README

Aritra Pal and Hadi Charkhgard

September 7, 2017

Contents

1 Understanding the source code:

This whole project has been implemented in Julia v0.6.0 and divided into six different subpackages: Modof.jl, Modolib.jl, FPBH.jl, CPLEXExtensions.jl, FPBHCPLEX.jl and Modoplots.jl for ease of understanding and maintainence. Let us now describe the purpose and major components of each of these subpackages:

1.1 Modof.jl

Modof.jl is a framework used for solving multiobjective mixed integer programs in Julia. Detailed documentation of Modof.jl is available here

1. ModoModel.jl extends JuMP.jl model for multiple objectives.
2. Types.jl has efficient data structures for storing instances and solutions of different classes of multiobjective optimization problems.

3. Utilities.jl contains functions for:
    1. selecting, sorting, writing and normalizing a nondominated frontier
    2. computing ideal and nadir points of a nondominated frontier
    3. computing the closest and the farthest point from the ideal and the nadir points respectively.
4. Quality_of_a_Frontier.jl contains functions for computing the quality of a nondominated frontier:
    1. exact (for biobjective and triobjective) and approximate (for more than 4 objectives) hypervolumes
    2. cardinality
    3. maximum and average coverage
    4. uniformity
5. MDLS.jl wraps the MDLS algorithm for solving multidimensional knapsack and biobjective set packing problems. MDLS must be compiled and the respective path of the binaries must be exported as `export PATH="path to mdls binaries:$PATH"`.

1.2 Modolib.jl

Modolib.jl is a collection of instances and their efficient frontiers (if available) of various classes of multiobjective mixed integer programs. It also has function for generating random several classes of random instances. Detailed documentation of Modolib.jl is available here

1. Api.jl wraps various types of instances and their efficient frontiers (if available) of various classes of multiobjective pure and mixed integer programs.
2. Generating_Instances.jl contains functions for generating several classes of random instances.

1.3 FPBH.jl

FPBH.jl is the source code of the Feasibility Pump based Heuristic for Multi-objective Mixed Integer Linear Programming. It is developed using MathProgBase.jl and hence supports any LP solver supported by MathProgBase.jl.

1. starting_solution_creators.jl is the source code of the different weighted sum methods.
2. feasibility_pumping.jl is the source code of the different feasibility pump methods.
3. local_search_operators.jl is the source code of the different local search operators.
4. decomposition_heuristics.jl is the source code of stage 1 of FPBH.jl including parallelization.
5. solution_polishing.jl is the source code of stage 2 of FPBH.jl including parallelization.
6. In Overall_Algorithm.jl all the different components of FPBH.jl are assembled together.

1.4 CPLEXExtensions.jl

CPLEXExtensions.jl extends CPLEX.jl for single-objective optimization by adding additional functionality like deleting constraints, changing coefficient on lhs and rhs of constraints, etc (using `del_constrs!`, `chg_coeffs!`, `set_rhs!`, `chg_coeff_of_obj!`, `chg_coeff_of_rhs!`, `chg_coeff!`, `get_rhs_coef`) and for multi-objective optimization (using `cplex_model`) for Modof.jl

1.5 FPBHCPLEX.jl

FPBHCPLEX.jl is the source code of FPBH using CPLEX.jl and CPLEXExtensions.jl. Thus, it only uses CPLEX to solve the underlying LP subproblems. It is important to note that some functions (for generating queues in local search and decomposition heuristics) in FPBH.jl are reused in FPBHCPLEX.jl

1. starting_solution_creators.jl is the source code of the different weighted sum methods.
2. feasibility_pumping.jl is the source code of the different feasibility pump methods.
3. local_search_operators.jl is the source code of the different local search operators.
4. decomposition_heuristics.jl is the source code of stage 1 of FPBHCPLEX.jl including parallelization.
5. solution_polishing.jl is the source code of stage 2 of FPBHCPLEX.jl including parallelization.
6. In Overall_Algorithm.jl all the different components of FPBHCPLEX.jl are assembled together.

1.6 Modoplots.jl

Plotting_Nondominated_Frontiers.jl uses PyPlot.jl and Seaborn for plotting nondominated frontiers of

1. Biobjective:
    1. discrete problems
    2. mixed problems
2. Triobjective problems

2 Installation:

It is important to note that the whole ecosystem has been tested using Julia v0.6.0 and hence we can cannot guarantee whether it will work with previous versions of Julia. Thus, it is important that Julia v0.6.0 is properly installed and available on your system.

2.1 If CPLEX is available:

CPLEX must be available in the local machine and CPLEX.jl must be properly installed, otherwise the installation will fail. Once, `Julia v0.6.0` and `CPLEX.jl` has been properly installed, the following instructions in a **Julia** terminal will install **FPBHCPLEX.jl** and its dependencies (**Modof.jl**, **Modolib.jl**, **FPBH.jl** and **CPLEXExtensions.jl**) on the local machine:

```
Pkg.clone("https://github.com/aritrasep/FPBHCPLEX.jl")
Pkg.build("FPBHCPLEX")
```

This will however not install **Modoplots.jl**, which must be installed separately if desired.

2.2 If CPLEX is not available:

If CPLEX is not available, FPBH.jl can be installed instead of FPBHCPLEX.jl. Once, `Julia v0.6.0` has been properly installed, the following instructions in a **Julia** terminal will install **FPBH.jl** and its dependencies (**Modof.jl**, and **Modolib.jl**) on the local machine:

```
Pkg.clone("https://github.com/aritrasep/FPBH.jl")
Pkg.build("FPBH")
```

FPBH.jl automatically installs GLPK by default, however any LP solver supported by MathProgBase.jl can be used as the underlying LP solver. If the user desires to use a LP solver other than GLPK, it must be separately installed. For example: if the user desires to use Clp, the following additional instructions in a **Julia** terminal will do so:

```
Pkg.add("Clp")
```

This does not install **Modoplots.jl**, which must be installed separately if desired.

3 Using FPBH.jl / FPBHCPLEX.jl:

Providing the following multiobjective mixed integer linear program as a JuMP Model:

$$\min x_1 + x_2 + y_1 + y_2 \tag{1}$$
$$\min \ -x_1 - x_2 - y_1 - y_2 \tag{2}$$
$$\min \ -x_1 - 2x_2 - y_1 - 2y_2 \tag{3}$$
$$\text{s.t. } x_1 + x_2 \leq 1 \tag{4}$$
$$y_1 + 2y_2 \geq 1 \tag{5}$$
$$x_1, x_2 \in \{0, 1\} \tag{6}$$
$$y_1, y_2 \geq 0 \tag{7}$$

Note: All objective functions must be minimizations. We are going to soon support both minimization and maximization

```
In [119]: using Modof, JuMP, FPBH
```

3.1 Using JuMP.jl Model

```
In [120]: # Creating the Model

          model = ModoModel() # Creating an empty Model
          @variable(model, x[1:2], Bin)
          @variable(model, y[1:2] >= 0.0)
          objective!(model, 1, :Min, x[1] + x[2] + y[1] + y[2])
          objective!(model, 2, :Min, - x[1] - x[2] - y[1] - y[2])
          objective!(model, 3, :Min, - x[1] - 2x[2] - y[1] - 2y[2])
          @constraint(model, x[1] + x[2] <= 1)
          @constraint(model, y[1] + 2y[2] >= 1)

          # Using GLPK as the underlying LP Solver, and imposing a maximum timelimit of 10.0

          @time solutions = fpbh(model, timelimit=10.0)

  0.084952 seconds (36.71 k allocations: 3.010 MiB)


Out[120]: 4-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
```

```
In [121]: # Writing nondominated frontier to a file

          write_nondominated_frontier(solutions, "Nondominated_frontier.txt")
```

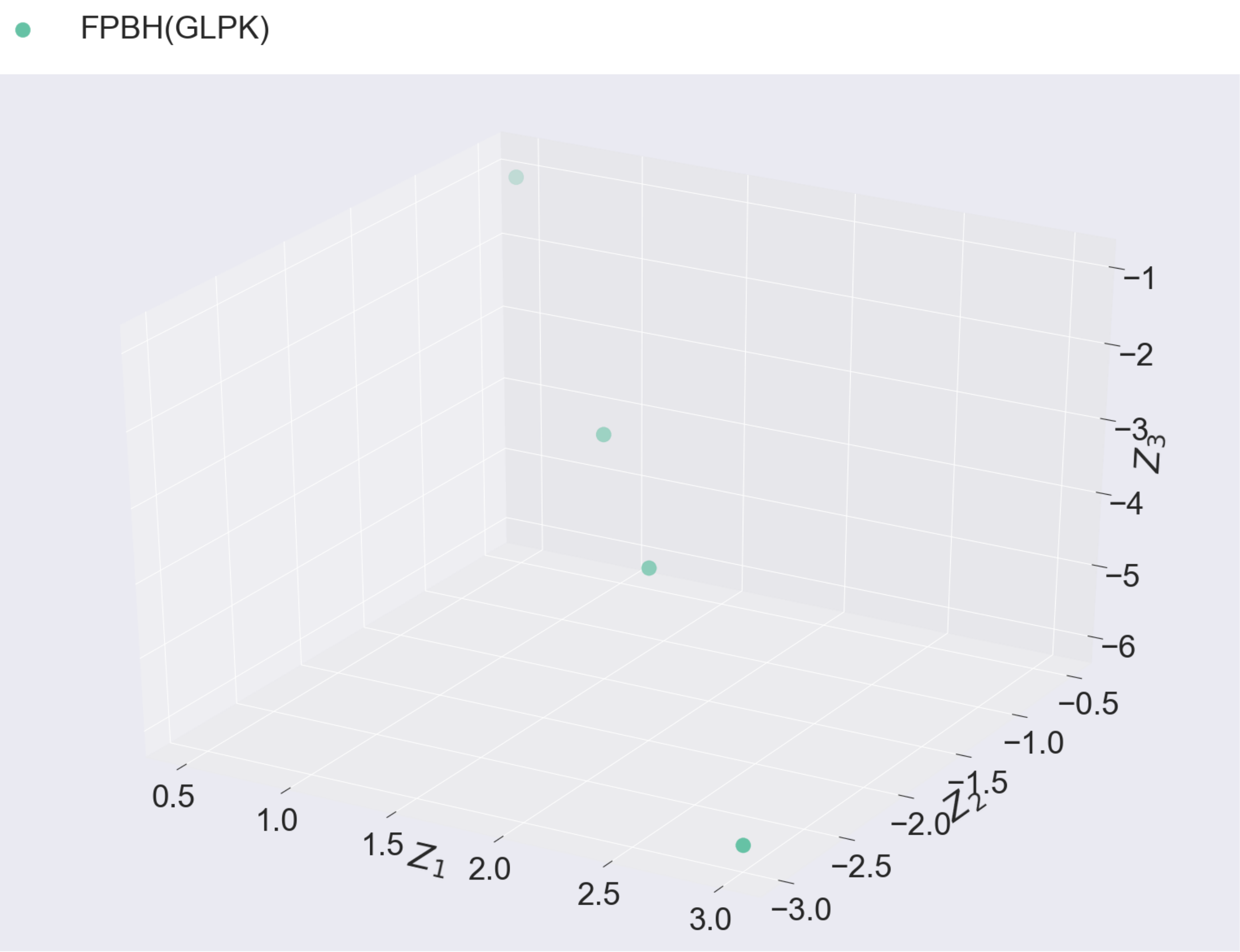
```
In [122]: # Writing nondominated solutions to a file

          write_nondominated_sols(solutions, "Nondominated_solutions.txt")
```

```
In [123]: # Nondominated frontier

          nondominated_frontier = wrap_sols_into_array(solutions)

Out[123]: 4×3 Array{Float64,2}:
           0.5  -0.5  -1.0
           1.5  -1.5  -3.0
           2.0  -2.0  -4.0
           3.0  -3.0  -6.0
```

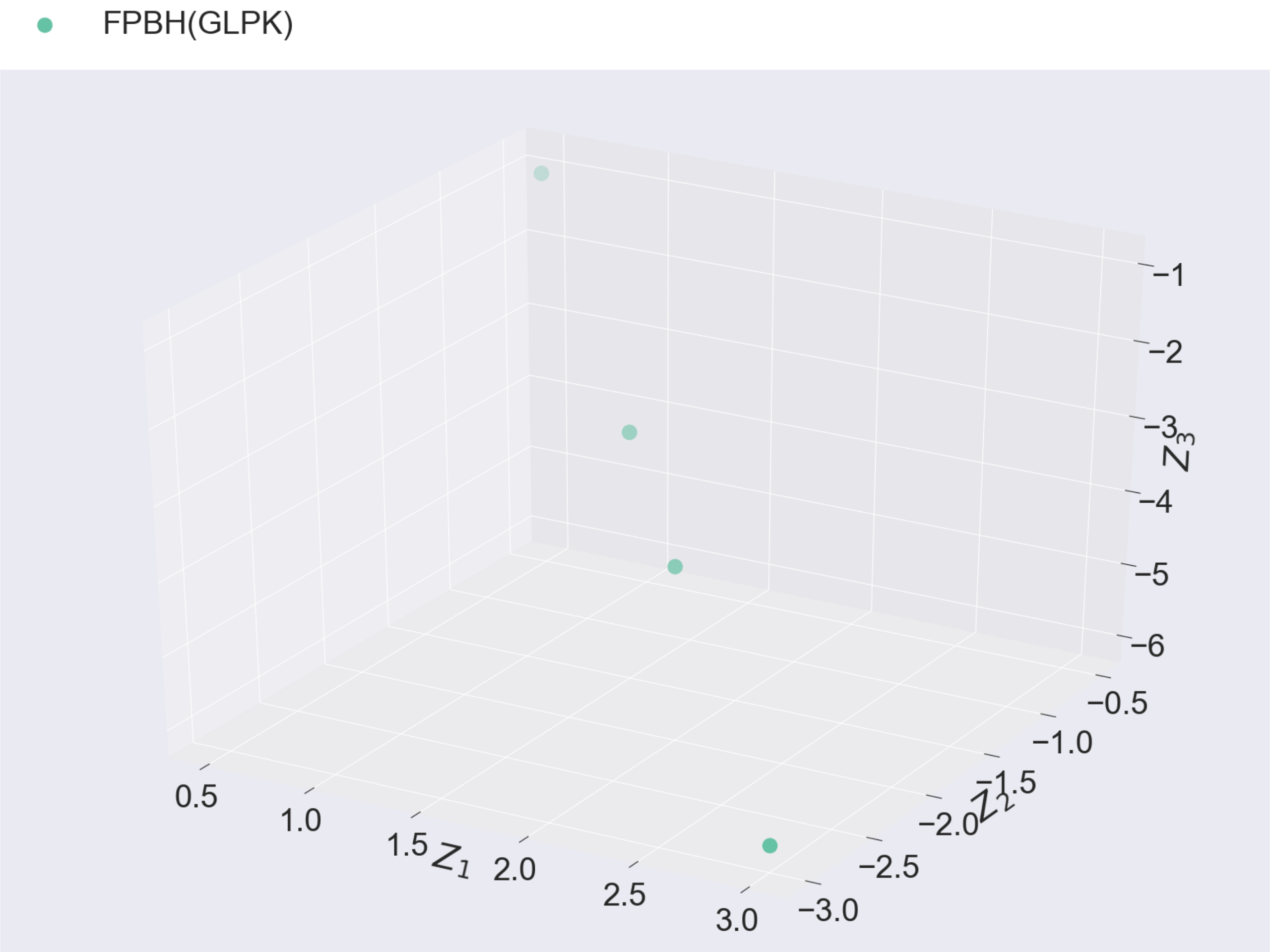
```
In [124]: hypervolume = compute_hypervolume_of_a_discrete_frontier(nondominated_frontier)
          println("Hypervolume of the nondominated frontier = $(hypervolume)")

Hypervolume of the nondominated frontier = 5.5
```

```
In [125]: # Plotting the nondominated frontier - Modoplots.jl must be installed

          using Modoplots
```

```
In [126]: plt_discrete_non_dom_frntr([nondominated_frontier], ["FPBH(GLPK)"]) # Only for IJulia
```



```
In [127]: plt_discrete_non_dom_frntr([nondominated_frontier], ["FPBH(GLPK)"], false, "Plot.eps")
```



```
In [128]: # Using Clp as the underlying LP Solver - Clp.jl must be installed

          using Clp

          @time solutions = fpbh(model, lp_solver=ClpSolver(), timelimit=10.0)

  6.809012 seconds (1.47 M allocations: 109.353 MiB, 1.03% gc time)


Out[128]: 48-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 0.0, 1.00009e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 1.50013e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 2.50022e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 6.00053e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
```

```
 ⋮
 Modof.MOPSolution([0.0, 1.0, 1.0, 1.0], [3.0, -3.0, -5.0])
 Modof.MOPSolution([0.0, 1.0, 1.0, 1.0], [3.0, -3.0, -5.0])
 Modof.MOPSolution([0.0, 1.0, 0.0, 3.0], [4.0, -4.0, -8.0])
 Modof.MOPSolution([0.0, 1.0, 0.0, 4.0], [5.0, -5.0, -10.0])
 Modof.MOPSolution([0.0, 0.0, 0.0, 6.0], [6.0, -6.0, -12.0])
 Modof.MOPSolution([0.0, 1.0, 0.0, 6.0], [7.0, -7.0, -14.0])
 Modof.MOPSolution([0.0, 0.0, 0.0, 9.0], [9.0, -9.0, -18.0])
 Modof.MOPSolution([0.0, 1.0, 0.0, 9.0], [10.0, -10.0, -20.0])
 Modof.MOPSolution([0.0, 0.0, 0.0, 6.25e13], [6.25e13, -6.25e13, -1.25e14])
 Modof.MOPSolution([0.0, 1.0, 0.0, 6.25e13], [6.25e13, -6.25e13, -1.25e14])
 Modof.MOPSolution([0.0, 1.0, 0.0, 2.5e14], [2.5e14, -2.5e14, -5.0e14])
 Modof.MOPSolution([0.0, 1.0, 0.0, 2.8125e14], [2.8125e14, -2.8125e14, -5.625e14])
```

```
In [129]: # Using FPBHCPLEX.jl - FPBHCPLEX.jl must be installed

          using FPBHCPLEX

          @time solutions = fpbhcplex(model, timelimit=10.0)
```

```
  0.236318 seconds (83.07 k allocations: 7.798 MiB)
```

```
Out[129]: 4-element Array{Modof.MOPSolution,1}:
 Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
 Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
 Modof.MOPSolution([0.0, 0.0, 0.0, 2.0], [2.0, -2.0, -4.0])
 Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
```

3.2 Using LP File Format

Providing the following multiobjective mixed integer linear program as a LP file:

$$\min x_1 + x_2 + y_1 + y_2 \tag{8}$$

$$\min \; -x_1 - x_2 - y_1 - y_2 \tag{9}$$

$$\min \; -x_1 - 2x_2 - y_1 - 2y_2 \tag{10}$$

$$\text{s.t. } x_1 + x_2 \leq 1 \tag{11}$$

$$y_1 + 2y_2 \geq 1 \tag{12}$$

$$x_1, x_2 \in \{0, 1\} \tag{13}$$

$$y_1, y_2 \geq 0 \tag{14}$$

1. All objective functions must be minimizations.
2. The first objective function should follow the convention of LP format of single objective optimization problem
3. **The other objective functions should be added as constraints with RHS = 0, at the end of the constraint matrix in the respective order**
4. Variables and constraints should follow the convention of LP format of single objective optimization problem

```
In [130]: # Writing the LP file of the above multiobjective mixed integer program to Test.lp

          write("Test.lp", "\\ENCODING=ISO-8859-1
          \\Problem name: TestInstance
```

```
         Minimize
          obj: x1 + x2 + x3 + x4
         Subject To
          c1: x1 + x2 <= 1
          c2: x3 + 2 x4 >= 1
          c3: - x1 - x2 - x3 - x4  = 0
          c4: - x1 - 2 x2 - x3 - 2 x4  = 0
         Binaries
          x1  x2
         End\n")
```

```
Out[130]: 218
```

```
In [131]: # Using GLPK as the underlying LP Solver, and imposing a maximum timelimit of 10.0

          @time solutions = fpbh("Test.lp", [:Min, :Min], timelimit=10.0)
```

```
Reading problem data from 'Test.lp'...
4 rows, 4 columns, 12 non-zeros
2 integer variables, all of which are binary
13 lines were read
  0.107765 seconds (36.21 k allocations: 2.963 MiB)
```

```
Out[131]: 4-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 2.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
```

```
In [132]: # Using Clp as the underlying LP Solver

          @time solutions = fpbh("Test.lp", [:Min, :Min], lp_solver=ClpSolver(), timelimit=10.0)
```

```
Reading problem data from 'Test.lp'...
4 rows, 4 columns, 12 non-zeros
2 integer variables, all of which are binary
13 lines were read
  6.822883 seconds (1.64 M allocations: 120.602 MiB, 1.26% gc time)
```

```
Out[132]: 52-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 1.00009e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 1.50013e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 2.75024e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 4.5004e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 6.50058e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
```

```
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           ⋮
           Modof.MOPSolution([0.0, 1.0, 0.5, 1.0], [2.5, -2.5, -4.5])
           Modof.MOPSolution([0.0, 1.0, 0.5, 1.0], [2.5, -2.5, -4.5])
           Modof.MOPSolution([0.0, 1.0, 0.5, 1.0], [2.5, -2.5, -4.5])
           Modof.MOPSolution([0.0, 1.0, 0.75, 1.0], [2.75, -2.75, -4.75])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
           Modof.MOPSolution([0.0, 1.0, 1.0, 1.0], [3.0, -3.0, -5.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 3.0], [4.0, -4.0, -8.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 4.0], [5.0, -5.0, -10.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 5.0], [6.0, -6.0, -12.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 8.0], [9.0, -9.0, -18.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 6.25e13], [6.25e13, -6.25e13, -1.25e14])
           Modof.MOPSolution([0.0, 1.0, 0.0, 6.25e13], [6.25e13, -6.25e13, -1.25e14])
```

```
In [133]: # Using FPBHCPLEX.jl

          @time solutions = fpbhcplex("Test.lp", [:Min, :Min], timelimit=10.0)
```

```
Reading problem data from 'Test.lp'...
4 rows, 4 columns, 12 non-zeros
2 integer variables, all of which are binary
13 lines were read
  0.158653 seconds (50.35 k allocations: 4.735 MiB)
```

```
Out[133]: 5-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 2.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 3.0], [4.0, -4.0, -8.0])
```

3.3 Using MPS File Format

Providing the following multiobjective mixed integer linear program as a MPS file:

$$\min x_1 + x_2 + y_1 + y_2 \tag{15}$$

$$\min \; -x_1 - x_2 - y_1 - y_2 \tag{16}$$

$$\min \; -x_1 - 2x_2 - y_1 - 2y_2 \tag{17}$$

$$\text{s.t. } x_1 + x_2 \leq 1 \tag{18}$$

$$y_1 + 2y_2 \geq 1 \tag{19}$$

$$x_1, x_2 \in \{0, 1\} \tag{20}$$

$$y_1, y_2 \geq 0 \tag{21}$$

1. All objective functions must be minimizations.
2. The first objective function should follow the convention of MPS format of single objective optimization problem
3. **The other objective functions should be added as constraints with RHS = 0, at the end of the constraint matrix in the respective order**
4. Variables and constraints should follow the convention of MPS format of single objective optimization problem

```
In [134]: # Writing the MPS file of the above multiobjective mixed integer program to Test.mps
```

```
          write("Test.mps", "NAME   TestInstance
          ROWS
           N  OBJ
           L  CON1
           G  CON2
           E  CON3
           E  CON4
          COLUMNS
              MARKER    'MARKER'                 'INTORG'
              VAR1  CON1  1
              VAR1  CON3  -1
              VAR1  CON4  -1
              VAR1  OBJ  1
              VAR2  CON1  1
              VAR2  CON3  -1
              VAR2  CON4  -2
              VAR2  OBJ  1
              MARKER    'MARKER'                 'INTEND'
              VAR3  CON2  1
              VAR3  CON3  -1
              VAR3  CON4  -1
              VAR3  OBJ  1
              VAR4  CON2  2
              VAR4  CON3  -1
              VAR4  CON4  -2
              VAR4  OBJ  1
          RHS
              rhs    CON1    1
              rhs    CON2    1
              rhs    CON3    0
              rhs    CON4    0
          BOUNDS
            UP BOUND VAR1 1
            UP BOUND VAR2 1
            PL BOUND VAR3
            PL BOUND VAR4
          ENDATA\n")
```

```
Out[134]: 635
```

```
In [135]: # Using GLPK as the underlying LP Solver, and imposing a maximum timelimit of 10.0

          @time solutions = fpbh("Test.mps", [:Min, :Min], timelimit=10.0)
```

```
Reading problem data from 'Test.mps'...
Problem: TestInstance
Objective: OBJ
5 rows, 4 columns, 16 non-zeros
2 integer variables, all of which are binary
37 records were read
One free row was removed
  0.110264 seconds (36.29 k allocations: 2.968 MiB)
```

```
Out[135]: 4-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
```

```
In [136]: # Using Clp as the underlying LP Solver

          @time solutions = fpbh("Test.mps", [:Min, :Min], lp_solver=ClpSolver(), timelimit=10.0)
```

```
Reading problem data from 'Test.mps'…
Problem: TestInstance
Objective: OBJ
5 rows, 4 columns, 16 non-zeros
2 integer variables, all of which are binary
37 records were read
One free row was removed
  6.788158 seconds (1.18 M allocations: 89.422 MiB, 0.98% gc time)
```

```
Out[136]: 51-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 1.00009e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 1.25011e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 1.50013e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 3.25029e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 0.0, 4.00036e-12, 1.0], [1.0, -1.0, -2.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.5], [1.5, -1.5, -3.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 2.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           ⋮
           Modof.MOPSolution([0.0, 1.0, 0.75, 1.0], [2.75, -2.75, -4.75])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
           Modof.MOPSolution([0.0, 1.0, 1.0, 1.0], [3.0, -3.0, -5.0])
           Modof.MOPSolution([0.0, 1.0, 1.0, 1.0], [3.0, -3.0, -5.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 3.0], [4.0, -4.0, -8.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 4.0], [5.0, -5.0, -10.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 6.0], [6.0, -6.0, -12.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 6.0], [7.0, -7.0, -14.0])
           Modof.MOPSolution([0.0, 0.0, 0.0, 6.25e13], [6.25e13, -6.25e13, -1.25e14])
           Modof.MOPSolution([0.0, 1.0, 0.0, 6.25e13], [6.25e13, -6.25e13, -1.25e14])
           Modof.MOPSolution([0.0, 1.0, 0.0, 9.375e13], [9.375e13, -9.375e13, -1.875e14])
           Modof.MOPSolution([0.0, 1.0, 0.0, 5.3125e14], [5.3125e14, -5.3125e14, -1.0625e15])
```

```
In [137]: # Using FPBHCPLEX.jl

          @time solutions = fpbhcplex("Test.mps", [:Min, :Min], timelimit=10.0)
```

```
Reading problem data from 'Test.mps'…
Problem: TestInstance
```

```
Objective: OBJ
5 rows, 4 columns, 16 non-zeros
2 integer variables, all of which are binary
37 records were read
One free row was removed
  0.100721 seconds (37.32 k allocations: 3.481 MiB)
```

```
Out[137]: 3-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 0.0, 0.0, 0.5], [0.5, -0.5, -1.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 1.0], [2.0, -2.0, -4.0])
           Modof.MOPSolution([0.0, 1.0, 0.0, 2.0], [3.0, -3.0, -6.0])
```

3.4 Solving biobjective mixed binary programs from literature:

```
In [138]: using Modolib
```

```
In [139]: instance, true_frontier = read_bomip_hadi(8) # Eighth instance
          @time solutions = fpbh(instance, timelimit=10.0)
```

```
  1.664441 seconds (2.68 M allocations: 185.816 MiB, 8.23% gc time)
```

```
Out[139]: 127-element Array{Modof.BOPSolution,1}:
           Modof.BOPSolution([1.59413, 0.0, 0.0, 0.0, 1.14704, 2.17134, 0.0, 0.0, 2.15108, 0.0  …  0.0, 1.0, 0.0, 0.0,
           Modof.BOPSolution([1.96505, 0.0, 0.0, 0.0, 0.0, 1.60246, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 0.0, 0.0, 1.0, 1.
           Modof.BOPSolution([1.59413, 0.0, 0.0, 0.0, 1.14704, 2.17134, 0.0, 0.0, 2.15108, 0.0  …  0.0, 1.0, 0.0, 0.0,
           Modof.BOPSolution([1.96505, 0.0, 0.0, 0.0, 0.0, 1.60246, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 0.0, 0.0, 1.0, 1.
           Modof.BOPSolution([1.59413, 0.0, 0.0, 0.0, 1.14704, 2.17134, 0.0, 0.0, 2.15108, 0.0  …  0.0, 1.0, 1.0, 0.0,
           Modof.BOPSolution([1.15065, 0.0, 0.0, 0.0, 0.970551, 2.13987, 0.0, 0.0, 2.08841, 0.0  …  0.0, 1.0, 1.0, 0.0
           Modof.BOPSolution([1.94684, 0.0, 0.0, 0.0, 0.0, 1.61594, 0.0, 0.0, 0.189943, 0.0  …  0.0, 1.0, 1.0, 0.0, 1.
           Modof.BOPSolution([1.96505, 0.0, 0.0, 0.0, 0.0, 1.60246, 0.0, 0.0, 9.27699e-15, 0.0  …  0.0, 1.0, 1.0, 0.0,
           Modof.BOPSolution([1.96505, 0.0, 0.0, 0.0, 0.0, 1.60246, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 0.0, 1.0, 1.
           Modof.BOPSolution([1.94806, 0.0, 0.0, 0.0, 0.0, 1.58672, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 0.0, 1.0, 1.
           Modof.BOPSolution([1.92994, 0.0, 0.0, 0.0, 0.0, 1.56995, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 0.0, 1.0, 1.
           Modof.BOPSolution([1.8703, 0.0, 0.0, 0.0, 0.0, 1.51471, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 0.0, 1.0, 1.0
           Modof.BOPSolution([1.44569, 0.0, 0.0, 0.0, 0.0, 1.12143, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 0.0, 1.0, 1.
           ⋮
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 1.30763, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 1.30763, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 1.50499, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 1.38623, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0
           Modof.BOPSolution([0.273536, 0.0, 0.0, 0.0, 0.0, 1.54872, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.522055, 0.0, 0.0, 0.0, 0.0, 1.75231, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.512971, 0.0, 0.0, 0.0, 0.0, 1.73647, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.512971, 0.0, 0.0, 0.0, 0.0, 1.73647, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.243542, 0.0, 0.0, 0.0, 0.0, 1.68046, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.243542, 0.0, 0.0, 0.0, 0.0, 1.68046, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 1.64558, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 1.60753, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0
```

```
In [140]: # Nondominated frontier

          nondominated_frontier = wrap_sols_into_array(solutions)
```

```
Out[140]: 127×2 Array{Float64,2}:
           -773.871      9.61621
           -771.082    -13.9046
           -765.871   -169.384
           -763.082   -192.905
           -762.871   -200.384
           -761.121   -200.982
           -760.548   -222.059
           -760.082   -223.905
           -760.082   -223.905
           -759.875   -224.34
           -759.653   -224.805
           -758.925   -226.334
           -753.737   -237.22
              ⋮
            -83.2633  -922.028
            -83.2633  -922.028
            -78.5609  -922.093
            172.399   -924.737
            205.034   -933.078
            219.239   -954.638
            224.341   -957.604
            224.341   -957.604
            231.814   -961.291
            231.814   -961.291
            233.094   -961.417
            233.923   -961.493
```

```
In [141]: # Quality of the frontier w.r.t. true frontier without normalization

          hg, c, mc, ac, u = compute_quality_of_apprx_frontier(nondominated_frontier, true_frontier, true)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

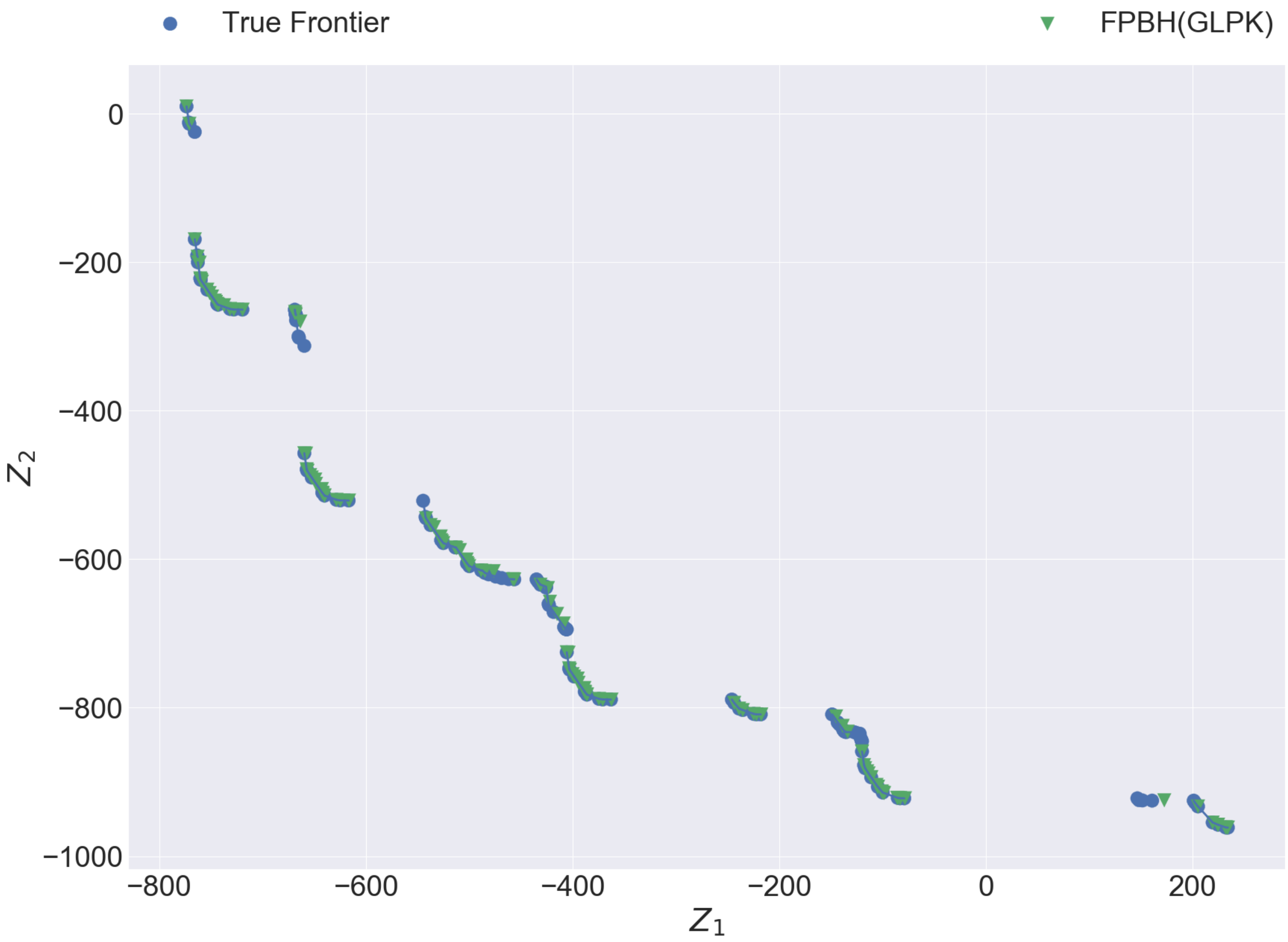
```
    Hypervolume Gap = 0.2770906 %
    Cardinality = 0.0 %
    Maximum Coverage = 32.9022683
    Average Coverage = 3.7708336
    Uniformity = 0.8818898
```

```
In [142]: # Quality of the frontier w.r.t. true frontier with normalization

          hg, c, mc, ac, u = compute_quality_of_norm_apprx_frontier(nondominated_frontier, true_frontier, true)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
```

```
          Average Coverage = $ac
          Uniformity = $u")
```

```
Hypervolume Gap = 0.2770906 %
Cardinality = 0.0 %
Maximum Coverage = 0.0338675
Average Coverage = 0.0038224
Uniformity = 0.9333333
```

```
In [143]: # Comparing the nondominated frontier of FPBH with the true frontier

          plt_non_dom_frntr_bomip([true_frontier, nondominated_frontier], ["True Frontier", "FPBH(GLPK)"]) # Only in 1
```



3.5 Solving biobjective uncapacitated facility location problems from literature:

```
In [144]: instance, true_frontier = read_bouflp_hadi(1) # First instance
          @time solutions = fpbh(instance, lp_solver=ClpSolver(), timelimit=10.0)
```

```
 10.019990 seconds (1.65 M allocations: 214.442 MiB, 0.99% gc time)
```

```
Out[144]: 37-element Array{Modof.BOPSolution,1}:
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           ⋮
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
           Modof.BOPSolution([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
```

```
In [145]: # Nondominated frontier

          nondominated_frontier = wrap_sols_into_array(solutions)
```

```
Out[145]: 37×2 Array{Float64,2}:
                9.33036e5       8.77978e5
                9.34538e5       8.68572e5
                9.36123e5       8.63268e5
                9.37075e5       8.6071e5
                9.37402e5       8.60121e5
                9.37904e5       8.58618e5
                9.38011e5       8.58497e5
                9.38292e5       8.58399e5
                9.39099e5       8.57451e5
                9.39192e5       8.55825e5
                9.39695e5       8.54322e5
                9.40145e5       8.53267e5
                9.40472e5       8.52678e5
                ⋮
                9.93493e5       8.22559e5
                9.93819e5       8.21971e5
                9.94137e5       8.21802e5
                9.94853e5       8.14534e5
                9.96889e5       8.14528e5
                9.97207e5  814359.0
                9.98437e5       8.14216e5
```

```
         9.98879e5       8.14014e5
   999297.0              8.13878e5
   999393.0              8.13649e5
         1.00034e6       8.11607e5
         1.00444e6       8.10198e5
```

```
In [146]: # Quality of the frontier w.r.t. true frontier without normalization

          hg, c, mc, ac, u = compute_quality_of_apprx_frontier(nondominated_frontier, true_frontier, true)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

```
    Hypervolume Gap = 15.9534143 %
    Cardinality = 13.7931034 %
    Maximum Coverage = 13011.3534925
    Average Coverage = 2540.3726614
    Uniformity = 0.6756757
```

```
In [147]: # Quality of the frontier w.r.t. true frontier with normalization

          hg, c, mc, ac, u = compute_quality_of_norm_apprx_frontier(nondominated_frontier, true_frontier, true)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

```
    Hypervolume Gap = 15.9534143 %
    Cardinality = 13.7931034 %
    Maximum Coverage = 0.1540316
    Average Coverage = 0.0311132
    Uniformity = 0.6756757
```

3.6 Solving multiobjective assignment problems from literature:

```
In [148]: instance, true_frontier = read_moap_kirlik(3, 5, 1) # 3 objective, 5 jobs, first instance
          @time solutions = fpbh(instance, timelimit=10.0)
```

```
  0.116379 seconds (43.01 k allocations: 4.678 MiB)
```

```
Out[148]: 8-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
           Modof.MOPSolution([0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
```

```
 Modof.MOPSolution([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0  …  0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0,
 Modof.MOPSolution([0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0,
 Modof.MOPSolution([0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
 Modof.MOPSolution([0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0  …  0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0,
 Modof.MOPSolution([0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0  …  0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```
In [149]: # Nondominated frontier

          nondominated_frontier = wrap_sols_into_array(solutions)
```

```
Out[149]: 8×3 Array{Float64,2}:
           16.0  61.0  47.0
           23.0  43.0  44.0
           24.0  39.0  45.0
           28.0  33.0  58.0
           29.0  29.0  59.0
           35.0  49.0  39.0
           43.0  51.0  31.0
           45.0  33.0  34.0
```

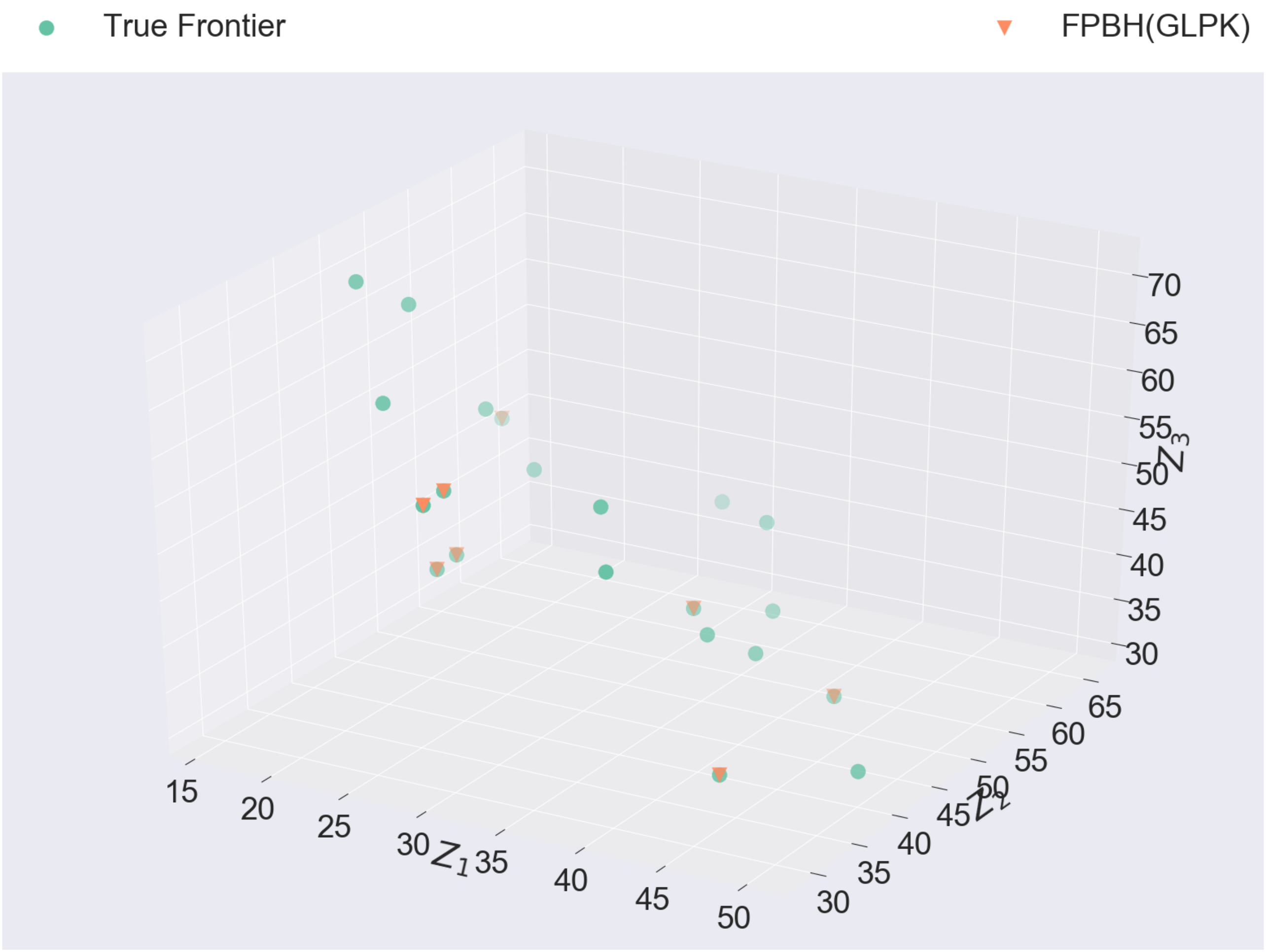
```
In [150]: # Quality of the frontier w.r.t. true frontier without normalization

          hg, c, mc, ac, u = compute_quality_of_apprx_frontier(nondominated_frontier, true_frontier)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

```

    Hypervolume Gap = 5.4025045 %
    Cardinality = 38.0952381 %
    Maximum Coverage = 19.7484177
    Average Coverage = 11.1556969
    Uniformity = 1.625
```

```
In [151]: # Quality of the frontier w.r.t. true frontier with normalization

          hg, c, mc, ac, u = compute_quality_of_norm_apprx_frontier(nondominated_frontier, true_frontier)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

```

    Hypervolume Gap = 5.4025045 %
    Cardinality = 38.0952381 %
    Maximum Coverage = 0.5322991
    Average Coverage = 0.3072659
    Uniformity = 1.625
```

```
In [152]: # Comparing the nondominated frontier of FPBH with the true frontier

          plt_discrete_non_dom_frntr([true_frontier, nondominated_frontier], ["True Frontier", "FPBH(GLPK)"]) # Only i
```



3.7 Solving multiobjective knapsack problems from literature:

```
In [153]: instance, true_frontier = read_mokp_kirlik(3, 10, 1) # 3 objective, 10 variables, first instance
          @time solutions = fpbhcplex(instance, timelimit=10.0)

  0.072766 seconds (78.30 k allocations: 16.146 MiB)


Out[153]: 7-element Array{Modof.MOPSolution,1}:
           Modof.MOPSolution([1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0], [-3394.0, -3817.0, -3408.0])
           Modof.MOPSolution([0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0], [-3042.0, -4627.0, -3189.0])
           Modof.MOPSolution([1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0], [-2997.0, -3539.0, -3509.0])
           Modof.MOPSolution([0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0], [-2854.0, -4636.0, -3076.0])
           Modof.MOPSolution([0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0], [-2854.0, -3570.0, -3714.0])
           Modof.MOPSolution([1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0], [-2706.0, -3857.0, -3304.0])
           Modof.MOPSolution([1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0], [-2518.0, -3866.0, -3191.0])
```

```
In [154]: # Nondominated frontier

          nondominated_frontier = wrap_sols_into_array(solutions)
```

```
Out[154]: 7×3 Array{Float64,2}:
           -3394.0  -3817.0  -3408.0
           -3042.0  -4627.0  -3189.0
           -2997.0  -3539.0  -3509.0
           -2854.0  -4636.0  -3076.0
           -2854.0  -3570.0  -3714.0
           -2706.0  -3857.0  -3304.0
           -2518.0  -3866.0  -3191.0
```

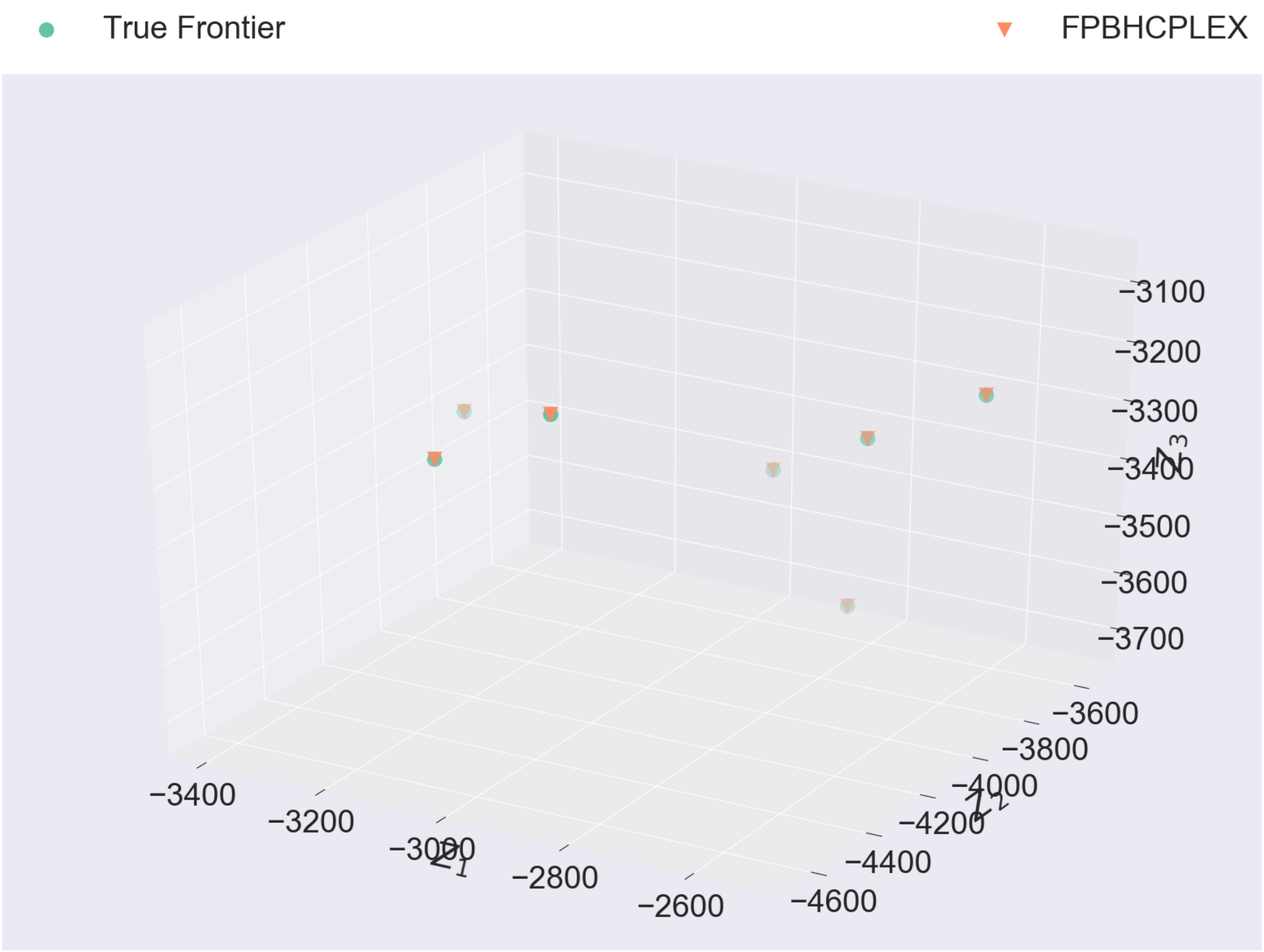
```
In [155]: # Quality of the frontier w.r.t. true frontier without normalization

          hg, c, mc, ac, u = compute_quality_of_apprx_frontier(nondominated_frontier, true_frontier)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

```

    Hypervolume Gap = 0.0 %
    Cardinality = 100.0 %
    Maximum Coverage = 0.0
    Average Coverage = 0.0
    Uniformity = 0.0
```

```
In [156]: # Quality of the frontier w.r.t. true frontier with normalization

          hg, c, mc, ac, u = compute_quality_of_norm_apprx_frontier(nondominated_frontier, true_frontier)
          println("
              Hypervolume Gap = $hg %
              Cardinality = $c %
              Maximum Coverage = $mc
              Average Coverage = $ac
              Uniformity = $u")
```

```

    Hypervolume Gap = 0.0 %
    Cardinality = 100.0 %
    Maximum Coverage = 0.0
    Average Coverage = 0.0
    Uniformity = 0.0
```

```
In [157]: # Comparing the nondominated frontier of FPBHCPLEX with the true frontier

          plt_discrete_non_dom_frntr([true_frontier, nondominated_frontier], ["True Frontier", "FPBHCPLEX"]) # Only i
```



4 Experiment

MOO_JA_2_Sup.jl contains the script for running the whole experiment and generating all the plots. Further, it also contains the Experimental Results and the resulting nondominated frontiers. This repository can be cloned as `git clone https://github.com/aritrasep/MOO_JA_2_Sup.jl`.

4.1 Dependencies:

1. Julia v0.6.0
2. Clp - v1.15
3. SCIP - v4.0.0
4. Gurobi - v7.5
5. CPLEX - v12.7
6. FPBHCPLEX.jl
7. MDLS must also be compiled and the respective path of its binaries must be exported as `export PATH="path to mdls binaries:$PATH"`.

4.2 Running the whole Experiment:

Once `Julia`, `Clp`, `SCIP`, `Gurobi` `CPLEX`, `FPBHCPLEX.jl` and `MDLS` has been properly installed, start a `julia` terminal inside the `src` directory with atleast 4 workers using `julia -p 4` and type the following to execute the whole experiment:

```
include("Experimental_Run.jl")
run_experiment()
```

The above commands using Experimental_Run.jl, will run all experiments related to MDLS, V1, V2, V3, V4 and V5 T1 in parallel using 4 (or more) workers. However, all experiments related to V5 T2, V5 T3 and V5 T4 will be run serially.

4.3 Experimental Results:

All results will be written as csv files inside `results` directory. Results is the detailed experimental results for our paper.

4.4 Nondominated Frontiers:

1. Nondominated Frontiers:
    1. V1
    2. V2
    3. V3
    4. V4
    5. V5:
        1. 1 Thread
        2. 2 Threads
        3. 3 Threads
        4. 4 Threads
    6. True / Reference Frontiers

4.5 Summarizing Results and Generating Plots:

Modoplots.jl must be installed on the local machine. If it has not been installed, it can be done so using the following instructions in a **Julia** terminal:

```
Pkg.clone("https://github.com/aritrasep/Modoplots.jl")
Pkg.build("Modoplots")
```

Start a `Julia` terminal inside the `src` directory and type the following in the terminal to generate plots used in the article

```
include("Summarizing_Results.jl")
```

The above commands using Summarizing_Results.jl, will generate all plots inside the `plots` directory.