ÖGOR Summer-Workshop for PhD-candidates and Post-Docs

An introduction to Julia and JuMP for Operations Research

Prof. Dr. Xavier Gandibleux

Nantes Université – France
Département Informatique – Faculté des Sciences et Technique

Topic 12

vOptSolver

`vOptGeneric.jl` and `vOptSpecific.jl`

Overview of vOptSolver

An ecosystem for modeling and solving multiobjective linear optimization problems (MOCO, MOIP, MOMIP, MOLP):

- it deals with structured and non-structured optimization problems with at least two objectives
- it integrates several specific and generic exact algorithms for computing efficient solutions
- Natural and intuitive use for mathematicians, informaticians, engineers
- Efficient, flexible, evolutive solver
- Aims to be easy to formulate a problem, to provide data, to solve a problem, to collect the outputs, to analyze the solutions
- Free, open source (MIT licence), multi-platform, reusing existing specifications
- Using usual free (GLPK, Clp/Cbc) and commercial (GUROBI, CPLEX) MILP solvers

Overview of vOptSolver

An ecosystem for modeling and solving multiobjective linear optimization problems (MOCO, MOIP, MOMIP, MOLP):

- it deals with structured and non-structured optimization problems with at least two objectives
- it integrates several specific and generic exact algorithms for computing efficient solutions
- Natural and intuitive use for mathematicians, informaticians, engineers
- Efficient, flexible, evolutive solver
- Aims to be easy to formulate a problem, to provide data, to solve a problem, to collect the outputs, to analyze the solutions
- Free, open source (MIT licence), multi-platform, reusing existing specifications
- Using usual free (GLPK, Clp/Cbc) and commercial (GUROBI, CPLEX) MILP solvers

`vOptGeneric.jl`

is archived and replaced by

`MultiObjectiveAlgorithms.jl`

Getting started with MultiObjectiveAlgorithms

Install:

```
using Pkg
```

```
Pkg.add("JuMP")
Pkg.add("MultiObjectiveAlgorithms")
```

```
Pkg.add("GLPK")
```

Setup:

```
using JuMP
import MultiObjectiveAlgorithms as MOA
```

```
using GLPK
```

Getting started with `MultiObjectiveAlgorithms`

Install:

```
using Pkg
```

```
Pkg.add("JuMP")
Pkg.add("MultiObjectiveAlgorithms")
```

```
Pkg.add("GLPK")
```

Setup:

```
using JuMP
import MultiObjectiveAlgorithms as MOA
```

```
using GLPK
```

Example with `MultiObjectiveAlgorithms`

For the bi-objective unidimensional 01 knapsack problem,

$$\max \left\{ (p^1 x,\ p^2 x) \mid wx \leqslant c,\ x \in \{0,1\}^n \right\}$$

with[1]

$$n = 5$$

$$p^1 = (10,\ 3,\ 6,\ 8,\ 2)$$

$$p^2 = (12,\ 9,\ 11,\ 5,\ 6)$$

$$w = (4,\ 5,\ 2,\ 5,\ 6)$$

$$c = 17$$

compute Y_N, the set of non-dominated points using the ϵ-constraint method.

[1] exercise 10.2, page 290 of *Multicriteria Optimization* (2nd edt), M. Ehrgott, Springer 2005

Setup the data

```
julia> p1 = [10,3,6,8,2]          # coef vct of the obj 1
julia> p2 = [12,9,11,5,6]         # coef vct of the obj 2
julia> w = [4,5,2,5,6]              # coef vct of weights
julia> c = 17                          # nominal capacity
julia> n = length(p1)                   # number of items
```

Setup the model

```
julia> kp = Model( )
julia> @variable(kp, x[1:n],Bin)
julia> @expression(kp, fct1, sum(p1[j]*x[j] for j=1:n))
julia> @expression(kp, fct2, sum(p2[j]*x[j] for j=1:n))
julia> @objective(kp, Max, [fct1, fct2])
julia> @constraint(kp, sum(w[j]*x[j] for j=1:n) ⩽ c)
```

Setup the solver (ϵ-constraint method; step=1)

```
julia> set_optimizer(kp,()->MOA.Optimizer(GLPK.Optimizer))
julia> set_attribute(kp, MOA.Algorithm(),
                         MOA.EpsilonConstraint())
```

Solve and display results (1/2)

Invoking the solver:

```
julia> optimize!(kp)
```

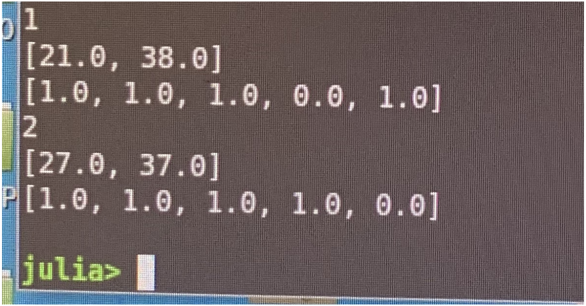
Summary of the resolution:

```
julia> solution_summary(kp)
```

Displaying the results (X_E and Y_N):

```
julia> for i in 1:result_count(kp)
julia>     println( i )
julia>     println( objective_value(kp; result = i) )
julia>     println( value.(x; result = i) )
julia> end
```

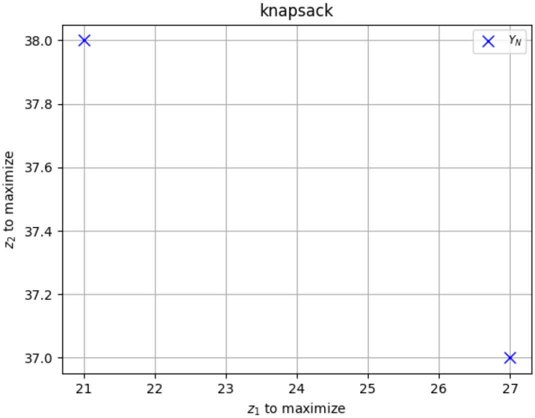
Solve and display results (2/2)



Plot results (1/2)

```
julia> using PyPlot
julia>
julia> cardYN = result_count(kp)
julia> z1 = [value(fct1; result = i) for i in 1:cardYN]
julia> z2 = [value(fct2; result = i) for i in 1:cardYN]
julia>
julia> PyPlot.title("Knapsack")
julia> PyPlot.xlabel(L"$z_1$ to maximize")
julia> PyPlot.ylabel(L"$z_2$ to maximize")
julia> grid()
julia> plot(z1,z2,"bx",markersize="8",label=L"$Y_N$")
julia> legend(loc=1,fontsize="small")
julia>
julia> show()
```

Plot results (2/2)



More about...

`JuMP`:

`https://jump.dev/`

`MultiObjectiveAlgorithms`:

`https://github.com/jump-dev/MultiObjectiveAlgorithms.jl`

Examples:

`https://jump.dev/JuMP.jl/stable/tutorials/linear/multi_objective_examples/`