This tutorial walks you through the steps you need to make to run the SESC algorithm. At the current stage SESC is written in multiple programming languages and uses WEKA and MATLAB packages, therefore multiple steps are necessary to complete the SESC process. Once a data set is given a script calls a specified clustering algorithm WEKA (Xmeans or EM) repeatedly until is created. A MATLAB script then reads the output of WEKA and creates the files the C++ code needs as input. The whole process is divided into 3 steps and explained in details here.

The code is compiled and tested under MacOS 10.6.8, Ubuntu 12.10 and CentOS 5.5.

I. Create a grid for a given data set

In order to create the grid for a given data set you need the data set file in *arff* and *dlm* format (a *dlm* file is a *csv* file where all commas are replaced with a single space, you can easily do this in any text editor. From an *arff* file remove the attributes description header, then replace all commas with spaces and you will get the *dlm* file)

1. Next step is to run a clustering algorithm that doesn't need the cluster count a priori (e.g. Xmeans or EM) on each attribute. The Shell script "**AttributeWiseClustering**" does this. Make sure to place this script within the WEKA installation folder. You need to provide the following input arguments to the script: where your data set is located without the trailing slash (*datasetPath*), data set name without the .arff extension (*datasetName*), attribute count of the data set (*AttributeCount*) and the clustering method, Xmeans or EM (*ClustererName*):

USAGE: *./AttributeWiseClustering datasetPath datasetName attributeCount clustererName*

This will output a single file for each attribute storing which 1D cluster each instance is assigned to within that attribute. These files are temporary files that WEKA creates and won't be used later on, so you can get rid of them once WEKA is done. Another output of the mentioned script is a file called "*dataset-resultsby-clusterer*", where *dataset* is the data set name and *clusterer* is either Xmeans or EM. This is the file that stores the 1D cluster centroids for each attribute separately. This file will be passed to a MATLAB script to extract only the 1D cluster centroids for each attribute.

2. Once 1D clustering is performed run the MATLAB “**prepare_dataset.m**” script to produce all the necessary files for the C++ code. This script uses two other scripts that are included in the script folder. The inputs to this script are: *dataset_name* which is the path along with the name of the data set in *dlm* format, *labels_provided* is 1 if the last column of the data set contains class labels and 0 otherwise, *clusterer_name* defines the clustering method used by WEKA, Xmeans or EM, and *delete_files* is set to 1 if we need all temporary files deleted after the script is done. It is a safe practice to withhold from deleting the temporary files until you are sure you do not need them. If for any reason you need to modify and re-run the script all temporary files are there, and

there's no need to redo the WEKA step again. The following script is run within MATLAB UI:

USAGE: *prepare_dataset(dataset_name, labels_provided, clusterer_name, delete_files);*

This will create 6 files, two of which are temporary files and can be deleted (they end in _*temp*). The other four files are as follow: "*dataset-AttributeWiseCentroids*" stores the 1D centroids for each attribute, the first value in each row is the number of 1D centroids and the following values are 1D centroids. "*dataset-AttributeWiseLabels*" stores the 1D cluster index of each data instance for that attribute, therefore it is the same size as the data set file. The third file is "*dataset-NoneligibleAttributes*" which is a vector of binary values and tells the ESC algorithm which attributes are not useful (non-eligible). A value of 1 for an attribute determines an attribute as not useful and 0 means it is useful and can be indexed by ESC. In our preprocessing script there are two reasons to mark an attribute as non-eligible; 1. an attribute has only one cluster (i.e. there is only one single constant value repeating for all instances), or 2. an attribute has a cluster that covers 99% or more of the instances. The fourth file "*dataset-Noneligible1DCentroids*" is similar to the previous mentioned file, however this file flags non-eligible 1D cluster centroids to be filtered out instead of non-eligible attributes.

If class labels exist, two other files will be created. Both files are only necessary for post-training performance evaluation for data sets with class labels. The first file, "*dataset-Labels*", only stores class labels while the second file, "*dataset-AttributeGroundTruth*" specifies relevant attributes to each subspace cluster. This file acts as the attribute ground truth for attribute relevance evaluation.

II. Initialized S-ESC parameters

Here's a list of ESC input parameters that you might want to modify in *createInputs.pl* script:

my $envType = "datasetEnv";

Do not modify the *envType* parameter.
==

Data set name, path, dimensionality and cardinality:

my $dataSetName = "/Users/vahdat/ESC/data/50d4c/EM/50d4c.dlm";
my $labelsProvided = 1;
my $dataSetDim = 50;
my $dataSetSize = 1289;

dataSetName is the name (along with the path) of the data set ending in .dlm
If the last column of the data set provides class labels set *labelsProvided* to 1, otherwise 0.

dataSetDim and *dataSetSize* refer to the dimensionality and cardinality of the data set respectively.
==

Input file paths:

my $attrWiseClusCentFileName = "~/data/50d4c-AttributeWiseCentroids";
my $attrWiseClusLabelsFileName = "~/data/50d4c-AttributeWiseLabels";
my $noneligibleAttrFileName = "~/data/50d4c-NoneligibleAttributes";
my $noneligibleClusFileName = "~/data/50d4c-Noneligible1DCentroids";

attrWiseClusCentFileName is the path (along with the name of the file that stores the 1D centroids. *attrWiseClusLabelsFileName* is the name of the file that stores 1D cluster labels for all instances. *noneligibleAttrFileName* is the name of the file that stores the binary vector determining the non-eligible attributes, and similarly *noneligibleClusFileName* stores the 1D cluster centroids that are not useful for ESC to index while forming subspace cluster centroids.
==

Number of levels and objectives in each level

my $numLevels = 2;
my $numObjectives0 = 0;
my $numObjectives1 = 2;

numLevels should be set to 2. *numObjectives0* determines the number of objectives at level 0 in case there is a multi-objective process going on for level 0. In our case it should be 0, however since this value is used for allocating memory purposes as well we set it to 2. *numObjectives1* is similarly the number of objectives for level 1. In our case it is 2.
==

Identifying objectives:

my $doDistortion = 1;
my $doConnectivity = 1;
my $doAttrCount = 0;
my $doClusterCount = 0;

doDistortion, *doConnectivity*, *doAttrCount*, and *doClusterCount* tell ESC which objectives to use. Currently we use distortion (compactness) as the primary objective and connectivity as the secondary objective. In some experiments Cluster count is used instead of connectivity. The objectives to be used are set to 1, and 0 otherwise.
==

Point population parameters:

my $doSubsampling = 1;
my $pointPopSize = 100;

doSubsampling determines whether subsampling is performed (set to 1) or not (set to 0), and *pointPopSize* determines the number of points (instances) in case subsampling is being used.
==

Host (team, clustering solution) level parameters:

my $hostPopSize = 100;
my $minSymbionts = 2;
my $maxSymbionts = 20;
my $pc = 1.0;
my $pm = 1.0;
my $psd = 0.1;
my $psa = 0.1;
my $pss = 0.1;
my $pmlm = 1;

hostPopSize is the number of clustering solutions (hosts / level1 individuals), *minSymbionts* and *maxSymbionts* being the min and max number of clusters (symbionts) within a clustering solution (host) respectively, i.e. the min and max value for *k*. *pc* is the probability of crossover (not used in current version of ESC). *pm* the overall probability of mutation (always set to 1.0). *psd*, *psa*, *pss* are the probability of symbiont deletion, symbiont addition, and symbiont swap/replace for the second time. They all get performed at least once, however they might are re-applied with the above-mentioned probabilities a second time. The probability of further re-application of each operator is multiplied by these values, 0.01, 0.001, etc. *pmlm* is also the probability of multi-level mutation (keep at 1.0 also.)
==

Symbiont (cluster) level parameters:

my $minAttributes = 2;
my $maxAttributes = 20;
my $pad = 0.1;
my $paa = 0.1;
my $pas = 0.1;
my $plc = 0.1;

minAttributes and *maxAttributes* are the min and max number of attributes to be indexed in a subspace cluster centroid (symbiont) respectively. *pad*, *paa*, *pas*, and *plc* are the probabilities of

attribute deletion, attribute addition, attribute swap, and 1D centroid change mutation operators respectively. Similar to host-level mutation operators all the above operators are applied at least once, and the above-mentioned probabilities are used if an operator is to be applied a second time. The probability of further re-application of each operator is multiplied by these values, 0.01, 0.001, etc.
==

Number of generations at each level:

my $numGenerations0 = 0;
my $numGenerations1 = 1000;

numGenerations0 is the number generations in case there is a multi-objective process assigned for level 0 (symbiont). In our case this is set to 0. *numGenerations1* is similarly the number of generations for level 1 (host) multi-objective process. A typical value for this is 1000, however this can be changed according to the task and data set.
==

Evolutionary parameters:

my $symbsToUse = 0;
my $exe = "esc";
my $seed = 1; # first seed number, last seed number is the input from command line
my $statMod = 100;
my $maxAttempts = 20;
my $epsilon = 0.001;
my $infinity = 1000000.00;

These set of parameters can be left unchanged. Set *symbToUse* to 0 to use all the symbiont population (recommended in case first level doesn't have a MOEA), 1 to use only the unique symbionts on PF of the level0 MOEA, and 2 to use all symbionts on PF of the level0 MOEA (recommanded if first level uses a MOEA). *exe* determines the name of the executable file once the code is compiled. *seed* determines the first seed number, the last seed number is defined as the second argument of the command line instruction.
==

III. Run the S-ESC C++ code

Once the 4 required files (*AttributeWiseCentroids*, *AttributeWiseLabels*, *NoneligibleAttributes*, and *Noneligible1DCentroids*) are created by WEKA and MATLAB, we can go ahead and run the SESC C++ core.

1. To simplify passing parameters to SESC, all parameters are defined in an input argument file

ending with “.arg”. The perl script “*createInputs.pl*” contains all SESC parameters. The path to the data set as well as the required four files along with all other SESC parameters such as population size, generation size, mutation probabilities, etc are defined here. Once all parameters are set, run this script with the prefix of output filenames, number of runs (*N*) and number of runs per batch (*M*) as input arguments:

USAGE: *perl createInputs.pl filename_prefix numRuns numBatch*

This script creates *N* .arg files and a single “.run” file that has the script to run the *N* runs *M* by *M*.

2. Run the “.run” script:

USAGE: *nohup sh filename_prefix.run > filename_prefix.out &*

This will put all the runs in a queue and run them *M* by *M* until *N* is reached. Each run generates a “.std” output file with all the algorithm output in it, and an “.err” file if an error occurred.

Once all runs are done you will end up with *N* .std files that store the outputs of SESC.