# Special problem, Grocery Store Location. ECE 719 Optimal systems


Spring 2016
Electrical engineering department
University of Wisconsin, Madison

Instructor: Professor B Ross Barmish

By

Nasser M. Abbasi


December 30, 2019

# Contents

0.1 Introduction . . . . . 3
0.2 Analysis of the problem . . . . . 3
0.3 Algorithm description . . . . . 6
  0.3.1 Description of test cases used in development . . . . . 8
0.4 Result applying the algorithm to the supplied input . . . . . 9
0.5 Conclusion . . . . . 10
0.6 References . . . . . 11
0.7 Appendix . . . . . 11

# List of Figures

1 Partitions found by kmeans++ with centroid as green dots and competitor sores as black dots . . . . . 5
2 Density of population with corresponding store locations found . . . . . 5
3 Partitions found by kmeans++ with centroid as green dots and competitor sores as black dots . . . . . 5
4 Density of population with corresponding store locations found . . . . . 5
5 Test case 1 . . . . . 10
6 Test case 2 . . . . . 10
7 Test case 3 . . . . . 10
8 Test case 4 . . . . . 10

# List of Tables

1 Summary of store location score of each test case . . . . . 9

**Abstract**

k-means++ cluster analysis was used to partition the population area such that the center of each partition minimizes the within the partition sum of distance squares of each point in the partition to the center of the partition. The number of customers that would visit our stores located at the center of the partitions was then determined. The number of partitions was increased and the calculation repeated on the larger set by trying all of the different combinations of allocating the stores in the new and larger set of partitions. The largest score was selected. Matlab's `kmeans` implementation in the `Statistics and Machine Learning Toolbox` was used to find the set of partitions and their centroids. kmeans++ clustering is known to be computationally NP-hard problem[1]. In addition, the time complexity to analyze each set of partitions is $\mathcal{O}\left(N\binom{p}{n}\right)$ where $p$ is the number partitions and $N$ is the size of the population. This number quickly becomes very large therefore the implementation limits the number of partitions $p$ to no more than 15. A number of small test cases with known optimal store locations were constructed and the algorithm was verified to be correct by direct observations. Locations of competitor stores do not affect the decision to where to locate our stores. Competitor stores locations affects the number of customers our stores will attract, but not the optimal locations of our stores.

***Index terms***— k-means++ clustering, NP-hard, optimal store locations

## 0.1 Introduction

The problem is the following: We want to locate $n$ stores in an area of given population distribution where there already exists $m$ number of competitor stores. We are given the locations of the competitor stores. We want to find the optimal locations of our $n$ stores such that we attract the largest number of customers by being close to as many as possible. We are given the locations (coordinates) of the population.

## 0.2 Analysis of the problem

The first important observation found is that the locations of the competitor stores did not affect the decision where the location of our stores should be. This at first seemed counter intuitive. But the optimal solution is to put our stores at the center of the most populated partitions even if the competitor store happened to also be in the same exact location.

The idea is that it is better to split large number of customers with the competition, than locate a store to attract all customers but in a less populated area. This was verified using small test cases (not shown here due to space limitation).

This is where cluster analysis using the kmeans++ algorithm was used. Cluster analysis is known algorithm that partitions population into number of clusters or partitions such that each cluster has the property that its centroid has minimum within-cluster sum of squares of the distance to each point in the cluster. The following is the formal definition of kmeans++ clustering algorithm taken from `https://en.wikipedia.org/wiki/K-means_clustering`

[1] non-deterministic polynomial-time hard

Given a set of observations $(x_1, x_1, \dots, x_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the $n$ observations into $k \leq n$ sets $S = \{S_1, S_2, \dots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center). In other words, its objective is to find:

$$\underset{S}{\arg\min} \sum_{i=1}^{k} \sum_{x \in S_i} \left\| \boldsymbol{x} - \mu_i \right\|^2$$

where $\mu_i$ is the mean of points in $S_i$

The main difficulty was in deciding on the number of partitions needed. Should we try smaller number than the number of stores, and therefore put more than one store in same location? Using smaller number of clusters than the number of stores was rejected, since it leads to no improvement in the score (Putting two stores in same location means other areas are not served since we have limited number of stores). Or should we try more partitions than the number of our stores, and then try all the combinations possible between these partitions to find one which gives the larger score? This the approach taken in this algorithm. It was found that by increasing the number of partitions than the number of stores, and trying all possible combinations $\binom{p}{n}$, where $p$ is the number of partitions, a set could be found which has higher than if we used the same number of partitions as the number of stores. The problem with this method is that it has $\mathcal{O}\binom{p}{n}$ time complexity. This quickly becomes large and not practical when $p > 15$. In the test cases used however, there was no case found where $p$ had to be more than two or three larger than $n$. The implementing limits the number of partitions to 15.

When a score is found which is smaller than the previous score, the search stops as this means the maximum was reached. This was determined by number of trials where it was found that once the score become smaller than before, making more partitions did not make it go up again. There is no proof of this, but this was only based on trials and observations. Therefore, the search stops when a score starts to decrease.

The implementation described here is essentially an illustration of the use of cluster analysis, as provided by Matlab, in order to solve the grocery stores location problem. The appendix contains the source code written to solve this problem.

Before describing the algorithm below, we show an example using the early test send to us to illustrate how this method works. This used $500,000$ population with 5 competitor stores (the black dots) in the plots that follows and $n = 4$ (the green dots).

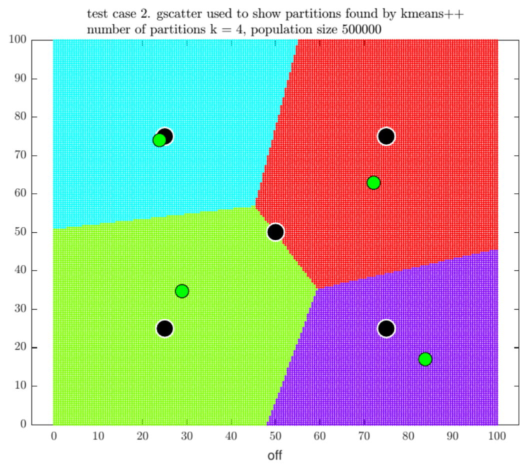



Figure 1: Partitions found by kmeans++ with centroid as green dots and competitor sores as black dots

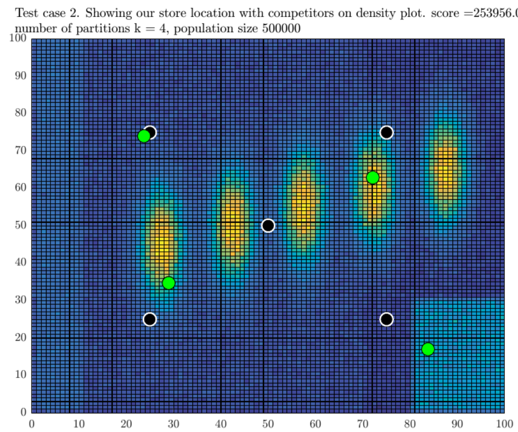



Figure 2: Density of population with corresponding store locations found

The partitions were now increased to 5 and $\binom{5}{4}$ different combinations were scored to find the 4 best store locations out of these. This resulted in the following result

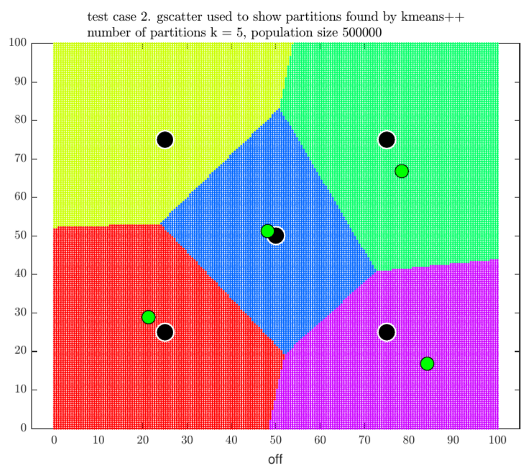



Figure 3: Partitions found by kmeans++ with centroid as green dots and competitor sores as black dots

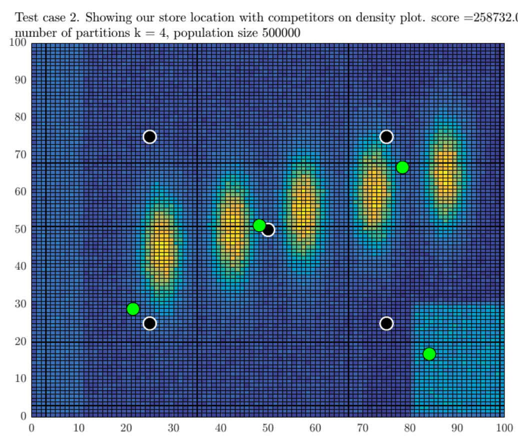



Figure 4: Density of population with corresponding store locations found

When trying 6 partitions, the score was decreased, so the search stopped. The program then printed the final result

```
J*=[ 258732.0000] = [%51.75]

 x              y
21.356   28.929
78.378   66.732
```

```
48.212  51.197
84.078  16.864
```

## 0.3 Algorithm description

This is a description of the algorithm which uses the kmeans++ cluster analysis function `kmeans()` as part of the Matlab `Statistics and Machine Learning Toolbox` toolbox, which is included in the student version. This is not a description of the kmeans++ algorithm itself, since that is well described and documented in many places such as in references [3,4]. This is a description of the algorithm using kmeans to solve the grocery location problem.

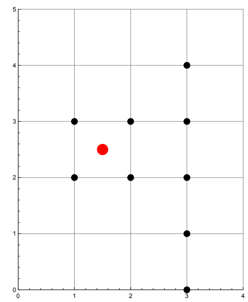
**Algorithm 1**: Cluster analysis using Kmeans++ for determining optimal store locations

**Input**: $n, X, Y$ where $n$ is the number of stores to allocate, $X$ is population coordinates, and $Y$ is competitor store location coordinates
**Output**: list of coordinates to locate our $n$ stores at, and $J^*$ which is size of population our stores will attract when placed at these locations

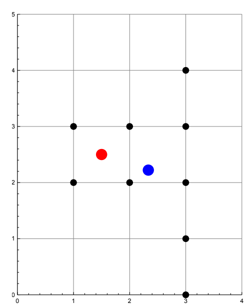
```
currentNumberOfPartitions ⟵ n
bestScore ⟵ 0
keepSearching ⟵ true
bestLocations ⟵ {}
while keepSearching do
    C ⟵ kmeans(currentNumberOfPartitions,X) /* C now contains the centroid of partitions found by kmeans++
        cluster analysis algorithm using Matlab toolbox */
    partitionSets ⟵ combnk(1:size(C,1),n) /* Find all possible combinations of partitions. Warning, this is binom(k, n)
        which will quickly grow. In practice, it was found we do not need k greater than n + 4 to find a
        maximum. n is limited to 10. */
    partitionScore ⟵ 0
    winningCombination ⟵ {}
    foreach e ∈ partitionSets do
        score ⟵ 0
        foreach x_i ∈ X do
            d_1 ⟵ shortest distance of x_i to any of the centroid of the partition e
            d_2 ⟵ shortest distance of x_i to any of competitor stores in Y
            if d_1 ≤ d_2                 /* win this customer or split it. Else competitor is closer */
             then
                if d_1 = d_2 then
                    score ⟵ score + 1/2
                else
                    score ⟵ score + 1
                end
            end
        end
        if score ≥ partitionScore then
            partitionScore ⟵ score
            winningCombination ⟵ e
        end
    end
    if partitionScore ≥ bestScore then
        /* score did not go down, keep searching. Increase number of population partitions by one and call
            kmeans++ (above) for new partitions */
        bestScore ⟵ partitionScore
        bestLocations ⟵ winningCombination
        if currentNumberOfPartitions = 15/* stop search if binom(k>=15ink, n) due to limitation */
         then
            keepSearching ⟵ false
        else
            currentNumberOfPartitions ⟵ currentNumberOfPartitions + 1
        end
    else
        /* when score goes down, it will not improve any more */
        keepSearching ⟵ false
    end
end
return bestScore,bestLocations
```

### 0.3.1 Description of test cases used in development

It was very important to check the correctness of the algorithm using small number of test cases to verify it is generating the optimal store locations as it is very hard to determine the optimal solution for any large size problem by hand. The following are some of the test problems used and the result obtained by the implementation, which shows that the optimal locations were found for each case.

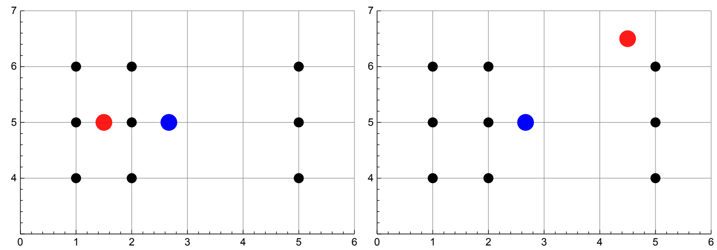
**test case 1** 

By direct observations, since we have one store only, then we see that by locating it in the center of the population, the score will be 6, which is optimal. The optimal store location found by the program is $\{2.333, 2.222\}$



**test case 2** This test case shows that the optimal location of our store do not change as the competition store location is changed. Since the optimal location depends on the clustering found and not on the competition location. In the following, the same configuration is used, but one had the competition store is at $\{1.5, 5\}$ and the other at $\{4.5, 6.5\}$. We see by direct counting and observation that the optimal store location is at $\{2, 5\}$ regardless. The only difference is the number of customers we attract in each case, but not the optimal store location itself. These two plots show this, with the score we obtain given below each configuration.

Clearly when the competitor store is away from the density area, our score will increase. Since the competition also wants to increase its score, then it should also have to locate its store in the same location, which is the kmeans++ optimal location.




Many other test cases where run, using more store locations and they were verified manually that the program result agrees with the finding. It is not possible to verify manually that the result will remain optimal for large population and large number of stores, but these tests cases at least shows that the algorithm works as expected. Now we will show result of large tests cases and the program output generated.

## 0.4 Result applying the algorithm to the supplied input

The following table summarizes the result of running the store location algorithm on the 5 test cases provided.

Table 1: Summary of store location score of each test case

| test case | n | m | X (population) | CPU time (minutes) | $J^*$ | percentage |
|---|---|---|---|---|---|---|
| trial/earlier one | 4 | 5 | 500,000 | 1.42 | 258,732 | 51.75% |
| 1 | 9 | 9 | 500,000 | 5.49 | 371,543 | 74.32% |
| 2 | 10 | 10 | 1,000,000 | 3.38 | 637,413 | 63.74% |
| 3 | 5 | 5 | 130,000 | 1.16 | 69,093 | 53.15% |
| 4 | 10 | 10 | 1,000,000 | 14.17 | 683,899 | 68.39% |

For illustration, the following four plots show the locations of our stores (the green dots) for the above final four test cases with the location of the competitor stores (black dots) and the final partitions selected.

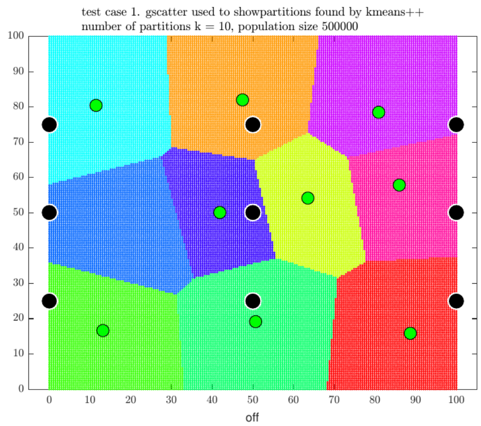



Figure 5: Test case 1

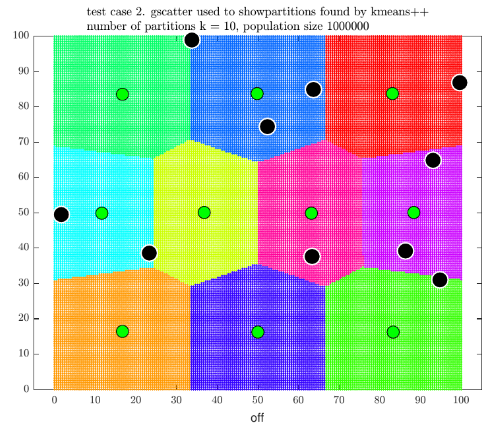



Figure 6: Test case 2

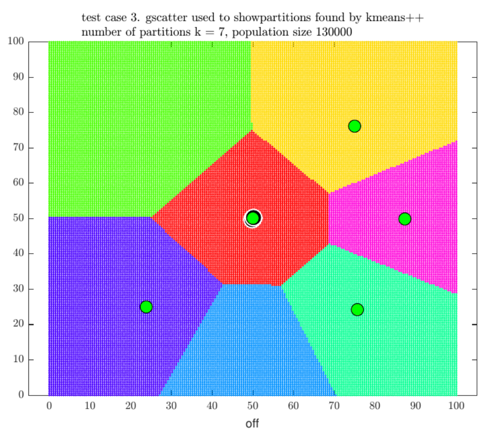



Figure 7: Test case 3

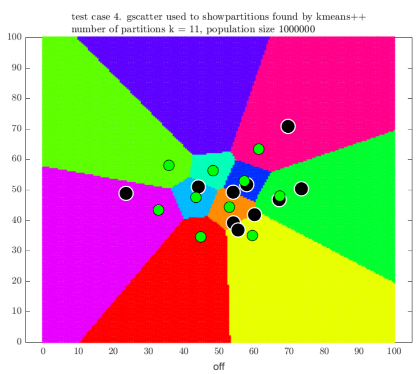



Figure 8: Test case 4

## 0.5 Conclusion

`kmeans++` algorithm for cluster analysis appears to be an effective method to use for finding an optimal store locations, but it is only practical for small $n$ as the algorithm used to obtain the partitions is NP-hard. In addition $\binom{p}{n}$ combinations of partitions needs to be searched to select the optimal set.

This implementation shows how `kmeans++` can be used to solve these types of problems. The location of the competitor stores has no influence on where to locate the stores, but it only affects the final possible score. Generating more partitions (using kmeans++) than the number of stores and selecting from them the best set can lead to improved score. It was found in the test cases used that no more than two of three additional partitions than the number of stores was needed to find the a combination of partitions which gave the maximum score. Generating additional partitions made the score go lower. The score used is the number of customers the stores attract out of the overall population. The algorithm was verified to be correct for small number of tests cases (not shown here due to space limitation). More research is needed to investigate how feasible this method can be for solving similar resource allocations problems.

## 0.6 References


**1** Matlab cluster analysis toolbox. Mathworks, Natick, MA

**2** https://en.wikipedia.org/wiki/K-means_clustering

**3** Seber, G.A.F. (1984) Multivariate Observations, Wiley, New York.


## 0.7 Appendix

```
function abbasi()
%Special problem. ECE 719, spring 2016
%by Nasser M. Abbasi
%Matlab 2016a
%

clear;      %start with clear env. just in case.
close all;  %is it ok to close all windows?
commandwindow;  %bring command window into focus
cd(fileparts(mfilename('fullpath')));

if license('test','Statistics_Toolbox') ~= 1
   error(['Warning, the needed toolbox does not', ...
       'seem to exist in your Matlab. This program needs',...
       'the Statistics and Machine Learning Toolbox as',...
       'it called kmeans++ cluster analysis\n',...
       'Please use the ver command to check you the toolbox\n']);
end

%this window will close when we are done. Ok to do.
fig = figure('Position',[370 400 400 30],...
  'Name','Optimal store locator. ECE 719. UW-Madison',...
  'NumberTitle','off');
set(fig, 'MenuBar', 'none');
set(fig, 'ToolBar', 'none');
h = uicontrol('Style','text','Position',[4 7 396 15],...
          'BackgroundColor','w',...
          'HorizontalAlignment','left');
drawnow;

DEBUG=true; %set to true to see plots
%change to false before code lockdown as plots slows down time.

```

```
OUT(h,'Starting store location program version 1.0.....');
OUT(h,'Checking for mat files.....');

if ~exist('n.mat','file')
    error(['file n.mat does not exist in current folder.',...
           'Please check for lower/upper case and location']);
end
if ~exist('X.mat','file')
    error(['file X.mat does not exist in current folder.',...
          'Please check for lower/upper case and location']);
end
if ~exist('Y.mat','file')
    error(['file n.mat does not exist in current folder.',...
           'Please check for lower/upper case and location']);
end


cd('../official_data/4/');
load('n');
load('X');
load('Y');
OUT(h,'mat files read ok.....');
cd(fileparts(mfilename('fullpath')));

rng(1); %for reproducability
KEEP_TRYING                = true;  %tells when to stop search
best_score_found_so_far    = 0;
best_locations             = [];
current_number_of_cluster = n;
tstart                     = tic;  %to keep track of CPU time
test_case                  = 4;
MAX_CPU                    = 15; %minutes CPU time limit.
status                     = true;
while KEEP_TRYING
    OUT(h,sprintf(['Best score so far: [%d]. calling kmeans++',...
                  'to make %d partitions...please wait...'],...
      round(best_score_found_so_far),current_number_of_cluster));
    [idx,C]  = make_cluster(X,current_number_of_cluster,...
                                                'sqeuclidean' );
    active_C = combnk(1:size(C,1),n);
    OUT(h,sprintf('created active_C, size is [%d,%d]....',...
            size(active_C,1),size(active_C,2)));

    [status,score,locations]=score_cluster(C,X,Y,active_C,...
                                               tstart,h,MAX_CPU);
```

```

    if ~status
        OUT(h,'Allowed CPU time exceeded, stopping the program');
        KEEP_TRYING = false;
    else
        if score>=best_score_found_so_far
            best_score_found_so_far=score;
            best_locations=locations;
            current_number_of_cluster=current_number_of_cluster+1;
            %stop search if size too large, or if number of
            %partitions too large this is due to using k choose m.
            %For k>15 it will need too much RAM.
            if current_number_of_cluster>=size(X,1)...
                                 ||current_number_of_cluster>=15
                KEEP_TRYING = false;
            end
            OUT(h,sprintf('current score %6.2f',...
                                           best_score_found_so_far));
            if DEBUG
               plot_result(test_case,best_locations,X,Y,...
                                          best_score_found_so_far,C);
            end
        else
          OUT(h,sprintf(...
             'Score is %6.2f. Less than last. Terminating..',...
              score));
          KEEP_TRYING = false;
        end
        telapsed = toc(tstart);
        if telapsed>MAX_CPU*60  % CPU limit
            OUT(h,'CPU time exceeded');
            KEEP_TRYING = false;
            status      = false;
        else
            OUT(h,sprintf(...
              'CPU time used to far %6.2f minutes',telapsed/60));
        end
    end
end

%final result
fprintf('n=%d, X=%d, Y=%d\n',n,size(X,1),size(Y,1));

fprintf('J*=[%6.2f] = [%%%4.2f]\n\n',...
  best_score_found_so_far,best_score_found_so_far/size(X,1)*100);
```

```
fprintf('optimal store coordinates\n');
fprintf(' x\t\t y\n');

for i=1:size(best_locations,1)
 fprintf('%3.3f\t%3.3f\n',best_locations(i,1),best_locations(i,2));
end

telapsed = toc(tstart);

if ~status
  fprintf('\nCPU limit reached. Elapsed time is %6.2f minutes\n',...
                                                    telapsed/60);
else
   fprintf('\nElapsed time is %6.2f minutes\n',telapsed/60);
end
if ishandle(fig)
    close(fig);
end
end
%============================
function d = distance_between_2_points(pt1,pt2)
%find distance between 2 points, assuming one can only
%move N-S or E-W, not diagonal.
x1 = pt1(1,1);
y1 = pt1(1,2);
x2 = pt2(1,1);
y2 = pt2(1,2);

d = abs(x1-x2) + abs(y1-y2);
end
%=============================================
function best_score_in_cluster = ...
                           find_my_score_in_each_cluster(C,X,Y)
%Takes center of each cluster (C) and customers locations (X)
%and competition store locations (Y) and returns how many
%customers I win in each cluster. Returns an array of number
%of customers we attract from competition in each cluster.

%to store score per cluster
best_score_in_cluster = zeros(size(C,1),1);

for i=1:size(X,1)  %loop of all population to see which we win
    %z1 is competitor, z2 is our store
    [~,z1]    = shortest_distance_to_stores(X(i,:),Y);
    [idx,z2] = shortest_distance_to_stores(X(i,:),C);
```

```
    if z2<=z1  %compare with competition to see if we are closer
     if z1==z2
       %oh well, split this customer between us and them
       best_score_in_cluster(idx)=best_score_in_cluster(idx)+0.5;
     else
       %good, we are closer, take this customer.
       best_score_in_cluster(idx)=best_score_in_cluster(idx)+1;
     end
    end
end
end

%================================
function [idx,d] =shortest_distance_to_stores(pt,stores_locations)
%find shortest distance from one customer to a set of stores.
%The stores can be ours or the competition. Returns the shortest
%distance in 'd' and the index of the store who is closest to
%this customer

d = inf;
for i=1:size(stores_locations,1)
    current_distance = distance_between_2_points(pt,...
                                         stores_locations(i,:));
    if current_distance <= d
        d   = current_distance;
        idx = i;
    end
end

end
%===========================================
function [status,best_score,locations]=score_cluster(...
                               C,X,Y,active_C,tstart,h,MAX_CPU)

best_score   = 0;
status       = true;
KEEP_TRYING  = true;

while KEEP_TRYING
  for i=1:size(active_C,1)
    OUT(h,sprintf(['scoring partition %d of %d in score_cluster() ',...
                       'Current best score %d'],...
                          i,size(active_C,1),round(best_score)));

        score = find_my_score_in_each_cluster(...
```

```
                                           C(active_C(i,:),:),X,Y);
        score=sum(score);
        if score>best_score
            best_score = score;
            locations  = C(active_C(i,:),:);
        end
        telapsed = toc(tstart);
        if telapsed>MAX_CPU*60
          OUT(h,sprintf('Exceeded CPU time limit in score_cluster'));
          KEEP_TRYING = false;
          status      = false;
        end
    end
    KEEP_TRYING = false;
end
end
%====================================
function [idx,C] = make_cluster(population,how_many,the_option)
%cluster the population. Number of cluster is same as
%number of our stores.  This was found to be optimal by many
%trials and errors. If we use more clusters than number of
%stores, the score actually goes down.
warning('off','all');
[idx,C] = kmeans(population,how_many,'Replicates',5,...
    'MaxIter',50,'Distance',the_option);
warning('on','all');
end

%======================================================
function plot_result(test_case,store_locations,X,Y,...
                                            overall_best_score,C)

%figure;
tmp = hist3(X, {0:100 0:100});
n1  = tmp';
n1(size(tmp,1), size(tmp,2)) = 0;
xb = linspace(0,100,101);
yb = xb;

figure;
pcolor(xb,yb,n1);
hold on;
plot(Y(:,1),Y(:,2),'o','MarkerSize',9,...
    'MarkerFaceColor','black',...
    'LineWidth',1,'MarkerEdgeColor','white');
```

```
plot(store_locations(:,1),store_locations(:,2),...
    'o','MarkerSize',9,'MarkerFaceColor','green',...
    'MarkerEdgeColor','black');
title( {sprintf(...
         ['Test case $%d$. Showing our store location with',...
          'competitors on density plot. score =%5.1f'],...
    test_case,overall_best_score),...
    sprintf('number of partitions k = $%d$, population size $%d$',...
    size(store_locations,1),size(X,1))},...
        'Fontsize',11,'interpreter','Latex');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
drawnow;
saveas(gcf, sprintf('../images/1_test_case_%d',test_case), 'pdf');
saveas(gcf, sprintf('../images/1_test_case_%d',test_case), 'png');

figure;
[x1G,x2G] = meshgrid(linspace(0,100,200),linspace(0,100,200));
XGrid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot
warning('off','all');
idx2Region = kmeans(XGrid,size(C,1),'MaxIter',1,'Start',C);
warning('on','all');
cmap = hsv(size(C,1));
gscatter(XGrid(:,1),XGrid(:,2),idx2Region,cmap,[],[],...
                                                 'doLeg','off');
hold on;
plot(Y(:,1),Y(:,2),'o','MarkerSize',12,...
               'MarkerFaceColor','black',...
               'LineWidth',1,'MarkerEdgeColor','white');
plot(store_locations(:,1),store_locations(:,2),...
    'o','MarkerSize',9,'MarkerFaceColor','green',...
    'MarkerEdgeColor','black');
title( {sprintf(['test case $%d$. gscatter used to show',...
                 'partitions found by kmeans++'],test_case),...
   sprintf('number of partitions k = $%d$, population size $%d$',...
   size(C,1),size(X,1))},'Fontsize',11,'interpreter','Latex');
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
drawnow;
saveas(gcf, sprintf('../images/2_test_case_%d',test_case), 'pdf');
saveas(gcf, sprintf('../images/2_test_case_%d',test_case), 'png');
end
%================================================
function OUT(h,the_string)
fprintf(the_string);
fprintf('\n');

```

```
%if ishandle(h)
%   set(h,'String',the_string);
%   drawnow;
%end
end
```

```
function nma_generate_output(test_case)
%Program to generate output to test special problem with
%Nasser M. Abbasi
%ECE 719, UW Madison

cd(fileparts(mfilename('fullpath')));

switch test_case
    case 1
        X=[1,2;
            1,3;
            2,2;
            2,3;
            3,0;
            3,1;
            3,2;
            3,3;
            3,4];
        Y=[1.5,2.5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

  case 105
        X=[1,2;
            1,3;
            2,2;
            2,3;
            3,0;
            3,1;
            3,2;
            3,3;
            3,4];
        Y=[1.5,2.5];
        n=2;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');
```

```

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2.5,2.5],X,Y))
    case 2
        X=[1,4;
            1,5;
            1,6;
            2,4;
            2,5;
            2,6;
            5,6;
            5,5;
            5,4];
        Y=[4,6.5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 3
        X=[1,4;
            1,5;
            1,6;
            2,4;
            2,5;
            2,6;
            5,6;
            5,5;
            5,4];
        Y=[1.5,5];
        n=1;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%        fprintf('best score is %3.3f\n',...
%            find_my_score_in_each_cluster([2,5],X,Y))

    case 4
        X=[1,1;
            1,2;
            2,1;
```

```
                2,2;
                4,3;
                4,4;
                5,3;
                5,4];
            Y=[3,2.5];
            n=1;
            save('n','n');
            save('X','X');
            save('Y','Y');
            save('test_case','test_case');

%             fprintf('best score is %3.3f\n',...
%                 find_my_score_in_each_cluster([2,5],X,Y))

        case 5
            X=[1,1;
                1,2;
                1,3;
                2,1;
                2,2;
                2,3;
                3,2;
                4,2;
                4,3;
                4,4;
                4,5;
                5,1;
                5,2;
                5,3;
                5,4;
                5,5;
                6,3;
                6,4;
                6,5];
            Y=[5,4];
            n=1;
            save('n','n');
            save('X','X');
            save('Y','Y');
            save('test_case','test_case');

%             fprintf('best score is %3.3f\n',...
%                 find_my_score_in_each_cluster([2,5],X,Y))

        case 6
            X=[ 1,1;
```

```
            2,1;
            3,1;
            1,2;
            2,2;
            3,2;
            1,3;
            2,3;
            3,3];
        Y=[2,2];
        n=2;
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

%         fprintf('best score is %3.3f\n',...
%             find_my_score_in_each_cluster([2,5],X,Y))

    case 7
        rng default; % For reproducibility
        N=10000;
        X=[ 30 + 2*randn(N,1),30 + 8*randn(N,1);
            40 + 2*randn(N,1),40 + 10*randn(N,1);
            25 + 2*randn(N,1), 50 + 4*randn(N,1);
            20 + 2*randn(N,1),30 + 4*randn(N,1);
            50 + 2*randn(N,1),50 + 4*randn(N,1)];

        n=9;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');


    case 8
        rng default; % For reproducibility
        N=1000;
        X=[30+randn(N,1),30+randn(N,1);
            40+randn(N,1),40+randn(N,1);
            25+randn(N,1),50+randn(N,1);
            20+randn(N,1),30+randn(N,1);
            50+randn(N,1),50+randn(N,1)];

        n=4;  %this gives 50%, since competition is allready optimal
```

```
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

    case 9
        rng default; % For reproducibility
        N=1000;
        X=[30+randn(N,1),30+randn(N,1);
            40+randn(N,1),40+randn(N,1);
            25+randn(N,1),50+randn(N,1);
            20+randn(N,1),30+randn(N,1);
            50+randn(N,1),50+randn(N,1)];

        n=7;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');

case 10
        rng default; % For reproducibility
        N=100000;
        X=[30+randn(N,1),30+randn(N,1);
            40+randn(N,1),40+randn(N,1);
            25+randn(N,1),50+randn(N,1);
            20+randn(N,1),30+randn(N,1);
            50+randn(N,1),50+randn(N,1)];

        n=10;  %this gives 50%, since competition is allready optimal
        Y=[41.6552   35.4282;
            24.5046    33.8534;
            30.5928   30.0431;
            40.5928   30.0431;
            70.5928   30.0431];
        save('n','n');
        save('X','X');
        save('Y','Y');
        save('test_case','test_case');
```

```
end
end
```