DTU

Modelling and decomposition

Julia And Optimization

Mathematical Programming (LP/MIP)

$$\texttt{Min} \quad \sum_j C_j \cdot x_j \tag{1}$$

s.t.

$$\sum_j A_{i,j} \cdot x_j \geq b_i \quad \forall i \tag{2}$$

$$x_j \in R^+ / Z^+ \tag{3}$$

If you use either Linear Programming or Mixed Integer Programming in your research, then this presentation should be interesting for you.

Modelling languages

We are used to write models up in mathematics, but need to feed it to the advanced solvers in an easy way. For this, modelling languages were made:

- GAMS
- AMPL
- Mosel
- Zimpl
- ....

Why use modelling languages ? Ease of programming. I am **sure** that the days of the dedicated modelling languages are comming to an end ...

Outline

- DTU: Our use of Julia
- Modelling
- Decomposition
- Mathheuristic
- Multi-objective modelling

OR Courses at the Technical University of Denmark (DTU)

- 42101 - Introduction to Operations Research (BSc)
- 42586 - Decisions under uncertainty (BSc)
- 42112 - Mathematical Programming Modelling (MSc)
- 42114 - Integer Programming (MSc)
- 42115 - Network Optimization (MSc)
- 42117 - Transport Optimization (MSc)
- 42136 - Large Scale Optimization using Decomposition (MSc)
- 42137 - Optimization using metaheuristics (MSc)
- 42142 - Recent Research Results in Management Science (MSc)

Today we use Julia/JuMP as the **only** language in **all** our courses

Why we use Julia/JuMP

- Our students were not good enough programmers ...
- Julia/JuMP was the only language which could be used in all our courses...
- JuMP is a **fantastic** package ...
- Julia and JuMP are open-source
- JuMP supports many different solvers ... including open-source solvers
- JuMP supports multi-objective modelling (as far as I know, as the first modelling language ....)

Out example in this lecture: Facility Location

Given $f \in F$ facilities and $c \in C$ customers. A volume of $Demand_c$ should be delivered to each customer from **one** depot. The cost for the whole demand from facility f to customer c is $DistCost_{f,c}$. Each facility f has a capacity of Cap_f and costs $FacCost_f$ to establish.

Mathematical programming model of Facility Location

Mixed Integer Programming model of the Facility Location problem:

$$\texttt{Min} \quad \sum_{f \in F} \sum_{c \in C} DistCost_{f,c} \cdot x_{f,c} + \sum_{f \in F} FacCost_f \cdot y_f \tag{4}$$

s.t.

$$\sum_{f \in F} x_{f,c} \geq 1 \quad \forall\, c \in C \tag{5}$$

$$\sum_{c \in C} Dem_c \cdot x_{f,c} \leq Cap_f \cdot y_f \quad \forall\, f \in F \tag{6}$$

$$x_{f,c}, y_f \in \{0, 1\} \tag{7}$$

JuMP model of Facility Location

```
FL = Model(HiGHS.Optimizer)
@variable(FL, x[1:C],Bin)
@variable(FL, y[1:F],Bin)
@objective(FL, Max, sum( DistCost[f,c]*x[f,c] for f=1:F,c=1:C) +
                    sum( FacCost[f]*y[f] for f=1:F))
@constraint(FL, [c=1:C], sum( x[f,c] for f=1:F) >= 1)
@constraint(FL, [f=1:F], sum( Dem[c]*x[f,c] for f=1:F) <= Cap[f]*y[f])
optimize!(FL)
```

Conditional constraint

$$\sum_{f \in F} x_{f,c} \geq 1 \quad \forall\, c \in C \mid Dem_c \geq 10$$

```
@constraint(FL, [c=1:C; Dem[c]>=10], sum( x[f,c] for f=1:F) >= 1)
```

Conditional sum

$$\sum_{f \in F | Cap_c >= 100} x_{f,c} \geq 1 \quad \forall\, c \in C$$

```julia
@constraint(FL, [c=1:C], sum( x[f,c] for f=1:F if Cap[c]>= 100) >= 1)
```

Conditional term

$$\sum_{f \in F} x_{f,c} \geq 1|(c \neq 2) + 2|(c = 2) \quad \forall\, c \in C$$

```
@constraint(FL, [c=1:C], sum( x[f,c] for f=1:F ) >= (c==2 ? 2 : 1) )
```

JuMP solvers (54)

Alpine.jl			Triad NS	(MI)NLP
Artelys Knitro	KNITRO.jl	Manual	Comm.	(MI)LP, (MI)SOCP, (MI)NLP
BARON	BARON.jl	Manual	Comm.	(MI)NLP
Bonmin	AmplNLWriter.jl		EPL	(MI)NLP
Cbc	Cbc.jl		EPL	(MI)LP
CDCS	CDCS.jl	Manual*	GPL	LP, SOCP, SDP
CDD	CDDLib.jl		GPL	LP
Clarabel.jl			Apache	LP, QP, SOCP, SDP
Clp	Clp.jl		EPL	LP
COPT	COPT.jl		Comm.	(MI)LP, SOCP, SDP
COSMO.jl			Apache	LP, QP, SOCP, SDP
Couenne	AmplNLWriter.jl		EPL	(MI)NLP
CPLEX	CPLEX.jl	Manual	Comm.	(MI)LP, (MI)SOCP
CSDP	CSDP.jl		EPL	LP, SDP
DAQP	DAQP.jl		MIT	(Mixed-binary) QP
DSDP	DSDP.jl		DSDP	LP, SDP
EAGO.jl			MIT	(MI)NLP
ECOS	ECOS.jl		GPL	LP, SOCP
FICO Xpress	Xpress.jl	Manual	Comm.	(MI)LP, (MI)SOCP
GLPK	GLPK.jl		GPL	(MI)LP
Gurobi	Gurobi.jl	Manual	Comm.	(MI)LP, (MI)SOCP
HiGHS	HiGHS.jl		MIT	(MI)LP, QP
Hypatia.jl			MIT	LP, SOCP, SDP
Ipopt	Ipopt.jl		EPL	LP, QP, NLP
Juniper.jl			MIT	(MI)SOCP, (MI)NLP
Loraine.jl			MIT	LP, SDP
MadNLP.jl			MIT	LP, QP, NLP
MAiNGO	MAiNGO.jl		EPL 2.0	(MI)NLP
Manopt.jl			MIT	NLP
MiniZinc	MiniZinc.jl	Manual	MPL-2	CP-SAT
Minotaur	AmplNLWriter.jl	Manual	BSD-like	(MI)NLP
MOSEK	MosekTools.jl	Manual	Comm.	(MI)LP, (MI)SOCP, SDP
NLopt	NLopt.jl		GPL	LP, QP, NLP
Octeract	AmplNLWriter.jl		Comm.	(MI)NLP
Optim.jl			MIT	NLP
OSQP	OSQP.jl		Apache	LP, QP
PATH	PATHSolver.jl		MIT	MCP
Pajarito.jl			MPL-2	(MI)NLP, (MI)SOCP, (MI)SDP
Pavito.jl			MPL-2	(MI)NLP
Penbmi	Penopt.jl		Comm.	Bilinear SDP
Percival.jl			MPL-2	NLP
PolyJuMP.KKT	PolyJuMP.jl		MIT	NLP
PolyJuMP.QCQP	PolyJuMP.jl		MIT	NLP
ProxSDP.jl			MIT	LP, SOCP, SDP
RAPOSa	AmplNLWriter.jl	Manual	RAPOSa	(MI)NLP
SCIP	SCIP.jl		Apache	(MI)LP, (MI)NLP
SCS	SCS.jl		MIT	LP, QP, SOCP, SDP
SDPA	SDPA.jl, SDPAFamily.jl		GPL	LP, SDP
SDPLR	SDPLR.jl		GPL	LP, SDP
SDPNAL	SDPNAL.jl	Manual*	CC BY-SA	LP, SDP
SDPT3	SDPT3.jl	Manual*	GPL	LP, SOCP, SDP
SeDuMi	SeDuMi.jl	Manual*	GPL	LP, SOCP, SDP
StatusSwitchingQP.jl			MIT	LP, QP
Tulip.jl			MPL-2	LP

JuMP solvers

JuMP supports many different kinds of optimization solvers: Linear programming, Quadratic programming, Second-order conic programming Mixed-complementarity programming, Nonlinear programming, Semidefinite programming, Mixed Integer Programming, Constraint programming and Boolean satisfiability. Our focus is LP and MIP. These solvers are relevant:

- HiGHS: Current best open-source LP/MIP solver
- Gurobi: Best commercial LP/MIP solver
- CPLEX: Good commercial LP/MIP solver
- Xpress: Good commercial LP/MIP solver
- COPT: Good commercial LP/MIP solver
- Obsolete solvers: Cbc, Clp and GLPK (open source solvers replaced by HiGHS)

JuMP

Decomposition

- Benders Decomposition:
  - Requires some theory and has limited application (Stochastic programming)
- Branch & Cut:
  - Very important, but hard to implement problem-specific: Which cut will you use ? JuMP supports generic cuts (Gurobi & CPLEX)
- Dantzig-Wolfe/Column Generation: A very important approach.

Dantzig-Wolfe (DZ) decomposition

$$\texttt{Min} \quad \sum_j C_j \cdot x_j \tag{8}$$

s.t.

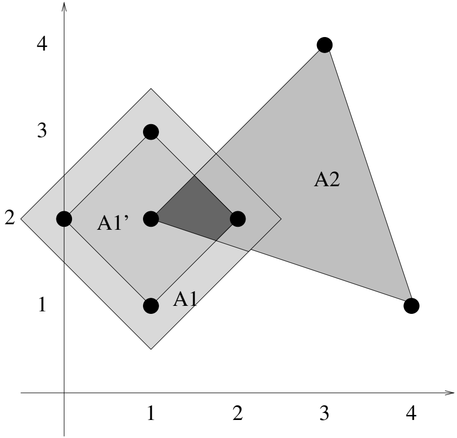
$$\sum_j A1_{i,j} \cdot x_j \geq b1_i \quad \forall i \tag{9}$$

$$\sum_j A2_{i,j} \cdot x_j \geq b2_i \quad \forall i \tag{10}$$

$$x_j \in R^+ \tag{11}$$

Why is DZ a good idea ?

Why is DZ decomposition a possible advantage



Dantzig-Wolfe (DZ) decomposition I

$$\texttt{Min} \quad \sum_{f\in F}\sum_{c\in C} DistCost_{f,c} \cdot x_{f,c} + \sum_{f\in F} FacCost_f \cdot y_f \tag{12}$$

s.t.

$$\sum_{f\in F} x_{f,c} \geq 1 \quad \forall\ c \in C \tag{13}$$

$$\sum_{c\in C} Dem_c \cdot x_{f,c} \leq Cap_f \cdot y_f \quad \forall\ f \in F \tag{14}$$

$$x_{f,c}, y_f \in [0,1] \tag{15}$$

Dantzig-Wolfe (DZ) decomposition II

$$\texttt{Min} \quad \sum_{f \in F} \sum_{p \in P} Cost_{p,f} \cdot \lambda_{p,f} \tag{16}$$

s.t.

$$\sum_{f \in F} x_{f,c} \geq 1 \quad \forall\, c \in C \tag{17}$$

$$x_{f,c} = \sum_{p \in P_f} \overline{x_{f,c}^{p}} \cdot \lambda_{p,f} \quad \forall\, f \in F, c \in C \tag{18}$$

$$\sum_{p \in P} \lambda_{p,f} \leq 1 \quad \forall\, f \in F \tag{19}$$

$$x_{f,c}, \lambda_{p,f} \in [0, 1] \tag{20}$$

Dantzig-Wolfe (DZ) decomposition III

$$\texttt{Min} \quad \sum_{f \in F, p \in P_f} Cost_{f,p} \cdot \lambda_{f,p} \tag{21}$$

s.t.

$$\sum_{f \in F} \sum_{p \in P_f} \overline{x^p_{f,c}} * \lambda_{f,p} \geq 1 \quad \forall\, c \in C \quad \alpha_c \in R^+ \tag{22}$$

$$\sum_{p \in P_f} \lambda_{f,p} \leq 1 \quad \forall\, f \in F \quad \beta_f \in R^- \tag{23}$$

$$\lambda_{f,p} \in [0, 1] \tag{24}$$

Column Generation

The model just shown, assume that we pre-generate all the variables $\lambda_{f,p}$ and their constants $\overline{x_{f,c}^{p}}$. There are however exponentially many of these, making this approach un-usable for even small problems. Instead the sub-problem finds them by minimizing the reduced cost:

$$
\begin{aligned}
& Min_f \quad \sum_{c \in C} DistCost_{f,c} \cdot x_c + FacCost_f && (25)\\
& \quad - \sum_{c \in C} \alpha_c \cdot x_c - \beta_f && (26)\\
& \text{s.t.} \quad \sum_{c} Dem_c \cdot x_c \leq Cap_f && (27)\\
& x_c \in \{0, 1\} && (28)
\end{aligned}
$$

JuMP model of sub-problem

```julia
function SolveSub(alpha,beta,f)
    sub=Model(HiGHS.Optimizer)
    @variable(sub, x[c=1:C], Bin)
    @objective(sub, Min, FacCost[f] + sum( DistCost[c,f]*x[c] for c=1:C )
                                - sum( alpha[c]*x[c] for c=1:C) - beta[f] )
    @constraint(sub, sum( Demand[c]*x[c] for c=1:C) <= Cap[f])
    optimize!(sub)
    if termination_status(sub) != MOI.OPTIMAL
        throw("Error: Non-optimal sub-problem status")
    end
    return (objective_value(sub),round.(Int,value.(x)),solve_time(sub))
end
```

But what about the first master-problem ?

$$Min \quad \sum_{c \in C} M \cdot slack_f \tag{29}$$

s.t.

$$slack_c \geq 1 \quad \forall\, c \in C \tag{30}$$

$$0 \leq 1 \quad \forall\, f \in F \tag{31}$$

$$slack_f \in R^+ \tag{32}$$

Initial Master Problem

```julia
master=Model(HiGHS.Optimizer)
set_silent(master)
@variable(master, s[c=1:C] >= 0)
@objective(master, Min, M*sum(s[c] for c=1:C) )
@constraint(master, CoverCustomer[c=1:C], s[c] >= 1 )
@constraint(master, FacLimit[f=1:F], 0 <= 1 )
```

Adding the new $\lambda_{f,p}$

```
function AddMasterVariable(xVal,f)
    cost=FacCost[f] + sum( DistCost[c,f]*Demand[c]*xVal[c] for c=1:C)
    oldvars = JuMP.all_variables(master)
    new_var = @variable(master, base_name="l_$(length(oldvars))_$(f)", lower_bound=0)
    set_objective_coefficient(master, new_var, cost)
    for c=1:C
        if xVal[c]==1
            set_normalized_coefficient(CoverCustomer[c], new_var, 1)
        end
    end
    set_normalized_coefficient(FacLimit[f], new_var, 1)
end
```

Core Column Generation Algorithm

```julia
improving=true
while improving
    optimize!(master)
    mas_obj=objective_value(master)
    alpha = dual.(CoverCustomer)
    beta = dual.(FacLimit)
    improving=false
    for f=1:F
        (redCost, xVal) = SolveSub(alpha,beta,f)
        if redCost < -0.001
            AddMasterVariable(xVal,f)
            improving = true
        end
    end
end
```

But we solved the relaxed problem !

In principle we should now use Branch & Price. This is more complicated so we choose the simple solution, hence we MIPIFY: Solve the master problem with the found column variables as binary variables: (we (probably) do not get the optimal solution, but we do get a gap):

```
for v=1:length(all_variables)
    set_binary(all_variables[v])
end
optimize!(master)
```

Mathheuristics

This is another approach: Use a MIP solver iteratively to find a heuristic solution. Here we will make a simple hill-climber, using the following model:

$$\texttt{Min} \quad \sum_{f \in F} \sum_{c \in C} DistCost_{f,c} \cdot x_{f,c} + \sum_{f \in F} FacCost_f \cdot y_f + M * \sum_{c \in C} q_c \tag{33}$$

s.t.

$$q_c + \sum_{f \in F} x_{f,c} \geq 1 \quad \forall\, c \in C \tag{34}$$

$$\sum_{c \in C} Dem_c \cdot x_{f,c} \leq Cap_f \cdot y_f \quad \forall\, f \in F \tag{35}$$

$$\sum_{f,c \mid \overline{x_{f,c}=0}} x_{f,c} + \sum_{f,c \mid \overline{x_{f,c}=1}} (1 - x_{f,c}) \leq K \tag{36}$$

$$x_{f,c}, y_f, q_c \in \{0, 1\} \tag{37}$$

A mathheuristic in Julia

If we make a function which optimize but limit the changes into a hamming distance of K:

```
it=1
while it<200
    (total,xVal,yVal,qVal)=AddKConstraintAndOptimize(xVal,K)
    if total>=old_total
        K=K+2
    end
    old_total=total
    it+=1
end
```

Optimizing with hamming distance constraint

```
function AddKConstraintAndOptimize(xVal,K)
    @constraint(compact, Kconstraint,
            sum( x[c,f] for f=1:F, c=1:C if xVal[c,f]==0) +
            sum( (1-x[c,f]) for f=1:F, c=1:C if xVal[c,f]==1)
            <= K)
    optimize!(compact)
    total=objective_value(compact)
    xVal=round.(Int,value.(x))
    yVal=round.(Int,value.(y))
    qVal=round.(Int,value.(q))
    delete(compact, Kconstraint)
    unregister(compact, :Kconstraint)
    return (total,xVal,yVal,qVal)
end
```

Multiple objectives

But there are actually two objectives:

$$\texttt{Min} \quad \sum_{f \in F} \sum_{c \in C} DistCost_{f,c} \cdot x_{f,c} \tag{38}$$

$$\texttt{Min} \quad \sum_{f \in F} FacCost_f \cdot y_f \tag{39}$$

s.t.

$$\sum_{f \in F} x_{f,c} \geq 1 \quad \forall\, c \in C \tag{40}$$

$$\sum_{c \in C} Dem_c \cdot x_{f,c} \leq Cap_f \cdot y_f \quad \forall\, f \in F \tag{41}$$

$$x_{f,c}, y_f \in \{0, 1\} \tag{42}$$

How can we solve this directly in JuMP ?

```
compact=Model()
@variable(compact, y[f=1:F],Bin)
@variable(compact, x[f=1:F,c=1:C],Bin)
@expression(compact, dist_expr, sum( Dist[c,f]*Demand[c]*x[f,c] for c=1:C,f=1:F ))
@expression(compact, fixed_expr, sum( FacCost[f]*y[f] for f=1:F))
@objective(compact, Min, [dist_expr, fixed_expr])
@constraint(compact, [c=1:C], sum( x[f,c] for f=1:F) ==1)
@constraint(compact, [f=1:F], sum( Demand[c]*x[f,c] for c=1:C) <= FCap[f]*y[f])
set_optimizer(compact, () -> MOA.Optimizer(HiGHS.Optimizer))
set_attribute(compact, MOA.Algorithm(), MOA.EpsilonConstraint())
set_attribute(compact, MOA.EpsilonConstraintStep(), 0.5)
optimize!(compact)
```

Conclusion

JuMP, in side the Julia language is really good:

- JuMP enables easy modelling
- Julia/JuMP enables easy implementation of decomposition algorithms
- Julia/JuMP enables easy implementation of mathheuristics
- Julia/JuMP/MultiObjectiveAlgorithms enables easy modelling and solution of multi-objective MIP models

But: Startup is slower (than GAMS) and index type failures are not found.

Where to go from here

If you are interested in learning modelling in Julia/JuMP, we (my college Richard Lusby and I) have written an open-source book: `https://www.man.dtu.dk/mathprogrammingwithjulia`

If you are interested in Dantzig-Wolfe/Column Generation, look at the book "Branch And Price", Desrosiers, Lübbecke, Desaulniers, & Gauthier [1]

If you are interested mathheuristics, there are a number of articles to take a look at [2, 3, 4, 5]

Appendix

Speed of Julia

We (our OR group) wanted to test the speed of Julia, applied to metaheuristics. Hence we tested a **simple** Simulated Annealing, on a simple standard TSP problem:

- 5 different TSP data-sets: berlin52, bier127, eil51, eil76 & st70
- Test every dataset for 30 sec. 10 times.

Notice, our interest is **not** solution quality, but speed, hence the number of SA iterations in 30 seconds.

Speed of Julia

- Implement a very simple Simulated Annealing algorithm (the simplest metaheuristic) on the most researched OR problem, TSP
- How many iterations can be done ?
- How fast is Julia compared to:
  - C, Thomas Stidsen, Bernd Dammann
  - C#, Simon Christensen
  - Java, Dario Pacino
  - Python, Niels Christian Fink Bagger
  - Julia, Stefan Røpke, Dario Pacino & Thomas Stidsen

Speed of Julia

Algorithm 1: Simple Simulated Annealing for TSP

ReadTSPDistanceMatrix()
cur=CreateRandomStartTour()
temp=StartTemperature()
while *time*() $<$ 30 **do**
    (i,j)=SelectTwoRandomDifferentCities()
    delta=SwitchCostIfCitySwap(cur,i,j) $\longrightarrow$ delta-evaluation
    if *delta* <0 *or exp*(*delta*/*temp*) $<$ *Rand*$(0,1)$ **then**
        cur=Swap(cur,i,j)
    temp=$\alpha \cdot t$

Results

Language	C	Java	Julia	C#	Python	O-Python	O-Julia
Mill. it pr. 30 sec.	978	480	542	485	9	732	963
Pr. sec.	32	16	18	16	0.3	24	32
C speed factor	1	2,05	1,81	2,02	104,71	1,35	1,02

My background

- Assoiciate professor at the Technical University of Denmark (DTU)
- Teaching mathematical modelling, decomposition and metaheuristics for 20 years
- Research focus: Scheduling/timetabling, manpower planning and multi-objective optimization
- Programming background:
  - Previously: GAMS and C++
  - Now: Julia

OR at DTU

- 8 faculty members, 10-15 PhD students
- Strong applied research in many areas: Transport, energy, timetables ...
- EURO-2024 was hosted by our group at DTU in Copenhagen, more than 3000 participants

Suggested literature

Suggested literature

Jacques Desrosiers, Marco Lübbecke, Guy Desaulniers, and Jean-Bertrand Gauthier.
Branch-and-price, volume G-2024-36.
Les cahiers du gerad edition, June 2024.

Matteo Fischetti and Andrea Lodi.
Local branching.
Mathematical programming, 98:23–47, 2003.

Enrico Angelelli, Renata Mansini, and M Grazia Speranza.
Kernel search: A general heuristic for the multi-dimensional knapsack problem.
Computers & Operations Research, 37(11):2017–2026, 2010.

Sharlee Climer and Weixiong Zhang.

Suggested literature

Cut-and-solve: An iterative search strategy for combinatorial optimization problems.
Artificial Intelligence, 170(8-9):714–738, 2006.

Matteo Fischetti and Michele Monaci.
Proximity search for 0-1 mixed-integer convex programming.
Journal of Heuristics, 20:709–731, 2014.