision space.

Optimal fitness values are stored in `fitness` property of mat-file `MOCones`.

15.1.5 MODTLZ1

This is the first benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 7$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.23}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = (1 + h(x_{M:D}))\, 0.5 \prod_{i=1}^{M-1} x_i, \tag{15.24}$$

$$f_{m=2:M-1}(\mathbf{x}) = (1 + h(x_{M:D}))\, 0.5 \left(\prod_{i=1}^{M-m} x_i\right)(1 - x_{M-m+1}), \tag{15.25}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D}))\, 0.5\, (1 - x_1), \tag{15.26}$$

$$h(\mathbf{z}) = 100\left\{k + \left[\sum_{i=1}^{k} (z_i - 0.5)^2 - \cos(20\pi(z_i - 0.5))\right]\right\}, \tag{15.27}$$

where $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.6 MODTLZ2

This is the second benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 12$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.28}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = (1 + h(x_{M:D})) \prod_{i=1}^{M-1} \cos(x_i \cdot \pi/2), \tag{15.29}$$

$$f_{m=2:M-1}(\mathbf{x}) = (1 + h(x_{M:D})) \left[\prod_{i=1}^{M-m} \cos(x_i \cdot \pi/2)\right] \sin(x_{M-m+1} \cdot \pi/2), \tag{15.30}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D})) \sin(x_1 \cdot \pi/2), \tag{15.31}$$

$$h(\mathbf{z}) = \sum_{i=1}^{k} (z_i - 0.5)^2, \tag{15.32}$$

where $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.7 MODTLZ3

This is the third benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 12$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.33}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = (1 + h(x_{M:D})) \prod_{i=1}^{M-1} \cos(x_i \cdot \pi/2), \tag{15.34}$$

$$f_{m=2:M-1}(\mathbf{x}) = (1 + h(x_{M:D})) \left[\prod_{i=1}^{M-m} \cos(x_i \cdot \pi/2)\right] \sin(x_{M-m+1} \cdot \pi/2), \tag{15.35}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D})) \sin(x_1 \cdot \pi/2), \tag{15.36}$$

$$h(\mathbf{z}) = 100 \left\{k + \left[\sum_{i=1}^{k} (z_i - 0.5)^2 - \cos(20\pi(z_i - 0.5))\right]\right\}, \tag{15.37}$$

where $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.8 MODTLZ4

This is the fourth benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 12$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.38}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = (1 + h(x_{M:D})) \prod_{i=1}^{M-1} \cos(x_i^{\alpha} \cdot \pi/2), \tag{15.39}$$

$$f_{m=2:M-1}(\mathbf{x}) = (1 + h(x_{M:D})) \left[\prod_{i=1}^{M-m} \cos(x_i^{\alpha} \cdot \pi/2)\right] \sin\left(x_{M-m+1}^{\alpha} \cdot \pi/2\right), \tag{15.40}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D})) \sin(x_1^{\alpha} \cdot \pi/2), \tag{15.41}$$

$$h(\mathbf{z}) = \sum_{i=1}^{k} (z_i - 0.5)^2, \tag{15.42}$$

where parameter $\alpha = 100$, $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.9 MODTLZ5

This is the fifth benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 12$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.43}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = (1 + h(x_{M:D})) \prod_{i=1}^{M-1} \cos(\theta_i \cdot \pi/2), \tag{15.44}$$

$$f_{m=2:M-1}(\mathbf{x}) = (1 + h(x_{M:D})) \left[\prod_{i=1}^{M-m} \cos(\theta_i \cdot \pi/2)\right] \sin(\theta_{M-m+1} \cdot \pi/2), \tag{15.45}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D})) \sin(\theta_1 \cdot \pi/2), \tag{15.46}$$

$$h(\mathbf{z}) = \sum_{i=1}^{k} (z_i - 0.5)^2, \tag{15.47}$$

$$\theta_i = \begin{cases} x_i & \text{for } i = 1, \\ \frac{1+2h \cdot x_i}{2(1+h)} & \text{for } i = 2,\ 3,\ \ldots (M-1), \end{cases} \tag{15.48}$$

where $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.10 MODTLZ6

This is the sixth benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 12$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.49}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = (1 + h(x_{M:D})) \prod_{i=1}^{M-1} \cos(\theta_i \cdot \pi/2), \tag{15.50}$$

$$f_{m=2:M-1}(\mathbf{x}) = (1 + h(x_{M:D})) \left[\prod_{i=1}^{M-m} \cos(\theta_i \cdot \pi/2)\right] \sin(\theta_{M-m+1} \cdot \pi/2), \tag{15.51}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D})) \sin(\theta_1 \cdot \pi/2), \tag{15.52}$$

$$h(\mathbf{z}) = \sum_{i=1}^{k} z_i^{0.1}, \tag{15.53}$$

$$\theta_i = \begin{cases} x_i & \text{for } i = 1, \\ \frac{1+2h \cdot x_i}{2(1+h)} & \text{for } i = 2,\ 3,\ \ldots (M-1), \end{cases} \tag{15.54}$$

where $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.11 MODTLZ7

This is the seventh benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in both - decision and objective - spaces. The default number of decision variables is $D = 22$ and the default number of objectives is $M = 3$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.55}$$

where D means the number of decision variables and it is arbitrary. Fitness functions:

$$f_{m=1:M-1}(\mathbf{x}) = x_m, \tag{15.56}$$

$$f_M(\mathbf{x}) = (1 + h(x_{M:D})) \left\{ M - \left[\sum_{i=1}^{M-1} \frac{f_i}{1 + h(x_{M:D})} (1 + \sin(3\pi \cdot f_i)) \right] \right\}, \tag{15.57}$$

$$h(\mathbf{z}) = 1 + \frac{9}{k} \sum_{i=1}^{k} z_i, \tag{15.58}$$

where $k = D - M + 1$ and M is the number of objectives.

Optimal solutions corresponds to $x_1 \in \langle 0, 1 \rangle$, $x_2 \in \langle 0, 1 \rangle$ and $z = \mathbf{0}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.12 MOFON

Fonseca and Fleming's test problem is taken from [2]. It is a problem with three decision variables and two objectives. Problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -4, 4 \rangle \text{ for } i = 1, 2, 3. \tag{15.59}$$

Fitness functions:

$$f_1(\mathbf{x}) = 1 - \exp\left[-\sum_{i=1}^{3} \left(x_i - \frac{1}{\sqrt{3}} \right)^2 \right], \tag{15.60}$$

$$f_2(\mathbf{x}) = 1 - \exp\left[-\sum_{i=1}^{3} \left(x_i + \frac{1}{\sqrt{3}} \right)^2 \right]. \tag{15.61}$$

Optimal solutions are stored in `position` property of mat-file `MOFON` and they are located on diagonal line defined by:

$$x_1 = x_2 = x_3 \in \left\langle -\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}} \right\rangle. \tag{15.62}$$

Optimal fitness values are stored in `fitness` property of mat-file `MOFON`.

15.1.13 MOKUR

Kursawe's test problem is taken from [2]. It is a problem with three decision variables and two objectives. Problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -5, 5 \rangle \text{ for } i = 1, 2, 3. \tag{15.63}$$

Fitness functions:

$$f_1(\mathbf{x}) = \left[\sum_{i=1}^{2} -10 \cdot \exp\left(-0.2 \cdot \sqrt{x_i^2 + x_{i+1}^2} \right) \right], \tag{15.64}$$

$$f_2(\mathbf{x}) = \left[\sum_{i=1}^{3} |x_i|^{0.8} + 5 \cdot \sin\left(x_i^3 \right) \right]. \tag{15.65}$$

It is difficult to explicitly state the optimal positions. More information can be found in [2]. Optimal positions are stored in `position` property of mat-file `MOKUR` and they were obtained thanks to very dense sampling of decision space.

Optimal fitness values are stored in `fitness` property of mat-file `MOKUR`.

15.1.14 MOLZ1

This is the first benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 10$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.66}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h(j, x_1)\right]^2, \tag{15.67}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h(j, x_1)\right]^2, \tag{15.68}$$

$$h(j) = x_1^{0.5\left(1+\frac{3j-6}{D-2}\right)} \text{ for } j = 1, 2, \ldots D, \tag{15.69}$$

$$J_1 = 3, 5, \ldots D, \tag{15.70}$$

$$J_2 = 2, 4, \ldots D. \tag{15.71}$$

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.15 MOLZ2

This is the second benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.72}$$

$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.73}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h(j, x_1)\right]^2, \tag{15.74}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h(j, x_1)\right]^2, \tag{15.75}$$

$$h(j) = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \text{ for } j = 1, 2, \ldots D, \tag{15.76}$$

$$J_1 = 3, 5, \ldots D, \tag{15.77}$$

$$J_2 = 2, 4, \ldots D. \tag{15.78}$$

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.16 MOLZ3

This is the third benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.79}$$

$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.80}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h_1(j, x_1)\right]^2, \tag{15.81}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h_2(j, x_1)\right]^2, \tag{15.82}$$

$$h_1(j) = 0.8 x_1 \cos\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.83}$$

$$h_2(j) = 0.8 x_1 \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.84}$$

$$J_1 = 3, 5, \ldots D, \tag{15.85}$$

$$J_2 = 2, 4, \ldots D. \tag{15.86}$$

Optimal solutions corresponds to:

$$x_j = \begin{cases} h_1(j) & \text{where } j \in J_1, \\ h_2(j) & \text{where } j \in J_2. \end{cases} \tag{15.87}$$

Optimal positions and fitness values are calculated during the problem initialization.

15.1.17 MOLZ4

This is the fourth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.88}$$

$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.89}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h_1(j, x_1)\right]^2, \tag{15.90}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h_2(j, x_1)\right]^2, \tag{15.91}$$

$$h_1(j) = 0.8 x_1 \cos\left(\frac{6\pi x_1 + \frac{j\pi}{D}}{3}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.92}$$

$$h_2(j) = 0.8 x_1 \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.93}$$

$$J_1 = 3, 5, \ldots D, \tag{15.94}$$

$$J_2 = 2, 4, \ldots D. \tag{15.95}$$

Optimal solutions corresponds to:

$$x_j = \begin{cases} h_1(j) & \text{for } j \in J_1, \\ h_2(j) & \text{for } j \in J_2. \end{cases} \tag{15.96}$$

Optimal positions and fitness values are calculated during the problem initialization.

15.1.18 MOLZ5

This is the fifth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.97}$$

$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \dots D, \tag{15.98}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - h_1(j, x_1)\right]^2, \tag{15.99}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - h_2(j, x_1)\right]^2, \tag{15.100}$$

$$h_1(j) = \left[0.3x_1^2\cos\left(24\pi x_1 + \frac{4j\pi}{D}\right) + 0.6x_1\right]\cos\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \dots D, \tag{15.101}$$

$$h_2(j) = \left[0.3x_1^2\cos\left(24\pi x_1 + \frac{4j\pi}{D}\right) + 0.6x_1\right]\sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \dots D, \tag{15.102}$$

$$J_1 = 3, 5, \dots D, \tag{15.103}$$

$$J_2 = 2, 4, \dots D. \tag{15.104}$$

Optimal solutions corresponds to:

$$x_j = \begin{cases} h_1(j) & \text{for } j \in J_1, \\ h_2(j) & \text{for } j \in J_2. \end{cases} \tag{15.105}$$

Optimal positions and fitness values are calculated during the problem initialization.

15.1.19 MOLZ6

This is the sixth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 10$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.106}$$

$$x_2 \in \langle 0, 1\rangle, \tag{15.107}$$

$$x_i \in \langle -2, 2\rangle \text{ for } i = 3, 4, \dots D, \tag{15.108}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - h(j, x_1, x_2)\right]^2, \tag{15.109}$$

$$f_2(\mathbf{x}) = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - h(j, x_1, x_2)\right]^2, \tag{15.110}$$

$$f_3(\mathbf{x}) = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left[x_j - h(j, x_1, x_2)\right]^2, \tag{15.111}$$

$$h(j) = 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{D}\right), \quad \text{for } j = 1, 2, \dots D, \tag{15.112}$$

$$J_1 = 4, 7, \dots D, \tag{15.113}$$

$$J_2 = 5, 8, \dots D, \tag{15.114}$$

$$J_3 = 3, 6, \dots D. \tag{15.115}$$

Optimal solutions corresponds to $x_j = h(j)$, where $j = 3, 4, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.20 MOLZ7

This is the seventh benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 10$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.116}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[4y_j^2 - \cos(8y_j\pi) + 1\right], \tag{15.117}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[4y_j^2 - \cos(8y_j\pi) + 1\right], \tag{15.118}$$

$$y_j = x_j - h(j) \quad \text{for } j = 2, 3, \ldots D, \tag{15.119}$$

$$h(j) = x_1^{0.5\left(1+\frac{3j-6}{D-2}\right)} \quad \text{for } j = 1, 2, \ldots D, \tag{15.120}$$

$$J_1 = 3, 5, \ldots D, \tag{15.121}$$

$$J_2 = 2, 4, \ldots D. \tag{15.122}$$

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.21 MOLZ8

This is the eighth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 10$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.123}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \left[4 \sum_{j \in J_1} y_j^2 - 2 \prod_{j \in J_1} \cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right], \tag{15.124}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \left[4 \sum_{j \in J_2} y_j^2 - 2 \prod_{j \in J_2} \cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right], \tag{15.125}$$

$$y_j = x_j - h(j) \quad \text{for } j = 2, 3, \ldots D, \tag{15.126}$$

$$h(j) = x_1^{0.5\left(1+\frac{3j-6}{D-2}\right)} \quad \text{for } j = 1, 2, \ldots D, \tag{15.127}$$

$$J_1 = 3, 5, \ldots D, \tag{15.128}$$

$$J_2 = 2, 4, \ldots D. \tag{15.129}$$

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.22 MOLZ9

This is the ninth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1\rangle \,, \tag{15.130}$$
$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \dots D, \tag{15.131}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - h(j, x_1)\right]^2 , \tag{15.132}$$
$$f_2(\mathbf{x}) = 1 - x_1^2 + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - h(j, x_1)\right]^2 , \tag{15.133}$$
$$h(j) = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \text{ for } j = 1, 2, \dots D, \tag{15.134}$$
$$J_1 = 3, 5, \dots D, \tag{15.135}$$
$$J_2 = 2, 4, \dots D. \tag{15.136}$$

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.23 MOPOL

Poloni's test problem is taken from [2]. It is a problem with two decision variables and two objectives. Problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -\pi, \pi\rangle \text{ for } i = 1, 2. \tag{15.137}$$

Fitness functions:

$$f_1(\mathbf{x}) = \left[1 + (A_1 - B_1(\mathbf{x}))^2 + (A_2 - B_2(\mathbf{x}))^2\right], \tag{15.138}$$
$$f_2(\mathbf{x}) = \left[(x_1 + 3)^2 + (x_2 + 1)^2\right], \tag{15.139}$$

where:

$$A_1 = 0.5\sin(1) - 2\cos(1) + \sin(2) - 1.5\cos(2)\,, \tag{15.140}$$
$$A_2 = 1.5\sin(1) - \cos(1) + 2\sin(2) - 0.5\cos(2)\,, \tag{15.141}$$
$$B_1(\mathbf{x}) = 0.5\sin(x_1) - 2\cos(x_1) + \sin(x_2) - 1.5\cos(x_2)\,, \tag{15.142}$$
$$B_2(\mathbf{x}) = 1.5\sin(x_1) - \cos(x_1) + 2\sin(x_2) - 0.5\cos(x_2)\,. \tag{15.143}$$

It is difficult to explicitly state the optimal positions. More information can be found in [2]. Optimal positions are stored in `position` property of mat-file `MOPOL` and they were obtained thanks to very dense sampling of decision space.

Optimal fitness values are stored in `fitness` property of mat-file `MOPOL`.

15.1.24 MOSCH

Schaffer's test problem is taken from [2]. It is a two-objective problem with only one decision variable. It is possible to redefine limits of this problem by problem input argument (see Subsection 6.1.1, specifically Listing 6.6).

Limits:

$$x_1 \in \langle -A, A \rangle\,, \tag{15.144}$$

where $A \in \langle 2, \infty \rangle$ defines the difficulty of the problem. The higher the A value is the more complicated the problem is.

Fitness functions:

$$f_1(\mathbf{x}) = x^2, \tag{15.145}$$

$$f_2(\mathbf{x}) = (x-2)^2\,, \tag{15.146}$$

Optimal solutions are stored in `position` property of mat-file `MOSCH` and they corresponds to $x_1 \in \langle 0, 2 \rangle$.

Optimal fitness values are stored in `fitness` property of mat-file `MOSCH`.

15.1.25 MOUF4

This is the fourth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1 \rangle\,, \tag{15.147}$$

$$x_i \in \langle -2, 2 \rangle \text{ for } i = 2, 3, \dots D, \tag{15.148}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} h(y_j)\,, \tag{15.149}$$

$$f_2(\mathbf{x}) = 1 - x_1^2 + \frac{2}{|J_2|} \sum_{j \in J_2} h(y_j)\,, \tag{15.150}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \text{ for } j = 1, 2, \dots D, \tag{15.151}$$

$$h(y) = \frac{|y|}{1 + \exp(2|y|)}, \tag{15.152}$$

$$J_1 = 3, 5, \dots D, \tag{15.153}$$

$$J_2 = 2, 4, \dots D. \tag{15.154}$$

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.26 MOUF5

This is the fifth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1 \rangle\,, \tag{15.155}$$

$$x_i \in \langle -1, 1 \rangle \text{ for } i = 2, 3, \dots D, \tag{15.156}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \left(\frac{1}{2N+\epsilon}\right)\left|\sin(2N\pi x_1)\right| + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j), \tag{15.157}$$

$$f_2(\mathbf{x}) = 1 - x_1 + \left(\frac{1}{2N+\epsilon}\right)\left|\sin(2N\pi x_1)\right| + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j), \tag{15.158}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.159}$$

$$h(y) = 2y^2 - \cos(4\pi y) + 1, \tag{15.160}$$

$$J_1 = 3, 5, \ldots D, \tag{15.161}$$

$$J_2 = 2, 4, \ldots D. \tag{15.162}$$

Variable N is an integer and it defines the number of the solutions of the true Pareto-front. The variable was set to $N = 10$, therefore the true Pareto-front has $2N + 1 = 21$ solutions. Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.27 MOUF6

This is the sixth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.163}$$

$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.164}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$\begin{aligned} f_1(\mathbf{x}) = x_1 + & \max\left\{0, 2\left(\frac{1}{2N} + \epsilon\right)\sin(2N\pi x_1)\right\} + \\ & \frac{2}{|J_1|}\left[4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right], \end{aligned} \tag{15.165}$$

$$\begin{aligned} f_2(\mathbf{x}) = 1 - x_1 + & \max\left\{0, 2\left(\frac{1}{2N} + \epsilon\right)\sin(2N\pi x_1)\right\} + \\ & \frac{2}{|J_2|}\left[4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right], \end{aligned} \tag{15.166}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.167}$$

$$J_1 = 3, 5, \ldots D, \tag{15.168}$$

$$J_2 = 2, 4, \ldots D. \tag{15.169}$$

Variable N is an integer and it defines the number of disconnected parts of the true Pareto-front. The variable was set to $N = 2$, therefore the true Pareto-front has $N + 1 = 3$ disconnected parts. Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.28 MOUF7

This is the seventh benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.170}$$
$$x_i \in \langle -1, 1 \rangle \text{ for } i = 2, 3, \ldots D, \tag{15.171}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = \sqrt[5]{x_1} + \frac{2}{|J_1|} \sum_{j \in J_1} y_j^2, \tag{15.172}$$
$$f_2(\mathbf{x}) = 1 - \sqrt[5]{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} y_j^2, \tag{15.173}$$
$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \text{ for } j = 1, 2, \ldots D, \tag{15.174}$$
$$J_1 = 3, 5, \ldots D, \tag{15.175}$$
$$J_2 = 2, 4, \ldots D. \tag{15.176}$$

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.29 MOUF9

This is the ninth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.177}$$
$$x_2 \in \langle 0, 1 \rangle, \tag{15.178}$$
$$x_i \in \langle -2, 2 \rangle \text{ for } i = 3, 4, \ldots D, \tag{15.179}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = 0.5\left[\max\left\{0, (1+\epsilon)\left(1 - 4(2x_1 - 1)^2\right)\right\} + 2x_1\right] x_2 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right)\right]^2, \tag{15.180}$$
$$f_2(\mathbf{x}) = 0.5\left[\max\left\{0, (1+\epsilon)\left(1 - 4(2x_1 - 1)^2\right)\right\} - 2x_1 + 2\right] x_2 + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right)\right]^2, \tag{15.181}$$
$$f_3(\mathbf{x}) = 1 - x_2 + \frac{2}{|J_3|} \sum_{j \in J_3} \left[x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right)\right]^2, \tag{15.182}$$

$$J_1 = 4, 7, \ldots D, \tag{15.183}$$
$$J_2 = 5, 8, \ldots D, \tag{15.184}$$
$$J_3 = 3, 6, \ldots D. \tag{15.185}$$

Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions are divided into two disconected parts where $x_1 \in [0, 0.25] \cup [0.75, 1]$, $0 \leq x_2 \leq 1$, and $x_j = 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 3, 4, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.30 MOUF10

This is the tenth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.186}$$
$$x_2 \in \langle 0, 1 \rangle, \tag{15.187}$$
$$x_i \in \langle -2, 2 \rangle \text{ for } i = 3, 4, \ldots D, \tag{15.188}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = \cos(0.5x_1\pi)\cos(0.5x_2\pi)\frac{2}{|J_1|}\sum_{j \in J_1}\left[4y_j^2 - cos(8\pi y_j) + 1\right], \tag{15.189}$$
$$f_2(\mathbf{x}) = \cos(0.5x_1\pi)\sin(0.5x_2\pi)\frac{2}{|J_2|}\sum_{j \in J_2}\left[4y_j^2 - cos(8\pi y_j) + 1\right], \tag{15.190}$$
$$f_3(\mathbf{x}) = \sin(0.5x_1\pi)\frac{2}{|J_3|}\sum_{j \in J_3}\left[4y_j^2 - cos(8\pi y_j) + 1\right], \tag{15.191}$$

$$y_j = x_j - 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right) \text{ for } j = 1, 2, \ldots D, \tag{15.192}$$
$$J_1 = 4, 7, \ldots D, \tag{15.193}$$
$$J_2 = 5, 8, \ldots D, \tag{15.194}$$
$$J_3 = 3, 6, \ldots D. \tag{15.195}$$

Optimal solutions corresponds to $x_j = 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 3, 4, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.31 MOWFG1

This is the first benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.196}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} \left[1 - \cos\left(\frac{\pi}{2} y_i\right)\right],^{*\text{s:convex}} \tag{15.197}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} \left[1 - \cos\left(\frac{\pi}{2} y_i\right)\right]\left[1 - \sin\left(\frac{\pi}{2} y_{M-1}\right)\right],^{*\text{s:convex}} \tag{15.198}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} \left[1 - \cos\left(\frac{\pi}{2} y_i\right)\right]\left[1 - \sin\left(\frac{\pi}{2} y_{M-1}\right)\right],^{*\text{s:convex}} \tag{15.199}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \left[1 - \cos\left(\frac{\pi}{2} y_1\right)\right]\left[1 - \sin\left(\frac{\pi}{2} y_{M-1}\right)\right],^{*\text{s:convex}} \tag{15.200}$$

$$f_M(\mathbf{x}) = y_M + 2M \left[1 - y_1 - \frac{\cos(2A\pi y_1 + \pi/2)}{2A\pi}\right]^B,^{*\text{s:mixed}} \tag{15.201}$$

where:

$$z_i = \frac{x_i}{2i} \quad \text{for } i = 1, 2, \ldots D,^{*\text{norm}} \tag{15.202}$$

$$t_{1,i} = \begin{cases} z_i & \text{if } i = 1, 2, \ldots, K, \\ \frac{|z_i - A|}{|\lfloor A - z_i \rfloor| + A}, & \text{if } i = K+1, \ldots, D,^{*\text{t:s:linear}} \end{cases} \tag{15.203}$$

$$t_{2,i} = \begin{cases} t_{1,i} & \text{if } i = 1, 2, \ldots, K, \\ A + \frac{A(B - t_{1,i}) \min\{0, \lfloor t_{1,i} - B \rfloor\}}{B} - \\ \frac{(1-A)(t_{1,i} - C) \min\{0, \lfloor C - t_{1,i} \rfloor\}}{1-C}, & \text{if } i = K+1, \ldots, D,^{*\text{t:b:flat}} \end{cases} \tag{15.204}$$

$$t_{3,i} = t_{2,i}^{A},^{*\text{t:b:poly}} \tag{15.205}$$

$$t_{4,i} = \begin{cases} h(t_{3,J_1}, 2J_1), & \text{if } i = 1, \ldots, M-1, \\ h(t_{3,J_2}, 2J_2), & \text{if } i = M, \end{cases} \tag{15.206}$$

$$h(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{V}|} w_i v_i \Big/ \sum_{i=1}^{|\mathbf{V}|} w_i,^{*\text{t:r:sum}} \tag{15.207}$$

$$J_1 = \frac{(i-1)K}{M-1} + 1, \ldots, \frac{iK}{M-1}, \tag{15.208}$$

$$J_2 = K + 1, \ldots, D, \tag{15.209}$$

$$y_i = \begin{cases} (t_{4,i} - 0.5)\max(1, t_{4,M}) + 0.5, & \text{if } i = 1, \ldots, M-1,^{*\text{p:param}} \\ t_{4,M}, & \text{if } i = M. \end{cases} \tag{15.210}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:convex**: Convex shape function.
- **s:mixed**: Mixed shape function. Input arguments: $A = 5$, $B = 1$.
- **norm**: Normalizing function.
- **t:s:linear**: Shift linear transition function. Input arguments: $A = 0.35$.
- **t:b:flat**: Bias flat transition function. Input arguments: $A = 0.8$, $B = 0.75$, $C = 0.85$.
- **t:b:poly**: Bias polynomial transition function. Input arguments: $A = 0.02$.
- **t:r:sum**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \ldots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.32 MOWFG2

This is the second benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.211}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} \left[1 - \cos\left(\frac{\pi}{2} y_i\right)\right],^{*\text{s:convex}} \tag{15.212}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} \left[1 - \cos\left(\frac{\pi}{2} y_i\right)\right]\left[1 - \sin\left(\frac{\pi}{2} y_{M-1}\right)\right],^{*\text{s:convex}} \tag{15.213}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} \left[1 - \cos\left(\frac{\pi}{2} y_i\right)\right]\left[1 - \sin\left(\frac{\pi}{2} y_{M-1}\right)\right],^{*\text{s:convex}} \tag{15.214}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \left[1 - \cos\left(\frac{\pi}{2} y_1\right)\right]\left[1 - \sin\left(\frac{\pi}{2} y_{M-1}\right)\right],^{*\text{s:convex}} \tag{15.215}$$

$$f_M(\mathbf{x}) = y_M + 2M \left[1 - y_1^B \cos^2\left(A\pi y_1^C\right)\right],^{*\text{s:disconnect}} \tag{15.216}$$

where:

$$z_i = \frac{x_i}{2i} \text{ for } i = 1, 2, \ldots D,^{*\text{norm}} \tag{15.217}$$

$$t_{1,i} = \begin{cases} z_i & \text{if } i = 1, 2, \ldots, K, \\ \frac{|z_i - A|}{|\lfloor A - z_i \rfloor| + A}, & \text{if } i = K+1, \ldots, D,^{*\text{t:s:linear}} \end{cases} \tag{15.218}$$

$$t_{2,i} = \begin{cases} t_{1,i} & \text{if } i = 1, 2, \ldots, K, \\ h_1\left(\left\{t_{1,K+2(i-K)-1}, t_{1,K+2(i-K)}\right\}, 2\right) & \text{if } i = K+1, \ldots, (D+K)/2, \end{cases} \tag{15.219}$$

$$t_{3,i} = \begin{cases} h_2\left(t_{3,J_1}, \{1, \ldots, 1\}\right), & \text{if } i = 1, \ldots, M-1, \\ h_2\left(t_{3,J_2}, \{1, \ldots, 1\}\right), & \text{if } i = M, \end{cases} \tag{15.220}$$

$$h_1(\mathbf{a}, A) = \frac{\sum_{j=1}^{|\mathbf{a}|}\left[a_j + \sum_{k=0}^{A-2}\left|a_j - a_{1+(j+k) \mod |\mathbf{a}|}\right|\right]}{\frac{|\mathbf{a}|}{A}\lceil A/2\rceil\left(1 + 2A - 2\lceil A/2\rceil\right)},^{*\text{t:r:nonsep}} \tag{15.221}$$

$$h_2(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{V}|} w_i v_i / \sum_{i=1}^{|\mathbf{V}|} w_i,^{*\text{t:r:sum}} \tag{15.222}$$

$$J_1 = \frac{(i-1)K}{M-1} + 1, \ldots, \frac{iK}{M-1}, \tag{15.223}$$

$$J_2 = K + 1, \ldots, (D+K)/2, \tag{15.224}$$

$$y_i = \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \ldots, M-1,^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} \tag{15.225}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:convex**: Convex shape function.
- **s:disconnect**: Disconnected shape function. Input arguments: $A = 5$, $B = 1$, $C = 1$.
- **norm**: Normalizing function.
- **t:s:linear**: Shift linear transition function. Input arguments: $A = 0.35$.
- **t:r:nonsep**: Reduction non-separable transition function. Input arguments: $A = 2$.
- **t:r:sum**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k + 1, k + 2, \dots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.33 MOWFG3

This is the third benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \;\text{ for } i = 1, 2, \dots D, \tag{15.226}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} [y_i]\,,^{*\text{s:linear}} \tag{15.227}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} [y_i] \;\; (1 - y_{M-1})^{*\text{s:linear}} \tag{15.228}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} [y_i] \;\; (1 - y_{M-2})\,,^{*\text{s:linear}} \tag{15.229}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \; [y_1] \;\; (1 - y_2)\,,^{*\text{s:linear}} \tag{15.230}$$

$$f_M(\mathbf{x}) = y_M + 2M \;\; (1 - y_1)\,,^{*\text{s:linear}} \tag{15.231}$$

where:

$$z_i = \frac{x_i}{2i} \quad \text{for } i = 1, 2, \dots D,^{*\text{norm}} \tag{15.232}$$

$$t_{1,i} = \begin{cases} z_i & \text{if } i = 1, 2, \dots, K, \\ \frac{|z_i - A|}{|\lfloor A - z_i \rfloor| + A}, & \text{if } i = K+1, \dots, D,^{*\text{t:s:linear}} \end{cases} \tag{15.233}$$

$$t_{2,i} = \begin{cases} t_{1,i} & \text{if } i = 1, 2, \dots, K, \\ \\ h_1\left(\left\{t_{1,K+2(i-K)-1}, t_{1,K+2(i-K)}\right\}, 2\right) & \text{if } i = K+1, \dots, (D+K)/2, \end{cases} \tag{15.234}$$

$$t_{3,i} = \begin{cases} h_2\left(t_{3,J_1}, \{1, \dots, 1\}\right), & \text{if } i = 1, \dots, M-1, \\ \\ h_2\left(t_{3,J_2}, \{1, \dots, 1\}\right), & \text{if } i = M, \end{cases} \tag{15.235}$$

$$h_1(\mathbf{a}, A) = \frac{\sum_{j=1}^{|\mathbf{a}|}\left[a_j + \sum_{k=0}^{A-2}\left|a_j - a_{1+(j+k) \mod |\mathbf{a}|}\right|\right]}{\frac{|\mathbf{a}|}{A}\lceil A/2 \rceil (1 + 2A - 2\lceil A/2 \rceil)},^{*\text{t:r:nonsep}} \tag{15.236}$$

$$h_2(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{V}|} w_i v_i / \sum_{i=1}^{|\mathbf{V}|} w_i,^{*\text{t:r:sum}} \tag{15.237}$$

$$J_1 = \frac{(i-1)K}{M-1} + 1, \dots, \frac{iK}{M-1}, \tag{15.238}$$

$$J_2 = K+1, \dots, (D+K)/2, \tag{15.239}$$

$$y_i = \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \dots, M-1,^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} \tag{15.240}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:linear**: Linear shape function.
- **norm**: Normalizing function.
- **t:s:linear**: Shift linear transition function. Input arguments: $A = 0.35$.
- **t:r:nonsep**: Reduction non-separable transition function. Input arguments: $A = 2$.
- **t:r:sum**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \dots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.34 MOWFG4

This is the fourth benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \quad \text{for } i = 1, 2, \dots D, \tag{15.241}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} [\sin(0.5\pi y_i)],^{*\text{s:concave}} \tag{15.242}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} [\sin(0.5\pi y_i)] \; [\cos(0.5\pi y_{M-1})]^{*\text{s:concave}} \tag{15.243}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} [\sin(0.5\pi y_i)] \; [\cos(0.5\pi y_{M-2})],^{*\text{s:concave}} \tag{15.244}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \; [\sin(0.5\pi y_1)] \; [\cos(0.5\pi y_2)],^{*\text{s:concave}} \tag{15.245}$$

$$f_M(\mathbf{x}) = y_M + 2M \; [\cos(0.5\pi y_1)],^{*\text{s:concave}} \tag{15.246}$$

where:

$$z_i = \frac{x_i}{2i} \quad \text{for } i = 1, 2, \ldots D,^{*\text{norm}} \tag{15.247}$$

$$t_{1,i} = \frac{1 + \cos\left[(4A+2)\pi\left(0.5 - \frac{|z_i - C|}{2(\lfloor C - z_i \rfloor + C)}\right)\right] + 4B\left(\frac{|z_i - C|}{2(\lfloor C - z_i \rfloor + C)}\right)^2}{B+2} \quad \text{for } i = 1, \ldots D,^{*\text{t:s:multi}} \tag{15.248}$$

$$t_{2,i} = \begin{cases} h(t_{1,J_1}, \{1, \ldots, 1\}), & \text{if } i = 1, \ldots, M-1, \\ h(t_{1,J_2}, \{1, \ldots, 1\}), & \text{if } i = M, \end{cases} \tag{15.249}$$

$$h(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{V}|} w_i v_i / \sum_{i=1}^{|\mathbf{V}|} w_i,^{*\text{t:r:sum}} \tag{15.250}$$

$$J_1 = \frac{(i-1)K}{M-1} + 1, \ldots, \frac{iK}{M-1}, \tag{15.251}$$

$$J_2 = K + 1, \ldots, D, \tag{15.252}$$

$$y_i = \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \ldots, M-1,^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} \tag{15.253}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:concave**: Concave shape function.
- **norm**: Normalizing function.
- **t:s:multi**: Shift multi-modal transition function. Input arguments: $A = 30$, $B = 10$, $C = 0.35$.
- **t:r:sum**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \ldots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.35 MOWFG5

This is the fifth benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \ \text{ for } i = 1, 2, \dots D, \tag{15.254}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$\begin{aligned}
f_1(\mathbf{x}) =& y_M + 2 \prod_{i=1}^{M-1} [\sin(0.5\pi y_i)], ^{*\text{s:concave}} & (15.255)\\
f_2(\mathbf{x}) =& y_M + 4 \prod_{i=1}^{M-2} [\sin(0.5\pi y_i)] \ [\cos(0.5\pi y_{M-1})]^{*\text{s:concave}} & (15.256)\\
f_3(\mathbf{x}) =& y_M + 6 \prod_{i=1}^{M-3} [\sin(0.5\pi y_i)] \ [\cos(0.5\pi y_{M-2})], ^{*\text{s:concave}} & (15.257)\\
f_{M-1}(\mathbf{x}) =& y_M + 2(M-1) \ [\sin(0.5\pi y_1)] \ [\cos(0.5\pi y_2)], ^{*\text{s:concave}} & (15.258)\\
f_M(\mathbf{x}) =& y_M + 2M \qquad [\cos(0.5\pi y_1)], ^{*\text{s:concave}} & (15.259)
\end{aligned}$$

where:

$$\begin{aligned}
z_i =& \frac{x_i}{2i} \ \text{ for } i = 1, 2, \dots D, ^{*\text{norm}} & (15.260)\\
t_{1,i} =& 1 + (|z_i - A| - B) \times \qquad \text{for } i = 1, \dots D, ^{*\text{t:s:decept}} & (15.261)\\
& \times \left[\frac{\lfloor z_i - A + B \rfloor \left(1 - C + \frac{A-B}{B}\right)}{A - B} + \frac{\lfloor z_i - A + B \rfloor \left(1 - C + \frac{A-B}{B}\right)}{1 - A - B} + \frac{1}{B} \right] &\\
t_{2,i} =& \begin{cases} h(t_{1,J_1}, \{1, \dots, 1\}), & \text{if } i = 1, \dots, M-1, \\ h(t_{1,J_2}, \{1, \dots, 1\}), & \text{if } i = M, \end{cases} & (15.262)\\
h(\mathbf{v}, \mathbf{w}) =& \sum_{i=1}^{|\mathbf{V}|} w_i v_i / \sum_{i=1}^{|\mathbf{V}|} w_i, ^{*\text{t:r:sum}} & (15.263)\\
J_1 =& \frac{(i-1)K}{M-1} + 1, \dots, \frac{iK}{M-1}, & (15.264)\\
J_2 =& K + 1, \dots, D, & (15.265)\\
y_i =& \begin{cases} (t_{3,i} - 0.5) \max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \dots, M-1, ^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} & (15.266)
\end{aligned}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:concave**: Concave shape function.
- **norm**: Normalizing function.
- **t:s:decept**: Shift deceptive transition function. Input arguments: $A = 0.35$, $B = 0.001$, $C = 0.05$.
- **t:r:sum**: Reduction weighted sum transition function.

- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \dots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.36 MOWFG6

This is the sixth benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \text{ for } i = 1, 2, \dots D, \tag{15.267}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$\begin{aligned}
f_1(\mathbf{x}) =& y_M + 2 \prod_{i=1}^{M-1} \left[\sin(0.5\pi y_i)\right], ^{*\text{s:concave}} && (15.268)\\
f_2(\mathbf{x}) =& y_M + 4 \prod_{i=1}^{M-2} \left[\sin(0.5\pi y_i)\right] \left[\cos(0.5\pi y_{M-1})\right]^{*\text{s:concave}} && (15.269)\\
f_3(\mathbf{x}) =& y_M + 6 \prod_{i=1}^{M-3} \left[\sin(0.5\pi y_i)\right] \left[\cos(0.5\pi y_{M-2})\right], ^{*\text{s:concave}} && (15.270)\\
f_{M-1}(\mathbf{x}) =& y_M + 2(M-1) \left[\sin(0.5\pi y_1)\right] \left[\cos(0.5\pi y_2)\right], ^{*\text{s:concave}} && (15.271)\\
f_M(\mathbf{x}) =& y_M + 2M \left[\cos(0.5\pi y_1)\right], ^{*\text{s:concave}} && (15.272)
\end{aligned}$$

where:

$$z_i = \frac{x_i}{2i} \text{ for } i = 1, 2, \dots D, ^{*\text{norm}} \tag{15.273}$$

$$t_{1,i} = \begin{cases} z_i & \text{if } i = 1, 2, \dots, K, \\ \frac{|z_i - A|}{|\lfloor A - z_i \rfloor| + A}, & \text{if } i = K+1, \dots, D, ^{*\text{t:s:linear}} \end{cases} \tag{15.274}$$

$$t_{2,i} = \begin{cases} h\left(\left\{t_{1,(i-1)K/(M-1)+1}, \dots, t_{1,iK/(M-1)}\right\}, K/(M-1)\right) & \text{if } i = 1, \dots, M-1, \\ \\ h\left(\left\{t_{1,K+1}, \dots, t_{1,D}\right\}, D-K\right) & \text{if } i = M, \end{cases} \tag{15.275}$$

$$h(\mathbf{a}, A) = \frac{\sum_{j=1}^{|\mathbf{a}|} \left[a_j + \sum_{k=0}^{A-2} \left|a_j - a_{1+(j+k) \mod |\mathbf{a}|}\right|\right]}{\frac{|\mathbf{a}|}{A} \lceil A/2 \rceil (1 + 2A - 2\lceil A/2 \rceil)}, ^{*\text{t:r:nonsep}} \tag{15.276}$$

$$y_i = \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \dots, M-1, ^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} \tag{15.277}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:concave**: Concave shape function.

- **norm**: Normalizing function.
- **t:s:linear**: Shift linear transition function. Input arguments: $A = 0.35$.
- **t:r:nonsep**: Reduction non-separate transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \dots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.37 MOWFG7

This is the seventh benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \text{ for } i = 1, 2, \dots D, \tag{15.278}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} [\sin(0.5\pi y_i)], ^{*\text{s:concave}} \tag{15.279}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} [\sin(0.5\pi y_i)] \ [\cos(0.5\pi y_{M-1})]^{*\text{s:concave}} \tag{15.280}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} [\sin(0.5\pi y_i)] \ [\cos(0.5\pi y_{M-2})], ^{*\text{s:concave}} \tag{15.281}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \ [\sin(0.5\pi y_1)] \ [\cos(0.5\pi y_2)], ^{*\text{s:concave}} \tag{15.282}$$

$$f_M(\mathbf{x}) = y_M + 2M \ [\cos(0.5\pi y_1)], ^{*\text{s:concave}} \tag{15.283}$$

where:

$$\begin{aligned}
z_i =& \frac{x_i}{2i} \quad \text{for } i = 1, 2, \ldots D,^{*\text{norm}} && (15.284)\\
t_{1,i} =& \begin{cases} A - [1 - 2h([z_{i+1}, \ldots, z_D], [1, \ldots, 1])] & \text{if } i = 1, 2, \ldots, K,^{*\text{t:b:param}} \\ |\lfloor 0.5 - h([z_{i+1}, \ldots, z_D], [1, \ldots, 1]) \rfloor + A| & \\ & \\ z_i, & \text{if } i = K+1, \ldots, D, \end{cases} && (15.285)\\
t_{2,i} =& \begin{cases} t_{1,i} & \text{if } i = 1, 2, \ldots, K, \\ \frac{|t_{1,i} - A|}{|\lfloor A - t_{1,i} \rfloor| + A}, & \text{if } i = K+1, \ldots, D,^{*\text{t:s:linear}} \end{cases} && (15.286)\\
t_{3,i} =& \begin{cases} h(t_{2,J_1}, \{1, \ldots, 1\}), & \text{if } i = 1, \ldots, M-1, \\ h(t_{2,J_2}, \{1, \ldots, 1\}), & \text{if } i = M, \end{cases} && (15.287)\\
h(\mathbf{v}, \mathbf{w}) =& \sum_{i=1}^{|\mathbf{V}|} w_i v_i / \sum_{i=1}^{|\mathbf{V}|} w_i,^{*\text{t:r:sum}} && (15.288)\\
J_1 =& \frac{(i-1)K}{M-1} + 1, \ldots, \frac{iK}{M-1}, && (15.289)\\
J_2 =& K+1, \ldots, D, && (15.290)\\
y_i =& \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \ldots, M-1,^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} && (15.291)
\end{aligned}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:concave**: Concave shape function.
- **norm**: Normalizing function.
- **t:b:param**: Bias parameter dependent transition function. Input arguments: u = t:r:sum, $A = 0.98/49.98$, $B = 0.02$, $C = 50$.
- **t:s:linear**: Shift linear transition function. Input arguments: $A = 0.35$.
- **t:r:sum**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \ldots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.38 MOWFG8

This is the eighth benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.292}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} \left[\sin(0.5\pi y_i)\right], ^{*\text{s:concave}} \tag{15.293}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} \left[\sin(0.5\pi y_i)\right] \left[\cos(0.5\pi y_{M-1})\right]^{*\text{s:concave}} \tag{15.294}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} \left[\sin(0.5\pi y_i)\right] \left[\cos(0.5\pi y_{M-2})\right], ^{*\text{s:concave}} \tag{15.295}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \left[\sin(0.5\pi y_1)\right] \left[\cos(0.5\pi y_2)\right], ^{*\text{s:concave}} \tag{15.296}$$

$$f_M(\mathbf{x}) = y_M + 2M \left[\cos(0.5\pi y_1)\right], ^{*\text{s:concave}} \tag{15.297}$$

where:

$$z_i = \frac{x_i}{2i} \quad \text{for } i = 1, 2, \dots D, ^{*\text{norm}} \tag{15.298}$$

$$t_{1,i} = \begin{cases} z_i, & \text{if } i = 1, 2, \dots, K, \\ A - \left[1 - 2h\left([z_{i+1}, \dots, z_D], [1, \dots, 1]\right)\right] & \\ \left|\lfloor 0.5 - h\left([z_{i+1}, \dots, z_D], [1, \dots, 1]\right)\rfloor + A\right| & \text{if } i = K+1, \dots, D, ^{*\text{t:b:param}} \end{cases} \tag{15.299}$$

$$t_{2,i} = \begin{cases} t_{1,i} & \text{if } i = 1, 2, \dots, K, \\ \frac{|t_{1,i} - A|}{|\lfloor A - t_{1,i} \rfloor| + A}, & \text{if } i = K+1, \dots, D, ^{*\text{t:s:linear}} \end{cases} \tag{15.300}$$

$$t_{3,i} = \begin{cases} h\left(t_{2,J_1}, \{1, \dots, 1\}\right), & \text{if } i = 1, \dots, M-1, \\ h\left(t_{2,J_2}, \{1, \dots, 1\}\right), & \text{if } i = M, \end{cases} \tag{15.301}$$

$$h(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{V}|} w_i v_i / \sum_{i=1}^{|\mathbf{V}|} w_i, ^{*\text{t:r:sum}} \tag{15.302}$$

$$J_1 = \frac{(i-1)K}{M-1} + 1, \dots, \frac{iK}{M-1}, \tag{15.303}$$

$$J_2 = K+1, \dots, D, \tag{15.304}$$

$$y_i = \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \dots, M-1, ^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} \tag{15.305}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:concave**: Concave shape function.
- **norm**: Normalizing function.
- **t:b:param**: Bias parameter dependent transition function. Input arguments: $u =$ t:r:sum, $A = 0.98/49.98$, $B = 0.02$, $C = 50$.
- **t:s:linear**: Shift linear transition function. Input arguments: $A = 0.35$.
- **t:r:sum**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \dots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.39 MOWFG9

This is the ninth benchmark problem from WFG test suite [21]. The problem is scalable in both, decision space and objective space. The default number of decision variables is $D = 10$ and the default number of objective functions is $M = 2$.

Limits:

$$x_i \in \langle 0, 2i \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.306}$$

where D means the number of decision variables and it is arbitrary.

To obtain a fitness function values, the positions are normalized at first. Afterwards, several transition functions are applied on the normalized positions and the position parameters are calculated. Finally, shape functions are applied to position parameters and the fitness values are deduced.

$$f_1(\mathbf{x}) = y_M + 2 \prod_{i=1}^{M-1} \left[\sin(0.5\pi y_i)\right], ^{*\text{s:concave}} \tag{15.307}$$

$$f_2(\mathbf{x}) = y_M + 4 \prod_{i=1}^{M-2} \left[\sin(0.5\pi y_i)\right] \left[\cos(0.5\pi y_{M-1})\right]^{*\text{s:concave}} \tag{15.308}$$

$$f_3(\mathbf{x}) = y_M + 6 \prod_{i=1}^{M-3} \left[\sin(0.5\pi y_i)\right] \left[\cos(0.5\pi y_{M-2})\right], ^{*\text{s:concave}} \tag{15.309}$$

$$f_{M-1}(\mathbf{x}) = y_M + 2(M-1) \left[\sin(0.5\pi y_1)\right] \left[\cos(0.5\pi y_2)\right], ^{*\text{s:concave}} \tag{15.310}$$

$$f_M(\mathbf{x}) = y_M + 2M \left[\cos(0.5\pi y_1)\right], ^{*\text{s:concave}} \tag{15.311}$$

where:

$$z_i = \frac{x_i}{2i} \text{ for } i = 1, 2, \ldots D, ^{*\text{norm}} \tag{15.312}$$

$$t_{1,i} = \begin{cases} A - \left[1 - 2h\left([z_{i+1}, \ldots, z_D], [1, \ldots, 1]\right)\right] & \text{if } i = 1, 2, \ldots, D-1, ^{*\text{t:b:param}} \\ \left|\lfloor 0.5 - h\left([z_{i+1}, \ldots, z_D], [1, \ldots, 1]\right)\rfloor + A\right| & \\ & \\ z_i, & \text{if } i = D, \end{cases} \tag{15.313}$$

$$t_{2,i} = \begin{cases} 1 + \left(|t_{1,i} - A| - B\right) \times & \text{for } i = 1, \ldots K, ^{*\text{t:s:decept}} \\ \times \left[\frac{\lfloor t_{1,i} - A + B \rfloor \left(1 - C + \frac{A-B}{B}\right)}{A-B} + \frac{\lfloor t_{1,i} - A + B \rfloor \left(1 - C + \frac{A-B}{B}\right)}{1-A-B} + \frac{1}{B}\right] & \\ & \\ \frac{1 + \cos\left[(4A+2)\pi\left(0.5 - \frac{|t_{1,i} - C|}{2\left(\lfloor C - t_{1,i} \rfloor + C\right)}\right)\right] + 4B\left(\frac{|t_{1,i} - C|}{2\left(\lfloor C - t_{1,i} \rfloor + C\right)}\right)^2}{B+2} & \text{for } i = 1, \ldots D, ^{*\text{t:s:multi}} \end{cases} \tag{15.314}$$

$$t_{3,i} = \begin{cases} h\left(\left\{t_{2,(i-1)K/(M-1)+1}, \ldots, t_{2,iK/(M-1)}\right\}, K/(M-1)\right) & \text{if } i = 1, \ldots, M-1, \\ & \\ h\left(\{t_{2,K+1}, \ldots, t_{2,D}\}, D-K\right) & \text{if } i = M, \end{cases} \tag{15.315}$$

$$h(\mathbf{a}, A) = \frac{\sum_{j=1}^{|\mathbf{a}|} \left[a_j + \sum_{k=0}^{A-2} \left|a_j - a_{1+(j+k) \mod |\mathbf{a}|}\right|\right]}{\frac{|\mathbf{a}|}{A} \lceil A/2 \rceil \left(1 + 2A - 2\lceil A/2 \rceil\right)}, ^{*\text{t:r:nonsep}} \tag{15.316}$$

$$y_i = \begin{cases} (t_{3,i} - 0.5)\max(1, t_{3,M}) + 0.5, & \text{if } i = 1, \ldots, M-1, ^{*\text{p:param}} \\ t_{3,M}, & \text{if } i = M. \end{cases} \tag{15.317}$$

Equations contains several marks denoting the following shape or transition functions known from [21]:

- **s:concave**: Concave shape function.
- **norm**: Normalizing function.
- **t:b:param**: Bias parameter dependent transition function. Input arguments: $u =$ t:r:sum, $A = 0.98/49.98$, $B = 0.02$, $C = 50$.
- **t:s:decept**: Shift deception transition function. Input arguments: $A = 0.35$, $B = 0.001$, $C = 0.05$.
- **t:s:multi**: Shift multi-modal transition function. Input arguments: $A = 0.35$, $B = 95$, $C = 0.35$.
- **t:r:nonsep**: Reduction weighted sum transition function.
- **p:param**: Calculate position parameters function.

Optimal solutions corresponds to $x_j = 2j0.35$, where $j = k+1, k+2, \ldots D$. Note that first k decision variables are called position parameters and its combination is responsible for the position of a solution of the Pareto-front. Remaining decision variables are called distance parameters and its combination is responsible for the distance of a solution from the true Pareto-front.

Optimal positions and fitness values are calculated during the problem initialization.

15.1.40 MOZDT1

This is the first benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in the decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_i \in \langle 0, 1 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.318}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.319}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}} \right], \tag{15.320}$$

$$h(\mathbf{x}) = 1 + 9 \frac{\sum_{i=2}^{D} x_i}{D-1}. \tag{15.321}$$

Optimal solutions corresponds to $x_1 \in \langle 0, 1 \rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.41 MOZDT2

This is the second benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in the decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_i \in \langle 0, 1 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.322}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.323}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \left(\frac{x_1}{h(\mathbf{x})} \right)^2 \right], \tag{15.324}$$

$$h(\mathbf{x}) = 1 + 9 \frac{\sum_{i=2}^{D} x_i}{D-1}. \tag{15.325}$$

Optimal solutions corresponds to $x_1 \in \langle 0, 1 \rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.42 MOZDT3

This is the third benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in the decision space. The default number of decision variables is $D = 30$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.326}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.327}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}} - \frac{x_1}{h(\mathbf{x})} \sin(10\pi \cdot x_1)\right], \tag{15.328}$$

$$h(\mathbf{x}) = 1 + 9\frac{\sum_{i=2}^{D} x_i}{D-1}. \tag{15.329}$$

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.43 MOZDT4

This is the fourth benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in the decision space. The default number of decision variables is $D = 10$.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.330}$$

$$x_i \in \langle -5, 5\rangle \text{ for } i = 2, 3, \dots D, \tag{15.331}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.332}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}}\right], \tag{15.333}$$

$$h(\mathbf{x}) = 1 + 10(D-1) + \left[\sum_{i=2}^{D} x_i^2 - 10\cos(4\pi \cdot x_i)\right]. \tag{15.334}$$

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.1.44 MOZDT6

This is the sixth benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in the decision space. The default number of decision variables is $D = 10$.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.335}$$

where D means the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = 1 - \exp(-4x_1)\sin^6(6\pi \cdot x_1), \tag{15.336}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \left(\frac{f_1(\mathbf{x})}{h(\mathbf{x})}\right)^2\right], \tag{15.337}$$

$$h(\mathbf{x}) = 1 + 9\left[\frac{\sum_{i=2}^{D} x_i}{D-1}\right]^{0.25}. \tag{15.338}$$

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.2 Single-objective

15.2.1 SOACK

Ackley's function is a single-objective problem [22]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -15, 30 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.339}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = -20 \exp\left[-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right] - \exp\left[\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi \cdot x_i)\right] + 20 + \exp(1). \tag{15.340}$$

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.2 SOALP

Alpine function is a single-objective problem [23]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.341}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = \sum_{i=1}^{D} |x_i \sin(x_i) + 0.1 x_i|. \tag{15.342}$$

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.3 SOBEA

Beale's function is a single-objective problem with two decision variables [24]. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -4.5, 4.5 \rangle \text{ for } i = 1, 2. \tag{15.343}$$

Fitness function:

$$f(\mathrm{x}) = (1.5 - x_1 + x_1 x_2)^2 + \left(2.25 - x_1 + x_1 x_2^2\right)^2 + \left(2.625 - x_1 + x_1 x_2^3\right)^2. \tag{15.344}$$

The optimal position is $\mathbf{x} = [3,\ 0.5]$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.4 SOBOH

Bohachevsky's function is a single-objective problem with two decision variables [25]. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -100, 100 \rangle \text{ for } i = 1, 2. \tag{15.345}$$

Fitness function:

$$f(\mathrm{x}) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi \cdot x_1) - 0.4\cos(4\pi \cdot x_2) + 0.7. \tag{15.346}$$

The optimal position is $x_i = 0$ for $i = 1, 2$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.5 SOBOO

Booth's function is a single-objective problem with two decision variables [25]. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2. \tag{15.347}$$

Fitness function:

$$f(\mathrm{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2. \tag{15.348}$$

The optimal position is $\mathbf{x} = [1,\ 3]$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.6 SOCan

This is a simple single-objective constrained optimization problem with two decision variables [2]. It delineates the design of a can where the objective function denotes the surface area of the can, which should be minimalized, while the constraint function limits the minimal volume of the can. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_1 \in \langle 0, 30 \rangle \text{ cm - is a radius of can,} \tag{15.349}$$

$$x_2 \in \langle 0, 30 \rangle \text{ cm - is a height of can.} \tag{15.350}$$

Fitness function:

$$f(\mathrm{x}) = 2\pi \cdot x_1^2 + 2\pi \cdot x_1 \cdot x_2. \tag{15.351}$$

Constraint function:

$$g(\mathrm{x}) = \pi \cdot x_1^2 \cdot x_2 > 300. \tag{15.352}$$

The optimal position is $\mathbf{x} = [3.5,\ 7.8]$ cm.

The optimal fitness value is $f(\mathrm{x}) = 248.5\,\mathrm{cm}^2$.

15.2.7 SODCS

Deflected Corrugated Spring function is a single-objective problem. The problem was taken from [26] and modified to have only one global minimum. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -20, 20 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.353}$$

where D means number of decision variables and is arbitrary.

Fitness function [26]:

$$f(\mathrm{x}) = \frac{1}{10} \sum_{i=1}^{D} (x_i - \alpha)^2 - \cos\left[k \sqrt{\sum_{i=1}^{D} (x_i - \alpha)^2}\right] + \frac{1}{10} \sum_{i=1}^{D} |x_i - \alpha|, \tag{15.354}$$

where $\alpha = 5$ and $k = 5$.

The optimal position is $x_i = \alpha$ for $i = 1, 2, \ldots D$.

The optimal fitness value is $f(\mathrm{x}) = -1$.

15.2.8 SODPF

Dixon-Price's function is a single-objective problem [27]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.355}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = (x_1 - 1)^2 + \sum_{i=2}^{D} i\left(2x_i^2 - x_{i-1}\right)^2. \tag{15.356}$$

The optimal position is located at

$$x_i = 2^{-\left[(2^i-2)/2^i\right]} \text{ for } i = 1, 2, \ldots D. \tag{15.357}$$

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.9 SOEAS

Easom's function is a single-objective problem with two decision variables [22]. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -100, 100 \rangle \text{ for } i = 1, 2. \tag{15.358}$$

Fitness function:

$$f(\mathrm{x}) = -\cos(x_1) \cdot \cos(x_2) \cdot \exp\left[-(x_1 - \pi)^2 - (x_2 - \pi)^2\right]. \tag{15.359}$$

The optimal position is $\mathbf{x} = [\pi, \pi]$.
The optimal fitness value is $f(\mathrm{x}) = -1$.

15.2.10 SOGRI

Griewangk's function is a single-objective problem [22]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -600, 600 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.360}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = \frac{1}{4000} \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1. \tag{15.361}$$

The optimal position is $x_i = 0$ for $i = 1, 2, \ldots D$.
The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.11 SOHUM

Camel function - Six Hump - is a single-objective problem with two decision variables [27]. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -5, 5 \rangle \text{ for } i = 1, 2. \tag{15.362}$$

Fitness function:

$$f(\mathrm{x}) = 4x_1^2 - 2.1x_1^4 + \frac{x_1^6}{3} + x_1 \cdot x_2 + 4x_2^4 - 4x_2^2 + 1.0316. \tag{15.363}$$

There are two global minima located at $\mathbf{x} = [-0.0898,\ 0.7126]$ and $\mathbf{x} = [0.0898,\ -0.7126]$.
The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.12 SOLEV

Levy's function is a single-objective problem [28]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.364}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$\begin{aligned} f(\mathrm{x}) = {} & \sin^2(\pi \cdot y_1) + \left\{ \sum_{i=1}^{D-1} (y_i - 1)^2 \cdot \left[1 + 10 \sin^2(\pi \cdot y_i + 1)\right] \right\} \\ & + (y_D - 1)^2 \cdot \left[1 + \sin^2(2\pi \cdot x_D)\right], \end{aligned} \tag{15.365}$$

$$y_i = 1 + \frac{x_i - 1}{4}, \text{ for } i = 1, 2, \dots D. \tag{15.366}$$

The optimal position is $x_i = 1$ for $i = 1, 2, \dots D$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.13 SOMAT

Matyas's function is a single-objective problem with two decision variables [27]. The problem is not scalable, therefore all input arguments will be ignored.

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2. \tag{15.367}$$

Fitness function:

$$f(\mathrm{x}) = 0.26 \left(x_1^2 + x_2^2\right) - 0.48 x_1 \cdot x_2. \tag{15.368}$$

The optimal position is $\mathbf{x} = [0,\ 0]$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.14 SORAS

Rastrigin's function is a single-objective problem [22]. The problem is scalable in the decision space. The default number of decision variables is $D = 5$.

Limits:

$$x_i \in \langle -5.12, 5.12 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.369}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = 10 \cdot D + \left[\sum_{i=1}^{D} x_i^2 - 10 \cos(2\pi \cdot x_i) \right]. \tag{15.370}$$

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.15 SOROS

Rosebrock's function is a single-objective problem [27]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -30, 30 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.371}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = \left[\sum_{i=1}^{D-1} 100 \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2 \right]. \tag{15.372}$$

The optimal position is $x_i = 1$ for $i = 1, 2, \dots D$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.2.16 SOSCH

Schwefel's function is a single-objective problem [22]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle -500, 500 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.373}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = \left[\sum_{i=1}^{D} -x_i \sin\left(\sqrt{|x_i|}\right) \right]. \tag{15.374}$$

The optimal position is $x_i = 420.968746$ for $i = 1, 2, \ldots D$.

The optimal fitness value is $f(\mathrm{x}) = -418.9829D$.

15.2.17 SOSPH

Sphere function is a single-objective problem [22]. The problem is scalable in the decision space. The default number of decision variables is $D = 2$.

Limits:

$$x_i \in \langle 0, 10 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.375}$$

where D means number of decision variables and is arbitrary.

Fitness function:

$$f(\mathrm{x}) = \sum_{i=0}^{D} x_i^2. \tag{15.376}$$

The optimal position is $x_i = 0$ for $i = 1, 2, \ldots D$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3 Single-objective Variable Number of Dimensions

15.3.1 VNDKDEJ

Modified De Jong's function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -50, 50 \rangle \ \text{ for } i = 1, 2, \dots D, \tag{15.377}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f(\mathrm{x}, D_{opt}) = \left(\sum_{i=1}^{D} i \cdot x_i^4 \right) + (D - D_{opt})^4, \tag{15.378}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of the problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.2 VNDKGIU

Modified Giunta's function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -500, 500 \rangle \ \text{ for } i = 1, 2, \dots D, \tag{15.379}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$\begin{aligned} f(\mathrm{x}, D_{opt}) = & \left\{ \sum_{i=1}^{D} \sin\left(\frac{16}{15}x_i - 1\right) + \sin^2\left(\frac{16}{15}x_i - 1\right) + \frac{1}{50}\sin\left[4\left(\frac{16}{15}x_i - 1\right)\right] + \frac{268}{1000} \right\} \\ & + \sqrt{|D - D_{opt}|}, \end{aligned} \tag{15.380}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0.46732$ for $i = 1, 2, \dots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.3 VNDKGRI

Modified Griewangk's function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -500, 500 \rangle \ \text{ for } i = 1, 2, \dots D, \tag{15.381}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left[\frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i+1}}\right)\right] + 0.2\left(D - D_{opt}\right)^2, \tag{15.382}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f\left(\mathrm{x}\right) = 0$.

15.3.4 VNDKRAS

Modified Rastrigin's function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -50, 50 \rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.383}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left[\sum_{i=1}^{D} 10 + x_i^2 - 10\cos\left(2\pi \cdot x_i\right)\right] + \left(D - D_{opt}\right)^4, \tag{15.384}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f\left(\mathrm{x}\right) = 0$.

15.3.5 VNDKROS

Modified Rosenbrock's function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -50, 50 \rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.385}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left[\sum_{i=1}^{D} 100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right] + \left(D - D_{opt}\right)^4, \tag{15.386}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 1$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f\left(\mathrm{x}\right) = 0$.

15.3.6 VNDKSCH

Modified Schwevel's function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -500, 500\rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.387}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10\rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left[418.9829D + \sum_{i=1}^{D} x_i \sin\left(\sqrt{|x_i|}\right)\right] + 40\left(D - D_{opt}\right)^2, \tag{15.388}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 420.968746$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f\left(\mathrm{x}\right) = 0$.

15.3.7 VNDKSPH

Modified Sphere function is a single-objective problem with the variable number of dimensions [29].

Limits:

$$x_i \in \langle -150, 150\rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.389}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10\rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left(\sum_{i=1}^{D} x_i^2\right) + \left(D - D_{opt}\right)^4, \tag{15.390}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f\left(\mathrm{x}\right) = 0$.

15.3.8 VNDMACK

Modified Ackley's function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle -32, 32\rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.391}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10\rangle$.

Fitness function:

$$\begin{aligned} f\left(\mathrm{x}, D_{opt}\right) = {} & 20\left(1 + \left|D - D_{opt}\right|\right) + \exp\left(1\right) \\ & - 20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left[\frac{1}{D}\sum_{i=1}^{D}\cos\left(2\pi\cdot x_i\right)\right], \end{aligned} \tag{15.392}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.9 VNDMALP

Modified Alpine function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle -10, 10 \rangle \;\text{ for } i = 1, 2, \ldots D, \tag{15.393}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f(\mathrm{x}, D_{opt}) = \left| \sum_{i=1}^{D} (x_i - \alpha_i) \sin(x_i - \alpha_i) + 0.1 (x_i - \alpha_i) \right| + |D - D_{opt}|, \tag{15.394}$$

$$\alpha_i(\mathrm{x}) = x_{i,L} + (x_{i,U} - x_{i,L}) \cdot 10^{-\frac{i}{D+1}}, \tag{15.395}$$

where D_{opt} is the optimal number of decision variables, $x_{i,L}$ and $x_{i,U}$ denotes lower and upper limits of i-th decision, respectively.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = \alpha_i$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.10 VNDMDCS

Modified Deflected Corrugated Spring function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle 0, 10 \rangle \;\text{ for } i = 1, 2, \ldots D, \tag{15.396}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f(\mathrm{x}, D_{opt}) = 1 + \frac{1}{D} \left\{ \sum_{i=1}^{D} (x_i - \alpha_i)^2 + 10 \sqrt{\sum_{i=1}^{D} (x_i - \alpha_i)^2} - \cos\left[\pi |D - D_{opt}| \left(\frac{1}{4} + \frac{37}{79} \right) \right] \right\}, \tag{15.397}$$

$$\alpha_i(\mathrm{x}) = \frac{i \cdot (x_{i,U} - x_{i,L})}{D + 1}, \tag{15.398}$$

where D_{opt} is the optimal number of decision variables, $x_{i,L}$ and $x_{i,U}$ denotes lower and upper limits of i-th decision, respectively.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = \alpha_i$ for $i = 1, 2, \ldots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.11 VNDMDPF

Modified Dixon-Price's function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.399}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left(D - D_{opt}\right)^2 + \left(x_1 - 1\right)^2 + \left(\sum_{i=2}^{D} 2x_i^2 - 2x_{i-1}^2\right), \tag{15.400}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position:

$$x_i = 2^{-\frac{2^i - 2}{2^i}} \text{ for } i = 1, 2, \dots D_{opt}. \tag{15.401}$$

The optimal fitness value is $f\left(\mathrm{x}\right) = 0$.

15.3.12 VNDMMIC

Modified Michalewicz's function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle 0, \pi \rangle \text{ for } i = 1, 2, \dots D, \tag{15.402}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = -10\left[\sum_{i=1}^{D} \sin\left(x_i\right) \cdot \sin^{20}\left(\frac{i \cdot x_i^2}{\pi}\right)\right] + \left|\left(D - D_{opt} + 2\right)^3\right|, \tag{15.403}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal positions and fitness values can be found in [30].

15.3.13 VNDMMUM

Modified Modified Multimodal function is a single-objective problem with variable number of dimensions [30].

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.404}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f\left(\mathrm{x}, D_{opt}\right) = \left(\sum_{i=1}^{D} \left|x_i\right| + \left|D - D_{opt}\right|\right) \cdot \left(\prod_{i=1}^{D} \left|x_i\right| + \frac{10}{D}\left|D - D_{opt}\right|\right), \tag{15.405}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.14 VNDMRAS

Modified Rastrigin's function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle -5.12, 5.12 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.406}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f(\mathrm{x}, D_{opt}) = 10 \cdot D \left[(D - D_{opt}) + 1 \right] + \left[\sum_{i=1}^{D} x_i^2 - 10 \cos(2\pi \cdot x_i) \right], \tag{15.407}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.15 VNDSPH1

Modified Sphere 1 function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.408}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f(\mathrm{x}, D_{opt}) = \left(\sum_{i=1}^{D} x_i^2 \right) + 10 (D - D_{opt})^2, \tag{15.409}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position is $x_i = 0$ for $i = 1, 2, \dots D_{opt}$.

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.3.16 VNDSPH2

Modified Sphere 2 function is a single-objective problem with the variable number of dimensions [30].

Limits:

$$x_i \in \langle -10, 10 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.410}$$

where D means the number of decision variables and it is varied during an optimization run within the range of the number of decision variables D_{range}. By default, $D_{range} \in \langle 2,\ 10 \rangle$.

Fitness function:

$$f(\mathrm{x}, D_{opt}) = \left[\sum_{i=1}^{D} (x_i - D_{opt})^2 + |x_i| (D - D_{opt})^2 \right], \tag{15.411}$$

where D_{opt} is the optimal number of decision variables.

The optimal number of decision variables D_{opt} is the first input argument of problem function, therefore it is arbitrary along with the range of the number of decision variables (second input argument).

By default, the optimal number of decision variables $D_{opt} = 5$.

The optimal position:

$$x_i = D_{opt}(D + 1) - \frac{1}{2}\left(D^2 + D_{opt}^2\right) \text{ for } i = 1, 2, \ldots D_{opt}. \tag{15.412}$$

The optimal fitness value is $f(\mathrm{x}) = 0$.

15.4 Multi-objective Variable Number of Dimensions

Multi-objective problems with Variable Number of Dimensions are based on a well known multi-objective benchmark problems. True Pareto-front of multi-objective problems is constituted by numerous solutions. Therefore, it is necessary to ensure, that a proper multi-objective problem with variable number of dimensions has also true Pareto-front constituted of solution with varying optimal dimensionalities. Such a difficulty leads to a complexity which is insoluble with common multi-objective algorithms.

The modifications of all the VNDMO problems in FOPS is based on methodology proposed in [31]. Methodology is described for both two and three objective test problems. Therefore, the FOPS contains two methods `getNOpt` and `getNOpt3D`. These methods returns the optimal dimensionality of a solution based on an angle between solution in objective space and the center of carthesian system.

Both methods has input arguments `nOptList`, `nParts` and `order`. The Pareto-front is divided into `nParts` parts, where all the dimensionalities listed in `nOptList` are covered either in ascending order (`order` is disabled) or in descending order (proporder is enabled).

If properties `nParts` and `order` are not specified in a problem definition, the `nParts` = 1 and `order` = false.

15.4.1 VNDMODTLZ1

This is the modified first benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.413}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = [1 + h(x_{M:D})]\, 0.5 x_1 \cdot x_2 + P, \tag{15.414}$$
$$f_2(\mathbf{x}) = [1 + h(x_{M:D})]\, 0.5 x_1 (1 - x_2) + P, \tag{15.415}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})]\, 0.5 (1 - x_1) + P, \tag{15.416}$$
$$h(\mathbf{z}) = 100 \left\{ k + \left[\sum_{i=1}^{k} (z_i - 0.5)^2 - 5\cos(20\pi (z_i - 0.5)) \right] \right\}, \tag{15.417}$$
$$P = 0.50 (D - D_{opt})^2 \tag{15.418}$$

where $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the `getOptN3D` function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.2 VNDMODTLZ2

This is the modified second benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.419}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = [1 + h(x_{M:D})] \cos(x_1\pi/2) \cos(x_2\pi/2) + P, \tag{15.420}$$
$$f_2(\mathbf{x}) = [1 + h(x_{M:D})] \cos(x_1\pi/2) \sin(x_2\pi/2) + P, \tag{15.421}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})] \sin(x_1\pi/2) + P, \tag{15.422}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} (z_i - 0.5)^2, \tag{15.423}$$
$$P = 0.05 (D - D_{opt})^2 \tag{15.424}$$

where $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the getOptN3D function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.3 VNDMODTLZ3

This is the modified third benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.425}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = [1 + h(x_{M:D})] \cos(x_1\pi/2) \cos(x_2\pi/2) + P, \tag{15.426}$$
$$f_2(\mathbf{x}) = [1 + h(x_{M:D})] \cos(x_1\pi/2) \sin(x_2\pi/2) + P, \tag{15.427}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})] \sin(x_1\pi/2) + P, \tag{15.428}$$
$$h(\mathbf{z}) = 100 \left\{ k + \left[\sum_{i=1}^{k} (z_i - 0.5)^2 - 5\cos(20\pi(z_i - 0.5)) \right] \right\}, \tag{15.429}$$
$$P = 0.80 (D - D_{opt})^2 \tag{15.430}$$

where $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the getOptN3D function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.4 VNDMODTLZ4

This is the modified fourth benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.431}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = [1 + h(x_{M:D})] \cos(x_1^\alpha \pi/2) \cos(x_2^\alpha \pi/2) + P, \tag{15.432}$$
$$f_2(\mathbf{x}) = [1 + h(x_{M:D})] \cos(x_1^\alpha \pi/2) \sin(x_2^\alpha \pi/2) + P, \tag{15.433}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})] \sin(x_1^\alpha \pi/2) + P, \tag{15.434}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} (z_i - 0.5)^2, \tag{15.435}$$
$$P = 0.05 (D - D_{opt})^2 \tag{15.436}$$

where parameter $\alpha = 100$, $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the getOptN3D function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.5 VNDMODTLZ5

This is the modified fifth benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.437}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = [1 + h(x_{M:D})] \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) + P, \tag{15.438}$$
$$f_2(\mathbf{x}) = [1 + h(x_{M:D})] \cos(\theta_1 \pi/2) \sin(\theta_2 \pi/2) + P, \tag{15.439}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})] \sin(\theta_1 \pi/2) + P, \tag{15.440}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} (z_i - 0.5)^2, \tag{15.441}$$
$$\theta_i = \begin{cases} x_i & \text{for } i = 1, \\ \frac{1+2h\cdot x_i}{2(1+h)} & \text{for } i = 2,\ 3,\ \dots(M-1), \end{cases} \tag{15.442}$$
$$P = 0.05 (D - D_{opt})^2 \tag{15.443}$$

where $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the getOptN3D function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \dots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.6 VNDMODTLZ6

This is the modified sixth benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \dots D, \tag{15.444}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = [1 + h(x_{M:D})] \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2) + P, \tag{15.445}$$
$$f_2(\mathbf{x}) = [1 + h(x_{M:D})] \cos(\theta_1 \pi/2) \sin(\theta_2 \pi/2) + P, \tag{15.446}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})] \sin(\theta_1 \pi/2) + P, \tag{15.447}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} z_i^{0.1}, \tag{15.448}$$
$$\theta_i = \begin{cases} x_i & \text{for } i = 1, \\ \frac{1+2h \cdot x_i}{2(1+h)} & \text{for } i = 2,\ 3,\ \ldots(M-1), \end{cases} \tag{15.449}$$
$$P = 0.08 (D - D_{opt})^2 \tag{15.450}$$

where $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the `getOptN3D` function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.7 VNDMODTLZ7

This is the modified seventh benchmark problem from Deb et al.'s test suite [18]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1\rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.451}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + P, \tag{15.452}$$
$$f_2(\mathbf{x}) = x_2 + P, \tag{15.453}$$
$$f_3(\mathbf{x}) = [1 + h(x_{M:D})] \left\{ M - \sum_{i=1}^{M-1} \left[\frac{f_i}{1 + h(x_{M:D})} (1 + \sin(3\pi \cdot f_i)) \right] \right\} + P, \tag{15.454}$$
$$h(\mathbf{z}) = 1 + \frac{9}{k} \sum_{i=1}^{k} z_i, \tag{15.455}$$
$$P = 0.30 (D - D_{opt})^2 \tag{15.456}$$

where $k = D - M + 1$, M is the number of objectives and D_{opt} is optimal dimension of a solution obtained by the `getOptN3D` function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0.5}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.8 VNDMOLI1

This is the benchmark problem from [31]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1\rangle \ \text{ for } i = 1, 2, \ldots D, \tag{15.457}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + h(x_{2:D}), \tag{15.458}$$
$$f_2(\mathbf{x}) = 1 - x_1 + h(x_{2:D}) + P, \tag{15.459}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} \left[z_i - \sin\left(\frac{D_{opt}}{2D}\pi\right)\right]^2, \tag{15.460}$$
$$P = 0.01\,(D - D_{opt})^2 \tag{15.461}$$

where $k = D - 1$ and D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_j = \sin\left(\frac{D_{opt}}{2D}\pi\right)$, where $j = 1, 2, \ldots, x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.9 VNDMOLI2

This is the benchmark problem from [31]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.462}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + h(x_{2:D}), \tag{15.463}$$
$$f_2(\mathbf{x}) = 1 - x_1 + h(x_{2:D}) + P, \tag{15.464}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} \left[z_i - \sin\left(\frac{D_{opt}}{2D}\pi\right)\right]^2, \tag{15.465}$$
$$P = 0.01\,(D - D_{opt})^2 \tag{15.466}$$

where $k = D - 1$ and D_{opt} is optimal dimension of a solution obtained by the getOptN function. In this case the nParts $= 2$ and order $= [\text{false}, \text{true}]$.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_j = \sin\left(\frac{D_{opt}}{2D}\pi\right)$, where $j = 1, 2, \ldots, x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.10 VNDMOLI3

This is the benchmark problem from [31]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.467}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = h(x_{3:D})\, x_1 (1 - x_2) + P, \tag{15.468}$$
$$f_2(\mathbf{x}) = h(x_{3:D})\, x_1 x_2 + P, \tag{15.469}$$
$$f_3(\mathbf{x}) = h(x_{3:D})\,(1 - x_1) + P, \tag{15.470}$$
$$h(\mathbf{z}) = \sum_{i=1}^{k} z_i^2, \tag{15.471}$$
$$P = 0.01\,(D - D_{opt})^2 \tag{15.472}$$

where $k = D - 2$ and D_{opt} is optimal dimension of a solution obtained by the getOptN3D function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$, $x_2 \in \langle 0, 1\rangle$ and $z = \mathbf{0}$, where $z = x_3, x_4, \ldots x_D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.11 VNDMOLZ1

This is the modified first benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.473}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h(j, x_1)\right]^2, \tag{15.474}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h(j, x_1)\right]^2 + P, \tag{15.475}$$

$$h(j) = x_1^{0.5\left(1+\frac{3j-6}{D-2}\right)} \text{ for } j = 1, 2, \ldots D, \tag{15.476}$$

$$J_1 = 3, 5, \ldots D, \tag{15.477}$$

$$J_2 = 2, 4, \ldots D, \tag{15.478}$$

$$P = 0.05\left(D - D_{opt}\right)^2, \tag{15.479}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.12 VNDMOLZ2

This is the modified second benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.480}$$

$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.481}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h(j, x_1)\right]^2, \tag{15.482}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h(j, x_1)\right]^2 + P, \tag{15.483}$$

$$h(j) = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \text{ for } j = 1, 2, \ldots D, \tag{15.484}$$

$$J_1 = 3, 5, \ldots D, \tag{15.485}$$

$$J_2 = 2, 4, \ldots D, \tag{15.486}$$

$$P = 0.06\left(D - D_{opt}\right)^2, \tag{15.487}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.13 VNDMOLZ3

This is the modified third benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.488}$$

$$x_i \in \langle -1, 1 \rangle \text{ for } i = 2, 3, \dots D, \tag{15.489}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h_1(j, x_1)\right]^2, \tag{15.490}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h_2(j, x_1)\right]^2 + P, \tag{15.491}$$

$$h_1(j) = 0.8 x_1 \cos\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \dots D, \tag{15.492}$$

$$h_2(j) = 0.8 x_1 \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \dots D, \tag{15.493}$$

$$J_1 = 3, 5, \dots D, \tag{15.494}$$

$$J_2 = 2, 4, \dots D, \tag{15.495}$$

$$P = 0.06 \left(D - D_{opt}\right)^2, \tag{15.496}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to:

$$x_j = \begin{cases} h_1(j) & \text{where } j \in J_1, \\ h_2(j) & \text{where } j \in J_2. \end{cases} \tag{15.497}$$

Optimal positions and fitness values are calculated during the problem initialization.

15.4.14 VNDMOLZ4

This is the modified fourth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.498}$$

$$x_i \in \langle -1, 1 \rangle \text{ for } i = 2, 3, \dots D, \tag{15.499}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h_1(j, x_1)\right]^2, \tag{15.500}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h_2(j, x_1)\right]^2 + P, \tag{15.501}$$

$$h_1(j) = 0.8 x_1 \cos\left(\frac{6\pi x_1 + \frac{j\pi}{D}}{3}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.502}$$

$$h_2(j) = 0.8 x_1 \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.503}$$

$$J_1 = 3, 5, \ldots D, \tag{15.504}$$

$$J_2 = 2, 4, \ldots D, \tag{15.505}$$

$$P = 0.05 \left(D - D_{opt}\right)^2, \tag{15.506}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to:

$$x_j = \begin{cases} h_1(j) & \text{for } j \in J_1, \\ h_2(j) & \text{for } j \in J_2. \end{cases} \tag{15.507}$$

Optimal positions and fitness values are calculated during the problem initialization.

15.4.15 VNDMOLZ5

This is the modified fifth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.508}$$

$$x_i \in \langle -1, 1 \rangle \ \text{ for } i = 2, 3, \ldots D, \tag{15.509}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - h_1(j, x_1)\right]^2, \tag{15.510}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - h_2(j, x_1)\right]^2 + P, \tag{15.511}$$

$$h_1(j) = \left[0.3 x_1^2 \cos\left(24\pi x_1 + \frac{4j\pi}{D}\right) + 0.6 x_1\right] \cos\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.512}$$

$$h_2(j) = \left[0.3 x_1^2 \cos\left(24\pi x_1 + \frac{4j\pi}{D}\right) + 0.6 x_1\right] \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.513}$$

$$J_1 = 3, 5, \ldots D, \tag{15.514}$$

$$J_2 = 2, 4, \ldots D, \tag{15.515}$$

$$P = 0.05 \left(D - D_{opt}\right)^2, \tag{15.516}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to:

$$x_j = \begin{cases} h_1(j) & \text{for } j \in J_1, \\ h_2(j) & \text{for } j \in J_2. \end{cases} \tag{15.517}$$

Optimal positions and fitness values are calculated during the problem initialization.

15.4.16 VNDMOLZ6

This is the modified sixth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1 \rangle, \tag{15.518}$$
$$x_2 \in \langle 0, 1 \rangle, \tag{15.519}$$
$$x_i \in \langle -2, 2 \rangle \text{ for } i = 3, 4, \dots D, \tag{15.520}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}[x_j - h(j, x_1, x_2)]^2 + P, \tag{15.521}$$
$$f_2(\mathbf{x}) = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}[x_j - h(j, x_1, x_2)]^2 + P, \tag{15.522}$$
$$f_3(\mathbf{x}) = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}[x_j - h(j, x_1, x_2)]^2 + P, \tag{15.523}$$
$$h(j) = 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{D}\right), \quad \text{for } j = 1, 2, \dots D, \tag{15.524}$$
$$J_1 = 4, 7, \dots D, \tag{15.525}$$
$$J_2 = 5, 8, \dots D, \tag{15.526}$$
$$J_3 = 3, 6, \dots D, \tag{15.527}$$
$$P = 0.12(D - D_{opt})^2, \tag{15.528}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN3D` function.

Optimal solutions corresponds to $x_j = h(j)$, where $j = 3, 4, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.17 VNDMOLZ7

This is the modified seventh benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1 \rangle \text{ for } i = 1, 2, \dots D, \tag{15.529}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left[4y_j^2 - \cos(8y_j\pi) + 1\right], \tag{15.530}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2}\left[4y_j^2 - \cos(8y_j\pi) + 1\right] + P, \tag{15.531}$$

$$y_j = x_j - h(j) \quad \text{for } j = 2,3,\dots D, \tag{15.532}$$

$$h(j) = x_1^{0.5\left(1+\frac{3j-6}{D-2}\right)} \quad \text{for } j = 1,2,\dots D, \tag{15.533}$$

$$J_1 = 3,5,\dots D, \tag{15.534}$$

$$J_2 = 2,4,\dots D, \tag{15.535}$$

$$P = 0.10\left(D - D_{opt}\right)^2, \tag{15.536}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2,3,\dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.18 VNDMOLZ8

This is the modified eighth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0,1\rangle \quad \text{for } i = 1,2,\dots D, \tag{15.537}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|}\left[4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right], \tag{15.538}$$

$$f_2(\mathbf{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\left[4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right] + P, \tag{15.539}$$

$$y_j = x_j - h(j) \quad \text{for } j = 2,3,\dots D, \tag{15.540}$$

$$h(j) = x_1^{0.5\left(1+\frac{3j-6}{D-2}\right)} \quad \text{for } j = 1,2,\dots D, \tag{15.541}$$

$$J_1 = 3,5,\dots D, \tag{15.542}$$

$$J_2 = 2,4,\dots D, \tag{15.543}$$

$$P = 0.35\left(D - D_{opt}\right)^2, \tag{15.544}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2,3,\dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.19 VNDMOLZ9

This is the modified ninth benchmark problem from Li and Zhang's test suite [19]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1\rangle\,, \tag{15.545}$$
$$x_i \in \langle -1, 1\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.546}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - h(j, x_1)\right]^2, \tag{15.547}$$
$$f_2(\mathbf{x}) = 1 - x_1^2 + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - h(j, x_1)\right]^2 + P, \tag{15.548}$$
$$h(j) = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.549}$$
$$J_1 = 3, 5, \ldots D, \tag{15.550}$$
$$J_2 = 2, 4, \ldots D, \tag{15.551}$$
$$P = 0.06\left(D - D_{opt}\right)^2, \tag{15.552}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_j = h(j)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.20 VNDMOUF4

This is the modified fourth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_1 \in \langle 0, 1\rangle\,, \tag{15.553}$$
$$x_i \in \langle -2, 2\rangle \text{ for } i = 2, 3, \ldots D, \tag{15.554}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j)\,, \tag{15.555}$$
$$f_2(\mathbf{x}) = 1 - x_1^2 + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) + P, \tag{15.556}$$
$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \ldots D, \tag{15.557}$$
$$h(y) = \frac{|y|}{1 + \exp(2|y|)}, \tag{15.558}$$
$$J_1 = 3, 5, \ldots D, \tag{15.559}$$
$$J_2 = 2, 4, \ldots D, \tag{15.560}$$
$$P = 0.05\left(D - D_{opt}\right)^2, \tag{15.561}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.21 VNDMOUF5

This is the modified fifth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1\rangle\,, \tag{15.562}$$
$$x_i \in \langle -1, 1\rangle \;\text{ for } i = 2, 3, \ldots D, \tag{15.563}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \left(\frac{1}{2N+\epsilon}\right)\left|\sin(2N\pi x_1)\right| + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j)\,, \tag{15.564}$$
$$f_2(\mathbf{x}) = 1 - x_1 + \left(\frac{1}{2N+\epsilon}\right)\left|\sin(2N\pi x_1)\right| + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) + P, \tag{15.565}$$
$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \;\text{ for } j = 1, 2, \ldots D, \tag{15.566}$$
$$h(y) = 2y^2 - \cos(4\pi y) + 1, \tag{15.567}$$
$$J_1 = 3, 5, \ldots D, \tag{15.568}$$
$$J_2 = 2, 4, \ldots D, \tag{15.569}$$
$$P = 0.15\left(D - D_{opt}\right)^2\,, \tag{15.570}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function. Variable N is an integer and it defines the number of the solutions of the true Pareto-front. The variable was set to $N = 10$, therefore the true Pareto-front has $2N+1 = 21$ solutions. Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.22 VNDMOUF6

This is the modified sixth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1\rangle\,, \tag{15.571}$$
$$x_i \in \langle -1, 1\rangle \;\text{ for } i = 2, 3, \ldots D, \tag{15.572}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1 + \max\left\{0, 2\left(\frac{1}{2N}+\epsilon\right)\sin(2N\pi x_1)\right\} + \frac{2}{|J_1|}\left[4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right], \tag{15.573}$$
$$f_2(\mathbf{x}) = 1 - x_1 + \max\left\{0, 2\left(\frac{1}{2N}+\epsilon\right)\sin(2N\pi x_1)\right\} + \frac{2}{|J_2|}\left[4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right] + P, \tag{15.574}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \dots D, \tag{15.575}$$
$$J_1 = 3, 5, \dots D, \tag{15.576}$$
$$J_2 = 2, 4, \dots D, \tag{15.577}$$
$$P = 0.50\left(D - D_{opt}\right)^2, \tag{15.578}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function. Variable N is an integer and it defines the number of disconnected parts of the true Pareto-front. The variable was set to $N = 2$, therefore the true Pareto-front has $N + 1 = 3$ disconnected parts. Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.23 VNDMOUF7

This is the modified seventh benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.579}$$
$$x_i \in \langle -1, 1\rangle \quad \text{for } i = 2, 3, \dots D, \tag{15.580}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = \sqrt[5]{x_1} + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2, \tag{15.581}$$
$$f_2(\mathbf{x}) = 1 - \sqrt[5]{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 + P, \tag{15.582}$$
$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{D}\right) \quad \text{for } j = 1, 2, \dots D, \tag{15.583}$$
$$J_1 = 3, 5, \dots D, \tag{15.584}$$
$$J_2 = 2, 4, \dots D, \tag{15.585}$$
$$P = 0.15\left(D - D_{opt}\right)^2, \tag{15.586}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_j = \sin\left(6\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.24 VNDMOUF9

This is the modified ninth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.587}$$
$$x_2 \in \langle 0, 1\rangle, \tag{15.588}$$
$$x_i \in \langle -2, 2\rangle \quad \text{for } i = 3, 4, \dots D, \tag{15.589}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$\begin{aligned} f_1(\mathbf{x}) =& 0.5\left[\max\left\{0,(1+\epsilon)\left(1-4(2x_1-1)^2\right)\right\}+2x_1\right]x_2+ && (15.590)\\ & \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{D}\right)\right]^2+P, \\ f_2(\mathbf{x}) =& 0.5\left[\max\left\{0,(1+\epsilon)\left(1-4(2x_1-1)^2\right)\right\}-2x_1+2\right]x_2+ && (15.591)\\ & \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{D}\right)\right]^2+P, \\ f_3(\mathbf{x}) =& 1-x_2+ && (15.592)\\ & \frac{2}{|J_3|}\sum_{j\in J_3}\left[x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{D}\right)\right]^2+P, \\ & J_1 = 4,7,\dots D, && (15.593)\\ & J_2 = 5,8,\dots D, && (15.594)\\ & J_3 = 3,6,\dots D, && (15.595)\\ & P = 0.25\left(D-D_{opt}\right)^2 \qquad , && (15.596) \end{aligned}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function. Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions are divided into two disconected parts where $x_1 \in [0, 0.25] \cup [0.75, 1]$, $0 \leq x_2 \leq 1$, and $x_j = 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 3, 4, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.25 VNDMOUF10

This is the modified tenth benchmark problem of test instances for the CEC 2009 [20]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$\begin{aligned} x_1 &\in \langle 0,1\rangle, && (15.597)\\ x_2 &\in \langle 0,1\rangle, && (15.598)\\ x_i &\in \langle -2,2\rangle \text{ for } i=3,4,\dots D, && (15.599) \end{aligned}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$\begin{aligned} f_1(\mathbf{x}) &= \cos(0.5x_1\pi)\cos(0.5x_2\pi)\frac{2}{|J_1|}\sum_{j\in J_1}\left[4y_j^2-cos(8\pi y_j)+1\right]+P, && (15.600)\\ f_2(\mathbf{x}) &= \cos(0.5x_1\pi)\sin(0.5x_2\pi)\frac{2}{|J_2|}\sum_{j\in J_2}\left[4y_j^2-cos(8\pi y_j)+1\right]+P, && (15.601)\\ f_3(\mathbf{x}) &= \sin(0.5x_1\pi) \qquad \frac{2}{|J_3|}\sum_{j\in J_3}\left[4y_j^2-cos(8\pi y_j)+1\right]+P, && (15.602)\\ y_j &= x_j-2x_2\sin\left(2\pi x_1+\frac{j\pi}{D}\right) \text{ for } j=1,2,\dots D, && (15.603)\\ J_1 &= 4,7,\dots D, && (15.604)\\ J_2 &= 5,8,\dots D, && (15.605)\\ J_3 &= 3,6,\dots D, && (15.606)\\ P &= 0.50\left(D-D_{opt}\right)^2, && (15.607) \end{aligned}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function. Constant $\epsilon > 0$ is set to $\epsilon = 0.1$.

Optimal solutions corresponds to $x_j = 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{D}\right)$, where $j = 3, 4, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.26 VNDMOZDT1

This is the modified first benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.608}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.609}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}}\right] + P, \tag{15.610}$$

$$h(\mathbf{x}) = 1 + 9\frac{\sum_{i=2}^{D} x_i}{D-1}, \tag{15.611}$$

$$P = 0.10\left(D - D_{opt}\right)^2, \tag{15.612}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.27 VNDMOZDT2

This is the modified second benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in decision space. The number of decision variables is defined by nVarsList property of problem. Optimal number of dimensions is defined by nOptList parameter along with nParts and order properties.

Limits:

$$x_i \in \langle 0, 1\rangle \text{ for } i = 1, 2, \ldots D, \tag{15.613}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.614}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \left(\frac{x_1}{h(\mathbf{x})}\right)^2\right] + P, \tag{15.615}$$

$$h(\mathbf{x}) = 1 + 9\frac{\sum_{i=2}^{D} x_i}{D-1}, \tag{15.616}$$

$$P = 0.10\left(D - D_{opt}\right)^2, \tag{15.617}$$

where D_{opt} is optimal dimension of a solution obtained by the getOptN function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.28 VNDMOZDT3

This is the modified third benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1\rangle \ \text{ for } i = 1, 2, \dots D, \tag{15.618}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.619}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}} - \frac{x_1}{h(\mathbf{x})} \sin(10\pi \cdot x_1)\right] + P, \tag{15.620}$$

$$h(\mathbf{x}) = 1 + 9 \frac{\sum_{i=2}^{D} x_i}{D-1}, \tag{15.621}$$

$$P = 0.10\,(D - D_{opt})^2, \tag{15.622}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.29 VNDMOZDT4

This is the modified fourth benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_1 \in \langle 0, 1\rangle, \tag{15.623}$$

$$x_i \in \langle -5, 5\rangle \ \text{ for } i = 2, 3, \dots D, \tag{15.624}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = x_1, \tag{15.625}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}}\right] + P, \tag{15.626}$$

$$h(\mathbf{x}) = 1 + 10(D-1) + \left[\sum_{i=2}^{D} x_i^2 - 10\cos(4\pi \cdot x_i)\right], \tag{15.627}$$

$$P = 0.20\,(D - D_{opt})^2, \tag{15.628}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1\rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \dots D$. Optimal positions and fitness values are calculated during the problem initialization.

15.4.30 VNDMOZDT6

This is the modified sixth benchmark problem from Zitzler et al.'s test suite [1]. The problem is scalable in decision space. The number of decision variables is defined by `nVarsList` property of problem. Optimal number of dimensions is defined by `nOptList` parameter along with `nParts` and `order` properties.

Limits:

$$x_i \in \langle 0, 1 \rangle \text{ for } i = 1, 2, \ldots D, \tag{15.629}$$

where D denotes the number of decision variables and it is arbitrary.

Fitness functions:

$$f_1(\mathbf{x}) = 1 - \exp(-4x_1) \sin^6(6\pi \cdot x_1), \tag{15.630}$$

$$f_2(\mathbf{x}) = h(\mathbf{x}) \left[1 - \left(\frac{f_1(\mathbf{x})}{h(\mathbf{x})}\right)^2\right] + P, \tag{15.631}$$

$$h(\mathbf{x}) = 1 + 9 \left[\frac{\sum_{i=2}^{D} x_i}{D-1}\right]^{0.25}, \tag{15.632}$$

$$P = 0.10 (D - D_{opt})^2, \tag{15.633}$$

where D_{opt} is optimal dimension of a solution obtained by the `getOptN` function.

Optimal solutions corresponds to $x_1 \in \langle 0, 1 \rangle$ and $x_i = \mathbf{0}$, where $i = 2, 3, \ldots D$. Optimal positions and fitness values are calculated during the problem initialization.

Chapter 16

Install FOPS Package

FOPS package is installed by following steps:

1. Copy whole directory with FOPS toolbox to an arbitrary destination.
2. Open a script `installFOPS` in `+internal/+install` directory.
3. Run the script.

This script adds current path to a MATLAB's search paths, therefore user can use FOPS package in an arbitrary location. Note that current path during install has to be the root folder of FOPS toolbox.

Script also modifies text files `problemFunctionPaths.txt` and `settingsFilePaths.txt` in `+internal/+install` directory. These text files contains paths which are scanned for problem functions and task settings files in FOPS constructor. Therefore, first line below header contains location (”default” path) where the FOPS package is installed. This specific line is modified by `installFOPS` script.

If user wants FOPS to scan another locations each time it is initialized, whole path to folder with problem functions and/or task settings files should be put below the line modified by FOPS installer.

Please note, that modification of ”default” path in `problemFunctionPaths.txt` and `settingsFilePaths.txt` files can lead to unavailable benchmark problems in FOPS package.

16.1 Uninstall FOPS Package

To uninstall FOPS package, use the `uninstallFOPS` script in the same directory. This script removes current path from the MATLAB paths. Text files `problemFunctionPaths.txt` and `settingsFilePaths.txt` in `+internal/+install` directory remain unmodified. Note that current path during uninstall has to be the root folder of FOPS toolbox.

Bibliography

[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.

[2] K. Deb, *Multi-objective optimization using evolutionary algorithms*, vol. 16. John Wiley & Sons, 2001.

[3] L. While, L. Bradstreet, and L. Barone, "A fast way of calculating exact hypervolumes," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 1, pp. 86–95, 2012.

[4] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[5] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*, vol. 7. John Wiley & Sons, 2000.

[6] J. H. Holland, "Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence," *Ann Arbor, MI: University of Michigan Press*, 1975.

[7] R. J. Kennedy and Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks IV*, vol. 1000, 1995.

[8] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[9] K. e. a. Price, *New ideas in optimization.* McGraw-Hill Ltd., UK, 1999.

[10] I. Zelinka, "Somaself-organizing migrating algorithm," in *New optimization techniques in engineering*, pp. 167–217, Springer, 2004.

[11] M. A. Luersen and R. Le Riche, "Globalized nelder–mead method for engineering optimization," *Computers & structures*, vol. 82, no. 23-26, pp. 2251–2260, 2004.

[12] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.

[13] S. Kukkonen and K. Deb, "A fast and effective method for pruning of non-dominated solutions in many-objective problems," in *PPSN*, vol. 4193, pp. 553–562, 2006.

[14] J. Moore and R. Chapman, "Application of particle swarm to multiobjective optimization," *Department of Computer Science and Software Engineering, Auburn University*, vol. 32, 1999.

[15] M. Reyes-Sierra and C. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International journal of computational intelligence research*, vol. 2, no. 3, pp. 287–308, 2006.

[16] S. Kukkonen and J. Lampinen, “Gde3: The third evolution step of generalized differential evolution,” in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 1, pp. 443–450, IEEE, 2005.

[17] V. ŠEDĚNKA and Z. RAIDA, “Critical comparison of multi-objective optimization methods: Genetic algorithms versus swarm intelligence.,” *Radioengineering*, vol. 19, no. 3, 2010.

[18] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, “Scalable multi-objective optimization test problems,” in *Evolutionary Computation, 2002. CEC’02. Proceedings of the 2002 Congress on*, vol. 1, pp. 825–830, IEEE, 2002.

[19] H. Li and Q. Zhang, “Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 13, no. 2, pp. 284–302, 2009.

[20] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, “Multiobjective optimization test instances for the cec 2009 special session and competition,” *University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report*, vol. 264, 2008.

[21] 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.

[22] M. Molga and C. Smutnicki, “Test functions for optimization needs,” *Test functions for optimization needs*, p. 101, 2005.

[23] B. Xin, J. Chen, Z. Peng, and F. Pan, “An adaptive hybrid optimizer based on particle swarm and differential evolution for global optimization,” *Science China Information Sciences*, vol. 53, no. 5, pp. 980–989, 2010.

[24] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, “Testing unconstrained optimization software,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 7, no. 1, pp. 17–41, 1981.

[25] S. Satapathy and A. Naik, “Improved teaching learning based optimization for global function optimization,” *Decision Science Letters*, vol. 2, no. 1, pp. 23–34, 2013.

[26] S. K. Mishra, “Performance of differential evolution and particle swarm methods on some relatively harder multi-modal benchmark functions,” 2006.

[27] M. Jamil and X.-S. Yang, “A literature survey of benchmark functions for global optimisation problems,” *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[28] C. Grosan, A. Abraham, and A. E. Hassainen, “A line search approach for high dimensional function optimization,” *Telecommunication Systems*, vol. 46, no. 3, pp. 217–243, 2011.

[29] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, “Fractional particle swarm optimization in multidimensional search space,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 2, pp. 298–319, 2010.

[30] P. Kadlec and V. Šeděnka, “Particle swarm optimization for problems with variable number of dimensions,” *Engineering Optimization*, pp. 1–18, 2017.

[31] H. Li and K. Deb, “Challenges for evolutionary multiobjective optimization algorithms in solving variable-length problems,” in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2217–2224, IEEE, 2017.

Index