# README

Aritra Pal and Hadi Charkhgard  

September 7, 2017

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Understanding the source code:</td>
</tr>
<tr>
<td>1.1</td>
<td>Modof.jl</td>
</tr>
<tr>
<td>1.2</td>
<td>Modolib.jl</td>
</tr>
<tr>
<td>1.3</td>
<td>FPBH.jl</td>
</tr>
<tr>
<td>1.4</td>
<td>CPLEXExtensions.jl</td>
</tr>
<tr>
<td>1.5</td>
<td>FPBHCPLEX.jl</td>
</tr>
<tr>
<td>1.6</td>
<td>Modoplots.jl</td>
</tr>
<tr>
<td>2</td>
<td>Installation:</td>
</tr>
<tr>
<td>2.1</td>
<td>If CPLEX is available:</td>
</tr>
<tr>
<td>2.2</td>
<td>If CPLEX is not available:</td>
</tr>
<tr>
<td>3</td>
<td>Using FPBH.jl / FPBHCPLEX.jl:</td>
</tr>
<tr>
<td>3.1</td>
<td>Using JuMP.jl Model</td>
</tr>
<tr>
<td>3.2</td>
<td>Using LP File Format</td>
</tr>
<tr>
<td>3.3</td>
<td>Using MPS File Format</td>
</tr>
<tr>
<td>3.4</td>
<td>Solving biobjective mixed binary programs from literature:</td>
</tr>
<tr>
<td>3.5</td>
<td>Solving biobjective uncapacitated facility location problems from literature:</td>
</tr>
<tr>
<td>3.6</td>
<td>Solving multiobjective assignment problems from literature:</td>
</tr>
<tr>
<td>3.7</td>
<td>Solving multiobjective knapsack problems from literature:</td>
</tr>
<tr>
<td>4</td>
<td>Experiment</td>
</tr>
<tr>
<td>4.1</td>
<td>Dependencies:</td>
</tr>
<tr>
<td>4.2</td>
<td>Running the whole Experiment:</td>
</tr>
<tr>
<td>4.3</td>
<td>Experimental Results:</td>
</tr>
<tr>
<td>4.4</td>
<td>Nondominated Frontiers:</td>
</tr>
<tr>
<td>4.5</td>
<td>Summarizing Results and Generating Plots:</td>
</tr>
</tbody>
</table>

## 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. **Apí.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 **CPLEX Extensions.jl**

**CPLEX Extensions.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_coeff!, 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 **CPLEX Extensions.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 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:

\[
\begin{align*}
\min & \quad x_1 + x_2 + y_1 + y_2 \\
\min & \quad -x_1 - x_2 - y_1 - y_2 \\
\min & \quad -x_1 - 2y_2 - y_1 - 2y_2 \\
\text{s.t.} & \quad x_1 + x_2 \leq 1 \\
& \quad y_1 + 2y_2 \geq 1 \\
& \quad x_1, x_2 \in \{0, 1\} \\
& \quad y_1, y_2 \geq 0
\end{align*}
\]

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

```julia
model = ModoModel() # Creating an empty Model
@variable(model, x[1:2], Bin)
@variable(model, y[1:2] >= 0.0)
@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

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

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

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

In [123]: # Nondominated frontier

```julia
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

```julia
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, 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, 0.0, 0.0, 2.5e14], [2.5e14, -2.5e14, -5.0e14])
Modof.MOPSolution([0.0, 1.0, 0.0, 2.5e14], [2.8125e14, -2.8125e14, -5.625e14])

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

    using FPBHCPLEX

    @time solutions = fpbhplex(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:

\[
\begin{align*}
\text{min} & \quad x_1 + x_2 + y_1 + y_2 \\
\text{min} & \quad -x_1 - x_2 - y_1 - y_2 \\
\text{min} & \quad -x_1 - 2x_2 - y_1 - 2y_2 \\
\text{s.t.} & \quad x_1 + x_2 \leq 1 \\
& \quad y_1 + 2y_2 \geq 1 \\
& \quad x_1, x_2 \in \{0, 1\} \\
& \quad y_1, y_2 \geq 0
\end{align*}
\]

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

7
Minimize

\[ \text{obj: } x_1 + x_2 + x_3 + x_4 \]

Subject To

\[ c1: x_1 + x_2 \leq 1 \]
\[ c2: x_3 + 2x_4 \geq 1 \]
\[ c3: -x_1 - x_2 - x_3 - x_4 = 0 \]
\[ c4: -x_1 - 2x_2 - x_3 - 2x_4 = 0 \]

Binaries

\[ x_1 x_2 \]

End

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])
## 3.3 Using MPS File Format

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

\[
\begin{align*}
\text{min } & x_1 + x_2 + y_1 + y_2 \\
\text{min } & -x_1 - x_2 - y_1 - y_2 \\
\text{min } & -x_1 - 2x_2 - y_1 - 2y_2 \\
\text{s.t. } & x_1 + x_2 \leq 1 \\
& y_1 + 2y_2 \geq 1 \\
& x_1, x_2 \in \{0, 1\} \\
& y_1, y_2 \geq 0
\end{align*}
\]

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.
write("Test.mps", "NAME  TestInstance
ROWS
N  OBJ
L  CON1
G  CON2
E  CON3
E  CON4
COLUMNS
   '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'     '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"
)

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, 0.0, 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], [1.0, -1.0, -2.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [1.0, 1.0, 2.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [2.0, 2.0, 3.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [3.0, 3.0, 4.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [4.0, 4.0, 5.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [5.0, 5.0, 6.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [6.0, 6.0, 7.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [7.0, 7.0, 8.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [8.0, 8.0, 9.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [9.0, 9.0, 10.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [10.0, 10.0, 11.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [11.0, 11.0, 12.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [12.0, 12.0, 13.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [13.0, 13.0, 14.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [14.0, 14.0, 15.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [15.0, 15.0, 16.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [16.0, 16.0, 17.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [17.0, 17.0, 18.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [18.0, 18.0, 19.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [19.0, 19.0, 20.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [20.0, 20.0, 21.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [21.0, 21.0, 22.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [22.0, 22.0, 23.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [23.0, 23.0, 24.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [24.0, 24.0, 25.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [25.0, 25.0, 26.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [26.0, 26.0, 27.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [27.0, 27.0, 28.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [28.0, 28.0, 29.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [29.0, 29.0, 30.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [30.0, 30.0, 31.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [31.0, 31.0, 32.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [32.0, 32.0, 33.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [33.0, 33.0, 34.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [34.0, 34.0, 35.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [35.0, 35.0, 36.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [36.0, 36.0, 37.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [37.0, 37.0, 38.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [38.0, 38.0, 39.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [39.0, 39.0, 40.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [40.0, 40.0, 41.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [41.0, 41.0, 42.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [42.0, 42.0, 43.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [43.0, 43.0, 44.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [44.0, 44.0, 45.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [45.0, 45.0, 46.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [46.0, 46.0, 47.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [47.0, 47.0, 48.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [48.0, 48.0, 49.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [49.0, 49.0, 50.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [50.0, 50.0, 51.0])
    Modof.MOPSolution([0.0, 0.0, 1.0, 1.0], [51.0, 51.0, 52.0])

In [137]: # Using FPBHCPLEX.jl
    
    @time solutions = fpbhcplex("Test.mps", [:Min, :Min], timelimit=10.0)

Reading problem data from 'Test.mps'...
Problem: TestInstance
3.4 Solving biobjective mixed binary programs from literature:

In [138]: using Modolib

In [139]: instance, true_frontier = read_bomip_hadi(# 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([0.0, 0.0, 0.0, 0.0, 0.0, 1.30763, 0.0, 0.0, 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 -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_bomp([true_frontier, nondominated_frontier], ["True Frontier", "FPBH(GLPK)"]) # Only in IJulia

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}:

<table>
<thead>
<tr>
<th>Solution 1</th>
<th>Solution 2</th>
<th>Solution 3</th>
<th>Solution 4</th>
<th>Solution 5</th>
<th>Solution 6</th>
<th>Solution 7</th>
<th>Solution 8</th>
<th>Solution 9</th>
<th>Solution 10</th>
<th>Solution 11</th>
<th>Solution 12</th>
<th>Solution 13</th>
<th>Solution 14</th>
<th>Solution 15</th>
<th>Solution 16</th>
<th>Solution 17</th>
<th>Solution 18</th>
<th>Solution 19</th>
<th>Solution 20</th>
<th>Solution 21</th>
<th>Solution 22</th>
<th>Solution 23</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td>(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In [145]: # Nondominated frontier

nondominated_frontier = wrap_sols_into_array(solutions)

Out[145]: 37x2 Array{Float64,2}:

| 0.33036e5 | 8.77978e5 |
| 0.34538e5 | 8.68572e5 |
| 0.36123e5 | 8.63268e5 |
| 0.37075e5 | 8.6071e5  |
| 0.37402e5 | 8.60121e5 |
| 0.37904e5 | 8.58618e5 |
| 0.38011e5 | 8.58497e5 |
| 0.38292e5 | 8.58399e5 |
| 0.39059e5 | 8.57451e5 |
| 0.39192e5 | 8.55825e5 |
| 0.39695e5 | 8.54322e5 |
| 0.40145e5 | 8.53267e5 |
| 0.40472e5 | 8.52678e5 |
| 0.93349e5 | 8.22559e5 |
| 0.93819e5 | 8.21971e5 |
| 0.94137e5 | 8.21802e5 |
| 0.94853e5 | 8.14534e5 |
| 0.96889e5 | 8.14528e5 |
| 0.97207e5 | 8.14359 e5 |
| 0.98437e5 | 8.14216e5 |
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, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.0)
  Modof.MOPSolution([0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.0)
  Modof.MOPSolution([0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0], 0.0)
In [149]: # Nondominated frontier

    nondominated_frontier = wrap_sols_into_array(solutions)

Out[149]: 8x3 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 in IJulia

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 FPBHCLEX with the true frontier

    plt_discrete_non_dom_frntr([true_frontier, nondominated_frontier], ["True Frontier", "FPBHCLEX"]) # Only in IJulia
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.