Julia tutorial (2)

Metaheuristic Implementation

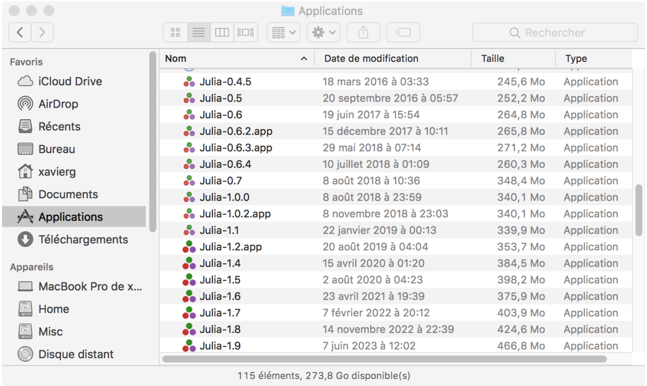
Prof. Dr. Xavier Gandibleux

Nantes Université, France

June 2024

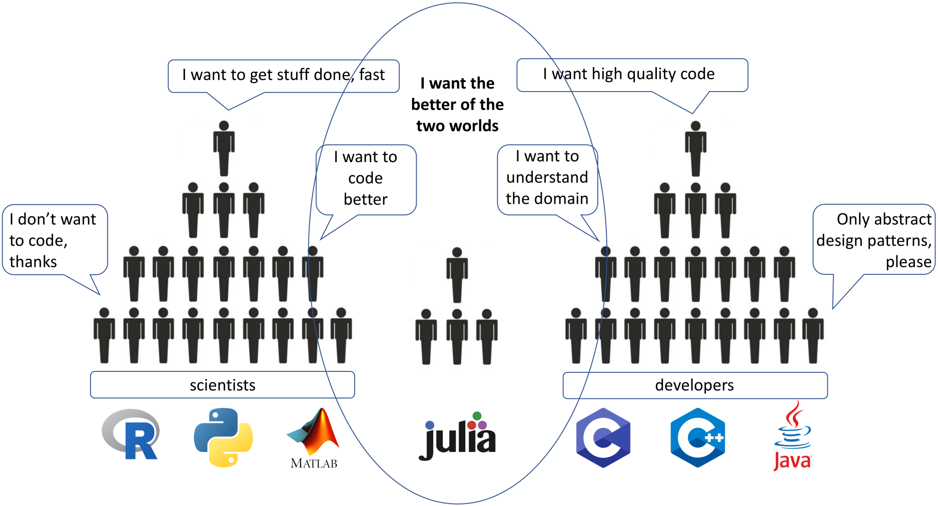
Julia

My story with Julia and JuMP: ROADEF’2016 (Feb 2016)



Julia

A programming language for optimization



Julia

LLVM: middleman between source code and compiled native code

source code

The **Julia** Programming Language

```
julia> 1+2
3
```

byte code
(agnostic of the CPU or GPU architecture)

```
julia> @code_llvm 1+2
;  @ int.jl:87 within `+`
define i64 @"julia_+_115"(i64 signext %0, i64 signext %1) #0 {
top:
  %2 = add i64 %1, %0
  ret i64 %2
}
```

The **LLVM** Compiler Infrastructure

native code
(assembly code just before 0/1)

```
julia> @code_native 1+2
        .section        __TEXT,__text,regular,pure_instructions
        .build_version macos, 13, 0
        .globl  "_julia_+_150"                  ; -- Begin function julia_+_150
        .p2align        2
"_julia_+_150":                         ; @"julia_+_150"
; ┌ @ int.jl:87 within `+`
        .cfi_startproc
; %bb.0:                                ; %top
        add     x0, x1, x0
        ret
        .cfi_endproc
; └
                                        ; -- End function
.subsections_via_symbols
```

The **CPU** or **GPU** architecture

```
01101100
01101111
01110110
01100101
```

Problem Support

Set Packing Problem (SPP):

- $J = \{1, \ldots, n\}$
- $I = \{1, \ldots, m\}$
- $$\left[\begin{array}{ll} Max\ z(x) = & \displaystyle\sum_{j \in J} c_j x_j \\ & \displaystyle\sum_{j \in J} a_{i,j} x_j \leqslant 1, \forall i \in I \\ & x_j \in \{0, 1\} \quad , \forall j \in J \\ & a_{i,j} \in \{0, 1\} \ , \forall i \in I, \forall j \in J \end{array}\right] \quad (SPP)$$

Problem Support

Example of a numerical instance:

```
7 6
7 2 4 6 3 1
3
1 2 3
3
2 5 6
3
1 2 4
3
2 3 5
2
4 6
3
3 4 5
2
1 5
```

Format OR-library:

`http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

number of rows (m), number of columns (n)

the cost of each column c(j),j=1,...,n

for each row i (i=1,...,m): the number of columns which cover row i

followed by a list of the columns which cover row i

Problem Support

Example of a numerical instance:

$$
\left[
\begin{array}{lccccccccccccccc}
\max z & = & 7x_1 & + & 2x_2 & + & 4x_3 & + & 6x_4 & + & 3x_5 & + & x_6 & & \\
s/c & & x_1 & + & x_2 & + & x_3 & & & & & & & \leqslant & 1 \\
 & & x_2 & + & x_5 & + & x_6 & & & & & & & \leqslant & 1 \\
 & & x_1 & + & x_2 & + & x_4 & & & & & & & \leqslant & 1 \\
 & & x_2 & + & x_3 & + & x_5 & & & & & & & \leqslant & 1 \\
 & & x_4 & + & x_6 & & & & & & & & & \leqslant & 1 \\
 & & x_3 & + & x_4 & + & x_5 & & & & & & & \leqslant & 1 \\
 & & x_1 & + & x_5 & & & & & & & & & \leqslant & 1 \\
 & & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & , & x_6 & = & (0,1)
\end{array}
\right]
$$

Problem Support

Reading a numerical instance:

```
7 6
7 2 4 6 3 1
3
1 2 3
3
2 5 6
3
1 2 4
3
2 3 5
2
4 6
3
3 4 5
2
1 5
```

```
julia> function loadSPP(fname)
julia>   f = open(fname)
julia>   m,n = parse.(Int, split(readline(f)) )
julia>   C = parse.(Int, split(readline(f)) )
julia>   A = zeros(Int, m, n)
julia>   for i=1:m
julia>     readline(f)
julia>     for valeur in split(readline(f))
julia>       j = parse(Int, valeur)
julia>       A[i,j] = 1
julia>     end
julia>   end
julia>   close(f)
julia>   return C, A
julia> end
```

Exercise 0

- **Solving the SPP instance with a MIP solver**

  Modeling the SPP with JuMP, calling your favorite MIP solver

Overview of JuMP

A modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear):

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: JuMP uses *MathOptInterface (MOI)*, an abstraction layer designed to provide a unified interface to mathematical optimization solvers.

  Currently supported solvers: Artelys Knitro, Baron, Bonmin, Cbc, Clp, Couenne, CPLEX, FICO Xpress, GLPK, Gurobi, SCIP, etc.
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

A modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear):

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: JuMP uses *MathOptInterface (MOI)*, an abstraction layer designed to provide a unified interface to mathematical optimization solvers.

  Currently supported solvers: Artelys Knitro, Baron, Bonmin, Cbc, Clp, Couenne, CPLEX, FICO Xpress, GLPK, Gurobi, SCIP, etc.
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

A modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear):

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: JuMP uses *MathOptInterface (MOI)*, an abstraction layer designed to provide a unified interface to mathematical optimization solvers.

  Currently supported solvers: Artelys Knitro, Baron, Bonmin, Cbc, Clp, Couenne, CPLEX, FICO Xpress, GLPK, Gurobi, SCIP, etc.
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

A modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear):

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: JuMP uses *MathOptInterface (MOI)*, an abstraction layer designed to provide a unified interface to mathematical optimization solvers.

  Currently supported solvers: Artelys Knitro, Baron, Bonmin, Cbc, Clp, Couenne, CPLEX, FICO Xpress, GLPK, Gurobi, SCIP, etc.
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Overview of JuMP

A modeling language for Mathematical Optimization (linear, mixed-integer, conic, semidefinite, nonlinear):

- **User friendliness**: syntax that mimics natural mathematical expressions.
- **Speed**: similar speeds to special-purpose modeling languages such as AMPL.
- **Solver independence**: JuMP uses *MathOptInterface (MOI)*, an abstraction layer designed to provide a unified interface to mathematical optimization solvers.

  Currently supported solvers: Artelys Knitro, Baron, Bonmin, Cbc, Clp, Couenne, CPLEX, FICO Xpress, GLPK, Gurobi, SCIP, etc.
- **Ease of embedding**: JuMP itself is written purely in Julia. Solvers are the only binary dependencies.

Getting Started

Install:

```
julia> using Pkg
```

```
julia> Pkg.add("JuMP")
```

```
julia> Pkg.add("HiGHS")
```

```
julia> Pkg.add("Gurobi")
```

Getting Started

Install:

```
julia> using Pkg
```

```
julia> Pkg.add("JuMP")
```

```
julia> Pkg.add("HiGHS")
```

```
julia> Pkg.add("Gurobi")
```

Getting Started

Install:

```
julia> using Pkg
```

```
julia> Pkg.add("JuMP")
```

```
julia> Pkg.add("HiGHS")
```

```
julia> Pkg.add("Gurobi")
```

Getting Started

Install:

```
julia> using Pkg
```

```
julia> Pkg.add("JuMP")
```

```
julia> Pkg.add("HiGHS")
```

```
julia> Pkg.add("Gurobi")
```

Writing a SPP Model and Solving an Instance

```
julia> using JuMP, HiGHS
```

```
julia> C,A = loadSPP(fname)
julia> m,n = size(A)
```

```
julia> spp = Model(HiGHS.Optimizer)
julia> @variable(spp, x[1:n], Bin)
julia> @objective(spp, Max, sum(c[j]*x[j] for j=1:n))
julia> @constraint(spp, cst[i=1:m], sum(A[i,j]*x[j] for j=1:n) ⩽ 1)
```

```
julia> print(model)
```

```
julia> optimize!(model)
```

```
julia> @show objective_value(model)
julia> @show value(x[2])
julia> @show value.(x)
```

Exercise 1

- **Construct a “good” initial solution**

  Iterative procedure: selection parts of a solution until it is feasible

- **Improve a “good” feasible solution**

  Iterative procedure: move from one solution to an other solution as long as required. The solution is the result of a move into a neighborhood structure.

Construction Algorithm

Algorithm 1: Greedy construction

$S \leftarrow \varnothing$

Initialize the candidate set $\mathcal{C}$, and evaluate the utility $u(e), \forall e \in \mathcal{C}$

while $(\mathcal{C} \neq \varnothing)$ **loop**

  Select the current *Best* element e from $\mathcal{C}$:

    $e \leftarrow \max_{e \in \mathcal{C}} u(e)$

  Incorporate e into the solution:

    $S \leftarrow S \cup \{e\}$

  Update the candidate set $\mathcal{C}$ and reevaluate the utility $u(e), \forall e \in \mathcal{C}$

    $\mathcal{C} \leftarrow \mathcal{C} \backslash \texttt{conflict}(\{e\})$

endWhile

return S

Example: construction (1/1)

$j =$	1	2	3	4	5	6	
$c_j =$	7	2	4	6	3	1	
$x_j =$	.	.	.	.	.	.	
	1	1	1	0	0	0	
	0	1	0	0	1	1	
	1	1	0	1	0	0	
$a_{ij} =$	0	1	1	0	1	0	
	0	0	0	1	0	1	
	0	0	1	1	1	0	
	1	0	0	0	1	0	
$\sum_{i \in I} a_{ij} =$	3	4	3	3	4	2	
$u(x_j) =$	$2,3$	$0,5$	$1,3$	2	$0,75$	$0,5$	
	$\times$						$\rightarrow \quad j^* = 1$

Example: construction (1/1)

$$
\begin{array}{rcccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 \\
x_j = & . & . & . & . & . & . \\
 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0 \\
\\
\sum_{i \in I} a_{ij} = & 3 & 4 & 3 & 3 & 4 & 2 \\
u(x_j) = & 2,3 & 0,5 & 1,3 & 2 & 0,75 & 0,5 \\
 & \times & & & & & & \rightarrow \quad j^* = 1
\end{array}
$$

Example: construction (1/1)

$$
\begin{array}{rcccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 \\
x_j = & 1 & . & . & . & . & . \\
 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0 \\
\sum_{i \in I} a_{ij} = & 3 & 4 & 3 & 3 & 4 & 2 \\
u(x_j) = & 2,3 & 0,5 & 1,3 & 2 & 0,75 & 0,5 \\
 & \times & & & & &
\end{array}
\quad \rightarrow \quad j^* = 1
$$

Example: construction (1/1)

$$
\begin{array}{rcccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 \\
x_j = & 1 & 0 & 0 & 0 & 0 & . \\
 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0 \\
\\
\sum_{i \in I} a_{ij} = & 3 & 4 & 3 & 3 & 4 & 2 \\
u(x_j) = & 2,3 & 0,5 & 1,3 & 2 & 0,75 & 0,5 \\
 & \times & & & & &
\end{array}
\quad \rightarrow \quad j^* = 1
$$

Example: construction (2/2)

$$
\begin{array}{rcccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 \\
x_j = & 1 & 0 & 0 & 0 & 0 & . \\
 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0 \\
\\
\sum_{i \in I} a_{ij} = & - & - & - & - & - & 2 \\
u(x_j) = & - & - & - & - & - & 0,5 \\
 & & & & & & \times \quad \rightarrow \quad j^* = 6
\end{array}
$$

Example: construction (2/2)

$$
\begin{array}{rcccccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 & & \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 & & \\
x_j = & 1 & 0 & 0 & 0 & 0 & 1 & & \\
 & 1 & 1 & 1 & 0 & 0 & 0 & & \\
 & 0 & 1 & 0 & 0 & 1 & 1 & & \\
 & 1 & 1 & 0 & 1 & 0 & 0 & & \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 & & \\
 & 0 & 0 & 0 & 1 & 0 & 1 & & \\
 & 0 & 0 & 1 & 1 & 1 & 0 & & \\
 & 1 & 0 & 0 & 0 & 1 & 0 & & \\
 & & & & & & & & \\
\sum_{i \in I} a_{ij} = & - & - & - & - & - & 2 & & \\
u(x_j) = & - & - & - & - & - & 0,5 & & \\
 & & & & & & \times & \rightarrow & j^* = 6
\end{array}
$$

Improvement Algorithm (Deepest Descent Method)

Algorithm 2: Local search algorithm (for minimization)

S, a feasible solution
localOptimum $\leftarrow$ false
repeat
  Choose $S' \in \mathcal{N}(\mathcal{S})$ with $f(S') < f(S)$
  if found(S')
    $S \leftarrow S'$
  else
    localOptimum $\leftarrow$ true
  endIf
until localOptimum
return S

Example: improvement (1/2)

A solution x ($x_j = (0,1),\ j = 1, \ldots, n$) :

- move:
  kp−exchange :
  k variables at 1 switched to 0 and p variables at 0 switched to 1
- relevant values of kp for WSPP:
  0-1-exchange
  1-1-exchange
  1-2-exchange
  2-1-exchange
  etc.

Example: improvement (2/2)

1-1-exchange:

$$
\begin{array}{rcccccccl}
j = & 1 & & 2 & 3 & & 4 & 5 & 6 & \\
c_j = & 7 & & 2 & 4 & & 6 & 3 & 1 & \\
x_j = & 1 & \searrow & 0 & 0 & \nearrow & 0 & 0 & 1 & \rightarrow z(x) = 8
\end{array}
$$

$$
a_{ij} = \begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{array}
\qquad
\begin{array}{ll}
\sum_{j \in J} a_{ij} x_j & = 1 \\
 & = 1 \\
 & = 1 \\
 & = 0 \\
 & = 1 \\
 & = 0 \\
 & = 1
\end{array}
$$

Exercise 2

- **Search a very "good" feasible solution with GRASP**[1]

  Iterative procedure: alternance between (1) greedy random construction and (2) local search.

[1] **GRASP**: Greedy randomized adaptative search procedure. Introduced by Th. Féo and M. Resende in 1989. Central idea : (Deepest) multistart descent method with initial solutions blending greedy and random.

GRASP Algorithm

Parameters :

- $\alpha \in [0, 1]$, the compromize between greedy and random.
- stoppingRule (ex : *nIter*, a number of iterations).

Algorithm 3: GRASP metaheuristic

$S^* \leftarrow \emptyset$, the best solution found
repeat
  $S \leftarrow$ `greedyRandomizedConstruction`(problem, α)
  $S' \leftarrow$ `localSearchImprovement`(S)
  `updateSolution`(S', S^*)
until `isFinished?`(StoppingRule)
return S^*

Construction algorithm

Algorithm 4: The greedy randomized construction

$S \leftarrow \varnothing$

Initialize the candidate set $\mathcal{C}$, and evaluate the utility $u(e), \forall e \in \mathcal{C}$

while $(\mathcal{C} \neq \varnothing)$ **loop**

  Build RCL, the restricted candidate list:

  $u_{Limit} \leftarrow \min_{e \in \mathcal{C}} u(e) + \alpha * (\max_{e \in \mathcal{C}} u(e) - \min_{e \in \mathcal{C}} u(e))$

  $RCL \leftarrow \{e \in \mathcal{C},\, u(e) \geqslant u_{Limit}\}$

  Select an element e from the RCL at random:

  $e \leftarrow \texttt{RandomSelect}(\mathrm{RCL})$

  Incorporate e into the solution:

  $S \leftarrow S \cup \{e\}$

  Update the candidate set $\mathcal{C}$ and reevaluate the utility $u(e), \forall e \in \mathcal{C}$

  $\mathcal{C} \leftarrow \mathcal{C} \backslash \texttt{conflict}(\{e\})$

endWhile

return S

GRASP : Greedy Randomized Adaptative Search Procedure

Exemple (Con't)

$$
\left[\begin{array}{lccccccccccccc}
\max z & = & 7x_1 & + & 2x_2 & + & 4x_3 & + & 6x_4 & + & 3x_5 & + & x_6 & & \\
s/c & & x_1 & + & x_2 & + & x_3 & & & & & & & \leqslant & 1 \\
 & & x_2 & + & x_5 & + & x_6 & & & & & & & \leqslant & 1 \\
 & & x_1 & + & x_2 & + & x_4 & & & & & & & \leqslant & 1 \\
 & & x_2 & + & x_3 & + & x_5 & & & & & & & \leqslant & 1 \\
 & & x_4 & + & x_6 & & & & & & & & & \leqslant & 1 \\
 & & x_3 & + & x_4 & + & x_5 & & & & & & & \leqslant & 1 \\
 & & x_1 & + & x_5 & & & & & & & & & \leqslant & 1 \\
 & & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & , & x_6 & = & (0,1)
\end{array}\right]
$$

With $\alpha = 0.70$

Example: construction

$$
\begin{array}{rcccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 \\
x_j = & . & . & . & . & . & . \\
 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0 \\
 & & & & & & \\
\sum_{i \in I} a_{ij} = & 3 & 4 & 3 & 3 & 4 & 2 \\
u(x_j) = & 2,3 & 0,5 & 1,3 & 2 & 0,75 & 0,5
\end{array}
$$

Example: construction

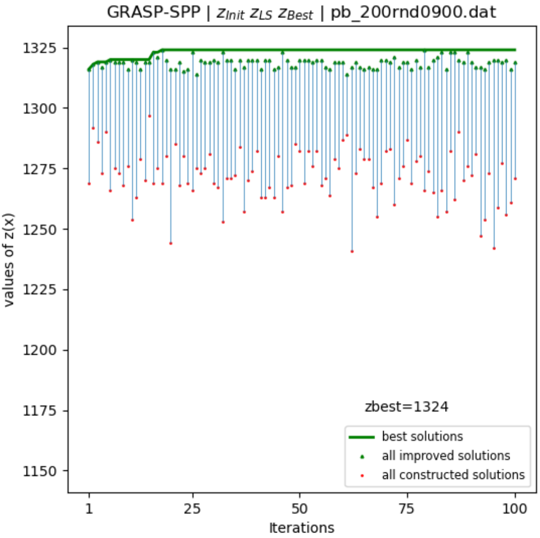
$$
\begin{array}{rcccccc}
j = & 1 & 2 & 3 & 4 & 5 & 6 \\
c_j = & 7 & 2 & 4 & 6 & 3 & 1 \\
x_j = & . & . & . & . & . & . \\
 & 1 & 1 & 1 & 0 & 0 & 0 \\
 & 0 & 1 & 0 & 0 & 1 & 1 \\
 & 1 & 1 & 0 & 1 & 0 & 0 \\
a_{ij} = & 0 & 1 & 1 & 0 & 1 & 0 \\
 & 0 & 0 & 0 & 1 & 0 & 1 \\
 & 0 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 0 & 1 & 0
\end{array}
$$

Selection n 1:

$$
\begin{array}{rcccccccc}
\sum_{i \in I} a_{ij} = & 3 & 4 & 3 & 4 & 3 & 2 & & \\
u(x_j) = & {\color{green}2,3} & 0,5 & 1,3 & 2,0 & 0,75 & {\color{red}0,5} & \Rightarrow & u_{Limit} = 1,76 \\
 & \times & & & \times & & & & \\
\text{RCL :} & 1 & 4 & & & & & \rightarrow & j^* = 4
\end{array}
$$

$$u_{Limit} = 0,7 \times (2,3 - 0,5) + 0.5 = 1,76$$

Example: output



References

- Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59: 65–98. 2017.
- Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*. 2023.
- Thomas A. Féo and Mauricio G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67-71,1989.
- Mauricio G.C. Resende and Celso C. Ribeiro. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures.* 332 pages. Springer, 2018.
- Xavier Delorme, Xavier Gandibleux, and Joaquin Rodriguez. GRASP for set packing problems. *European Journal of Operational Research*, 153(3):564-580, 2004.