ISE 533 Transshipment

•••

Group 5: Yueru Zhang, Xuchen Shao, Yongjun Kim, Zhaoqi Xiao

Workflow

Transshipment formulation

↓

Discretize the normal distributed demand
& Form the scenarios

↓

all-in-one LP

↓

Benders decomposition

- Solve master problem / set stopping criteria
- Solve subproblems to obtain the dual extreme points
- Build Benders Cuts and add cut to master; repeat

↓

Compare the results

Problem Statement

Problem:

There is 1 supplier and 7 non-identical retailers who face customer demands. The demand distribution at each retailer in a period is assumed to be known and stationary over time. The system inventory is reviewed periodically, and replenishment orders are placed with the supplier.

Goal:

Find the optimal order-up-to stock level quantity that minimize the expected long-run average cost (including the transshipment, holding, and penalty costs) over an infinite periods.

Data

Demand: the demand at each location are independent of each other and normally distributed

Location	1	2	3	4	5	6	7
Mean	100	200	150	170	180	170	170
Std Dev	20	50	30	50	40	30	50

Costs: transshipment cost, holding cost and shortage cost are across all locations

Symbol	Unit	Meaning	Value
h	dollar/unit	Per unit holding cost when there is excess inventory at one location	1.0
c	dollar/unit	Per unit transshipment cost to move inventory from one location to satisfy demand at another location	0.5
p	dollar/unit	Per unit shortage cost when demand cannot be met at one location	4.0

Data

Discretize the normal distribution:

For each location, we discretize the levels into low, medium, high, using quantiles

```
D_sample=[]
for i = 1:length(D)
    append!(D_sample,[quantile.(D[i],[1/6, 3/6, 5/6])])
end
```

Low, Medium, High

We get 3^7=2187 different scenarios

```
fans=[]
function arrangement(x,ans)
    if length(x)>0
        for i=1:length(D_sample[1])
            push!(ans,x[1][i])
            #println(i)
            #println(ans)
            if length(x)>=2
                arrangement(x[2:length(x)],ans[:])

            else
                push!(fans,ans[:])

            end
            pop!(ans)
            #print(i)
        end

    end
end
arrangement(D_sample,[])
```

All-in-One Model

$$\min_{S\geq 0} E[h(S,\tilde{D})],$$

$$h(S,D) = \min \sum_i h_i e_i + \sum_{i\neq j} c_{ij} t_{ij} + \sum_i p_i r_i.$$

Variables:

s_i order-up-to

e_i = ending inventory held at retailer i.
f_i = stock at retailer i used to satisfy demand at retailer i.
q_i = inventory at retailer i increased through replenishment.
r_i = amount of shortage met after replenishment at retailer i.
t_{ij} = stock at retailer i used to meet demand at retailer j, using the transshipment option.

h_i = unit cost of holding inventory at retailer i.
c_{ij} = unit cost of transshipment from retailer i to j.
p_i = penalty cost for shortage at retailer i.

$$f_i + \sum_{j\neq i} t_{ij} + e_i = s_i, \quad \forall i$$

$$f_i + \sum_{j\neq i} t_{ji} + r_i = d_i, \quad \forall i$$

$$\sum_i r_i + \sum_i q_i = \sum_i d_i$$

$$e_i + q_i = s_i, \quad \forall i$$

$$e_i, f_i, q_i, r_i, s_i, t_{ij} \geq 0, \quad \forall i, j.$$

All-in-One Model-Stochastic Quasi-Gradient (SQG) Method

Steps:

1. Initialize a random S
2. Solve the dual problem $\text{maximize} \quad \sum_i s_i B_i + \sum_i d_i M_i + \sum_i d_i R + \sum_i s_i E_i$ with S and different demand scenarios 1......M.
3. Get the sum of dual multipliers $B^{k,m}$ and $E^{k,m}$ for each scenario. Here, k is iteration and m is scenario.
4. Compute unbiased estimate of a subgradient. $\hat{\xi}^k = \frac{1}{M} \sum_{m=1}^{M} (B^{k,m} + E^{k,m}).$
5. Update s with the iterative scheme $S^{k+1} = P_+(S^k - \alpha_k \hat{\xi}^k),$
6. Repeat step 2 - 5 until maximum number of iteration is met.

Choose α_k to be 0.001 because of the conditions $\alpha_k \to 0, \quad \sum_k \alpha_k \to \infty, \quad \text{and} \quad \sum_k \alpha_k^2 < \infty.$

All-in-One Model-Stochastic Quasi-Gradient (SQG) Method

```
function get_A_D_dual(s,d,m)


    model = JuMP.Model(CPLEX.Optimizer)

    set_silent(model)

    @assert length(s)==N
    @assert length(d)==N

    t_set = [(i,j) for i=1:N for j=1:N if i!=j]

    @variables(model,begin
        f[1:N] >= 0
        e[1:N] >= 0
        t[t_set] >=0
        q[1:N] >=0
        r[1:N] >=0
    end)

    @constraints(model,begin

    A[i=1:N], f[i] + sum(t[(i,j)] for j=1:N if i!=j) + e[i] == s[i]
    B[i=1:N], f[i] + sum(t[(j,i)] for j=1:N if j!=i) + r[i] == d[i]
    C,sum(r)+sum(q)==sum(d)
    D[i=1:N], e[i] + q[i] == s[i]

    end)

    @objective(model,Min, h*sum(e) + c*sum(t) + p*sum(r))

    optimize!(model)

    pi_A = dual.(A)
    pi_D = dual.(D)

    return pi_A+pi_D

end
```

```
s = fans[length(fans)]
d_sam = fans
N = length(s)
alpha = 0.001
grad = []
#grad::Vector{Float64}=zero(N)

@time for j=1:100
            grad=[]
            for i=1:length(d_sam)
                push!(grad,get_A_D_dual(s,d_sam[i],i))
                if rem(i,1000)==0
                    println(i/length(d_sam))
                end
            end
            #println(grad)
            #println(mean(grad))
            ns = s - (mean(grad).*alpha)
            #println(ns)
            for k in eachindex(ns)
                if ns[k]>0
                    s[k]=ns[k]
                end
            end
            println("iter:",j)
        end

println("result_S: ",s)
```

All-in-One Model-Stochastic Quasi-Gradient (SQG) Method

```
]: s1 = [round(ss) for ss in s]

7-element Vector{Float64}:
 126.0
 264.0
 188.0
 234.0
 231.0
 208.0
 234.0
```

```
obj_avg

126.83333333333346
```

Benders Decomposition

Divide the objective function into two parts;

- Master problem: $\min_{S \geq 0} \eta$
- Subproblem: $\min \sum_{i} h_i e_i + \sum_{i \neq j} c_{ij} t_{ij} + \sum_{i} p_i r_i.$

  Solve the Master problem while solving the Subproblem to determine Benders cut and adding it as a new constraint.

  Start with random initial S (S >= 0) and iterate to solve the problem until a gap ((UB - LB)/LB) is smaller than Epsilon (0.000001).

  Set a dual objective value as upper bound and eta as lower bound.

Benders Decomposition

```
a,b,d_obj = get_aggregate_cuts(s_k)
```

```
@objective(model, Min, eta)
```

```
for i in eachindex(alpha)
    @constraint(model, alpha[i]+sum(beta[i][j]*s[j] for j=1:N)<=eta)
end
```

```
upper_bound = min(d_obj,upper_bound)
lower_bound = objective_value(model)
```

```
gap = (upper_bound -lower_bound)/lower_bound
if gap<0.000001
    break
end
```

Result Comparison

Model	SQG	Benders Decomposition
Running time (sec)	65.6448	61.1864
Objective value	126.8333	127.8360
S (Order-up-to quantity)	126 264 188 234 231 208 234	100 200 150 218 209 170 170
Iteration	100	73

Thank you!