A graph approach to find the best cluster-set for each possible number of clusters in a specified range

Cornel Barna[*] Haneen Dabain[†]

December 23, 2009

Abstract

In this paper, we propose new enhancements to the BBc and BBcH algorithms [1] in a way that would decrease the running time. We added a range limit on the number of clusters to reduce the size of the search space. we also added an error rate feature to BBcH algorithm to enhance the time performance by finding an approximate solution. The drawback of other clustering methods, like Ward's algorithm or k-Means, is that they do not find the optimum solution and provide no measure of how far from the optimum the solution is. So, by adding this error rate to the BBcH algorithm, we were able to get a solution with a predefined error rate.

Keywords: clustering; combinatorial algorithms; branch-and-bound.

1 Introduction

Given a set of n observations points, $X = \{x_1, \ldots, x_n\}$, a cluster-set can be defined as a partition of the indices $1, \ldots, n$ as follows:

$$\mathcal{C} = \{C_1, \ldots, C_g\}, \quad \text{where} \quad \begin{cases} 1 \le g \le n \\ C_i \subseteq \{1, \ldots, n\}, \quad i = 1, \ldots, g \\ C_i \cap C_j = \emptyset, \quad 1 \le i, j \le g, \quad i \ne j \\ C_1 \cup \cdots \cup C_g = \{1, \ldots, n\} \end{cases}$$

The aim of clustering is to find the best cluster set. I.e., to find the a partition $\mathcal{C} = \{C_1, \ldots, C_g\}$ such that each cluster C_i $(i = 1, \ldots, g)$ is as compact as possible. The number of clusters, g, is in most cases unknown, and thus, it is also to be determined. The measure of compactness of a group (or cluster) in general is problem dependent, and it has high impact on the quality of the solution. A widely used measure of compactness is the Error Sum of Squares (ESS). For a single cluster this is defined as:

$$ESS_{C_i} = \sum_{j \in C_i} \|x_j - \bar{x}_i\|^2, \qquad \bar{x}_i = \frac{1}{|C_i|} \sum_{j \in C_i} x_j,$$

[*]Department of Computer Science and Engineering, York University, Canada. E-mail: cornel@cse.yorku.ca.
[†]Department of Computer Science and Engineering, York University, Canada. E-mail: haneend@cse.yorku.ca.

while for a cluster set it becomes:

$$ESS_{\mathcal{C}} = \sum_{i=1}^{g} ESS_{C_i}.$$

Here, $\|\cdot\|$ and $|\cdot|$, denote the Euclidean norm and the cardinality, respectively. Note that the ESS measures the distance of each point in a cluster from its centroid, and does not take in account the distance between clusters. The clustering problem can be seen as a trade-off between fewer clusters versus less homogeneity. The latter can be expressed as:

$$\text{find} \quad g \quad \text{and} \quad \widehat{\mathcal{C}} = \underset{\mathcal{C}}{\operatorname{argmin}}\, ESS_{\mathcal{C}} \quad \text{s.t.} \quad |\mathcal{C}| = g. \tag{1}$$

In the next subsection, two other clustering algorithms are briefly presented. In section 2, we summarize the BBc algorithm [1]. In subsection 2.1 we propose another algorithm to solve the clustering problem when we are interested only on cluster-sets with the size in a specified range. We also propose some other versions of this algorithm. The experimental results and conclusions are presented in section 2.2.

Section 3 contains a description of our program and how it can be used.

1.1 Related work

An non-exact method to finding a good cluster set of n observations is the Ward algorithm [3]. It is a hierarchical agglomerative method and, from an algorithmically point of view, it is an n steps iterative greedy strategy. The algorithm starts with n clusters, each cluster having exactly one observation. Each step merges two clusters that yields the smallest increase in the ESS. The iterative process stops when a single cluster containing all the n observations is obtained. The solution is provided by an n-list of the cluster sets corresponding to the each number of possible groups. Algorithm 1 illustrates this procedure.

The drawback of the Ward method is that it does not find the optimum solution and provides no measure of how far the solution is from the optimum.

Algorithm 1: Ward's method for clustering n observations

$C_i \leftarrow \{i\}, \quad i = 1, \ldots, n \quad$ and let $\quad \mathcal{C} \leftarrow \{C_1, \ldots, C_n\}$
$g \leftarrow n$
while $g \geq 2$ **do**
  $\underset{1 \leq i,j \leq g}{\operatorname{argmin}}\, ESS_{[\mathcal{C}-\{C_i,C_j\}]\cup\{C_i \cup C_j\}} \quad \backslash\backslash \quad \binom{g}{2}$ possibilities
  $\mathcal{C} \leftarrow \left[\mathcal{C} - \{C_i, C_j\}\right] \cup \{C_i \cup C_j\}$
  $g \leftarrow g - 1$
end

Another popular method for clustering is k-Means [2]. In k-Means the algorithm receives the size k of the cluster-set as a parameter. The algorithm starts by selecting at random k observations to be the centers of the clusters. Then, for each observation, the distance to all centers is computed and the observation is assigned to the closest cluster. When all observations have been assigned to a cluster, a new center for each cluster is computed. Also the solution is evaluated (usually the ESS is used to measure the solution, but other functions can be used).

The algorithm will proceed to the next iteration, using the new computed centers for each cluster. The algorithm stops when a stopping criteria has been met (a maximum number of iterations has been achieved, the new solution is the same with the solution from the previous iteration, the improvement in the new solution is too small, etc.).

Algorithm 2: The k-Means algorithm for clustering n observations.

Input: k – the number of clusters in the cluster-set.
Output: A cluster-set with k clusters.

```
choose initial centers for clusters: c_1, c_2, ..., c_k;
while stopping condition not met do
    for i = 1, ..., n do
        assign the point with index i to the cluster whose center is the closest;
    end
    for j = 1, ..., k do
        calculate the new center for cluster j;
    end
end
```

In this paper we present an algorithm to find the best solution to the clustering problem when we are interested in cluster-sets with a size in a specified range. The new algorithm provides an increase in the solution quality compared with Ward's method or k-Means at the expense of execution time.

The k-Means has the same problem as the Ward's method – finding a suboptimal solution with no measure of the error from the optimum. Another drawback of k-Means is that it is highly dependant on the initial centers of clusters.

2 The Branch-and-Bound clustering

The Branch-and-Bound algorithm (BBc) [1] was presented to find the optimum solution to the clustering problem when the number of clusters is not specified. The solved problem can be written formally as:

$$\text{for all} \quad g = 1, \ldots, n \quad \text{find} \qquad \mathcal{C}_g = \underset{\mathcal{C}\text{ – cluster set}}{\operatorname{argmin}} \; ESS_{\mathcal{C}} \quad \text{s.t.} \quad |\mathcal{C}| = g. \tag{2}$$

The BBc algorithm organizes the searching space as a directed graph. A vertex represents a cluster-set and is denoted by V_1, V_2, V_3, ...The arc $A_{ij} = (V_i, V_j)$ exists and is directed from V_i to V_j if and only if V_j is obtained by merging some clusters in V_i. Each arc A_{ij} is associated a computational cost, i.e., the cost of (re-)computing ESS_{V_j} given ESS_{V_i}. The graph has a root vertex, without any incoming arcs and corresponds to the cluster set with each observation forming its own cluster. The remaining vertices are obtained by merging the clusters in all possible ways.

The graph is organized on levels, all vertices on a level represent cluster-sets of the same size.

Because the search space is very large, the BBc algorithm presents a strategy to traverse this graph such that the best cluster-set on each level is found without evaluating all possible cluster-sets. The strategy also guarantees that no cluster-set will be evaluated twice by finding a spanning tree of the graph. By providing an initial solution, the algorithm can cut larger sections from the graph and finds the optimum solution faster (this faster version of the BBc algorithm is called BBc with Hint or BBcH).

2.1 Branch-and-Bound clustering with Range

The BBc and BBcH are computationaly demanding because the searching space is an exponential function on the size of the input. Because we are not interested in all cluster-sets with sizes from 1 to n, where n is the number of observations to be clustered, we modify the BBc algorithm to find the optimum solution when the size of the cluster-set is in a given range. The new problem can be formulated as:

$$\text{for all} \quad g = g_1, \ldots, g_2 \quad \text{find} \qquad \mathcal{C}_g = \underset{\mathcal{C}\text{ – cluster set}}{\operatorname{argmin}} \; ESS_{\mathcal{C}} \quad \text{s.t.} \quad |\mathcal{C}| = g, \tag{3}$$

where $[g_1,\ g_2]$ is the range we are interested in, and $1 \leq g_1 \leq g_2 \leq n$. Note that if $g_1 = 1$ and $g_2 = n$ the problem is identical with the one solved by BBc.

The algorithm we are proposing is called BBc with Range (BBcR) and is illustrated by Algorithm 3. It exploits the graph monotonicity property (just like BBc)

$$\exists\, A_{i,j} \quad \Longrightarrow \quad ESS_{V_i} \leq ESS_{V_j} \quad , \tag{4}$$

that is, merging clusters increases the ESS of the resulting cluster set (where V_i and V_j are vertices in the support graph and $A_{i,j}$ is an arc between them). That is, when traversing the graph from root node following the arcs, the ESS of the vertices on the path will increase. The objective of the algorithm is to minimize the ESS. Therefore, property 4 will be used to prune parts of the graph when the vertices on those parts cannot improve the current solution. The algorithm will use a look-up table, $\rho_1, \cdots, \rho_n$ to store the ESS of the current solution corresponding to each number of clusters.

Algorithm 3: BBcRange(g_1, g_2) – Branch-and-Bound algorithm to find the best cluster sets with the size in the specified range.

Input: $[g_1, g_2]$ – the range for the cluster size.
Output: ρ – a vector with the best cluster-sets with sizes in the specified range.

1 $C_i \leftarrow \{i\}, \quad i = 1, \ldots, n \quad$ and let $\quad \mathcal{C} \leftarrow \{C_1, \ldots, C_n\}$;
2 initialize the ESS table: $\rho_i = \infty, \quad i = 1, \ldots, n$;
3 $minClusterSetSize = g_1$;
4 $maxClusterSetSize = g_2$;
5 ProcessClusterSet($\mathcal{C}$, 1);
6 **return** $\rho_{g_1,\ldots,g_2}$;

The BBcR algorithm uses the same support graph and the same spanning tree as BBc, but we cut more by limiting the number of children a node can have. Also the cutting test is modified, thus, only the levels we are interested in are taken in consideration.

As in BBc, our algorithm associates a value k to each vertex, marking a position in the cluster-set. In the subtree with the root node (V, k), no two clusters that have an index less than k can be joind. This way we make sure that we do not visit a cluster-set more than once. Also, this k will give us the size of the cluster-set on the last level of the subtree. We will use this in the cutting test on line 12 of procedure *ProcessClusterSet*, where i is the marked position in the child vertex. However, if the last level of the subtree is outside the specified range for the algorithm, we will compare with the best solution for the last level in the range ($ESS_{\mathcal{C}'} < \rho_{\max\{minClusterSize,\ i\}}$).

The second part of the cutting test is on line 10: if we are already on the last level of the allowed range, we don't continue further in this subtree.

Procedure `ProcessClusterSet(`$\mathcal{C}, k$`)` – procedure used to process a cluster set.

Input: $\mathcal{C}$ – a cluster-set to be investigated;
Input: k – a marked position in the cluster set.

```
1  for i = k + 1, ..., min{|C|, maxClusterSetSize + 1} do
2      for j = 1, ..., i − 1 do
3          create C' from C by merging clusters C_i and C_j and keeping the rest;
4          compute ESS_C';
6          if ESS_C' < ρ_|C'| then
7              ρ_|C'| = ESS_C';
8          end
10         if |C'| > minClusterSize then
12             if ESS_C' < ρ_max{minClusterSize, i} then
13                 ProcessClusterSet(C', i - 1);
14             end
15         end
16     end
17 end
```

Lets consider the example where the input set has 4 observations. The tree built by BBc is shown in Figure 1. This is also the support tree for BBcR. If we set $[1, n]$ as the range, the algorithm will visit the entire tree. But if we set the range to be $[2, 2]$ (that is, get the best cluster-set with 2 clusters), then nodes with a rectangle shape will be cut without having to evaluate any node (the nodes with 3 clusters will be cut by condition from line 1 and the node with one cluster will be cut by condition from line 10). These 2 conditions are dependent only on the apecified range and act as prepruning conditions, whilst the condition on line 12 will cut parts from the tree depennding on the ESS value.

The BBcR algorithm can be improved by initializing the look-up table ρ with an approximate solution. In our experiments, we used the solution provided by Ward's method.

We also tried to improve the time performance of the algorithm by allowing approximate solutions. With a modification in the cutting test we can run the algorithm to get a solution at most $e\%$ from the optimum (where e can be given as a parameter). The cutting test from line 12 will become

$$(1 + e) \times ESS_{\mathcal{C}'} < \rho_{\max\{minClusterSize,\, i\}} \tag{5}$$

2.2 Experimental results and conclusions

The algorithms have been implemented using C++. The tests were run on a machine with Intel Core 2 CPU, quad core, with 3 GB RAM, on a Windows environment. Artificial data was used in the experiments.

The results are shown in the table below, each row in a table represents the average on 50 different datasets.

Table 1 shows the execution times of the algorithms compared with BBcH. As expected, by initializing the look-up table with some solution (solution provided by Ward's method) the algorithms become significantly faster.

In Table 2 we see that, by allowing some error, the algorithm can be sped up even further. For example,

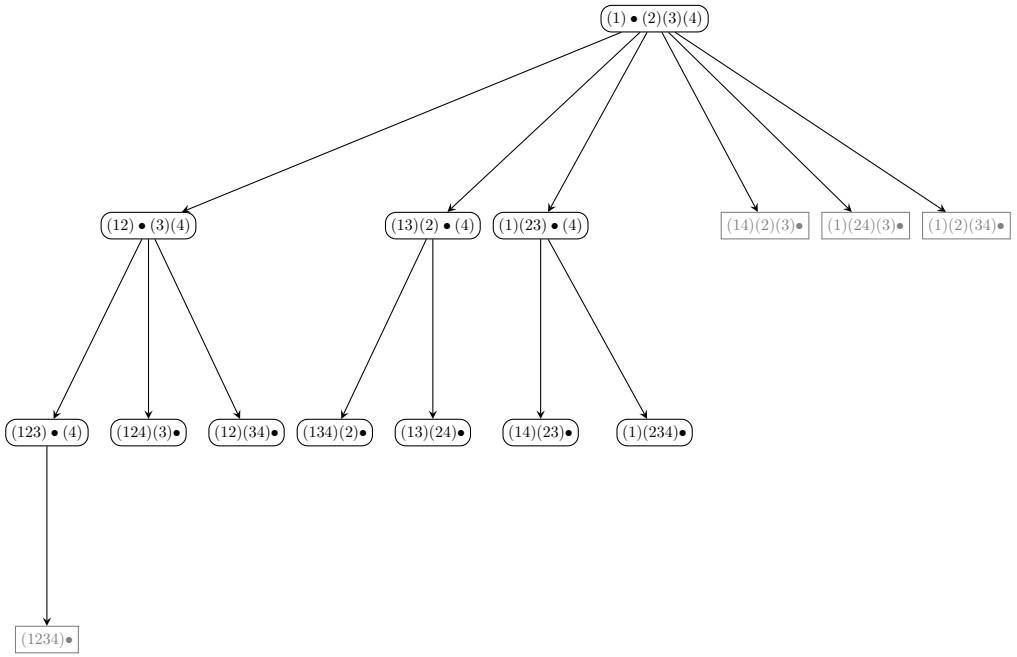

Figure 1: The tree that is traversed by algorithm while searching for the best cluster-set ($n = 4$). The position of the bullet (•) represents the value k associated with each node. The algorithm will execute a *depth first search* in this tree.

allowing an error of 10% when we have 40 observations and the range $[9, 11]$, the algorithm (BBcRH10%) gets an increase of more than 36% in the execution speed compared with BBcRH.

Considering that the other clustering methods, like k-Means or Ward's algorithm, that are very fast, can get a solution with a big error, providing no information about the error rate, we conclude that BBcR (BBcRH) can be a very useful clustering method when the number of observations is not that big.

In our tests, k-Means found solutions with very big errors, even 50%. It seems that k-Means performs well if the clusters are well defined (compact, and far from each other), but gets a poor solution if not. Also we noticed that if one cluster is very small (compared with the number of observations) it is very likely that k-Means will fail to find it. This is because k-Means is sensitive to the initial cluster centers, and the probability to choose a center in the small cluster is low.

Our algorithms don't have this problems – the optimum solution will always be found. But this come at the expense of execution time. Even if we cut very big parts of the graph, the searching space is very big [1]. For example, for $n = 40$ the searching space has more than 10^{40} possible cluster-sets. Our algorithm find the optimum solution by evaluating circa 10^{10}. Although the number of evaluated candidates is very small compared with the searching space, 10^{10} is still a big number. Also the algorithm has an exponential complexity.

We think that this complexity can be further reduced, and this can be done with further research on the problem. We have tried to arrange the observations such as when we generate the children of a node, we will generate first the cluster-set with the smallest ESS (or the largest ESS). We named these algorithms $BBcR^+$ – generate and visit the children in an *increasing* order according to ESS – and $BBcR^-$ – generate and visit

n	range	Time (sec.)					Time improvement over BBcH (%)			
		BBcH	BBcR	BBcRH	BBcRHE 5%	BBcRHE 10%	1 - $\frac{BBcR}{BBcH}$	1 - $\frac{BBcRH}{BBcH}$	1 - $\frac{BBcRHE5\%}{BBcH}$	1 - $\frac{BBcRHE10\%}{BBcH}$
	3-5	.003	.005	.001	.001	.001	-97.51	52.65	57.50	61.87
15	6-8	.005	.005	.001	.001	.001	-14.35	67.03	68.87	70.77
	11-13	.004	.0004	.0001	.0001	.00009	90.79	97.56	97.81	97.96
	4-6	.044	.051	.018	.016	.013	-15.80	58.12	62.95	69.73
20	9-11	.048	.038	.007	.006	.005	20.05	85.25	87.31	89.46
	14-16	.030	.005	.0007	.0006	.0005	83.69	97.58	97.93	98.14
	5-7	.422	.427	.114	.096	.084	-1.22	72.90	77.09	79.92
25	11-13	.612	.315	.096	.075	.069	48.47	84.19	87.66	88.59
	17-19	.676	.023	.004	.003	.003	96.50	99.40	99.48	99.55
	7-9	6.091	5.300	1.877	1.583	1.354	12.99	69.18	74.01	77.76
30	14-16	8.111	2.407	.559	.537	.432	70.32	93.11	93.37	94.67
	22-24	13.441	.041	.004	.003	.003	99.69	99.97	99.97	99.97
	8-10	68	46.664	22.802	18.447	13.260	31.79	66.67	73.04	80.62
35	17-19	53	11.031	1.851	1.506	1.217	79.26	96.52	97.17	97.71
	26-28	45	0.142	0.016	.013	.011	99.69	99.96	99.97	99.97
	9-11	583	619	113	96	71	-6.24	80.61	83.46	87.71
40	19-21	1 003	166	27	20	15	83.42	97.24	97.91	98.48
	29-31	448	958	.107	.093	.075	99.79	99.98	99.98	99.98

Algorithms:		
	BBcH	Branch-and-Bound clustering with Hint
	BBcR	Branch-and-Bound clustering with Range
	BBcRH	Branch-and-Bound clustering with Range and Hint
	BBcRHE x%	Branch-and-Bound clustering with Range, Hint and the solution is at most x% from the optimum

Table 1: Execution times of the algorithms and the time improvement over BBcH. BBcR and BBcRH will find the optimum solution. The execution times are shown with 3 significant digits when the time was less than a second.

the children in a *decreasing* order according to ESS.

We found out that the increase in time performance can be sometimes more than 90% over BBcHR (with one or both algorithms: $BBcR^+$, $BBcR^-$) and some other times we noticed a decrease in performance of more than 10 times. Our conclusion is that the order of the observations is important and can be exploited to improve the algorithm.

Research is required to find what are the properties of the input data that would allow as to reorder the observations so we can cut more and sooner from the graph.

3 Implementation

The program is written in C++ (using Microsoft Visual Studio 2008) and can be compiled only in Windows (We have implemented a performance timer to get the execution times of the algorithms. Because the standard C++ functions to get time are not precise enough, we used some functions from WinAPI to measure time with a precision of microseconds. Because of the use of this WinAPI, the code cannot be compiled on Linux. If this functionality is not required, then can be removed and the code will compile in other operating systems.)

n	range	Time (sec.)			Time improvement over BBcRH (%)	
		BBcRH	BBcRHE 5%	BBcRHE 10%	1 - $\frac{BBcRHE5\%}{BBcRH}$	1 - $\frac{BBcRHE10\%}{BBcRH}$
	3-5	.001	.001	.001	10.25	19.47
15	6-8	.001	.001	.001	5.59	11.34
	11-13	.0001	.0001	.00009	10.28	16.70
	4-6	.018	.016	.013	11.54	27.72
20	9-11	.007	.006	.005	13.39	28.56
	14-16	.0007	.0006	.0005	14.75	23.05
	5-7	.114	.096	.084	15.47	25.93
25	11-13	.096	.075	.069	21.96	27.84
	17-19	.004	.003	.003	13.06	25.02
	7-9	1.877	1.583	1.354	15.67	27.84
30	14-16	.559	.537	.432	3.81	22.65
	22-24	.004	.003	.003	17.74	19.99
	8-10	22.802	18.447	13.260	19.10	41.85
35	17-19	1.851	1.506	1.217	18.61	34.27
	26-28	0.016	.013	.011	19.65	30.53
	9-11	113	96	71	14.71	36.61
40	19-21	27	20	15	24.44	44.93
	29-31	.107	.093	.075	13.13	29.81

Algorithms:		
	BBcRH	Branch-and-Bound clustering with Range and Hint
	BBcRHE e%	Branch-and-Bound clustering with Range, Hint and the solution is at most e% from the optimum

Table 2: Execution times and the time improvement of the algorithms over BBcRH when we allow some error in the solution. BBcRH will find the optimum solution.

The implementation was focused on execution speed, therefore we eliminated all the unnecessary operations, created custom memory alocators, we used pointers and `memcpy` (is hardware implemented and is fast) to manipulate clusters and cluster-sets. The speed of the program can be *significantly* increased by using some other programming techniques (like memory caching and optimizations in memory manipulations for the recursive model of the algorithm).

The program does not check for bad user input (like incorrect setting of parameters), since it was designed to test the performance of an algorithm and not to be used for clustering. If there is no error while setting the parameters for algorithms, the program han handle very large datasets (is limited only by the limitations of the operating system), but the algorithm cannot (because of the exponential time complexity).

To run the algorithms, the code in file `program.cpp` needs to be modified. Then the program has to be compiled and run.

The modifications that need to be done are as following:

- `#define VAR_CNT 40` – sets the number of observations that need to be clustered (40 in this case). The program will generate at random a vector of numbers of the specified size, and that will be the input for the algorithms.
- Set the options for the algorithm to run.
  - To set a hint (we are using the Ward's method to get the hint):

```
theOptions->SetSolutionHint (resultsWards.GetSolutions());
theOptions->SetUseSolutionHint(true);
```

– To use the range version of the algorithm:

```
theOptions->SetUseRange(true);
theOptions->SetRange(11, 13); // find cluster-sets of size 11 to 13.
```

– To allow some error in the solution (the parameter must be a number between 0 and 1; only works with the range version of the algorithm):

```
theOptions->SetErrorRate(0.1); // accept an error of 10%
```

To run the BBc algorithm using a Hint, accepting an error of 10% to find a solution in range [11, 13], we need to write the folloding lines of code.

```
theOptions->Reset();
theOptions->SetUseRange(true);
theOptions->SetSolutionHint (resultsWards.GetSolutions());
theOptions->SetUseSolutionHint(true);
theOptions->SetErrorRate(.1);
theOptions->SetRange(11, 13);

CResults resultsBbcRHE10 = CBbcAlgorithm::Run(theInstance, theOptions);
resultsBbcRHE10.SetAlgName("BBc R H E 10");

results.push_back(&resultsBbcRHE10);
```

The results, stored in an object of type `CResults`, will be added to the vector `results` to be printed in a file. *If the results of an algorithm are not added to this vector the program won't output anythig for that algorithm.*

The `main` function contains commented code with sample for how to run each version of the algorithm.

When the evaluation finishes, results files are created containing all the information and observed measurements:

- `complete.txt` – will contain tables with the ESS of the solutions found by all algorithms. The first column of the table is the size of the cluster-set, the other columns contain the ESS of the solution, for the specified cluster-size, found by an algorithm. If the algorithm was given a name, the name will appear as a header of the table. The tables in this file are easy to import in Excel for further processing.

  Below is a snippet from this file.

```
            Ward        BBc R H    BBc R H E 5   BBc R H E 10
11       19399.04       17162.38       17162.38       17304.36
12       15927.61       14511.58       14511.58       14653.56
```

In the above example, we see that the BBcRH algorithm (BBc using Range and Hint) found a solution with 11 clusters, and the ESS of that solution is 17162.38. For the same cluster-set size we see that the ESS of the solution found by BBcRHE10% (BBc using Range, Hint and accepting an error of 10%) is 17304.36.

The last line of the table contains the execution times for each algorithm (in microseconds).

The results are appended to the file, so if we have more than one execution of the algorithms, then this file will contain a table for each execution.

- `solutions (run = 1).txt` – will have the complete solutions found by all algorithms used in a single run (`run = 1` in the filename, means that the results corespond to the first run of the algorithms).

  The name of each algorithm – if specified – is printed,

  For each algorithm is printed the name (if a name was specified), following is the input vector (the observations to be classified), the list of the solutions containing the size of the cluster-set, the ESS of the solution and the cluster-set (indices are used to specify the observations in clusters). Then the number of evaluated cluster-sets and execution time are displayed (i.e. `Computed 2240706439 nodes in 1,380,068,486` means that 2,240,706,439 cluster-sets were evaluated in 1,380,068,486 microseconds).

The entire code in function `main` is put in a `for` loop. This loop is used to have multiple runs of the algorithms. (We make a distinction between a run of the algorithms and a run of the program – in a single run of the program the algorithms can run multiple times and this loop controls how many times.)

In each run the information is appended to the output files, therefore it is recommended to delete the output files *before* running the program.

References

[1] Cornel Barna, Cristian Gatu, and Erricos J. Kontoghiorghes. A graph approach to find the best grouping for each possible number of clusters. 2009.

[2] Nargess Memarsadeghi and Dianne P. O'Leary. Classified information: The data clustering problem. *Computing in Science and Engg.*, 5(5):54–60, 2003.

[3] Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.