In [10]:
```
using JuMP
using NLPModelsJuMP, NLPModels
using Gurobi
using Distributions
using Random
using DataFrames
using CSV
using Printf

function print_iteration(k, args...)
    f(x) = Printf.@sprintf("%12.4e", x)
    println(lpad(k, 9), " ", join(f.(args), " "))
    return
end

################################################################
# PARAMETERS
################################################################
p = 1

H = 4
J = 3
Tᵈ = 1
Tˢ = 1
Ξ = 1

s = [1.0  0.0  0.0;  0.0  1.0  0.0; 0.0  0.0  1.0; 1.0  0.0  0.0]

d = [
    0.7  0.8  0.8  0.6  0.9  1.0  1.0  0.9  0.0  1.0
    0.2  0.2  1.0  0.9  0.7  0.7  0.4  1.0  0.3  0.7
    0.5  0.6  0.5  0.1  0.4  0.6  0.4  0.0  0.5  0.6
    0.5  1.4  0.3  1.0  0.8  0.8  0.2  0.8  0.6  0.6
    0.8  0.2  1.3  0.6  0.3  0.4  0.6  0.8  0.4  0.8
    0.3  0.4  0.9  0.2  0.6  1.0  0.4  0.3  0.4  0.6
]


Random.seed!(1234)

#μ = [1.2, 1.8, 1.6]
#Σ = [0.5 0.0 0.0; 0.0 0.5 0.0; 0.0 0.0 0.5]
#dξ = reshape(rand(MvNormal(μ,Σ), Ξ ), (J,Tˢ,Ξ) )
#dξ = round.(dξ , digits = 2)

dˢ = [1.2 , 1.8 , 3.2]

χᵐᵃˣ = fill(1, H ,J)


zᵖ = [7; 9; 11;7 ] * 1000
zᶜ = zᵖ *2

zᵐ = fill(0, H, J)

for h=1
```

```
        zᵐ[h , 2] = (1/2) * zᵖ[h]
        zᵐ[h , 3] = (3/4) * zᵖ[h]
end

for h=2
        zᵐ[h , 3] = (1/2) * zᵖ[h]
        zᵐ[h , 1] = (3/4) * zᵖ[h]
end

for h=3
        zᵐ[h , 1] = (1/2) * zᵖ[h]
        zᵐ[h , 2] = (3/4) * zᵖ[h]
end

for h=4
        zᵐ[h , 2] = (1/2) * zᵖ[h]
        zᵐ[h , 3] = (3/4) * zᵖ[h]
end

;


##########################################
# main model
##########################################

main = Model(Gurobi.Optimizer)
@variable(main, ψ[1:H , 1:J], Bin)
@variable(main, 0 ≤ χ[1:H , 1:J])
@variable(main, α[1:H , 1:J , 1:Tᵈ], Bin)
@variable(main, -1000000 ≤ θ)


@objective(main, Min, sum( zᵖ[h] * ψ[h,j] * (Tᵈ + Tˢ) + zᵐ[h,j] * χ[h,j]
                           for h in 1:H for j in 1:J) + θ);

con1_s1 = @constraint(main, demand_met[ j in 1:J , t in 1:Tᵈ ],
    sum(α[h,j,t] * (ψ[h,j] + χ[h,j]) for h in 1:H ) ≥ d[j,t] );

con2_s1 = @constraint(main, permanent_pool[h in 1:H , j in 1:J], ψ[h,j] ≤ s[h,j]

con3_s1 = @constraint(main, allocation_recruitment_training[h in 1:H , j in 1:J ,
                      α[h,j,t] ≤ ψ[h,j] + 10 * χ[h,j] );

con4_s1 = @constraint(main, no_more_than_one_station[h in 1:H , t in 1:Tᵈ],
              sum(α[h,j,t] for j in 1:J) ≤ 1  )

con4_s1 = @constraint(main, single_skilling_at_recruitment[h in 1:H , j in 1:J],
    χ[h,j] ≤ ( 1 - ψ[h,j] ) * χᵐᵃˣ[h,j] );

con5_s1 = @constraint(main, training_recruited[ h in 1:H ],
    sum(χ[h,j] for j in 1:J ) ≤ sum(ψ[h,j] for j in 1:J) * J  );


##################################################
# this function optimize the sub problem and generates the value for αˢ
# αˢ stochastic allocation
```

```
# γˢ stochastic casual
################################################

function sub_solve(ψᵏ , χᵏ)
    sub = Model(Gurobi.Optimizer)
    @variable(sub, αˢ[1:H , 1:J , 1:Tˢ , 1:Ξ], Bin )
    @variable(sub, 0 ≤ γˢ[1:J , 1:Tˢ , 1:Ξ]);
    @objective(sub, Min,  (1/Ξ) * sum(zᶜ[j] * γˢ[j,t,ξ] for j in 1:J for t in 1:T
    con1_S2 = @constraint(sub, demand_met[ j in 1:J , t in 1:Tˢ , ξ in 1:Ξ],
     sum(αˢ[h,j,t,ξ] * (ψᵏ[h,j] + χᵏ[h,j]) for h in 1:H ) + γˢ[j,t,ξ] ≥ dˢ[j,t,ξ]
    con2_s2 = @constraint(sub, permanent_allocability[h in 1:H , j in 1:J , t in
            αˢ[h,j,t,ξ] ≤ ψᵏ[h,j] + 10 * χᵏ[h,j] );
    con3_s2 = @constraint(sub, no_more_than_one_station[h in 1:H , t in 1:Tˢ , ξ
           sum( αˢ[h,j,t,ξ] for j in 1:J) ≤ 1 );
    optimize!(sub);
    αˢ = value.(αˢ)
    γˢ = value.(γˢ)
    oˢ = objective_value(sub)
    αˢ = round.(αˢ , digits = 2)
    γˢ = round.(γˢ , digits = 2)
    oˢ = round.(oˢ , digits = 2)
    return (αˢ , γˢ , oˢ)
end




################################################################
# defining a function for second stage integer dual
###########################################

# when we want to get the dual the x (first stage variables are considered as fix
# when we want to get the coefficents of x, x should be variable
# therefore, we need to define two models one for dual and one for coefficient
###########################################
###############################################################





function sub_dual(ψᵏ , χᵏ , αˢᵏ)
    sub = Model(Gurobi.Optimizer)
    @variable(sub,  αˢ[1:H , 1:J , 1:Tˢ , 1:Ξ])
    @variable(sub, 0 ≤ γˢ[1:J , 1:Tˢ , 1:Ξ])
    @objective(sub, Min,  sum(zᶜ[j] * γˢ[j,t,ξ] for j in 1:J for t in 1:Tˢ for ξ

    for h in 1:H
            for j in 1:J
                for t in 1:Tˢ
                    for ξ in 1:Ξ
                        if αˢᵏ[h , j , t , ξ] == 0
                            con = @constraint(sub, αˢ[h,j,t,ξ] ≤ 0)
                        else
                            con = @constraint(sub, αˢ[h,j,t,ξ] ≤ 1)
                            con = @constraint(sub, - αˢ[h,j,t,ξ] ≤ - 1)
                        end
                    end
                end
```

```
            end
        end

        con1_S2 = @constraint(sub, demand_met[ j in 1:J , t in 1:Tˢ , ξ in 1:Ξ],
        - sum(αˢ[h,j,t,ξ] * (ψᵏ[h,j] + χᵏ[h,j]) for h in 1:H ) - γˢ[j,t,ξ]  ≤ - dˢ
    con2_s2 = @constraint(sub, permanent_allocability[h in 1:H , j in 1:J , t in
                αˢ[h,j,t,ξ] ≤ ψᵏ[h,j] + 10 * χᵏ[h,j] );
    con3_s2 = @constraint(sub, no_more_than_one_station[h in 1:H , t in 1:Tˢ , ξ
               sum( αˢ[h,j,t,ξ] for j in 1:J) ≤ 1 );
    optimize!(sub)


    con_equal = all_constraints(sub, AffExpr, MOI.EqualTo{Float64})
    con_less = all_constraints(sub, AffExpr, MOI.LessThan{Float64})
    λ1 = dual.(con_equal)
    λ2 = dual.(con_less)
    λ = append!(λ1 , λ2)

    #no_con_equal  = length(con_equal)
    #no_con_less = length(con_less)
    #no_all_con = no_con_equal + no_con_less;
    #@show no_con_equal
    #@show no_con_less
    #@show no_all_con;

    #print(sub)
    return λ
end





################################################
# function for coefficients of  ψ  and  χ
################################################

function sub_coef( αˢᵏ)
    sub = Model(Gurobi.Optimizer)
    @variable(sub,    ψ[1:H , 1:J], Bin )
    @variable(sub, 0 ≤ χ[1:H , 1:J])
    @variable(sub,  αˢ[1:H, 1:J , 1:Tˢ, 1:Ξ], Bin)
    @variable(sub, 0 ≤ γˢ[1:J , 1:Tˢ , 1:Ξ])
    @objective(sub, Min,  sum(zᶜ[j] * γˢ[j,t,ξ] for j in 1:J for t in 1:Tˢ for ξ
    for h in 1:H
            for j in 1:J
                for t in 1:Tˢ
                    for ξ in 1:Ξ
                        if αˢᵏ[h , j , t , ξ] == 0
                            con = @constraint(sub, αˢ[h,j,t,ξ] ≤ 0)
                        else
                            con = @constraint(sub, αˢ[h,j,t,ξ] ≤ 1)
                            con = @constraint(sub, - αˢ[h,j,t,ξ] ≤ - 1)
                        end
                    end
                end
            end
```

```
        end

        con1_S2 = @NLconstraint(sub, demand_met[ j in 1:J , t in 1:Tˢ , ξ in 1:Ξ]
      - sum(αˢ[h,j,t,ξ] * (ψ[h,j] + χ[h,j]) for h in 1:H ) - γˢ[j,t,ξ] ≤ - dˢ[j,t
    con2_s2 = @constraint(sub, permanent_allocability[h in 1:H , j in 1:J , t in
                αˢ[h,j,t,ξ] ≤ ψ[h,j] + 10 * χ[h,j] );
    con3_s2 = @constraint(sub, no_more_than_one_station[h in 1:H , t in 1:Tˢ , ξ
               sum( αˢ[h,j,t,ξ] for j in 1:J) ≤ 1 );

    vr = all_variables(sub)
    vr_index = [vr[i].index.value for i in 1:length(vr)]
    df = DataFrame(variable = vr , index = vr_index);
    #@show df

    nlp = MathOptNLPModel(sub)
    q = zeros(nlp.meta.nvar)
    jac(nlp, q)
    T = jac(nlp, q)[ : , 1:24]
    h1 = nlp.meta.ucon
    h2 = cons(nlp , q)
    h = h1 + h2
    return (T,h1, h2, h)
end





function initiate()
     for k in 1:5
        # main #########
        optimize!(main);
        ψᵏ = value.(ψ)
        χᵏ = value.(χ)
        θᵏ = value(θ)
        all_var = all_variables(main)
        xᵏ = all_var[1: (H*J*2)]
        # sub_solve #####
        αˢᵏ = sub_solve(ψᵏ , χᵏ)[1]
        γˢᵏ = sub_solve(ψᵏ , χᵏ)[2]
        oˢ = sub_solve(ψᵏ , χᵏ)[3]
        # sub_dual ######
        λᵏ = sub_dual(ψᵏ , χᵏ , αˢᵏ)
        # sub_coef ######
        Tᵏ = sub_coef(αˢᵏ)[1]
        hᵏ = sub_coef(αˢᵏ)[4]
        eᵏ = p * (λᵏ)' * hᵏ
        Eᵏ = p * (λᵏ)' * Tᵏ
        # w ? ϑ ##############
        wᵏ = eᵏ - Eᵏ * value.(xᵏ)
        println("**********************************************************")
        println("         k    θᵏ    wᵏ ")
        print_iteration( k ,   θᵏ  ,  wᵏ)
        println("**********************************************************")
        if θᵏ > wᵏ
            println("*********************************")
            println("       we have optimality        ")
```

```
            println("**********************************")
            break
        end
        cut = @constraint(main, θ ≥ eᵏ - Eᵏ * xᵏ)
        @info "we add the cut $(cut) "
    end
end


initiate()
```

```
  Objective range  [1e+04, 2e+04]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+01]
Presolve removed 34 rows and 15 columns
Presolve time: 0.00s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    6.5600000e+04   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.00 seconds (0.00 work units)
Optimal objective  6.560000000e+04

User-callback calls 29, time in user-callback 0.00 sec
Set parameter Username
Academic license - for non-commercial use only - expires 2023-09-05
Set parameter Username
Academic license - for non-commercial use only - expires 2023-09-05
*********************************************************
        k    θᵏ     wᵏ
        2  -4.5700e+05  -4.6800e+05
```

In [ ]: