Decomposition Exercise 4: Answers

Q1: The direct Newsboy model

Below the direct model for the problem, implmented in Julia is given:
Given the data:

- Scenario: $s \in S$: There are 10 scenarios, 1 to 10
- π_s: Probability of scenario s
- $c = 20$: Purchase cost for the newsboy
- $p = 70$: Salesprice for the newsboy
- $h = 10$: Scrap value for the newsboy, for newspapers he does not manage to sell
- D_s: Demand of newspapers in scenario s

Max:

$$\sum_s \pi_s(p \cdot x_s - c \cdot y + h \cdot (y - x_s)) = \sum_s \pi_s(p-h)x_s + \sum_s \pi_s(h-c)y$$

S.T.:

$$\begin{array}{rcl} x_s & \leq & D_s \quad \forall\, s \\ x_s & \leq & y \quad \forall\, s \\ x \in R^+ & & y \in Z^+ \end{array}$$

Comment:

- The objective function consists (in the first formulation) of three terms: The sales made minus the purchase cost plus the money made from returning scrap newspapers.
- There are just two constraints: You cannot sell more than the demand in each scenario and you cannot sell more than the newspapers you have.

The Direct model in Julia:

```
small = Model(solver=GLPKSolverLP())

@variable(small, y >= 0) # bought newspapers

@variable(small, x[1:S] >= 0) # sold newspapers

# maximize profit from sales and scrap minus cost
@objective(small, Max, sum( prob[s]*((p-h)*x[s] + (h-c)*y) for s=1:S))

# cant sell more than the demand
@constraint(small, [s=1:S], x[s] <= demand[s])

# cant sell more newspapers than purchased
@constraint(small, lim[s=1:S], x[s] <= y)
```

Q2: The dual of the sub-problem

Max:

$$\sum_s \pi_s(p \cdot x_s - c \cdot \overline{y} + h \cdot (\overline{y} - x_s)) = \sum_s \pi_s(p-h)x_s + \sum_s \pi_s(h-c)\overline{y}$$

S.T.:

$$\begin{array}{rcl} x_s & \leq & D_s \quad \forall\, s \\ x_s & \leq & \overline{y} \quad \forall\, s \\ x \in R^+ & & y \in Z^+ \end{array}$$

Below is the julia version of the model, please notice:

- We replace the variable y with the constant ybar, which we choose to fix to the value 21
- We use the getdual function to get the dual values. **HOWEVER:** to do this, we need to name the constraints, see below.

```
# News Boy: Direct Model
# Notice: This is a maximization problem

#-------------------------------------------------------------------
# Intro definitions
using JuMP
using GLPKMathProgInterface
#-------------------------------------------------------------------

#************************************************************************
# Data
# Demand of newspapers in each scenario
demand=[ 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
S=length(demand)
# probability of scenario
prob=[0.05, 0.10, 0.10, 0.10, 0.15, 0.15, 0.10, 0.10, 0.10, 0.05 ]
c=20 # purchase price
p=70 # selling price
h=10 # scrap value
#************************************************************************

#-------------------------------------------------------------------
# direct problem

direct = Model(solver=GLPKSolverLP())
```

```
ybar=21
@variable(direct, x[1:S] >= 0) # sold newspapers

# maximize profit from sales and scrap minus cost
@objective(direct, Max, sum( prob[s]*((p-h)*x[s] + (h-c)*ybar) for s=1:S))

# cant sell more than the demand
@constraint(direct, dem[s=1:S], x[s] <= demand[s])

# cant sell more newspapers than purchased
@constraint(direct, bought[s=1:S], x[s] <= ybar)

solve(direct)

println("Obj. Val.: ", getobjectivevalue(direct))
println("x = ", getvalue(x))
println("dem dual = ", getdual(dem))
println("bought dual = ", getdual(bought))
#-------------------------------------------------------------------
```

Q3: The Benders sub-problem

We arrive at the Benders sub-problem by:

- Replacing the y variables with the $\overline{y}$ constants
- Since the y variables has dissapeared, we can remove a constant part of the objective, the term: $\sum_s \pi_s(h-c)y$
- Dualize the rest:
  - The dual problem is a Minimization problem
  - Since we have only one variable (x_s) we get only one type of constraints, and one actual constraint for each scenario s.
  - Since the 2 constraints in the direct model were $\leq$ for a maximization problem, we get positive variables: $(\alpha_s, \beta_s \in R^+)$

Min:

$$\sum_s D_s \cdot \alpha_s + \sum_s \overline{y} \cdot \beta_s$$

S.T.:

$$\begin{aligned} \alpha_s + \beta_s \quad &\geq \quad \pi_s(p-h) \quad \forall\ s \\ \alpha_s, \beta_s \in R^+ \end{aligned}$$

Q4: The first version of the Benders algorithm

Now we can implement the first version of the Benders algorithm:

```
# News Boy Benders algorith
# Notice: This is a maximization problem

#-------------------------------------------------------------------
# Intro definitions
using JuMP
using GLPKMathProgInterface
#-------------------------------------------------------------------

#************************************************************************
# Data
# Demand of newspapers in each scenario
demand=[ 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
S=length(demand)
# probability of scenario
prob=[0.05, 0.10, 0.10, 0.10, 0.15, 0.15, 0.10, 0.10, 0.10, 0.05 ]
c=20 # purchase price
p=70 # selling price
h=10 # scrap value
#************************************************************************

#-------------------------------------------------------------------
# Master problem
mas=Model(solver=GLPKSolverMIP())

# Variables
@variable(mas, q )

@variable(mas, 0 <= y <= 30, Int)
@objective(mas, Max, (h-c)*y + q)

function solve_master( alphabar, betabar )
    # Add Constraints
    @constraint(mas, sum( demand[s]*alphabar[s] for s=1:S) +
                     sum( y*betabar[s] for s=1:S) >= q)

    solve(mas)

    return getobjectivevalue(mas)

end
#-------------------------------------------------------------------
```

```
#-------------------------------------------------------------------
# Sub problem
function solve_sub( ybar )
    sub = Model(solver=GLPKSolverLP())

    @variable(sub, alpha[1:S] >= 0)

    @variable(sub, beta[1:S] >= 0)

    @objective(sub, Min, sum( demand[s]*alpha[s] for s=1:S) +
                         sum( ybar*beta[s] for s=1:S) )

    @constraint(sub, [s=1:S], alpha[s] + beta[s] >= prob[s]*(p-h) )

    solution = solve(sub)

    return (getobjectivevalue(sub), getvalue(alpha), getvalue(beta) )
end
#-------------------------------------------------------------------



#-------------------------------------------------------------------
# main code
let
    UB=Inf
    LB=-Inf
    Delta=0
    ybar=0
    it=1
    while (UB-LB>Delta)

        (sub_obj, alpha, beta)=solve_sub(ybar)

        LB=max(LB,sub_obj + (h-c)*ybar )

        mas_obj=solve_master(alpha,beta)

        ybar=getvalue(y)
        UB=mas_obj

        println("It: $(it) UB: $(UB) LB: $(LB)  Sub: $(sub_obj)")
        it+=1
```

```
    end
end
println("Correct Ending")
```

Q5: The Second version of the Benders algorithm

In the second version, we solve the sub-problem seperately for each scenario. The sub-problem then becomes:

Max_s:

$$D_s \cdot \alpha_s + \overline{y} \cdot \beta_s$$

S.T.:

$$\begin{aligned} \alpha_s + \beta_s &\geq \pi_s(p-h) \\ \alpha_s, \beta_s \in R^+ \end{aligned}$$

Inserting this in the previous Benders algorithm only changes it slightly:

```
# News Boy Benders algorith
# Notice: This is a maximization problem

#--------------------------------------------------------------------
# Intro definitions
using JuMP
using GLPKMathProgInterface
#--------------------------------------------------------------------

#*************************************************************************
# Data
# Demand of newspapers in each scenario
demand=[ 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
S=length(demand)
# probability of scenario
prob=[0.05, 0.10, 0.10, 0.10, 0.15, 0.15, 0.10, 0.10, 0.10, 0.05 ]
c=20 # purchase price
p=70 # selling price
h=10 # scrap value
#*************************************************************************

#--------------------------------------------------------------------
# Master problem
mas=Model(solver=GLPKSolverMIP())

# Variables
@variable(mas, q )

@variable(mas, 0 <= y <= 30, Int)
@objective(mas, Max, (h-c)*y + q)
```

```
function solve_master( alphabar, betabar )
    # Add Constraints
    @constraint(mas, sum( demand[s]*alphabar[s] for s=1:S) +
                     sum( y*betabar[s] for s=1:S) >= q)

    solve(mas)

    return getobjectivevalue(mas)

end
#-------------------------------------------------------------------

#-------------------------------------------------------------------
# Sub problem
function solve_sub_scenario( ybar, s)
    sub = Model(solver=GLPKSolverLP())

    @variable(sub, alpha >= 0)

    @variable(sub, beta >= 0)

    @objective(sub, Min, demand[s]*alpha + ybar*beta )

    @constraint(sub,  alpha + beta >= prob[s]*(p-h) )

    solution = solve(sub)

    return (getobjectivevalue(sub), getvalue(alpha), getvalue(beta) )
end
#-------------------------------------------------------------------

#-------------------------------------------------------------------
# main code
let
    UB=Inf
    LB=-Inf
    Delta=0
    ybar=0
    sub_obj=zeros(Float64,S)
    alpha=zeros(Float64,S)
    beta=zeros(Float64,S)
    it=1
    while (UB-LB>Delta)

        for s=1:S
            (sub_obj[s], alpha[s], beta[s])=solve_sub_scenario(ybar,s)
```

```
        end
        LB=max(LB,sum(sub_obj[s] for s=1:S) + (h-c)*ybar )

        mas_obj=solve_master(alpha,beta)

        ybar=getvalue(y)
        UB=mas_obj

        println("It: $(it) UB: $(UB) LB: $(LB)  Sub: $(sub_obj)")
        it+=1
    end
end
println("Correct Ending")
```

Q6: The third version of the Benders algorithm

If we look at the secnario dependent sub-problem, we realize that we do not need to solve an LP: If the demand $D_s > \overline{y}$ we set $\alpha = 0$ and $\beta = \pi_s(p-h)$ and if $D_s \leq \overline{y}$ we set $\alpha = \pi_s(p-h)$ and $\beta = 0$. The changed algorithm now looks as:

```
# News Boy Benders algorith
# Notice: This is a maximization problem

#---------------------------------------------------------------------
# Intro definitions
using JuMP
using GLPKMathProgInterface
#---------------------------------------------------------------------

#**************************************************************************
# Data
# Demand of newspapers in each scenario
demand=[ 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
S=length(demand)
# probability of scenario
prob=[0.05, 0.10, 0.10, 0.10, 0.15, 0.15, 0.10, 0.10, 0.10, 0.05 ]
c=20 # purchase price
p=70 # selling price
h=10 # scrap value
#**************************************************************************

#---------------------------------------------------------------------
# Master problem
mas=Model(solver=GLPKSolverMIP())

# Variables
@variable(mas, q )

@variable(mas, 0 <= y <= 30, Int)
@objective(mas, Max, (h-c)*y + q)

function solve_master( alphabar, betabar )
    # Add Constraints
    @constraint(mas, sum( demand[s]*alphabar[s] for s=1:S) +
                     sum( y*betabar[s] for s=1:S) >= q)

    solve(mas)

    return getobjectivevalue(mas)
```

```
end
#------------------------------------------------------------------

#------------------------------------------------------------------
# Sub problem
function solve_sub_scenario_ALGORITHMIC( ybar, s)
    if demand[s]>ybar
        alpha=0
        beta=prob[s]*(p-h)
    else
        alpha=prob[s]*(p-h)
        beta=0
    end
    sub_scen_algo=demand[s]*alpha + ybar*beta

    return (sub_scen_algo, alpha, beta )
end
#------------------------------------------------------------------

#------------------------------------------------------------------
# main code
let
    UB=Inf
    LB=-Inf
    Delta=0
    ybar=0
    sub_obj=zeros(Float64,S)
    alpha=zeros(Float64,S)
    beta=zeros(Float64,S)
    it=1
    while (UB-LB>Delta)

        for s=1:S
            (sub_obj[s], alpha[s], beta[s])=solve_sub_scenario_ALGORITHMIC(ybar,s)
        end
        LB=max(LB,sum(sub_obj[s] for s=1:S) + (h-c)*ybar )

        mas_obj=solve_master(alpha,beta)

        ybar=getvalue(y)
        UB=mas_obj

        println("It: $(it) UB: $(UB) LB: $(LB)  Sub: $(sub_obj)")
        it+=1
    end
```

```
end
println("Correct Ending")
```