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.
```

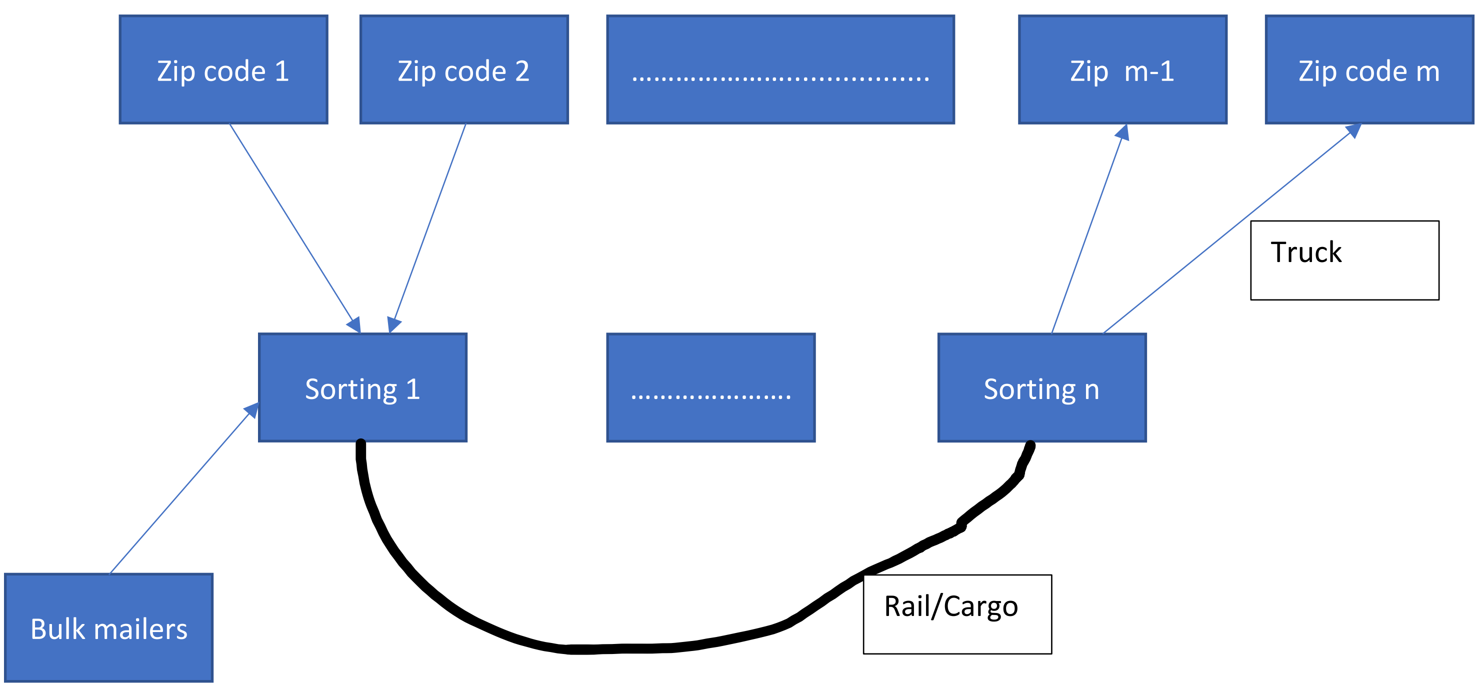
```
####### Defining Constraints #######

## 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




```
###### 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.
```