CUES : Clustering Using Evolutionary Search

(User Notes by Richard Forsyth, February 2015)

This program uses an evolutionary (Darwinian) optimization technique to perform clustering, i.e. it identifies within a dataset groups of items which in some sense belong together. An important point about CUES is that it decides on the number of groups as part of the optimization process without having to be given the number to find as input -- unlike many well-established clustering algorithms. It has been written in Python3 and is released under the GNU Public License for general usage.

Why I Wrote this Software

I have used clustering software over many years and during that time have tried many different clustering programs; for instance, those provided as standard within SPSS and R. However, I have never been entirely happy with any of the standard methods for choosing the number of groups. As Everitt has said:

> "In many applications researchers will want to select the partition with the 'best' number of groups for a data set ... A number of indices have been suggested for this purpose but it remains a difficult problem." (Everitt, 1993 : 89).

It is still an unresolved problem more than two decades later.

Recently, I wrote EASTIRS, a special-purpose clustering program for serial data which tries to identify natural break-points in sequential data sets, thus partitioning a series into segments, or phases, with similar values. This gave me the opportunity to work on this problem within an optimization framework. From single-dimensional clustering it was natural to move on to more general clustering, exploring quality indices that would balance the number of groups with the cohesiveness of the clusters. CUES is the result.

Setting Up

First you need Python3. If you don't have it already, the latest version can be downloaded and installed from the Python website: www.python.org. This is usually quite straightforward. The only snag is if you have Python2 and want to keep using it. Then you'll probably have to set up a specific command to run whichever version you use less frequently.

Next step is to unpack the cues.zip file. After unpacking it (into a top-level folder called "cues", unless you want to do lots of editing), you should find the following subfolders.

datasets
op
p3
parapath

The programs are in p3. Sample data sets for testing will be found in subfolder datasets. Subfolder op is the default location for output files and parapath is a convenient place for storing parameter files, which will be explained later. In Windows, it is most convenient to install cues at the top level of the C:\ drive, at least to start with; otherwise you'll have to edit the sample parameter files to make sure their datfile parameters point to the correct locations.

Data Format

The program expects to read its input values from data files such as can be exported from R (R Core Team, 2013) or Excel, with a header line giving column names, using the tab character as a delimiter. Data files can also be created in a text editor such as Notepad++ (http://notepad-plus-plus.org/),

preferably in utf-8 encoding.

The first four and last four lines of the sample datafile iris.dat are listed below to illustrate this format.

```
typename     sl     sw     pl     pw
setosa       5.1    3.5    1.4    0.2
setosa       4.9    3      1.4    0.2
setosa       4.7    3.2    1.3    0.2
....
virgin       6.3    2.5    5      1.9
virgin       6.5    3      5.2    2
virgin       6.2    3.4    5.4    2.3
virgin       5.9    3      5.1    1.8
```

This dataset is a well-studied collection of 150 cases known as "Fisher's Iris Data". It was originally collected by Edgar Anderson who gathered the data to study the morphological variation of Iris flowers of three related species. Two of the three species were collected in the Gaspé peninsula in Quebec (Anderson, 1935). The dataset consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features are measured from each sample: the length and the width of the sepals and petals, in centimetres. In the context of cluster analysis the points at issue are (1) whether the data naturally falls into 3 groups and (2) if so whether those groups reproduce the assigned species labels.

This iris dataset is an example of a rectangular 'flat file' with instances as rows and attributes as columns, a format used by many machine-learning and statistical packages. When CUES reads a rectangular file of this type, one of its first actions is to compute from it a distance matrix which will then be used to guide the clustering process.

CUES can also deal with an alternative input format: it is possible to give it a dissimilarity or distance matrix directly. This gives a user the chance to employ a special-purpose distance or dissimilarity measure which may suit the particular problem concerned. Such a data matrix should also be presented to CUES as a tab-delimited file, with case labels on the initial line and the same number of data rows as columns. The main diagonal, i.e. the dissimilarity score between each case and itself, should contain only zeroes. This is how CUES recognizes that it has been given a dissimilarity/distance matrix.

The first five lines of the file eurodist.dat, provided as an example in the datasets subfolder, are reproduced below as illustration.

```
Amsterdam    Athens Barcelona    Brussels     Calais Cherbourg    Cologne
       Copenhagen    Geneva Gibraltar    Hamburg      Hook of Holland     Lisbon
       Lyons  Madrid Marseilles    Milan  Munich Paris  Rome   Stockholm    Vienna
       Warsaw
0      2821   1531   202    362    789    259    788    982    2429   465    77
       2232   925    1769   1233   1074   824    502    1651   1435   1147   1188
2821   0      3313   2963   3175   3339   2762   3276   2610   4485   2977   3030
       4532   2753   3949   2865   2282   2179   3000   817    3927   1991   2348
1531   3313   0      1318   1326   1294   1498   2218   803    1172   2018   1490
       1305   645    636    521    1014   1365   1033   1460   2868   1802   2347
202    2963   1318   0      204    583    206    966    677    2256   597    172
       2084   690    1558   1011   925    747    285    1511   1616   1175   1300
....
```

This dataset gives the distances by road, in kilometres, between a selection of 23 towns and cities in Europe.

A small file containing an R function called distout() is provided in the main CUES folder. This will take a distance matrix as computed by the R function dist() and print it in the format suitable to be read into CUES. The reason this may be needed is that R only holds by default the upper diagonal entries of a distance matrix whereas cues.py expects a full N-by-N matrix, where N is the number of cases. And the reason CUES does that is to allow processing of asymmetric dissimilarity matrices where the dissimilarity between item i and item j isn't necessarily the same as the dissimilarity between item j and item i -- as with road distances, airline travel times and so forth. (Of course, since distout() is intended to print the results of using R's dist() calculations, its output will in fact be symmetrical. But users may have their own ways of generating asymmetric dissimilarity matrices, appropriate to certain problems.)

Preparing a Parameter File

When you run cues.py it will ask for the name of a parameter file. Below is a listing of parameter file iris.txt which comes with the distribution in parapath.

```
comment  cues testing on iris data :
jobname  iris
ntrials  131072
optimode  1
scaling  2
distmode  eu
datfile  c:\cues\datasets\iris.dat
skipvars  typename,sw
```

A parameter file is just a plain text file with one item per line. Each line should begin with the parameter name, then 1 or more blank spaces, then the parameter value. The following table interprets the above parameter file, line by line.

Parameter	Default value	Function
comment	[None]	This (or in fact any unrecognized parameter name, e.g. "##") can be used to insert reminders about what the file is meant to do.
jobname	cues	This gives the job a name. Any text string can be the value. It isn't necessary but it is useful as the jobname will be used as a prefix to the program's output files, so it can be seen that they form a group.
ntrials	131072	CUES uses an evolutionary algorithm to optimize the subdivision of the data, according to the fitness function selected by optimode. This parameter specifies how many evolutionary trials to make, i.e. the total number of gene-strings that will be created during the optimization process.
optimode	1	This selects which 'fitness function' is to be used. Three functions have been implemented (see below). Number 1 is the fastest and the mode that I would recommend, at least initially, as discussed below.
scaling	1	There are 3 scaling modes, 0 to 2: 0 indicates no scaling, i.e. using the data just as input; 1 requests scaling each column to give a value from 0 to 1 as a proportion of the range between the minimum and maximum values in that column; 2 causes each value to be standardized by subtracting the mean and dividing by the standard deviation of the column concerned. Note that scaling is not applied when distances matrices are input directly, only when the input is a 'normal' rectangular file.
distmode	city	This tells the system which distance mode to use. Any string other

		than "city" will cause Euclidean distances to be computed; otherwise city-block or 'Manhattan' distances will be calculated -- the default. Note that this parameter has no effect when an input dissimilarity/distance matrix is given directly as input.
datfile	[None]	This should be the full file specification of a file where the input data is stored (in tab-delimited form with a header line naming the columns).
skipvars	[None]	This should be a line of variable names separated by commas, identifying columns which are **not** to be used in the distance calculations. Note that at present only numeric columns are considered by CUES, so it isn't necessary to include string variables in this list. (Though that is done in the example above it will make no difference.) This parameter should not be used when a distance matrix is input directly to the program.

Running CUES

When you run cues.py it will ask for a parameter file. If this file is in the same directory as the program or in its parapath subfolder you won't have to give the full path specification, just its name (.txt extension will be presumed if no extension is given).

You should see on screen something like the listing below, which comes from a run using the iris.txt parameter file discussed in the previous section. This runs the program on Fisher's Iris data, excluding variable sw (sepal width) which is highly correlated with sepal length and has the weakest association with the species category of the four morphological features. (If you regard that as cheating, you can try running cues.py with sw included.)

```
c:\datwork>python c:\cues\p3\cues.py
C:\cues\p3\cues.py 1.7 Fri Jan 30 14:20:04 2015
command-line args. = 1
prepath : C:\cues\p3
working folder:  C:\datwork
script usage:  python C:\cues\p3\cues.py <parafile>
please give parameter file name : c:\cues\parapath\iris
Paths to search for parameter file :
['C:\\datwork\\parapath', 'C:\\datwork', '..', '.', 'C:\\Users\\Richard\\parapath',
'C:\\Users\\Richard']
 iris
trying to open : C:\cues\parapath\iris.txt
C:\cues\parapath\iris.txt opened for reading.
151 5
data rows = 150
data cols = 5
column names :
['typename', 'sl', 'sw', 'pl', 'pw']

variables to be used :
1 sl
id        0
mean      5.8433
std1      0.8253
std2      0.8281
var1      0.6811
var2      0.6857
xmax      7.9
xmin      4.3
xr        3.6

3 pl
id        0
mean      3.758
```

```
std1      1.7594
std2      1.7653
var1      3.0955
var2      3.1163
xmax      6.9
xmin      1
xr        5.9

4 pw
id        0
mean      1.1993
std1      0.7597
std2      0.7622
var1      0.5771
var2      0.581
xmax      2.5
xmin      0.1
xr        2.4

11175 distances.
number of pals = 12
mean number of comrades = 32.0267
150 18
cycles,subsize = 5 61
loop 1 26214 trials
SEAL incr 3
234 0.05633 !
6553 0.31258 !
13107 0.75158 !
19661 0.76228 !
26214 0.7698 !
26214 trials made.
0.7697972
loop 1 local score = 0.7697972
5 groups.
loop 1 global score = 0.8396219
DDDCDDCDCDDDDCDDDDDDDDCDDDDDDCDDDDDDDDCDDCCDDDDCDDBBBABABABAABABABAABABBBBBBBBBBAAAA
BABBBAAABAAAAABAAEBEBEEAEEEEEEBBEEEEBEBEBEEBBEEEEEBBEEEBEEEBEEEEEEB
loop 2 26214 trials
SEAL incr 3
234 0.05824 !
6553 0.27017 !
13107 0.6788 !
19661 0.80612 !
26214 0.82066 !
26214 trials made.
0.8206597
loop 2 local score = 0.8206597
4 groups.
loop 2 global score = 0.8961262
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDBBBCBCBCBCCBCBCBCCBCBBBBBBBBBCCCC
BCBBBCCCBCCCCCBCCABABAACAAABBABBABAABABABAABBAAAAABBAABBAAABAAABBBB
loop 3 26214 trials
SEAL incr 3
234 0.03776 !
6553 0.43739 !
13107 0.73381 !
19661 0.77356 !
26214 0.77356 !
26214 trials made.
0.7735582
loop 3 local score = 0.7735582
5 groups.
loop 3 global score = 0.8244502
loop 4 26214 trials
SEAL incr 3
234 0.0455 !
6553 0.50536 !
```

```
13107 0.81089 !
19661 0.83985 !
26214 0.86327 !
26214 trials made.
0.8632674
loop 4 local score = 0.8632674
3 groups.
loop 4 global score = 0.9063978
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBABBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBABBBBABCBCBCCBCCCBBCBBCBCCBCBCBCCBBCCCCBBBCBBBCCCBCCCBBBB
loop 5 26216 trials
SEAL incr 3
234 0.04717 !
6554 0.353 !
13108 0.82305 !
19662 0.86808 !
26216 0.86808 !
26216 trials made.
0.8680804
loop 5 local score = 0.8680804
4 groups.
loop 5 global score = 0.8543987
3 groups.
190.258 secs elapsed.
150
post-tidying loop 1 begins with best score 0.9063978
post-tidying loop 1 ends with best score 0.9279532
92
post-tidying loop 2 begins with best score 0.9279532
post-tidying loop 2 ends with best score 0.9313336
56
post-tidying loop 3 begins with best score 0.9313336
post-tidying loop 3 ends with best score 0.9313336
3 clusters.
8 changes made in post-tidying.
results dumped on  C:\op\iris_dump.dat
C:\cues\p3\cues.py done on Fri Jan 30 14:23:19 2015
after 192.083 seconds.
```

The only user input required here has been marked in **bold face**, i.e. the program launch command and the name of the parameter file, iris.txt, which is found in folder c:\cues\parapath\ with .txt extension. The main output file (iris_dump.dat) will be exactly the same as the input data except that an extra column, with the name "clus_ter" in the header line, will have been added. This column contains the integer cluster labels assigned by cues.py to each row. Such output files can be read into R for further analysis by an R command such as

cuesout = read.delim("c:\\cues\\op\\iris_dump",fileEncoding="utf-8")

where the precise file specification will be what cues.py displays just before it ends.

Interpreting the output

The listing above begins by displaying information about the run's settings, including basic statistics of the three variables chosen for analysis, sl, pl & pw. (See Upton & Cook (2006) if uncertain about statistical terminology.) It confirms that there are five columns and 150 rows in this dataset.

The main body of the listing shows the progress of the clustering. The program always does more than one cycle of the evolutionary optimization process, the exact number of cycles is the integer part of the natural logarithm of the number or rows of data. In this case there were 5 evolutionary cycles, each applied to a random subset of the whole dataset. The parameter file requested 131072 evolutionary trials: these are divided into four runs of 26214 trials and one of 26216 trials. This approach, of spreading computational resources over a few shorter runs rather than a single long

run, is a heuristic that has proved effective in a large number of stochastic optimization applications.

During each cycle the program prints the best fitness score achieved during that cycle at five points: firstly near the start of the procedure, then when a quarter, a half, and three quarters of the cycles have been used, and then the final score. The difference between a "local score" and "global score" after each loop is the difference between the fitness score on the selected subset and the fitness score once all instances in the dataset have been allocated. If the global score of the latest cycle is the best yet, a string representation of the best clustering is also displayed, with letters as cluster labels, in the same sequence as the input data cases.

Towards the end of the listing are the results of one or more "post-tidying loops". These show that cues.py is in fact a hybrid method. The main processing is done by an evolutionary algorithm, but experience shows that this often leaves 1 or 2 items in a cluster from which they appear distant to the human eye (at least in 2-dimensional examples). Thus I have appended a greedy hill-climbing algorithm which usually finds a way of making a few reallocations that improve the overall fitness score.

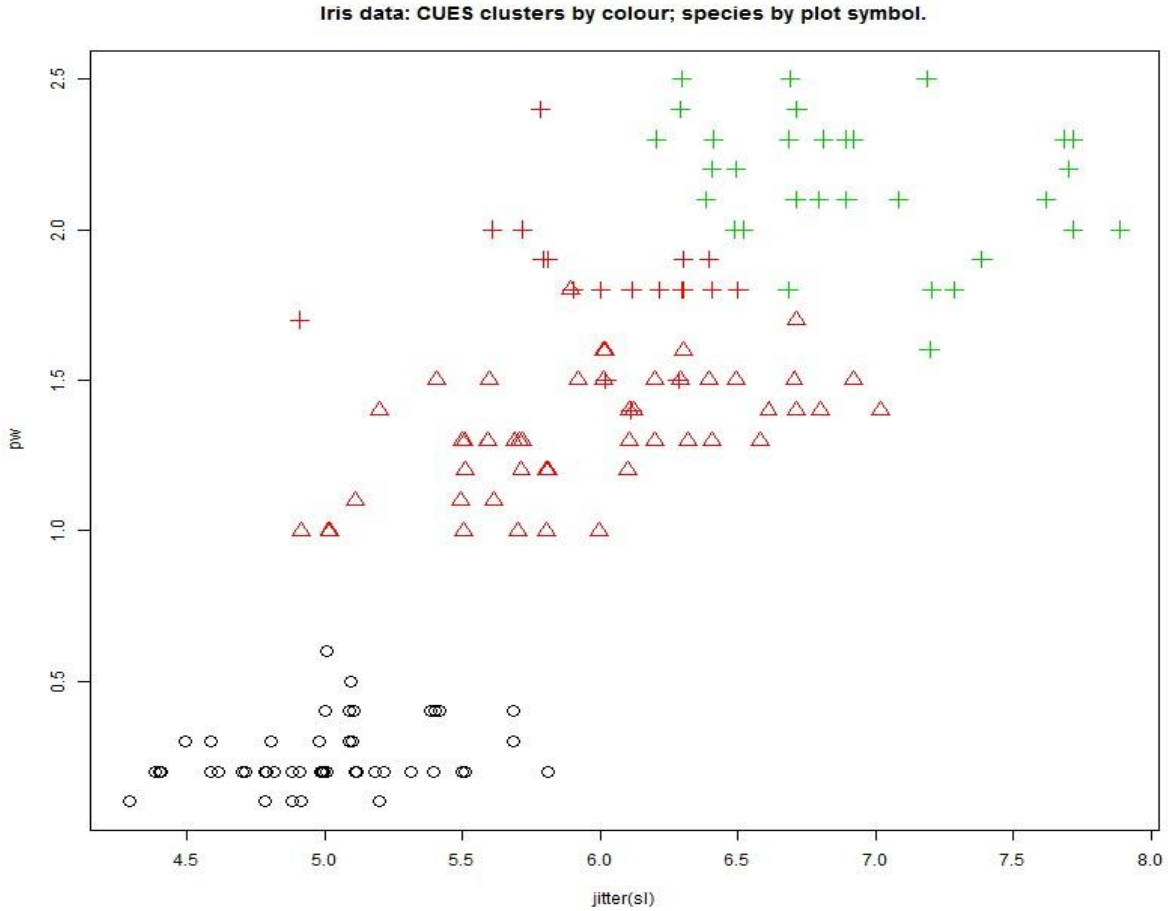
(If you choose a small value for the ntrials parameter, you can force the post-tidying routine to do most of the work. Then you will find that it is rather slow and not very efficient. However, it is pretty effective at reassigning a few obvious misplacements in a nearly correct clustering.)

As the above will have made clear, CUES is a stochastic process; thus it does not guarantee an optimal solution. It is good at 'satisficing', however, which means that it can normally be relied upon to arrive at a near-optimal clustering. In the present example, it has found a solution with 3 groups; but those groups aren't identical with the species categories. A tabulation of the species categories against the CUES clusters found in the run above is given below.

CUES group:	0	1	2
Setosa	50	0	0
Versicolor	0	50	0
Virginica	0	19	31

All fifty Iris Setosa cases have been put into a coherent group, and 31 of the Iris Virginica cases are in a group of their own; but 19 of the Virginica instances fall into a group which contains all fifty of the Iris Versicolor instances. In other words, some Virginica plants seem more like Versicolor than their own species.

Below is a 2D scatter plot which illustrates this by showing the results in the space of 2 of the features, pw & sl. Here the colours distinguish the three CUES groupings while the plot symbols distinguish the pre-existing categories. It will be seen that Versicolor and Virginica examples do overlap in this feature space. A botanist might respond to this result by re-visiting the species assignment, or considering whether indeed Iris Versicolor and Virginica are distinct species.



How the program works

The problem of clustering is relatively well suited to solution by an evolutionary approach, since a clustering can be described by a string of integers, one for each data case, with each integer indicating the class to which that particular case belongs. Strings of this type are exactly what evolutionary/genetic algorithms work with most effectively. They can be chopped up and recombined or mutated very simply to produce fresh candidate solutions, thus searching the space of all possible clusterings.

The evolutionary optimizer used here can be found in the py3seal.py library, on folder p3. It is a streamlined version of a technique described in section 4.1 of Forsyth (1996).
http://www.richardsandesforsyth.net/pubs/ppsn1996.pdf

To apply evolutionary optimization to this kind of problem, a 'fitness function' must be defined. This gives a quantitative evaluation of the quality of any given clustering. The current version of CUES implements three different fitness functions, one of which is selected by giving a number, 1, 2 or 3, as value of the optimode parameter in the parameter file. If none is give, mode 1 will be used.

Optimode 1 is an original invention. The algorithm has a sociological or tribal flavour: it tries to ensure that as many cases as possible are grouped with other cases with which they have affinities (as defined by the dissimilarity/distance matrix) and apart from cases from which they are most different. It is also the fastest of the three modes and currently the default setting. The idea behind it is to create, for each item in the dataset, four sets of related items, namely 'pals', 'foes', 'comrades' and 'outcasts'. Pals and foes are of fixed size, nf, where nf is the rounded square root of the number of rows in the dataset. For each case, its pals and foes are simply the nf nearest other

cases to it and furthest from it, respectively. For each item its comrades are those items closer to it than the 25th percentile (lower quartile) of all the inter-item distances in the whole dataset. For each case its outcasts are simply those items whose distances from it are greater than the median (50% level) of all inter-item distances in the data. Thus the sizes of sets comrades and outcasts vary between cases, dependent on the density of the region around each case.

Having established these four sets for each case in the data, the quality of a clustering is computed as follows. Each case contributes (A + B - C) to an overall total
where
A = (pg - fg) / nf
B = c / co
C = o / oc
and where
pg is the number of its pals in the current case's cluster;
fg is the number of its foes in the current case's cluster;
nf is the size of the sets pals & foes, as above;
c is the number of its comrades in the current case's cluster;
co is the number of comrades of the current case;
o is the number of its outcasts in the current case's cluster;
oc is the number of outcasts of the current case.

The contributions of every case are summed and divided by twice the number of cases under consideration to give an overall fitness score which will be between -1 and 1. This may sound long-winded, but it is faster to compute, in general, than the other two fitness functions.

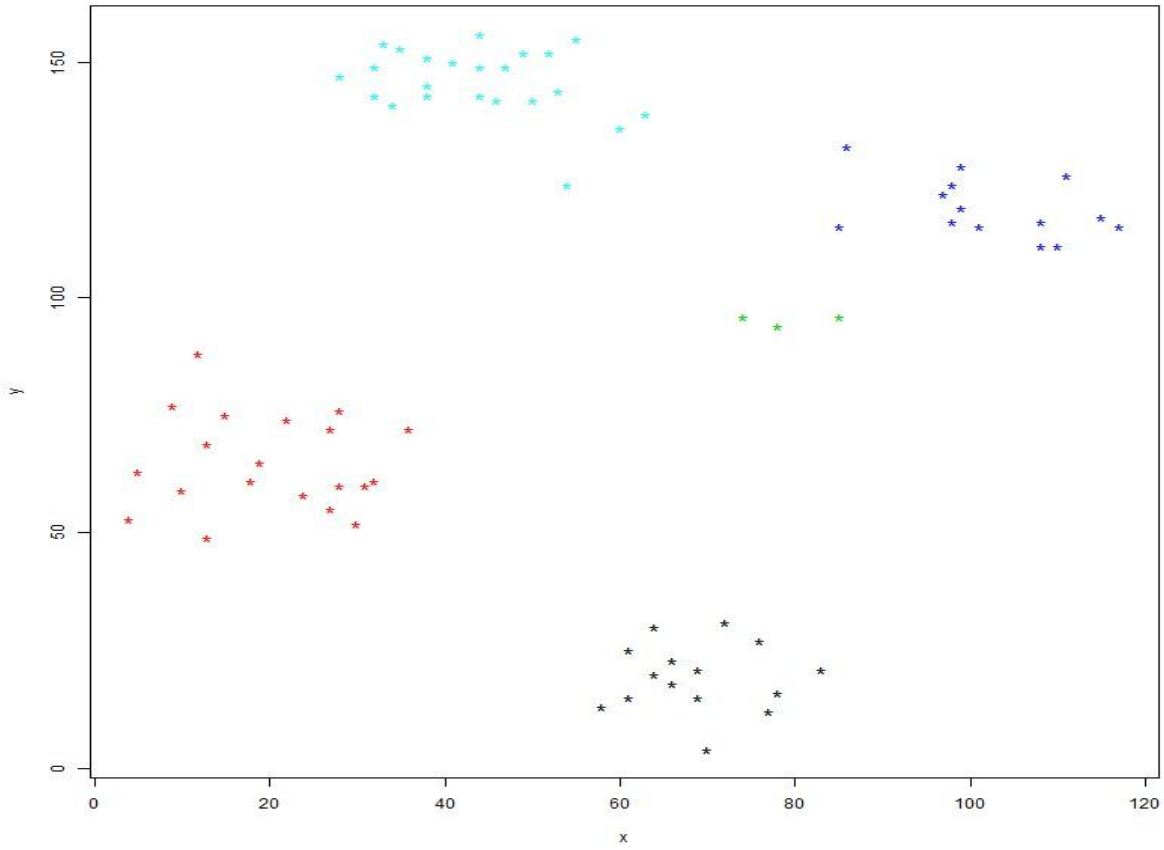
Optimode 2 was intended as a robust version of optimode 3, below, but it does not perform very satisfactorily in practice, so I may well remove it at some stage. Available at user's risk for experimentation.

Optimode 3 is conceptually simple. It computes the mean value of all within-cluster inter-item distances and the mean value of all between-cluster inter-item distances then subtracts the former from the latter and divides by an estimate of the maximum expected difference. It increases with the proximity of cluster members to each other and with their distances to members of other clusters. It also lies in the range -1 to +1, though rarely getting close to those extremes.

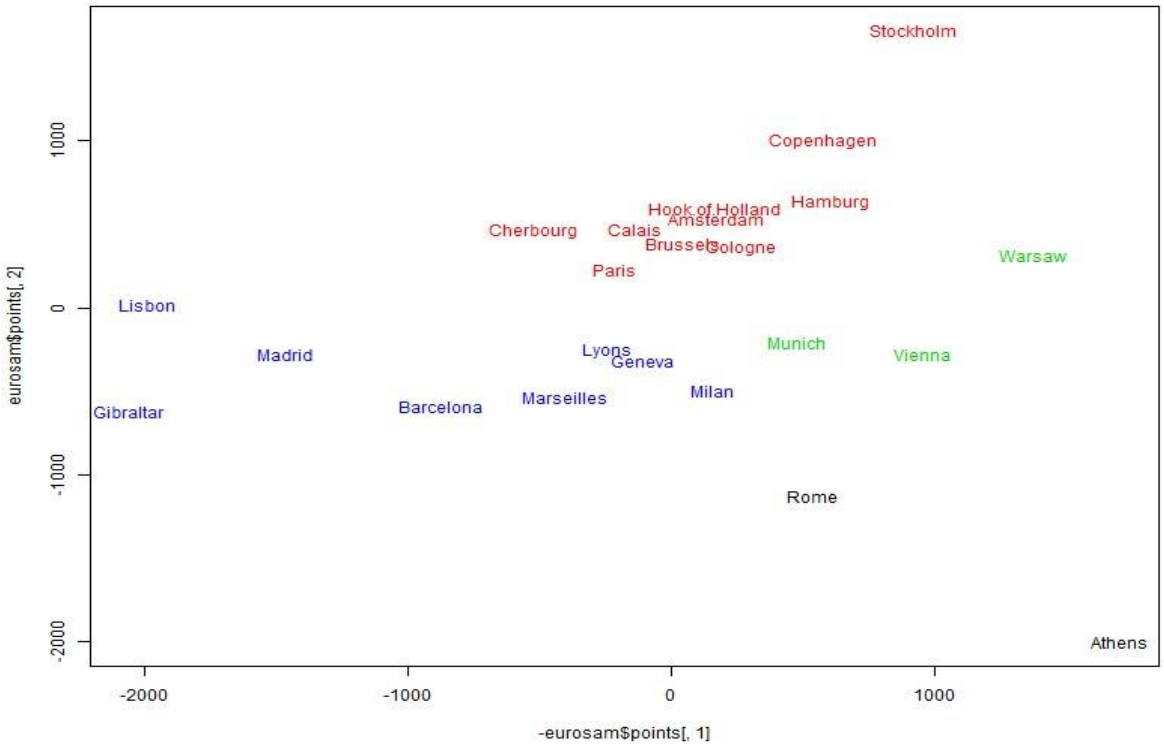
Notice that there is no explicit penalty in any of these formulas for the actual number of clusters. Nevertheless it is clear that, unless all inter-item distances are equal, the extremes of putting every item in the same cluster or putting every item in a different cluster cannot achieve a maximum score. In practice, optimode 1 tends to find its highest value near the natural logarithm of the number of data cases, and optimode 3 tends to find its highest value near the square root of the number of data cases.

Example results
Two further illustrations follow. The first is the classic Ruspini dataset (Ruspini, 1970) provided as ruspini.dat in the datasets subfolder. This has just 2 dimensions so is suited to plotting. With optimode 1, cues.py normally finds the 'standard' four-group solution; but with optimode 3, it tends to converge on a five-group solution, as illustrated here. Arguably, based on visual impression, a 6-group solution would also make sense. You can judge for yourself by looking at the graph.

Ruspini data clustered by CUES mode 3.



Euro cities coloured by group after CUES, optimode 1.



The second graph displays a solution found by applying cues.py to the eurodist.dat sample data file using optimode 1. Here the input is a distance matrix (road distances in kilometres). In order to display it as a 2D plot, I used R's sammon() multidimensional scaling function, from the MASS library, on the distances. This produced 2 dimensions which do roughly correspond to north-south and east-west. The latter happens to be inverted from a geographic point of view, so its negation has been used as the x-axis. Here the program has found a four-group solution, as indicated by the colours. There is no definitive right answer, but this looks reasonable to my eyes.

These graphs were obtained by reading the output file from cues.py, including the extra column headed 'clus_ter', into R and using R's plotting facilities. This is how I imagine users will generally employ the CUES software -- dumping data from R with write.table(), generating a clustering with cues.py, then re-reading the data with cluster labels back into R with read.delim() for further analyses. (I thought of writing the whole thing in R, but Python is faster and handles sets more neatly. Also I know Python better.)

Acknowledgements

Thank you for reading this far. :-)

References

Anderson, E. (1935). "The irises of the Gaspé Peninsula". *Bulletin of the American Iris Society* 59: 2–5.
http://en.wikipedia.org/wiki/Iris_flower_data_set

Everitt, B.S. (1993). *Cluster Analysis*, third edition. London: Arnold.

Forsyth, R.S. (1996). IOGA: an instance-oriented genetic algorithm. In: Voight, H.-M., Ebeling, W., Rechenberg, I. & Schwefel, H.-P. (eds.) *Parallel Problem Solving from Nature -- PPSN IV.* Berlin: Springer.
http://www.richardsandesforsyth.net/pubs/ppsn1996.pdf

R Core Team (2013). R: A language and environment for statistical computing. *R Foundation for statistical Computing*, Vienna, Austria.
http://www.R-project.org/.

Ruspini, E.H. (1970). Numerical methods for fuzzy clustering. *Information Science,* 2, 319–350.

Turchin, P., Peiros, I. & Gell-Mann, M. (2009). Analyzing Genetic Connections between Languages by Matching Consonant Classes. *Cliodynamics*, October 2009.
http://cliodynamics.info/PDF/ConsClass.pdf

Upton, G. & Cook, I. (2006). *Oxford Dictionary of Statistics*, second edition. Oxford: Oxford University Press.

Wong, M.A. & Lane, T. (1983). A kth nearest neighbour clustering procedure. *J. Royal Statistical Society*, B, 45, 362-368.

Appendix 1 : Parameter Files

Parameters used by **cues**.py are described below.

Parameter	Default value	Function
comment	[None]	This (or in fact any unrecognized parameter name, e.g. "##") can be used to insert reminders about what the file is meant to do.
datfile	[None]	This should be the full file specification of a file that indicates where the input data is stored (in tab-delimited form with a header line naming the columns).
distmode	city	This tells the system which distance mode to use. Any string other than "city" will cause Euclidean distances to be computed; otherwise city-block or 'Manhattan' distances will be calculated -- the default. Note that this parameter has no effect when an input matrix is given directly as input.
dumpfile	[None]	This is the specification of the file where the main output will be written. Its contents will be the same as the input data (datfile) except that an additional column (headed 'clus_ter') will be appended, containing the numeric cluster codes. If none is specified, this file will have the jobname followed by "_dump.dat".
jobname	cues	This gives the job a name. Any text string can be the value. It isn't necessary but it is useful as the jobname will be used as a prefix to the program's output files, so it can be seen that they form a group.
listfile	[None]	File mainly for debugging output. If none is specified, the file will have the jobname followed by "_list.txt".
longtime	3600	If the time elapsed, in seconds, is as long as or longer than this when the evolutionary phase has finished, only a highly abbreviated version of the post-tidying process will be performed. If less than longtime seconds have elapsed when the evolutionary phase finishes, the full post-tidying procedure will be executed.
ntrials	131072	CUES uses an evolutionary algorithm to optimize the subdivision of the data, according to the fitness function selected by optimode. This parameter specifies how many evolutionary trials to make, i.e. how many gene-strings will be created during the evolutionary optimization process. [*]
optimode	1	This selects one of three fitness functions to be used (1,2,3). Optimode 1 is the default. It is also the fastest. Optimode 3 tends to create a larger number of clusters. These modes are explained above, in the subsection headed "How the program works".
outfile	..\op\cues.txt	File that will receive parameter setting information, for reference.
outpath	..\op\	Directory to receive output files. By default this is the op subfolder of the parent directory of the program.
scaling	1	There are 3 scaling modes, 0 to 2: 0 indicates no scaling, i.e. using the data just as input; 1 requests scaling each column to give a value from 0 to 1 as a proportion of the range between the minimum and maximum values in that column; 2 causes each value to be standardized by subtracting the mean and dividing by the standard deviation of the column concerned. Note that scaling is not applied when distance matrices are input directly, only when the input is a 'normal' rectangular file.
skipvars	[None]	This should be a line of variable names separated by commas,

		identifying columns which are **not** to be used in the distance calculations. Note that at present only numeric columns are considered by CUES, so it isn't necessary to include string variables in this list. This parameter should not be used when a distance matrix is input directly to the program.

[*] With modest-sized data sets, up to about 256 data points, CUES is surprisingly effective. However, the search space expands multiplicatively with the number of data points, so theory as well as practical experience suggests that the system will hit the 'combinatorial wall' somewhere around a thousand data points.

N.B. At present, if you have more than 1 blank lines in a parameter file, the input routine chokes. I do plan to fix this at some stage; in the mean time, it is quite easy to delete empty lines using a text editor.

Appendix 2 : Sample Datafiles

These are provided in the datasets subfolder, so that users can experiment with the system before using it on their own data sets.

eurodist.dat
This is a distance-matrix file containing road distances in km between a selection of 23 European towns and cities.

iris.dat
This is a rectangular data file with five columns containing information about three species of iris flowers (Anderson, 1935). It is commonly known as Fisher's Iris Data since it was studied by Sir Ronald Fisher in his development of the technique of linear discriminant analysis (LDA).

lingdis1.dat
This is a distance-matrix file derived from an article about relatedness among languages by Turchin et al. (2009). Their method involved taking 100 meanings in 53 languages (several of them reconstructed 'proto-languages') and recording the phonetic classes of first 2 consonants of the (most standard?) word with that meaning. I wrote a little Python program to compute from this table the number of mismatches between these 100 pairs of consonant classes for each language pair, thus constructing a dissimilarity matrix, held in the lingdis1.dat file. If you apply cues.py to this datafile with optimode 1, you will rediscover some of the classic language 'families' proposed by linguists, such as Altaic, Indo-European and Semitic. With optimode 3 you will rediscover many subfamilies, such as Germanic, Romance and Slavic. For reference, the original data is also provided in the file Turchin09CCM.txt, which can also be found online as an Excel file at:
http://cliodynamics.info/data/SuppInfo.xls

page113.dat
This is a 2-dimensional dataset produced by random sampling from 2 different bivariate normal distributions -- 50 observations from each -- with the same parameter values as described in (Everitt, 1993) and illustrated in figure 6.1 on page 113 of that book. It isn't exactly the same as the data that gave rise to that figure, but it is a sample of the same size with the same characteristics.

ruspini.dat
This is the classic 2-dimensional dataset used by Ruspini (1970) to test his clustering algorithm which has become something of a benchmark example in the field of cluster analysis. (Illustrated above.)

wonglane.dat
I produced this dataset of 52 items by measuring as well as I could the x & y coordinates, in millimetres, of the data points in figure 7.1 on page 129 of (Everitt, 1993). This figure derives from an article by Wong & Lane (1983), which I haven't read, though I'd be willing to wager a half pint of beer that at least 100 of the 104 coordinates that I transferred from the page will prove to be within plus or minus half a millimetre (appropriately scaled) of their correct values. First bet-taker only. ;-)

zoobase.dat
This is a home-brew dataset that I came up with many moons ago, which somehow found its way into the UCI machine-learning repository. It describes 101 animal species in terms of 16 attributes, most of them binary. Zoologically speaking it is rather amateurish, but it still serves as an instructive test case. Further details can be found in file zoobase.txt, in the same subfolder.

Four datasets intended for use with EASTIRS have also been included, though they are less relevant to CUES: see Eastirs User Notes for details.

aircraft.dat
This dataset contains information (16 columns) concerning 103 makes of military aircraft used in World War II. For further details, see aircraft.txt in the same subfolder.

lynxdat.dat
This is the classic Canadian lynx time series, as obtained from the lynx dataset provided with the R package. http://www.R-project.org/.

michuron.dat
This data records the annual mean level of Lake Huron (which is the same as Lake Michigan) over the years 1918 to 2013, obtained from the U.S. National Oceanic and Atmospheric Administration website. http://www.glerl.noaa.gov/data/now/wlevels/levels.html

spots101.dat
This data series records the mean daily sunspot numbers observed during the period 1913 to 2013 inclusive. Original source: WDC-SILSO, Royal Observatory of Belgium, Brussels.
http://www.sidc.be/silso/datafiles