HW 4

Submitted by Aman Jain

aj644

29th Sept 2019

Problem5

####################### Base model for part 1 ################

# master sets of customers and locations

**set** customer; #set of customers

**set** location; #set of candidate locations

#time periods

**param** n > 0 **integer**; #Number of locations

**param** m > 0 **integer**; #Number of customers

**param** T > 0 **integer**; #Number of time periods

####### Defining params #######

## Operation data of locations

**param** f\_operating {location,1..T} >= 0; #Fixed cost of operating at a location during a timeperiod.

**param** f\_plus {location,1..T} >= 0; #Fixed cost of Opening at a location during a timeperiod.

**param** f\_minus {location,1..T} >= 0; #Fixed cost of Closing at a location during a timeperiod.

**var** y {location,0} >= 0; # Data whether a location is in service in the beginning, T= 0.

## Capacity data of locations

**param** b {location} >= 0; # incremental capacity that can be added at a location.

**param** m {location} >= 0; # minimum capacity at a location.

**param** M {location} >= 0; # maximum capacity at a location.

**param** h {location} >= 0; # current capacity at a location.

**param** g\_operating {location,1..T} >= 0; # cost of operating an additional unit of capacity at an operating location during a timeperiod.

**param** g\_plus {location,1..T} >= 0; # cost of operating an additional unit at a new location opened during a timeperiod.

**param** g\_minus {location,1..T} >= 0; # cost of operating an additional unit at a new location opened during a timeperiod.

## Customer demand data

**param** c {location,customer,1..T} >= 0; # Fixed cost of serving a customer from a location during a timeperiod.

**param** e {location,customer,1..T} >= 0; # Variable cost of serving a customer from a location during a timeperiod.

**param** d {customer,1..T} >= 0; # Demand of a customer during a timeperiod.

####### Defining Variables #######

## Operation variable of locations

**var** y {location,1..T} >= 0 **binary**; # whether a location is being operated in a time period.

**var** y\_plus {location,1..T} >= 0 **binary**; # whether a location is Open in a time period.

**var** y\_minus {location,1..T} >= 0 **binary**; # whether a location is Closed in a time period.

## Capacity variable of locations

**var** z {location,1..T} >= 0 **integer**; # amount of additional capacity being operated at a location in a time period.

**var** z\_plus {location,1..T} >= 0 **integer**; # amount of additional capacity added to a location being opened in a time period.

**var** z\_minus {location,1..T} >= 0 **integer**; # amount of additional capacity added to a location being closed in a time period.

#We will be adding constraint to take care of z\_minus does logically coincides with y\_minus.

## Customer demand variables

**var** x {location,customer,1..T} >= 0 **binary**; # whether a location fulfills demand of a customer in a time period.

**var** w {location,customer,1..T} >= 0; # how much of demand of a customer is fulfilled by a location in a time period.

####### Defining Objective #######

**minimize** total\_cost:

**sum** {t **in** 1..T}

( **sum** {i **in** location}

(y[i,t]\*f\_operating[i,t] + y\_plus[i,t]\*f\_plus[i,t] + y\_minus[i,t]\*f\_minus[i,t] #cost of operating/open/close a location.

+ z[i,t]\*g\_operating[i,t] + z\_plus[i,t]\*g\_plus[i,t] + z\_minus[i,t]\*g\_minus[i,t]) #cost of operating/open/close additional capacities.

+ **sum** {j **in** customer}

(x[i,j,t]\*c[i,j,t] + w[i,j,t]\*e[i,j,t]))); #fixed and variable costs of serving demand from a locaitons.

####### Defining Constraints #######

## Operation constraint of locations

# Constraint\_1 : Can only close an open facility

**s.t.** Constraint\_1 {i **in** location, t **in** 0..T}:

y\_minus[i,t] <= y[i,t-1];

# Constraint\_2 : Can only open a closed facility

**s.t.** Constraint\_2 {i **in** location, t **in** 0..T}:

y\_plus[i,t] <= 1 - y[i,t-1];

# Constraint\_3 : balance equaiton for locations from t-1 to t

**s.t.** Constraint\_3 {i **in** location, t **in** 0..T}:

y[i,t] = y[i,t-1] + y\_plus[i,t] - y\_minus[i,t];

## Capacity constraint of locations

# Constraint\_4 : Can only have capacity at an open location

**s.t.** Constraint\_4{i **in** location, t **in** 1..T}:

z[i,t] <= y[i,t]\*((M[i]-h[i])/b[i]);

# Constraint\_5 : balance equaiton for additional capacity from t-1 to t

**s.t.** Constraint\_5{i **in** location, t **in** 0..T}:

z[i,t] = z[i,t-1] + z\_plus[i,t] - z\_minus[i,t];

## Customer demand constraints

# Constraint\_6 : Demand balance constraint - demand fulfilled from all locaiton equals total demand of customer.

**s.t.** Constraint\_6{j **in** customer, t **in** 1..T}:

**sum** {i **in** location} w[i,j,t] = d[j,t];

# Constraint\_7 : if we are using a location to fill demand, we ensure to take fixed cost into account

**s.t.** Constraint\_7{i **in** location, j **in** customer, t **in** 1..T}:

w[i,j,t] <= d[j,t]\*x[i,j,t];

# Constraint\_8 : For all locations, demand served is less than equal to current + additional capacity

**s.t.** Constraint\_8{i **in** location, t **in** 1..T}:

**sum** {j **in** customer} w[i,j,t] <= h[i]\*y[i,t] + b[i]\*z[i,t];

# Constraint\_9 : For all locations, current + additional capacity <= max capacity

**s.t.** Constraint\_8{i **in** location, t **in** 1..T}:

h[i]\*y[i,t] + b[i]\*z[i,t] <= M[j];

# Constraint\_10 : Ensuring minimum utilization of capacity

**s.t.** Constraint\_10{i **in** location, t **in** 1..T}:

**sum** {j **in** customer} w[i,j,t] >= m[i]\*y[i,t];

# Constraint\_11 : All customers are served from somewhere

**s.t.** Constraint\_11 {j **in** customer, t **in** 1..T}:

**sum** {i **in** location} x[i,j,t] = 1;

############# Additional Changes for part 2 #############

# Constraint\_2\_a : Limits on the total number of facilities open in a time period. At least l and at most L

**s.t.** Constraint\_2\_a\_1 {t **in** 1..T}:

**sum** {i **in** location} y\_plus[i,t] >= l;

**s.t.** Constraint\_2\_a\_2 {t **in** 1..T}:

**sum** {i **in** location} y\_plus[i,t] <= L;

# Constraint\_2\_b : Limits on the number of facilities open in a time period in some subsets of locations

**s.t.** Constraint\_2\_b\_1 {t **in** 1..T}:

**sum** {i **in** location\_subset} y\_plus[i,t] >= l;

**s.t.** Constraint\_2\_b\_2 {t **in** 1..T}:

**sum** {i **in** location\_subset} y\_plus[i,t] <= L;

#and the special case of at most one of these locations.

**s.t.** Constraint\_2\_b\_3 {t **in** 1..T}:

**sum** {i **in** location\_subset} y\_plus[i,t] <= 1;

#and the special case of at least one of these locations.

**s.t.** Constraint\_2\_b\_4 {t **in** 1..T}:

**sum** {i **in** location\_subset} y\_plus[i,t] >= 1;

# Constraint\_2\_c : Limits on the number of distinct customers served at a facility in any time period

**s.t.** Constraint\_2\_c {i **in** location, t **in** 1..T}:

**sum** {j **in** customer} x[i,j,t] <= max\_customer\_can\_be\_served\_from\_a\_location;

# Constraint\_2\_d : Limits on the number of distinct customers from some specified set served at a location

#in a time periods, including the special case of these customers must all be served by different facilities

**s.t.** Constraint\_2\_d {i **in** location, t **in** 1..T}:

**sum** {j **in** customer\_set} x[i,j,t] = 1;

# Constraint\_2\_e : Limits on the number of facilities used during some interval to serve a single customer

**s.t.** Constraint\_2\_e {j **in** customer, t **in** 1..T}:

**sum** {i **in** location} x[i,j,t] <= max\_location\_used\_to\_serve\_a\_customer;

# Constraint\_2\_f : Limits on the number of facilities used during some interval to serve a set of customers.

**s.t.** Constraint\_2\_f {t **in** 1..T}:

**sum** {i **in** location, j **in** customer\_set} x[i,j,t] <= max\_location\_used\_to\_serve\_a\_customer;

############# Additional Changes for part 3 #############

###### Answer part 3(a)

## Additional Customer location assignment variables

**var** xij {location,customer,0..T} >= 0 **binary**; # whether a location fulfills demand of a customer at time period.

**var** xij\_plus {location,customer,1..T} >= 0; # if a location starts to fullfill demand of a customer at time period T.

**var** xij\_minus {location,customer,1..T} >= 0; # if a location stops to fullfill demand of a customer at time period T.

## Additional assignment constraints

# Constraint\_3\_a\_1 : Can only stop an ongoing assignment

**s.t.** Constraint\_3\_a\_1 {i **in** location, j **in** customer, t **in** 1..T}:

xij\_minus[i,j,t] <= xij[i,j,t-1];

# Constraint\_3\_a\_2 : Can only start a no current assignment

**s.t.** Constraint\_3\_a\_2 {i **in** location, j **in** customer, t **in** 1..T}:

xij\_plus[i,j,t] <= 1 - xij[i,j,t-1];

# Constraint\_3\_a\_3 : balance equaiton for assignment from t-1 to t

**s.t.** Constraint\_3\_a\_3 {i **in** location, j **in** customer, t **in** 1..T}:

xij[i,j,t] = xij[i,j,t-1] + xij\_plus[i,j,t] - xij\_minus[i,j,t];

# Constraint\_3\_a\_4 : Limit on the number of customers switching facilities in any time period.

# note that a customer shifting from one location to another is considered as 2 switches.

# one for stopping with current location and second for starting at a new location.

**s.t.** Constraint\_3\_a\_4 {t **in** 1..T}:

**sum** {i **in** location, j **in** customer} (xij\_plus[i,j,t] + xij\_minus[i,j,t]) <= max\_number\_of\_switchs;

###### Answer part 3(b)

# Constraint\_3\_b : Limit on the number of switches by a customer over time.

# note that a customer shifting from one location to another is considered as 2 switches.

# one for stopping with current location and second for starting at a new location.

**s.t.** Constraint\_3\_a\_4 {j **in** customer}:

**sum** {i **in** location, t **in** 1..T} (xij\_plus[i,j,t] + xij\_minus[i,j,t]) <= max\_number\_of\_switchs\_by\_a customer;

Problem 6

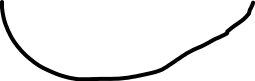
Zip code 2

Zip code 1

Bulk mailers

Truck

Rail/Cargo



………………….

…………………..................

Sorting n

Sorting 1

Zip m-1

Zip code m

###### PART 1 #######

########################## Base model ########################

# master sets of zip codes and sorting facilities

**set** zip; #set of zip codes

**set** sf\_current; #current set of sorting facility (around 800)

**set** sf\_potential; #set of potentially new sorting facility (around 100)

**set** sf = sf\_current **union** sf\_potential; #set of all potential sorting facility

**set** congress\_district; # set of all congress districts.

#parameters

**param** n > 0 **integer**; #Number of sorting facilities

**param** m > 0 **integer**; #Number of zip codes

####### Defining params #######

## Origination and Destination demand volumes.

**param** volume\_originated {zip} >= 0; #Volume originating from a zip code.

**param** volume\_destinated {zip} >= 0; #Volume to be delivered to (or destined to) a zip code.

## Open/close of sorting facilites

**param** prior\_status{sf} >= 0 ; # Current sorting facilities (sf\_current) marked as 1 and rest as 0.

**param** sf\_open {sf\_potential} >= 0; #Fixed cost of setting up new sorting facility.

**param** sf\_shut {sf\_current} >= 0; #Fixed cost of shutting down a current facility.

## sorting facility capacities

**param** capacity{sf} >= 0; # Max capacities for all sorting facilities

**param** sf\_handling\_cost{sf} >= 0; # Per unit handling costs at each sorting facility.

## transportation costs; assuming transportation costs are same in either direction for a pair.

**param** zip\_to\_sf\_transportation {zip, sf} >= 0; # Transportation cost from each zip to sorting facilities.

**param** sf\_to\_sf\_transportation {sf, sf} >= 0; # Transportation cost from a sorting facility to another sorting facility.

## district to sf mapping

**param** district\_to\_sf\_mapping {congress\_district,sf} >= 0; # Binary mapping, 1 if a sf falls in that congress district else 0.

####### Defining Variables #######

**var** new\_status{sf} >= 0 **binary**; # whether a particular sf is open or close in final answer.

**var** zip\_sf\_mapping{zip, sf} >= 0 **binary**; # which all zipcodes assigned to which all sf in new arrangement.

###### Defining Objective #######

**minimize** total\_cost:

(**sum** {i **in** zip}

(volume\_originated[i]\* **sum**{j **in** sf} (zip\_to\_sf\_transportation[i,j]\*zip\_sf\_mapping[i,j]) #cost of first leg of transportation.

+ (volume\_destinated[i]\***sum**{j **in** sf} (zip\_to\_sf\_transportation[i,j]\*zip\_sf\_mapping[i,j])) #cost of last leg of transportation.

+ #cost of middle leg of transportation.(sf to sf)

+ (**sum** {j **in** sf, i **in** zip} sf\_handling\_cost[j]\*(volume\_originated[i]\*zip\_sf\_mapping[i,j] + volume\_destinated[i]\*zip\_sf\_mapping[i,j])

# cost of handling at each sf. Note that cost of handling at zipcode level are not considered here as those costs are not

# dependent on the variable decision of this problem.

+ (**sum** {j **in** sf\_current} sf\_shut[j]\*(1-new\_status[j])) #cost of shutting down an existing sf.

+ (**sum** {j **in** sf\_potential} sf\_open[j]\*new\_status[j]) #cost of opening a new sf.

####### Defining Constraints #######

# Constraint\_1: One zip code is mapped to one sf only.

**s.t.** Constraint\_1 {i **in** zip}:

**sum** {j **in** sf} zip\_sf\_mapping[i,j] = 1;

# Constraint\_2: incoming volume at a hub is within capacity.

**s.t.** Constraint\_2 {j **in** sf}:

**sum** {i **in** zip} (volume\_originated[i]\*zip\_sf\_mapping[i,j]) <= capacity[j];

# Constraint\_3: outgoing volume from a hub is within capacity.

**s.t.** Constraint\_3 {j **in** sf}:

**sum** {i **in** zip} (volume\_destinated[i]\*zip\_sf\_mapping[i,j]) <= capacity[j];

# Constraint\_4 : Current open sf can only be open or close # Redundant though as new\_status is defined as binary only.

**s.t.** Constraint\_4 {j **in** sf\_current}:

new\_status[j] <= 1

###### PART 2 #######

#As shown in table below origination and destination data can be obtained by having to\_zip and from\_zip cross data.

If To and from data between each pair is given we can obtain, zip origination and zip destination volumes as follow:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| From / To | Z1 | Z2 | Z3 | Zip origination |
| Z1 | 10 | 20 | 30 | 60 |
| Z2 | 5 | 10 | 15 | 30 |
| Z3 | 10 | 10 | 30 | 50 |
| Zip destination | 25 | 40 | 75 |  |

## To and from demand volumes.

**param** volume\_from\_zip\_to\_zip {zip, zip} >= 0; #Volume originating from a zip code toward another zip code.

# Modify Objective function to include cost of movement between

**minimize** total\_cost = cost function **in** part A

+ (**sum** {i **in** zip, j **in** zip, k **in** sf} volume\_from\_zip\_to\_zip[i,j]\*sf\_to\_sf\_transportation[zip\_sf\_mapping[i,k], zip\_sf\_mapping[j,k]]);

###### PART 3 #######

# Constraint\_5 : Additional constraint to ensure atleast one facility be located within each congressman district

**s.t.** Constraint\_5 {i **in** congress\_district}:

**sum** {j **in** sf} district\_to\_sf\_mapping[i,j]\*new\_status[j] >= 1

###### PART 4 #######

# Constraint\_6 : Balancing between congressman district, one of doing that is to restrict each congress district to some range.

**s.t.** Constraint\_6 {i **in** congress\_district}:

**sum** {j **in** sf} district\_to\_sf\_mapping[i,j]\*new\_status[j] <= 3 # each district will have between 1,2 or 3 sorting facilities.

###### PART 5 ######

#Modify your model to select from amongst the solutions that minimize the USPS cost, the solution that minimizes the large mailer costs.

**set** bulk\_senders: # set of all bulk senders.

**param** bulk\_to\_sf\_transportation {bulk\_senders, sf} >= 0; # Transportation cost from a bulk senders to send to each sorting facility.

**param** bulk\_to\_sf\_volume {bulk\_senders, zip } >= 0; # Transportation cost from a bulk senders to send to each sorting facility.

# Modify Objective function:

**minimize** total\_cost: C1\*cost function **in** part A

+ C2\* **sum** {i **in** bulk\_senders, j **in** zip} (bulk\_to\_sf\_volume[j,i]\***sum** {k **in** sf} zip\_sf\_mapping[j,k]\*bulk\_to\_sf\_transportation[i,k]

# C1 and C2 represents the relative weight to be given to USPS and bulk sender's costs.

###### PART 6 ######

#The bulk mailers also have concerns over fairness. None wants to incur unnecessary costs for the benefit of the others

**param** bulk\_senders\_costs {bulk\_senders} >= 0; # Current costs of bulk senders.

# Constraint\_7 : Putting constriant to ensure total cost for each bulk sender is within certain rainge of current costs.

**s.t.** Constraint\_7 {i **in** bulk\_senders}:

**sum** {j **in** zip, k **in** sf} (bulk\_to\_sf\_volume[j,i]\***sum** {k **in** sf} zip\_sf\_mapping[j,k]\*bulk\_to\_sf\_transportation[i,k]) <= 1.2\*bulk\_senders\_costs(i)

# ensures new costs to be within 20% of original costs.