COMP6925 Applied Operations Research

Linear Programming Tools (JuMP)

Inzamam Rahaman
inzamam.rahaman@sta.uwi.edu

Why tools?

- Solving optimization problems are hard by hand
- Algorithms are repetitive, tedious, and when used by humans, error-prone
- Optimization problems usually have “canonical” forms of representation (amenable to tools that can divorced from particular problem instances!)

Solvers

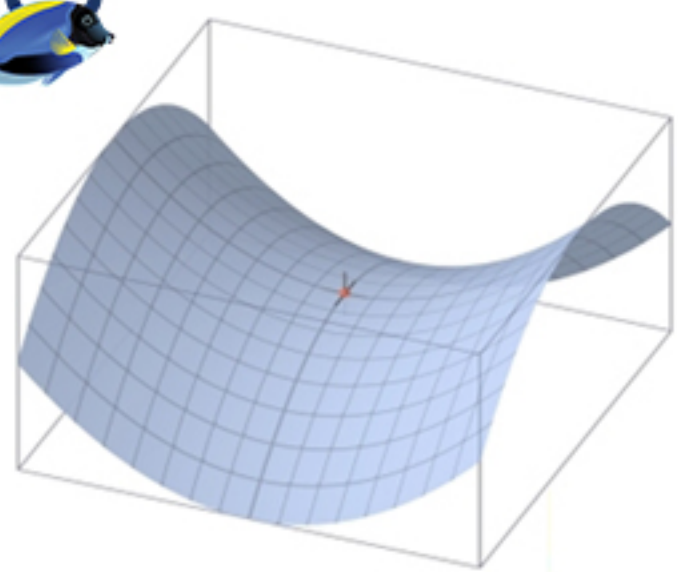
- Optimization problems are solved using tools referred to as “solvers”
- Can be written in any language, but usually written in C, C++, or FORTRAN
- Usually command line tool that accepts a file written in a specific format
- Excel can also be used for LP

COIN|OR

IBM

CPLEX

KNITRO®



The most advanced solver
for nonlinear optimization

Higher level languages

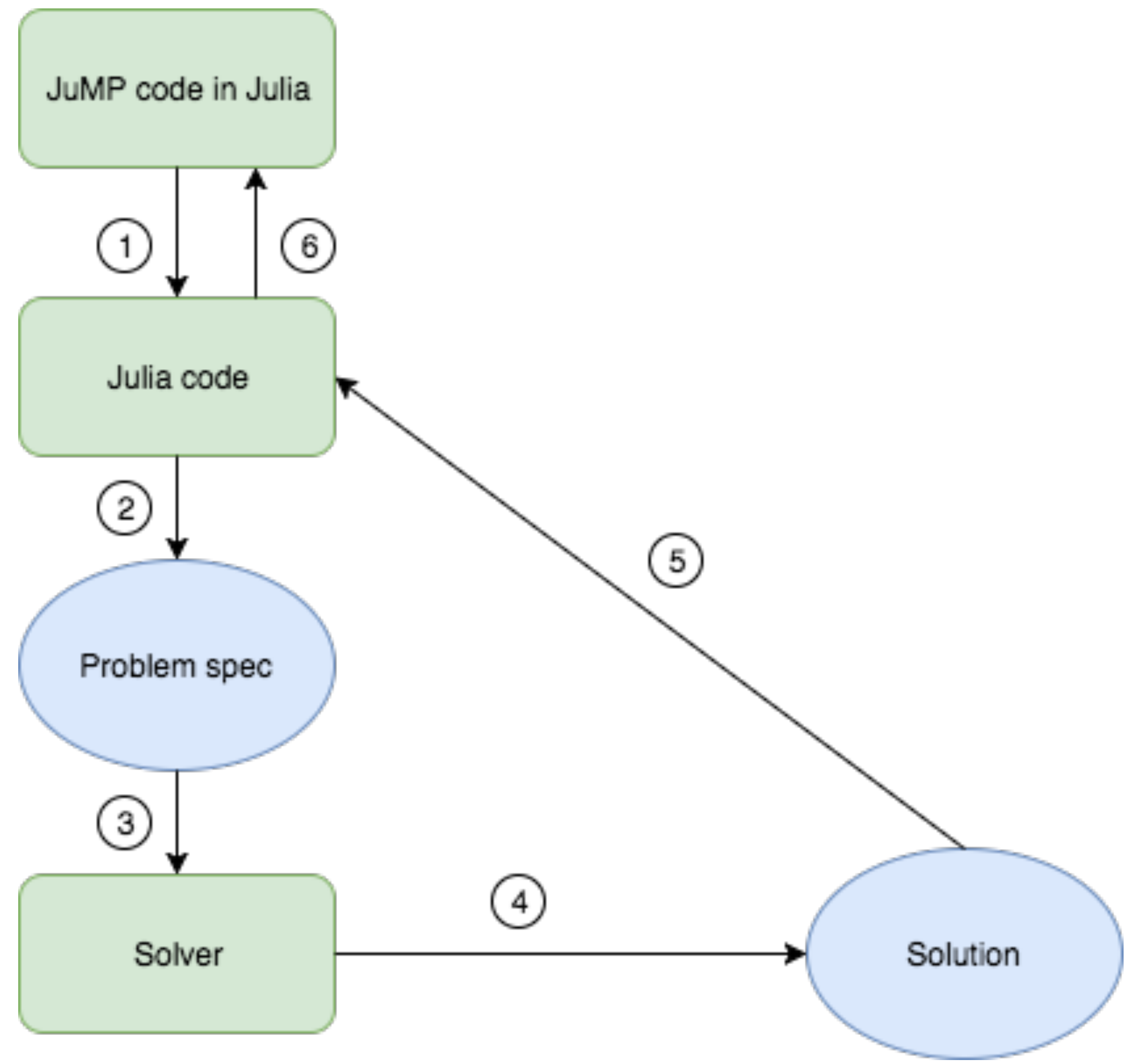
- We lose flexibility if we use these low-level solvers
- Solve optimization problems as part of a larger workflow
- Solution: abstract over tools in higher-level languages such as Python, R, MATLAB, and Julia

JuMP

- Optimization library in Julia (more specifically embedded DSL)
- Wraps over an assortment of tools (listed at http://www.juliaopt.org/JuMP.jl/0.18/installation.html)
- Can solve different types of optimization problems:
  - LP - Linear Programming
  - SDP - Semidefinitive Programming
  - MILP - Mixed-Integer Linear Programming
  - NLP - Non-linear Programming (e.g. quadratic programming)

JuMP Pipeline

1. JuMP DSL expands to conventional Julia code
2. Julia code generates problem spec in correct format for chosen solver
3. Problem spec piped to solver
4. Solver writes solution
5. Julia code reads solution
6. Solution is then stored in data structure for further use/ presentation



Model Specifications

- variables - macro to define variable name and range
- objective - macro to specify objective; whether Max or Min; function
- constraints - constraints on relationships between variables; variables need to be defined first with variable macro

```julia
using JuMP
using Clp

m = Model(solver = ClpSolver())

# specify variables and their ranges
@variable(m, x1 >= 0)
@variable(m, x2 >= 0)

# specify the objective function to optimize and whether to minimize or maximize
@objective(m, Min, 0.4x1 + 0.5x2)

# specify the constraints
@constraint(m, 0.3x1 + 0.1x2 <= 2.7)
@constraint(m, 0.5x1 + 0.5x2 == 6)
@constraint(m, 0.6x1 + 0.4x2 >= 6)

println(m)

status = solve(m)

println("Solution status: ", status)

println("Objective value: ", getobjectivevalue(m))
println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
```

```julia
# example taken from page 47
using JuMP
#using Clp
using Ipopt

#m = Model(solver = ClpSolver())
m = Model(solver = IpoptSolver())

# specify variables and their ranges
@variable(m, x1 >= 0)
@variable(m, x2 >= 0)
@variable(m, x3 >= 0)
@variable(m, x4 >= 0)
@variable(m, x5 >= 0)
@variable(m, x6 >= 0)
@variable(m, x7 >= 0)
@variable(m, x8 >= 0)
@variable(m, x9 >= 0)

# specify the objective function to optimize and whether to minimize or maximize
@objective(m, Max, 1000(x1 + x2 + x3) + 750(x4 + x5 + x6) + 250(x7 + x8 + x9))

# specify the constaints
@constraint(m, x1 + x4 + x7 <= 400)
@constraint(m, x2 + x5 + x8 <= 600)
@constraint(m, x3 + x6 + x9 <= 300)


@constraint(m, 3x1 + 2x4 + x7 <= 600)
@constraint(m, 3x2 + 2x5 + x8 <= 800)
@constraint(m, 3x3 + 2x6 + x9 <= 375)

@constraint(m, x1 + x2 + x3 <= 600)
@constraint(m, x4 + x5 + x6 <= 500)
@constraint(m, x7 + x8 + x9 <= 325)

@constraint(m, 3(x1 +  x4 +  x7) -  2(x2 +  x5 + x8) ==  0)
@constraint(m, (x2 +  x5 +  x8) -  2(x3 +  x6 + x9) ==  0)
@constraint(m, (x2 + x5 +  x8) - 2(x3 +  x6 +  x9)  ==  0)

print(m)

status = solve(m)

println("Solution status: ", status)

println("Objective value: ", getobjectivevalue(m))
println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
println("x3 = ", getvalue(x3))
println("x4 = ", getvalue(x4))
println("x5 = ", getvalue(x5))
println("x6 = ", getvalue(x6))
println("x7 = ", getvalue(x7))
println("x8 = ", getvalue(x8))
println("x9 = ", getvalue(x9))
```

```julia
# example taken from page 49
using JuMP
using Clp

m = Model(solver = ClpSolver())

# specify variables and their ranges
@variable(m, 0 <= x1 <= 1)
@variable(m, 0 <= x2 <= 1)
@variable(m, 0 <= x3 <= 1)
@variable(m, 0 <= x4 <= 1)
@variable(m, 0 <= x5 <= 1)
@variable(m, 0 <= x6 <= 1)

# specify the objective function
@objective(m, Min, 8x1 + 10x2 + 7x3 + 6x4 + 11x5 + 9x6)

# specify the constraints
@constraint(m, 12x1 + 9x2 + 25x3 + 20x4 + 17x5 + 13x6 >= 60)
@constraint(m, 35x1 + 42x2 + 18x3 + 31x4 +  56x5 +  49x6 >= 150)
@constraint(m, 37x1 + 53x2 + 28x3 + 24x4  + 29x5 + 20x6 >= 125)


print(m)

status = solve(m)

println("Solution status: ", status)

println("Objective value: ", getobjectivevalue(m))
println("x1 = ", getvalue(x1))
println("x2 = ", getvalue(x2))
println("x3 = ", getvalue(x3))
println("x4 = ", getvalue(x4))
println("x5 = ", getvalue(x5))
println("x6 = ", getvalue(x6))
```

References

- Code: https://github.com/InzamamRahaman/COMP6925
- JuMP docs: https://jump.readthedocs.io/en/latest/index.html
- Examples taken from Hillier and Lieberman's Introduction to Operations Research (pages noted in comments on source code on GitHub)