JuMP
MIP Programming

Przemysław Szufel
https://szufel.pl/

Use case scenario

The Subway restaurant chain in Las Vegas has a total of 118 restaurants in different parts of the city.

Company’s manager plans to visit all restaurants during a single day.

What is the optimal order that restaurants should be visited?

Traveling salesman problem

- Variables:
  - c_{ft} – cost of travel from "f" to "t"
  - x_{ft} – binary variable indicating 1 when agent travels from "f" to "t"

$$\text{Min} \sum_{f=1}^{N} \sum_{t=1}^{N} c_{ft} x_{ft}$$

The mathematical problem framing based on: Julia Programming for Operations Research A Primer on Computing, Changhyun Kwon, 2018, see: http://opensourc.es/blog/mip-tsp

TSP

$$\text{Min} \sum_{f=1}^{N} \sum_{t=1}^{N} c_{ft} x_{ft}$$

Variables:

- c_{ft} – cost of travel from "f" to "t"
- x_{ft} – binary variable indicating 1 when agent travels from "f" to "t"

Each city visited once

$$\sum_{t=1}^{N} x_{ft} = 1 \quad \forall f \in \{1, \ldots, N\}$$

$$\sum_{f=1}^{N} x_{ft} = 1 \quad \forall t \in \{1, \ldots, N\}$$

City cannot visit itself

$$x_{ff} = 0 \quad \forall f \in \{1, \ldots, N\}$$

Avoid two-city cycles

$$x_{ft} + x_{tf} <= 1 \quad \forall f, t \in \{1, \ldots, N\}$$

Other cycles: /dynamically add a constraint whenever a cycle occurs/

For more details see: http://opensourc.es/blog/mip-tsp

JuMP implementation

```
m = Model(solver=GLPKSolverMIP())
@variable(m, x[f=1:N, t=1:N], Bin)
@objective(m, Min, sum( x[i, j]*distance_mx[i,j] for i=1:N,j=1:N))
@constraint(m, notself[i=1:N], x[i, i] == 0)
@constraint(m, oneout[i=1:N], sum(x[i, 1:N]) == 1)
@constraint(m, onein[j=1:N], sum(x[1:N, j]) == 1)
for f=1:N, t=1:N
    @constraint(m, x[f, t]+x[t, f] <= 1)
end
```

Getting a cycle

```
function getcycle(m, N)
    x_val = getvalue(x)
    cycle_idx = Vector{Int}()
    push!(cycle_idx, 1)
    while true
        v, idx = findmax(x_val[cycle_idx[end], 1:N])
        if idx == cycle_idx[1]
            break
        else
            push!(cycle_idx, idx)
        end
    end
    cycle_idx
end
```

Adding a constraint…

```
function solved(m, cycle_idx, N)
    println("cycle_idx: ", cycle_idx)
    println("Length: ", length(cycle_idx))
    if length(cycle_idx) < N
        cc = @constraint(m, sum(x[cycle_idx,cycle_idx])
           <= length(cycle_idx)-1)
        println("added a constraint")
        return false
    end
    return true
end
```

Iterating over the model

```
while true
    status = solve(m)
    println(status)
    cycle_idx = getcycle(m, N)
    if solved(m, cycle_idx,N)
        break;
    end
end
```

Gurobi.jl

- Commercial software
- Free for academic use
- Integrates with JuMP via Gurobi.jl

- Supports JuMP Lazy constraints (http://www.juliaopt.org/JuMP.jl/0.18/callbacks.html)

Gurobi callbacks

```
function callbackhandle(cb)
    cycle_idx = getcycle(cb, N)
    println("Callback! N= $N cycle_idx: ", cycle_idx)
    println("Length: ", length(cycle_idx))
    if length(cycle_idx) < N
        @lazyconstraint(cb, sum(x[cycle_idx,cycle_idx]) <= length(cycle_idx)-1)
        println("added a lazy constraint")
    end
end

addlazycallback(m, callbackhandle)
solve(m)
```

TravelingSalesmanHeuristics.jl

```
using TravelingSalesmanHeuristics
sol = TravelingSalesmanHeuristics.solve_tsp(
distance_mx,quality_factor =100)
```

More info:
http://evanfields.github.io/TravelingSalesmanHeuristics.jl/latest/heuristics.html