**PMCLUSTER:**

**A Collection of MATLAB Programs for *p*-median Clustering**

**MICHAEL J. BRUSCO**

**Florida State University**

**HANS-FRIEDRICH KÖHN AND DOUGLAS STEINLEY**

**University of Missouri-Columbia**

**April 2009**

**TABLE OF CONTENTS**

**PURPOSE Page 3**

**WHY CHOOSE P-MEDIAN CLUSTERING? Page 4**

**OVERVIEW OF DATA STRUCTURES Page 5**

**PART I. PROGRAMS FOR TRADITIONAL CLUSTERING Page 6**

**PART II. PROGRAMS FOR OTHER DATA STRUCTURES Page 9**

**REFERENCES Page 11**

**APPENDICES – MATLAB m-files Page 13**

**PURPOSE**

The goal of our PMCLUSTER project is to provide a collection of freely-available MATLAB m-files that research analysts can use to conduct cluster analyses based on the *p*-median model. Although our principal target audience is researchers in the social sciences (especially psychology and business), the program are apt to be useful for other scientific disciplines as well. The programs in this suite are based on our recent research efforts focusing on the *p*-median problem. The relevant citations to our work include:

Brusco, M.J., & Köhn, H.-F. (2008a). Optimal partitioning of a data set based on the *p*-median model. *Psychometrika*, 73, 89-104.

Brusco, M.J., & Köhn, H.-F. (2008b). Comment on ‘Clustering by passing messages between data points.’ *Science*, 319, 726c.

Brusco, M.J., & Köhn, H.-F. (2009). Exemplar-based clustering via simulated annealing. *Psychometrika*, DOI: 10.1007/s11336-009.9115-2.

Köhn, H.-F., Steinley, D., & Brusco, M. J. (2009). The *p*-median model as a tool for clustering psychological data. Manuscript submitted for publication.

**WHY CHOOSE P-MEDIAN CLUSTERING?**

Perhaps the most well known methods for partitioning a data set fall under the umbrella of *K*-means cluster analysis (Hartigan & Wong, 1979; MacQueen, 1967; Steinhaus, 1956; Thorndike, 1953). Generally speaking, the goal in *K*-means clustering is to find a partition that minimizes the within-cluster sum of squared deviations between objects and their centroids (see Steinley, 2006 for an extensive review of *K*-means clustering and Brusco & Steinley, 2007 for a comparison of heuristic methods for this problem).

The *p*-median clustering approach is an alternative to *K*-means clustering and differs in the following respects:

1) The centroids of the *p*-median algorithm are real objects rather than averages, and these objects serve as **“exemplars”** for their clusters.

2) The *p*-median model is purported as being generally robust to outliers (Kaufman & Rousseeuw, 1990).

3) Whereas *K*-means is generally applied to dissimilarity data under the assumption of squared Euclidean distances, the *p*-median model can be applied for any distance measure, and can even be used in cases where there are violations of the triangle inequality and/or symmetry.

Excellent coverage of heuristic methods for the *p*-median problem is provided by Mladenovic et al. (2007). Kaufman and Rousseeuw (1990, 2005) provide an in-depth treatment of *p*-median (or, partitioning around medoids) methods for clustering (see also Alba & Dominguez, 2006; Klastorin, 1985; Mulvey & Crowder, 1979).

Although tremendous progress has been made in the development of exact solution methods for the *p*-median problem (Avella et al., 2007; Beltran et al., 2006), we restrict our attention to heuristic methods.

**DATA STRUCTURES**

We offer multiple versions of our programs so that psychological scientists and other researchers can select those that are best-suited for their applications. All programs in this MATLAB library use algorithms that operate on **nonnegative dissimilarity matrices with a main diagonal of zeros**. Although these options *can* be relaxed, doing so can create some rather unusual solutions and is, therefore, not recommended. Dissimilarity data are characterized by smaller values indicating greater similarity and larger values suggesting less similarity. The most common examples are Euclidean or squared-Euclidean distances between objects.

Different versions of our programs make different assumptions about the input format of the data. In **Part I of our program suite**, the programs read data in the traditional *n* × *d* data matrix format. The data matrix, **X**, consists of measurements for *n* objects on each of *d* variables. The analyst can choose between Euclidean or squared-Euclidean distances to conduct the analysis.

In **Part II of our program suite**, the programs directly read in an *n* × *n* dissimilarity matrix of non-negative elements. This allows application of the *p*-median model to more flexible data structures, including dissimilarities that violate the triangle inequality or are asymmetric. Some programs in this suite also allow users to specify preference weights for the objects to serve as exemplars (see, for example, Frey & Dueck, 2007).

**PART I. PROGRAMS FOR TRADITIONAL CLUSTERING**

**MATLAB m-files**

**1) MFI.m** – this program runs a “multistart fast interchange” heuristic. This procedure is based on the vertex substitution heuristic described by Teitz and Bart (1968), but uses the fast interchange rules described by Hansen and Mladenovic (1997), which are based on Whitaker’s (1983) work. The program that reads an *n* × *d* matrix (X) of measurements for *n* objects each measured on *d* metric variables, produces a dissimilarity matrix, and clusters the data. The calling function is:

>> [exemplars, Z, P, silhouette\_index] = MFI(X, K, M, restarts);

The inputs are the *n* × *d* matrix (X), the desired number of clusters (K), the desired distance measurement (M), and the desired number of restarts (restarts – unless n or K are large, we recommend 20-50 restarts). A choice of M = 1 will result in a dissimilarity matrix based on squared Euclidean distance, whereas any other choice results in Euclidean distance.

The outputs are a *K* × 1 vector of cluster centers (exemplars), the best-found objective function value (Z), an *n* × 1 vector of cluster assignments (P), and Kaufman and Rousseeuw’s (1990) silhouette index.

**2) SA.m** – this program runs a “simulated annealing” heuristic. This procedure is based on the algorithm described by Brusco and Köhn (2009). The program that reads an *n* × *d* matrix (X) of measurements for *n* objects each measured on *d* metric variables, produces a dissimilarity matrix, and clusters the data. The calling function is:

>> [exemplars, Z, P, silhouette\_index] = SA(X, K, M, cool);

The inputs are the *n* × *d* matrix (X), the desired number of clusters (K), the desired distance measurement (M), and the cooling parameter for the simulated annealing algorithm (cool). A choice of M = 1 will result in a dissimilarity matrix based on squared Euclidean distance, whereas any other choice results in Euclidean distance. We recommend cool = .9, or possibly .8 if the number of objects exceeds 2000.

The outputs are a *K* × 1 vector of cluster centers (exemplars), the best-found objective function value (Z), an *n* × 1 vector of cluster assignments (P), and Kaufman and Rousseeuw’s (1990) silhouette index.

**3) LRA.m** – this program runs a “Lagrangian relaxation algorithm” heuristic for the p-median problem. This procedure is based on the work of Cornuejols, Fisher, and Nemhauser (1977), Mulvey and Crowder (1979), and Hanjoul and Peeters (1985), too name a few. The specific implementation is the one described by Brusco and Köhn (2008a). The program that reads an *n* × *d* matrix (X) of measurements for *n* objects each measured on *d* metric variables, produces a dissimilarity matrix, and runs the algorithm. The calling function is:

>> [exemplars, ZLB, E, terminate] = LRA(X, M, K, ZUB);

The inputs are the *n* × *d* matrix (X), the desired number of clusters (K), the desired distance measurement (M), and an upper bound on the objective function (ZUB). A choice of M = 1 will result in a dissimilarity matrix based on squared Euclidean distance, whereas any other choice results in Euclidean distance. We recommend running MFI.m or SA.m first and using the resulting objective value from these procedures as ZUB.

The outputs are a *K* × 1 vector of cluster centers (exemplars), the best-found lower-bound on the objective function value (ZLB), the percentage deviation of ZUB above ZLB (E), and a flag (terminate) that assumes a value of 1 for optimality and 0 otherwise.

Recommendations and Cautions when using LRA.m

* LRA.m works best when using a tight upper bound. We recommend running MFI.m or SA.m first and using the resulting objective value from these procedures as ZUB.
* The LRA.m can be used to ***confirm*** that a heuristic solution is, in fact, optimal. This confirmation occurs when an “ε-optimality” condition such that ZUB – ZLB < ε. Our default value of ε = .00001. Depending on the precision of the data, optimality might be guaranteed. For example, suppose that all data in the dissimilarity matrix is measured to a precision of two decimal places, and ZUB = 57.54 and ZLB = 57.539991, then the heuristic solution corresponding to ZUB must be optimal. On the other hand, if the precision of the proximity data is to 8 decimal places and = 57.55399967 and ZLB = 57.539991, then we only know that the solution corresponding to ZUB is “ε-optimal”. Caution – when the algorithm terminates based on the “ε-optimality” condition, the ‘exemplars’ returned are not necessarily optimal.
* The LRA.m can also ***produce*** an optimal solution in its own right, even if the input value of ZUB is suboptimal. This occurs when the algorithm finds a solution that is “ε-feasible”. That is, if the sum of the squared constraint feasibility values are less than ε, then the algorithm concludes that the optimal solution has been found.
* If the algorithm either confirms or provides an optimal solution, then terminate = 1 and E = 0. Otherwise, terminate = 0 and E is some positive value.

**Scalability**

Both the MFI.m and SA.m algorithms have been evaluated on test problems with thousands of objects and hundreds of clusters. For example, Brusco & Köhn (2009) recently showed that the simulated annealing approach was competitive with some of the best-known heuristics for *p*-median clustering (Hansen et al., 2001; Resende & Werneck, 2004; Taillard, 2003) for problems with up to *n* = 5,900 objects and *K* = 900 clusters.

The maximum number of objects that can be handled by our procedures is a function of available computer memory. For example, we are able to handle problems with just over 6,000 objects on a microcomputer with 3GB of SDRAM. Although the choice of *K* does not affect computer memory limits, it can have a pronounced effect on computation time.

**Choosing Between MFI.m and SA.m**

For most applications in the psychological sciences, 50 restarts of MFI.m will produce the same partition as SA.m when used with a cooling parameter value of .8 or greater. In other words, both methods tend to be extremely effective for psychological applications because applications in psychology typically require 5 or fewer clusters, and certainly no more than 10 clusters.

The big performance differences between MFI.m and SA.m occur when the number of clusters increases far beyond 10, particularly as *K* increases beyond 50. However, we are unaware of any psychological applications where analysts require 50 or more clusters.

The MFI.m program can run multiple restarts, whereas the SA.m program is set for a single restart because of its much greater computation time. However, if an analysts wants to run multiple restarts of the SA.m heuristic, then this can be accomplished simply by changing the ‘state’ parameter in the code to a different number for each restart.

**Choosing the Number of Clusters**

Choosing the number of clusters remains one of the most vexing problems in applied cluster analysis. One approach is to consider the percentage decreases in the objective function as K is increased and look for a large gap (e.g., if the objective function decreases sharply when going from 3 to 4 clusters, but only slightly when going from 4 to 5, then choose 4 clusters). As an alternative approach, MFI.m and SA.m report the silhouette index (Kaufman & Rousseeuw, 1990), which can facilitate the choice of *K*. An analyst can select the value of *K* that maximizes the index, or consider the change in the index as *K* is increased.

**PART II. PROGRAMS FOR OTHER DATA STRUCTURES**

**MATLAB m-files**

**4) MFI\_FC.m** – this program runs a “multistart fast interchange” heuristic for p-median problems with a “fixed charge” (FC) added for selected exemplars . This procedure is based on the vertex substitution heuristic described by Teitz and Bart (1968), but uses the fast interchange rules described by Hansen and Mladenovic (1997), which are based on Whitaker’s (1983) work. The program is an adaptation of Brusco and Köhn’s (2008b) method for a dissimilarity context and reads an *n* × *n* dissimilarity matrix (S) and clusters the data. The calling function is:

>> [exemplars, Z, P, silhouette\_index] = MFI\_FC(S, K, R, restarts);

The inputs are the *n* × *n* dissimilarity matrix (S), the desired number of clusters (K), an *n* × 1 vector of fixed charge “costs” representing preferences for objects serving as exemplars (R), and the desired number of restarts (restarts – unless *n* or *K* are large, we recommend 20-50 restarts). If all objects have equal preference for serving as an exemplar, then R is a vector of zeros.

The outputs are a *K* × 1 vector of cluster centers (exemplars), the best-found objective function value (Z), an *n* × 1 vector of cluster assignments (P), and Kaufman and Rousseeuw’s (1990) silhouette index.

**5) SA\_FC.m** – this program runs a “simulated annealing” heuristic. This procedure is based on the algorithm described by Brusco and Köhn (2009). The program that reads an *n* × *n* dissimilarity matrix (S) and clusters the data. The calling function is:

>> [exemplars, Z, P, silhouette\_index] = SA\_FC(S, K, R, cool);

The inputs are the *n* × *n* dissimilarity matrix (S), the desired number of clusters (K), an *n* × 1 vector of fixed charge “costs” representing preferences for objects serving as exemplars (R) and the cooling parameter for the simulated annealing algorithm (cool). We recommend cool = .9, or possibly .8 if the number of objects exceeds 2000.

The outputs are a *K* × 1 vector of cluster centers (exemplars), the best-found objective function value (Z), an *n* × 1 vector of cluster assignments (P), and Kaufman and Rousseeuw’s (1990) silhouette index.

**6) LRA\_GEN.m** – this program runs a “Lagrangian relaxation algorithm” heuristic for the *p*-median problem. This procedure is based on the work of Cornuejols, Fisher, and Nemhauser (1977), Mulvey and Crowder (1979), and Hanjoul and Peeters (1985), too name a few. The specific implementation is the one described by Brusco and Köhn (2008a), which clusters input data provided in the form of an *n* × *n* dissimilarity matrix (S). The calling function is:

>> [exemplars, ZLB, E, terminate] = LRA\_GEN(S, K, ZUB);

The inputs are the *n* × *n* dissimilarity matrix (S), the desired number of clusters (K), and an upper bound on the objective function (ZUB). We recommend running MFI\_FC.m or SA\_FC.m first and using the resulting objective value from these procedures as ZUB.

The outputs are a *K* × 1 vector of cluster centers (exemplars), the best-found lower-bound on the objective function value (ZLB), the percentage deviation of ZUB above ZLB (E), and a flag (terminate) that assumes a value of 1 for optimality and 0 otherwise.

**When to Use MFI\_FC.m or SA\_FC.m instead of MFI.m or SA.m**

If an *n* × *n* nonnegative dissimilarity matrix already exists or has been formed in a previous step, then MFI\_FC.m and SA\_FC.m are the algorithms of choice. The dissimilarity data could be formed based on Euclidean, squared-Euclidean, city-block, or some other type of distance measure. **Moreover, the nonnegative dissimilarity data could even have triangle inequalities or asymmetries (e.g., air travel distances)**. Accordingly, the MFI\_FC.m and SA\_FC.m algorithms have tremendous flexibility in the data structures that they can accommodate – without any loss of scalability or computational efficiency.

The MFI\_FC.m and SA\_FC.m algorithms have roughly the same scalability and computational properties as MFI.m and SA.m, respectively. The same metrics for choosing the number of clusters also apply. Moreover, in the case of nonzero preferences, the objective function does not necessarily monotonically decrease as *K* increases, and thus the value of *K* that results in the minimum objective value could also be chosen (see Frey & Dueck, 2007).

**When to Use LRA\_GEN.m instead of LRA.m**

If an *n* × *n* dissimilarity matrix already exists or has been formed in a previous step, then MFI\_FC.m and SA\_FC.m are the algorithms of choice. The dissimilarity data could be formed based on Euclidean, squared-Euclidean, city-block, or some other type of distance measure. **Moreover, we have also had success with the model when using it for dissimilarity data that have triangle inequalities and asymmetries; however, we have limited experience with such problems – so caution is advised)**.

Note: LRA\_GEN cannot handle the “fixed charge” case of differential preferences.

**REFERENCES**

Alba, E., & Dominguez, E. (2006). Comparative analysis of modern optimization tools for the *p*-median problem. *Statistics and Computing*, *16*, 251-260.

Avella, P., Sassano, A., & Vasil’ev, I. (2007). Computational study of large-scale *p*-median problems. *Mathematical Programming A*, 109, 89-114.

Beltran, C., Tadonki, C., & Vial, J. (2006). Solving the *p*-median problem with a semi-Lagrangian relaxation. *Computational Optimization and Applications*, June 5, 2006, DOI: 10.1007/s 10589-006-6513-6.

Brusco, M. J., & Köhn, H.-F. (2008a). Optimal partitioning of a data set based on the *p*-median model. *Psychometrika*, *73*, 89-105.

Brusco, M. J., & Köhn, H.-F. (2008b). Comment on ‘Clustering by passing messages between data points.’ *Science*, *319*, 726.

Brusco, M. J., & Köhn, H.-F. (2009). Exemplar-based clustering via simulated annealing. *Psychometrika*, (in press, DOI: 10.1007/s11336-009.9115-2).

Brusco, M. J., & Steinley, D. (2007). A comparison of heuristic procedures for minimum within-cluster sums of squares partitioning. *Psychometrika*, *72*, 583-600.

Cornuejols, G., Fisher, M. L., & Nemhauser, G. L. (1977). Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. *Management Science*, *23*, 789-810.

Frey, B., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*, 972-976.

Hanjoul, P., & Peeters, D. (1985). A comparison of two dual-based procedures for solving the p-median problem. *European Journal of Operational Research*, *20*, 387-396.

Hansen, P., & Mladenoviĉ, N. (1997). Variable neighborhood search for the *p*-median. *Location Science*, *5*, 207-226.

Hansen, P., Mladenoviĉ, N., & Perez-Brito, D. (2001). Variable neighborhood decomposition search. *Journal of Heuristics*, *7*, 335-350.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS136: A *k*-means clustering program. *Applied Statistics*, *28*, 100-128.

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis*. New York: Wiley.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671-680.

Klastorin, T. (1985). The *p*-median problem for cluster analysis: a comparative test using the mixture model approach. *Management Science*, *31*, 84-95.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281-297), Berkeley, CA: University of California Press.

Mladenoviĉ, N., Brimberg, J., Hansen, P., & Moreno-Pérez, J. A. (2007). The *p*-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*, *179*, 927-939.

Mulvey, J. M., & Crowder, H. P. (1979). Cluster analysis: an application of Lagrangian relaxation. *Management Science*, *25*, 329-340.

Resende, M. G. C., & Werneck, R. F. (2004). A hybrid heuristic for the *p*-median problem. *Journal of Heuristics*, *10*, 59-88.

Steinhaus, H. (1956). Sur la division des corps matétiels en parties. *Bulletin de l’Académie Polonaise des Sciences*, Classe III, vol. IV, no. 12, 801-804.

Steinley, D. (2006). K-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, *59*, 1-34.

Teitz, M. B., & Bart, P. (1968). Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, *16*, 955-961.

Thorndike. R. L. (1953). Who belongs in the family? *Psychometrika*, *18*, 267-276.

Whitaker, R. (1983). A fast algorithm for the greedy interchange of large-scale clustering and median location problems. *INFOR*, *21*, 95-108.

**APPENDICES – MATLAB m-files**

**1) MFI.m**

function [exemplars, Z, P, silohouette\_index] = MFI(X, M, K, restarts);

% MFI.m: A "multistart fast interchange" procedure for the

% p-median problem. The procedure is the Teitz and

% Bart (1968) VERTEX SUBSTITUTION HEURISTIC using the

% using fast updates from Hansen and Mladenovic (1997),

% which are based on Whitaker's (1983) work.

% INPUTS: X = an n x d matrix of measurements for n objects on

% each of d variables

% M = 0 for squared Euc. distance, otherwise Euclidean

% K = the desired number of medians/clusters/exemplars

% restarts = the desired number of restarts

% OUTPUTS: exemplars = the best found set of medians/exemplars

% Z = the best-found objective function value

% silohoette\_index = index from Kaufman & Rousseeuw (1990)

% P an n by 1 vector of cluster assignments

% NOTE: If M = 0, then the DISSIMILARITY matrix is formed based on

% Squared Euclidean distances; otherwise, the matrix is formed

% based on Euclidean distances.

state = 1; % fix state for testing

rand('state', state);

tic;

[n,d] = size(X);

S = zeros(n,n);

for i = 1:n-1

for j = i+1:n

dist = 0;

for k = 1:d

dist = dist + (X(i,k)-X(j,k)).^2;

end

if M ~= 0

S(i,j) = sqrt(dist);

else

S(i,j) = dist;

end

S(j,i) = S(i,j);

end

end

nreps = restarts; gbest = 9.9e+12; fstore = zeros(nreps,1);

c1 = zeros(n,1); c2 = zeros(n,1); p = K;

for klk = 1:nreps

s = randperm(n); % Randomly order the exemplar

a1 = S(:,s(1:p)); % candidates and let the first

fbest = 0; % p be the selected subset

for i = 1:n; % Note: In this program

dmin = 9.9e+12; % i indexes the rows, which

for j = 1:p % assigned to exemplars or

if a1(i,j) < dmin % selected columns indexed by j

dmin = a1(i,j); jsel = j;

end

end

fbest = fbest + dmin; c1(i) = s(jsel); % fbest = initial objective;

end % c1(i) = exemplar to which

for i = 1:n % object i is most similar

dmin = 9.9e+12; % c2(i) = exemplar to which

for j = 1:p % object is 2nd most similar

jj = s(j); % is computed in this block

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

trig = 0;

while trig == 0 % The exchange process begins here

wstar = 0;

% The block below finds, for each unseleted point (goin), the best exemplar

% to remove (goout) if adding goin. The best (goin, goout) pair is

% identified and (wstar) is the resulting change in the objective function.

% If wstar >= 0, then the algorithm terminates because there is no viable

% exchange that will improve the objective function further.

for goin = p+1:n

ii = s(goin); w = 0; u = zeros(n,1);

for i = 1:n

if S(i,ii) < S(i,c1(i))

w = w + S(i,ii) - S(i,c1(i));

else

u(c1(i)) = u(c1(i)) + min(S(i,ii),S(i,c2(i))) - S(i,c1(i));

end

end

dmin = 9.9e+12;

for j = 1:p

jj = s(j);

if u(jj) < dmin

dmin = u(jj); goout = j;

end

end

w = w + dmin;

if w < wstar

wstar = w; goinb = goin; gooutb = goout;

end

end

if wstar >= -.00001

trig = 1;

continue

end

% The block below updates the c1 and c2 vectors if an (goin, goout) swap

% results in an improvement.

fbest = fbest + wstar; goinc = s(goinb); gooutc = s(gooutb);

idum = s(goinb); s(goinb) = s(gooutb); s(gooutb) = idum;

for i = 1:n

if c1(i) == gooutc

if S(i,goinc) <= S(i,c2(i))

c1(i) = goinc;

else

c1(i) = c2(i);

dmin = 9.9e+12;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

else

if S(i,c1(i)) > S(i,goinc)

c2(i) = c1(i); c1(i) = goinc;

elseif S(i,goinc) < S(i,c2(i))

c2(i) = goinc;

elseif c2(i) == gooutc

dmin = 9.9e+12;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

end

end

end

if fbest < gbest

gbest = fbest; sbest = s; cbest = c1;

end

fstore(klk) = fbest;

end

% DOUBLE-CHECK THE BEST-FOUND SOLUTION

P = zeros(n,1); Z = 0; exemplars = sbest(1:p); % Store best solution

for i = 1:n

dmin = 9.9e+12;

for j = 1:p

g = exemplars(j);

dist = 0;

for k = 1:d

dist = dist + (X(i,k)-X(g,k)).^2;

end

if dist < dmin

P(i) = j; dmin = dist;

end

end

if M ~= 0

Z = Z + sqrt(dmin);

else

Z = Z + dmin;

end

end

if M == 0

u = silhouette(X,P);

else

u = silhouette(X,P,'Euclidean');

end

W = 1:n; % This block is important for singletons

for k = 1:K % MATLAB's default is that objects in

y = W(P==k); ylen = length(y); % singletons have silhoettes = 1,

if ylen == 1 % but Kaufman and Rousseeuw (1990, p. 85)

u(y) = 0; % assume silhouettes = 0.

end

end

silohouette\_index = mean(u);

toc

**2) SA.m**

function [exemplars, Z, P, silohouette\_index] = SA(X, M, K, cool);

%SA -- Simulated Annealing Heuristic for the P-median Problem

% We use the method published by Brusco & Koehn (2009),

% except that this program operates on DISSIMILARITY matrix.

% INPUTS: X = an n x d matrix of measurements for n objects on

% each of d variables

% M = 0 for squared Euc. distance, otherwise Euclidean

% K = the desired number of medians/clusters/exemplars

% cool = the simulated annealing parameter. We recommend

% 0.9, but 0.8 often suffices and is faster.

% OUTPUTS: exemplars = the best found set of medians/exemplars

% Z = the best-found objective function value

% silohoette\_index = index from Kaufman & Rousseeuw (1990)

% p an n by 1 vector of cluster assignments

% NOTE: If M = 0, then the DISSIMILARITY matrix is formed based on

% Squared Euclidean distances; otherwise, the matrix is formed

% based on Euclidean distances.

% NOTE: There is no 'tmin' in this program -- stops when no inferior

% solution is accepted

state = 1; % fix state for testing

rand('state', state);tic;

[n,d] = size(X); p = K;

a = zeros(n,n);

for i = 1:n-1

for j = i+1:n

dist = 0;

for k = 1:d

dist = dist + (X(i,k)-X(j,k)).^2;

end

if M ~= 0

a(i,j) = sqrt(dist);

else

a(i,j) = dist;

end

a(j,i) = a(i,j);

end

end

[m,n] = size(a);

t = randperm(n); % Initial random permutation

s = t(1:p); % s = the current set of exemplars

x = t(p+1:n); % u = unselected objects that are not exemplars

z = sum(max(a(:,s),[],2)); % z = the current objective function value

zbest = z;

sbest = s;

pbest = p; % The starting solution

tempmax = 0; % Find an initial temperature by sampling

for ijk = 1:200

i3 = int32(rand.\*(n-p)+1);

i = double(i3);

if i > n-p

i = n-p;

end

goin = x(i); % goin is the entering object

j3 = int32(rand\*p+1);

goout = double(j3);

if goout > p

goout = p; % goout is the exemplar position where goin enters

end

strial = s;

strial(goout) = goin;

ztrial = sum(max(a(:,strial),[],2));

if abs(z-ztrial) > tempmax

tempmax = abs(z-ztrial);

end

end

iloop = 10.\*n; % SA temperature length

temp = tempmax; % SA initial temperature

c1 = zeros(m,1); c2 = zeros(m,1);

a1 = a(:,s);

z = 0;

for i = 1:m; % Note: In this program

dmin = 9.9e+15; % i indexes the rows, which

for j = 1:p % assigned to exemplars or

if a1(i,j) < dmin % selected columns indexed by j

dmin = a1(i,j); jsel = j;

end

end

z = z + dmin; c1(i) = s(jsel); % z = initial objective;

end % c1(i) = exemplar to which

for i = 1:m % object i is most similar

dmin = 9.9e+15; % c2(i) = exemplar to which

for j = 1:p % object is 2nd most similar

jj = s(j); % is computed in this block

if c1(i) == jj

continue

end

if a(i,jj) < dmin

dmin = a(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

% THE SA ALGORITHM BEGINS ON THE NEXT LINE

terminate = 0;

while terminate == 0

terminate = 1;

for kkk = 1:iloop

goinidx = ceil(rand\*(n-p));

goin = x(goinidx);

w = 0; u = zeros(n,1);

for i = 1:m

if a(i,goin) < a(i,c1(i))

w = w + a(i,goin) - a(i,c1(i));

else

u(c1(i)) = u(c1(i)) + min(a(i,goin),a(i,c2(i))) - a(i,c1(i));

end

end

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if u(jj) < dmin

dmin = u(jj); gooutidx = j;

end

end

w = w + dmin; goout = s(gooutidx);

if w <= 0

x(goinidx) = goout;

s(gooutidx) = goin;

z = z + w;

for i = 1:m

if c1(i) == goout

if a(i,goin) <= a(i,c2(i))

c1(i) = goin;

else

c1(i) = c2(i);

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if a(i,jj) < dmin

dmin = a(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

else

if a(i,c1(i)) > a(i,goin)

c2(i) = c1(i); c1(i) = goin;

elseif a(i,goin) < a(i,c2(i))

c2(i) = goin;

elseif c2(i) == goout

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if a(i,jj) < dmin

dmin = a(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

end

end

if z < zbest

zbest = z;

sbest = s;

end

else

s1 = rand;

rcrit = exp(-w./temp);

if s1 <= rcrit

terminate = 0;

x(goinidx) = goout;

s(gooutidx) = goin;

z = z + w;

for i = 1:m

if c1(i) == goout

if a(i,goin) <= a(i,c2(i))

c1(i) = goin;

else

c1(i) = c2(i);

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if a(i,jj) < dmin

dmin = a(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

else

if a(i,c1(i)) > a(i,goin)

c2(i) = c1(i); c1(i) = goin;

elseif a(i,goin) < a(i,c2(i))

c2(i) = goin;

elseif c2(i) == goout

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if a(i,jj) < dmin

dmin = a(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

end

end

end

end

end

temp = temp .\* cool; % Temperature reduction

end

% DOUBLE-CHECK THE BEST-FOUND SOLUTION

P = zeros(n,1); Z = 0; exemplars = sbest(1:p); % Store best solution

for i = 1:n

dmin = 9.9e+12;

for j = 1:p

g = exemplars(j);

dist = 0;

for k = 1:d

dist = dist + (X(i,k)-X(g,k)).^2;

end

if dist < dmin

P(i) = j; dmin = dist;

end

end

if M ~= 0

Z = Z + sqrt(dmin);

else

Z = Z + dmin;

end

end

if M == 0

u = silhouette(X,P);

else

u = silhouette(X,P,'Euclidean');

end

W = 1:n; % This block is important for singletons

for k = 1:K % MATLAB's default is that objects in

y = W(P==k); ylen = length(y); % singletons have silhoettes = 1,

if ylen == 1 % but Kaufman and Rousseeuw (1990, p. 85)

u(y) = 0; % assume silhouettes = 0.

end

end

silohouette\_index = mean(u);

toc

**3) LRA.m**

function [exemplars, ZLB, E, terminate] = LRA(X, M, K, ZUB);

% LRA.m: A "Lagrangian relaxation algorithm" for the

% p-median problem. The procedure is based on the work of

% Cornuejols et al. (1977), Mulvey & Crowder (1979), Hanjoul &

% Peeters (1985). This specific implementation is the one

% described by Brusco and Koehn (2008a).

% INPUTS: X = an n x d matrix of measurements for n objects on

% each of d variables

% M = 0 for squared Euc. distance, otherwise Euclidean

% K = the desired number of medians/clusters/exemplars

% ZUB = is an "upper bound" on the optimal objective function

% value -- obtained from MFI program.

% OUTPUTS: exemplars = the best found set of medians/exemplars

% ZLB = the final lower bound obtained from the algorithm

% E = the percentage deviation between ZUB and ZLB

% terminate = 1 if the solution is confirmed optimal, 0 otherwise

% RECOMMENDATIONS FOR USING AND UNDERSTANDING THIS PROGRAM:

% 1) Run the MFI algorithm first and obtain an upper bound, ZUB.

% 2) Run this program with ZUB as the upper bound

% - If E = 0, the algorithm has obtained an optimal solution

% with objective value = ZLB. If E > 0, then terminate = 0 and

% an optimal solution is not guaranteed.

% 3) Optimal solutions in this program are "Epsilon-optimal".

% An optimal solution is "confirmed" if the ZUB - ZLB difference

% is less than epsilon, OR, if the sum of squared infeasibilities

% is less than epsilon. Be careful! Only in the second case are

% the reported exemplars necessarily the optimal ones.

state = 1; % fix state for testing

rand('state', state);

tic;

eps = 1e-5;

[n,d] = size(X);

S = zeros(n,n);

for i = 1:n-1

for j = i+1:n

dist = 0;

for k = 1:d

dist = dist + (X(i,k)-X(j,k)).^2;

end

if M ~= 0

S(i,j) = sqrt(dist);

else

S(i,j) = dist;

end

S(j,i) = S(i,j);

end

end

restarts = 5; maxit = 4000; taumax = 2; maxno = 200; zstar = -999;

terminate = 0;

LB = zeros(n,1); % Vector of lagrange multipliers

x = zeros(n); % n x n matrix of assignments

for ijk = 1: restarts

tau = taumax; % Reset tau to taumax

noimp = 0; % Counter for iters with no improvement

L = LB; % Reinstall best-found multipliers

ZR = -9.9e+12; % Initialize best Z for restarts

XL = repmat(L,1,n); % matrix of n replicated columns of L

H = S - XL; % distance matrix - mulitpliers

HH = min(H(:,:),0); % find negative values in H

Hsum = sum(HH); % Column sums of the HH matrix

[a, b] = sort(Hsum); % Sort the column sums of Hsum

s = b(1:K); u=b(K+1:n); % Select the first K exemplars in list

x(:,u) = 0; % x(i,j) columns are zeros for unselected

x(:,s) = (HH(:,s)<0); % mark <0 entries in selected columns

for h = 1:maxit

ZLB = sum(L)+sum(sum(H.\*x)); % Compute objective function

sumx = ones(n,1) - sum(x,2); % Compute infeasibilities

sumxx = sumx.^2; % Compute squared infeasibilities

denom = sum(sumxx); % Compute sum-of-squared infeas.

if ZUB - ZLB < eps || denom < eps % Check termination condition

terminate = 1;

break

end

if ZLB > ZR + eps

noimp = 0; ZR = ZLB; % If the bound improves, then store it

else % and reset "noimp" counter to 0

noimp = noimp + 1; % Otherwise, increment "noimp" counter

if noimp == maxno

noimp = 0; tau = tau/2; % Cut tau in half when iteration

end % limit is reached

end

t = tau.\*((ZUB-ZLB)./denom); % This block updates the Lagrange

for i = 1:n % multipliers for the next iter.

L(i) = L(i) + t.\*sumx(i);

end

XL = repmat(L,1,n); % matrix of n replicated columns of L

H = S - XL; % distance matrix - mulitpliers

HH = min(H(:,:),0); % find negative values in H

Hsum = sum(HH); % Column sums of the HH matrix

[a,b] = sort(Hsum); % Sort the column sums of Hsum

s = b(1:K); u=b(K+1:n); % Select the first K exemp in list

x(:,u) = 0; % x(i,j) col are zeros for unselected

x(:,s) = (HH(:,s)<0); % mark <0 entries in selected columns

end

if terminate == 1;

break

end

LB = L;

end

if terminate == 1

E = 0; exemplars = s;

else

E = 100.\*((ZUB - ZLB)./ZLB); exemplars = s;

end

toc

**4) MFI\_FC.m**

function [exemplars, Z, P, silohouette\_index] = MFI\_FC(S, K, R, restarts);

% MFI\_FC.m: A "multistart fast interchange" procedure for the

% p-median problem. The procedure is the Teitz and

% Bart (1968) VERTEX SUBSTITUTION HEURISTIC using the

% using fast updates from Hansen and Mladenovic (1997),

% which are based on Whitaker's (1983) work.

% NOTE: This program differs from MFI.m in two respects:

% 1) This program reads in an n x n DISSIMILARITY matrix directly.

% Accordingly, this program is not restricted to Euclidean distances,

% and even accommodates ASYMMETRIC matrices. To avoid "unusual"

% solutions, however, elements should be nonnegative.

% 2) This program reads a vector of 'costs' for objects to serve as

% exemplars. Thus, the 'FC' stands for the 'fixed charge' for

% selecting an object as an exemplar. Objects with lower costs have

% greater "preference" for serving as an exemplar. If there are no

% differences among exemplars with respect to preference, we recommend

% setting the n x 1 cost vector to all zeros.

% INPUTS: S = an n x n matrix of DISSIMILARITIES for n objects

% K = the desired number of medians/clusters/exemplars

% R = an n x 1 vector of exemplar preferences

% restarts = the desired number of restarts

% OUTPUTS: exemplars = the best found set of medians/exemplars

% Z = the best-found objective function value (including the

% fixed costs of the exemplars).

% silohoette\_index = index from Kaufman & Rousseeuw (1990)

% P an n by 1 vector of cluster assignments

state = 1; % fix state for testing

rand('state', state);

tic;

[n,n1] = size(S);

nreps = restarts; gbest = 9.9e+12; fstore = zeros(nreps,1);

c1 = zeros(n,1); c2 = zeros(n,1); p = K;

for klk = 1:nreps

s = randperm(n); % Randomly order the exemplar

a1 = S(:,s(1:p)); % candidates and let the first

fbest = sum(R(s(1:p))); % p be the selected subset

for i = 1:n; % Note: In this program

dmin = 9.9e+12; % i indexes the rows, which

for j = 1:p % assigned to exemplars or

if a1(i,j) < dmin % selected columns indexed by j

dmin = a1(i,j); jsel = j;

end

end

fbest = fbest + dmin; c1(i) = s(jsel); % fbest = initial objective;

end % c1(i) = exemplar to which

for i = 1:n % object i is most similar

dmin = 9.9e+12; % c2(i) = exemplar to which

for j = 1:p % object is 2nd most similar

jj = s(j); % is computed in this block

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

trig = 0;

while trig == 0 % The exchange process begins here

wstar = 0;

% The block below finds, for each unseleted point (goin), the best exemplar

% to remove (goout) if adding goin. The best (goin, goout) pair is

% identified and (wstar) is the resulting change in the objective function.

% If wstar >= 0, then the algorithm terminates because there is no viable

% exchange that will improve the objective function further.

for goin = p+1:n

ii = s(goin); w = R(ii); u = zeros(n,1);

for i = 1:n

if S(i,ii) < S(i,c1(i))

w = w + S(i,ii) - S(i,c1(i));

else

u(c1(i)) = u(c1(i)) + min(S(i,ii),S(i,c2(i))) - S(i,c1(i));

end

end

dmin = 9.9e+12;

for j = 1:p

jj = s(j);

if u(jj)-R(jj) < dmin

dmin = u(jj)-R(jj);

goout = j;

end

end

w = w + dmin;

if w < wstar

wstar = w; goinb = goin; gooutb = goout;

end

end

if wstar >= -.00001

trig = 1;

continue

end

% The block below updates the c1 and c2 vectors if an (goin, goout) swap

% results in an improvement.

fbest = fbest + wstar; goinc = s(goinb); gooutc = s(gooutb);

idum = s(goinb); s(goinb) = s(gooutb); s(gooutb) = idum;

for i = 1:n

if c1(i) == gooutc

if S(i,goinc) <= S(i,c2(i))

c1(i) = goinc;

else

c1(i) = c2(i);

dmin = 9.9e+12;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

else

if S(i,c1(i)) > S(i,goinc)

c2(i) = c1(i); c1(i) = goinc;

elseif S(i,goinc) < S(i,c2(i))

c2(i) = goinc;

elseif c2(i) == gooutc

dmin = 9.9e+12;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

end

end

end

if fbest < gbest

gbest = fbest; sbest = s; cbest = c1;

end

fstore(klk) = fbest;

end

% DOUBLE-CHECK THE BEST-FOUND SOLUTION

P = zeros(n,1); exemplars = sbest(1:p); Z = sum(R(exemplars));

for i = 1:n

dmin = 9.9e+12;

for j = 1:p

g = exemplars(j);

if S(i,g) < dmin

P(i) = j; dmin = S(i,g);

end

end

Z = Z + dmin;

end

X = [];

TRI = zeros(n.\*(n-1)./2,1);

ict = 0;

for i = 1:n-1

for j = i+1:n

ict = ict + 1; TRI(ict) = (S(i,j)+S(j,i))./2;

end

end

u = silhouette(X, P, TRI');

W = 1:n; % This block is important for singletons

for k = 1:K % MATLAB's default is that objects in

y = W(P==k); ylen = length(y); % singletons have silhoettes = 1,

if ylen == 1 % but Kaufman and Rousseeuw (1990, p. 85)

u(y) = 0; % assume silhouettes = 0.

end

end

silohouette\_index = mean(u);

toc

**5) SA\_FC.m**

function [exemplars, Z, P, silohouette\_index] = SA\_FC(S, K, R, cool);

%SA\_FC.m -- Simulated Annealing Heuristic for the P-median Problem

% We use the method published by Brusco & Koehn (2009),

% except that this program operates on DISSIMILARITY matrix.

% INPUTS: S = an n x n matrix of DISSIMILARITIES for n objects

% K = the desired number of medians/clusters/exemplars

% R = an n x 1 vector of exemplar preferences

% cool = the simulated annealing parameter. We recommend

% 0.9, but 0.8 often suffices and is faster.

% OUTPUTS: exemplars = the best found set of medians/exemplars

% Z = the best-found objective function value

% silohoette\_index = index from Kaufman & Rousseeuw (1990)

% P an n by 1 vector of cluster assignments

% NOTE: This program differs from SA.m in two respects:

% 1) This program reads in an n x n DISSIMILARITY matrix directly.

% Accordingly, this program is not restricted to Euclidean distances,

% and even accommodates ASYMMETRIC matrices. To avoid "unusual"

% solutions, however, elements should be nonnegative.

% 2) This program reads a vector of 'costs' for objects to serve as

% exemplars. Thus, the 'FC' stands for the 'fixed charge' for

% selecting an object as an exemplar. Objects with lower costs have

% greater "preference" for serving as an exemplar. If there are no

% differences among exemplars with respect to preference, we recommend

% setting the n x 1 cost vector to all zeros.

% NOTE: There is no 'tmin' in this program -- stops when no inferior

% solution is accepted

state = 1; % fix state for testing

rand('state', state);tic;

[m,n] = size(S); p = K;

t = randperm(n); % Initial random permutation

s = t(1:p); % s = the current set of exemplars

x = t(p+1:n); % u = unselected objects that are not exemplars

z = sum(R(s(1:p))); % fixed cost of selected exemplars

z = z + sum(max(S(:,s),[],2)); % z = the current objective function value

zbest = z;

sbest = s;

pbest = p; % The starting solution

tempmax = 0; % Find an initial temperature by sampling

for ijk = 1:200

i3 = int32(rand.\*(n-p)+1);

i = double(i3);

if i > n-p

i = n-p;

end

goin = x(i); % goin is the entering object

j3 = int32(rand\*p+1);

goout = double(j3);

if goout > p

goout = p; % goout is the exemplar position where goin enters

end

strial = s;

strial(goout) = goin;

ztrial = sum(max(S(:,strial),[],2));

ztrial = ztrial + sum(R(strial(1:p)));

if abs(z-ztrial) > tempmax

tempmax = abs(z-ztrial);

end

end

iloop = 10.\*n; % SA temperature length

temp = tempmax; % SA initial temperature

c1 = zeros(m,1); c2 = zeros(m,1);

a1 = S(:,s);

z = sum(R(s(1:p)));

for i = 1:m; % Note: In this program

dmin = 9.9e+15; % i indexes the rows, which

for j = 1:p % assigned to exemplars or

if a1(i,j) < dmin % selected columns indexed by j

dmin = a1(i,j); jsel = j;

end

end

z = z + dmin; c1(i) = s(jsel); % z = initial objective;

end % c1(i) = exemplar to which

for i = 1:m % object i is most similar

dmin = 9.9e+15; % c2(i) = exemplar to which

for j = 1:p % object is 2nd most similar

jj = s(j); % is computed in this block

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

% THE SA ALGORITHM BEGINS ON THE NEXT LINE

terminate = 0;

while terminate == 0

terminate = 1;

for kkk = 1:iloop

goinidx = ceil(rand\*(n-p));

goin = x(goinidx);

w = R(goin); u = zeros(n,1);

for i = 1:m

if S(i,goin) < S(i,c1(i))

w = w + S(i,goin) - S(i,c1(i));

else

u(c1(i)) = u(c1(i)) + min(S(i,goin),S(i,c2(i))) - S(i,c1(i));

end

end

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if u(jj)-R(jj) < dmin

dmin = u(jj)-R(jj); gooutidx = j;

end

end

w = w + dmin; goout = s(gooutidx);

if w <= 0

x(goinidx) = goout;

s(gooutidx) = goin;

z = z + w;

for i = 1:m

if c1(i) == goout

if S(i,goin) <= S(i,c2(i))

c1(i) = goin;

else

c1(i) = c2(i);

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

else

if S(i,c1(i)) > S(i,goin)

c2(i) = c1(i); c1(i) = goin;

elseif S(i,goin) < S(i,c2(i))

c2(i) = goin;

elseif c2(i) == goout

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

end

end

if z < zbest

zbest = z;

sbest = s;

end

else

s1 = rand;

rcrit = exp(-w./temp);

if s1 <= rcrit

terminate = 0;

x(goinidx) = goout;

s(gooutidx) = goin;

z = z + w;

for i = 1:m

if c1(i) == goout

if S(i,goin) <= S(i,c2(i))

c1(i) = goin;

else

c1(i) = c2(i);

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

else

if S(i,c1(i)) > S(i,goin)

c2(i) = c1(i); c1(i) = goin;

elseif S(i,goin) < S(i,c2(i))

c2(i) = goin;

elseif c2(i) == goout

dmin = 9.9e+15;

for j = 1:p

jj = s(j);

if c1(i) == jj

continue

end

if S(i,jj) < dmin

dmin = S(i,jj); jsel = jj;

end

end

c2(i) = jsel;

end

end

end

end

end

end

temp = temp .\* cool; % Temperature reduction

end

% DOUBLE-CHECK THE BEST-FOUND SOLUTION

P = zeros(n,1); exemplars = sbest(1:p); Z = sum(R(exemplars));

for i = 1:n

dmin = 9.9e+12;

for j = 1:p

g = exemplars(j);

if S(i,g) < dmin

P(i) = j; dmin = S(i,g);

end

end

Z = Z + dmin;

end

X = [];

TRI = zeros(n.\*(n-1)./2,1);

ict = 0;

for i = 1:n-1

for j = i+1:n

ict = ict + 1; TRI(ict) = (S(i,j)+S(j,i))./2;

end

end

u = silhouette(X, P, TRI');

W = 1:n; % This block is important for singletons

for k = 1:K % MATLAB's default is that objects in

y = W(P==k); ylen = length(y); % singletons have silhoettes = 1,

if ylen == 1 % but Kaufman and Rousseeuw (1990, p. 85)

u(y) = 0; % assume silhouettes = 0.

end

end

silohouette\_index = mean(u);

toc

**6) LRA\_GEN.m**

function [exemplars, ZLB, E, terminate] = LRA\_GEN(S, K, ZUB);

% LRA\_GENB.m: A "Lagrangian relaxation algorithm" for the

% p-median problem. The procedure is based on the work of

% Corneujols et al. (1977), Mulvey and Crowder (1979), and Hanjoul

% and Peeters (1985). This specific version is based on the

% implementation of Brusco and Koehn (2008a).

% INPUTS: S = an n x n matrix of DISSIMILARITIES for n objects

% K = the desired number of medians/clusters/exemplars

% ZUB = an upper bound on the objective function value obtained

% from the MFI\_FC algorithm.

% OUTPUTS: exemplars = the best found set of medians/exemplars

% Z = the best-found objective function value (including the

% fixed costs of the exemplars).

% ZLB = the lower bound on the objective function.

% E = percentage deviation of ZUB from ZLB

% terminate = 1 an optimal solution was confirmed and 0 otherwise

% RECOMMENDATIONS FOR USING AND UNDERSTANDING THIS PROGRAM:

% 1) Run the MFI\_FC algorithm first and obtain an upper bound, ZUB.

% 2) Run this program with ZUB as the upper bound

% - If E = 0, the algorithm has obtained an optimal solution

% with objective value = ZLB. If E > 0, then terminate = 0 and

% an optimal solution is not guaranteed.

% 3) Optimal solutions in this program are "Epsilon-optimal".

% An optimal solution is "confirmed" if the ZUB - ZLB difference

% is less than epsilon, OR, if the sum of squared infeasibilities

% is less than epsilon. Be careful! Only in the second case are

% the reported exemplars necessarily the optimal ones.

% NOTES:

% 1) This program reads in a generic n x n DISSIMILARITY matrix directly,

% rather than producing one, and that is how it differs from LRA.m.

% Accordingly, this program is not restricted to Euclidean distances,

% The algorithm has not been tested on ASYMMETRIC matrices, or matrices

% with negative elements - CAUTION is advised for such cases.

% 2) This program differs from LRA\_GENB in that an optimal solution is

% only returned for EITHER "epslion feasibility" or "epsilon lower

% bound vs upper bound" differences.

state = 1; % fix state for testing

rand('state', state);

tic;

eps = 1e-5; % Set epsilon for optimality tolerance

[n,n1] = size(S); % Obtain size of the data matrix

restarts = 5; maxit = 4000; taumax = 2; maxno = 200;

terminate = 0;

LB = zeros(n,1); % Vector of lagrange multipliers

x = zeros(n); % n x n matrix of assignments

for ijk = 1: restarts

tau = taumax; % Reset tau to taumax

noimp = 0; % Counter for iters with no improvement

L = LB; % Reinstall best-found multipliers

ZR = -9.9e+12; % Initialize best Z for restarts

XL = repmat(L,1,n); % matrix of n replicated columns of L

H = S - XL; % distance matrix - mulitpliers

HH = min(H(:,:),0); % find negative values in H

Hsum = sum(HH); % Column sums of the HH matrix

[a, b] = sort(Hsum); % Sort the column sums of Hsum

s = b(1:K); u=b(K+1:n); % Select the first K exemplars in list

x(:,u) = 0; % x(i,j) columns are zeros for unselected

x(:,s) = (HH(:,s)<0); % mark <0 entries in selected columns

for h = 1:maxit

ZLB = sum(L)+sum(sum(H.\*x)); % Compute objective function

sumx = ones(n,1) - sum(x,2); % Compute infeasibilities

sumxx = sumx.^2; % Compute squared infeasibilities

denom = sum(sumxx); % Compute sum-of-squared infeas.

if ZUB - ZLB < eps || denom < eps % Check termination condition

terminate = 1;

break

end

if ZLB > ZR + eps % If the bound improves, then store it

noimp = 0; ZR = ZLB; % and reset "noimp" counter to 0.

else

noimp = noimp + 1; % Otherwise, increment "noimp" counter

if noimp == maxno

noimp = 0; tau = tau/2; % Reduce the scaling factor if the

end % max iterations with no improvement

end % is reached.

t = tau.\*((ZUB-ZLB)./denom); % This block updates the Lagrange

for i = 1:n % multipliers for the next iteration

L(i) = L(i) + t.\*sumx(i);

end

XL = repmat(L,1,n); % matrix of n replicated columns of L

H = S - XL; % distance matrix - mulitpliers

HH = min(H(:,:),0); % find negative values in H

Hsum = sum(HH); % Column sums of the HH matrix

[a,b] = sort(Hsum); % Sort the column sums of Hsum

s = b(1:K); u=b(K+1:n); % Select the first K exemp in list

x(:,u) = 0; % x(i,j) col are zeros for unselected

x(:,s) = (HH(:,s)<0); % mark <0 entries in selected columns

end

if terminate == 1;

break

end

LB = L; % Reset multipliers for next restart

end

if terminate == 1 % Compute "error" as percent deviation

E = 0; exemplars = s; % between ZUB and ZLB.

else

E = 100.\*((ZUB-ZLB)./ZLB); exemplars = s;

end

toc