- **Faculdade de Engenharia da Universidade do Porto**

FEUP

Solving Unstructured Classification Problems with Multicriteria Decision Aiding

Rui Pedro Rodrigues Sebastião

VERSÃO FINAL

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Prof. Dr. José Soeiro Ferreira
Co-orientador: Dra. Iryna Yevseyeva

29/07/2011

Abstract

In the framework of multicriteria decision aiding, a lot of interest has been paid to the assignment of alternatives to predefined categories, i.e. ordered groups of alternatives. This will be referred to the multicriteria classification or sorting problem. On the other hand, similar problem with no information about classes, called ordered clustering problem received less attention. In this work we will formalize our problem as an optimization problem and we will propose a related approach, inspired by split and merge processes. Our approach will be tested on artificial datasets as well as on an example related to the field of business failure risk.

- *Keywords: Multicriteria decision aid, ordered clustering problems, outranking methods*

Acknowledgment

I want to direct my first word of thanks to my supervisor, Dr. Iryna Yevseyeva for all the availability to hear me, for help me in times of doubt and uncertain and for the awakening of new ideas, that without them, this work would not be possible.

To my parents, thank you very much for always trusted in me and supported me in my decisions, for always instilled in me a sense of honesty, work and humility and for always been present in every moment of my life.

I want to thank to my friends, especially to David, who always said words of encouragement, and always have been present in the moments of greatest difficulty.

Finally, I want to thank INESC Porto and University of Porto, for the availability of resources and for hosting me.

Contents

List of figures

List of tables

Abbreviations

List of abbreviations (in alphabetic order)

DM	Decision maker
Electre	Elimination Et Choix Traduisant la Realité (Elimination and choice expressing reality)
MC	Multicriteria
MCDA	Multicriteria decision aid
MCOC	Multicriteria ordered clustering
Promethee	Preference Ranking Organization Method for Enrichment Evaluations

Chapter 1

Introduction

The goal of grouping “similar” objects into homogeneous clusters is commonly encountered in different fields such as the finance sector, the medical sector, in the agriculture, in marketing, in image processing, etc. For example based on new unknown earlier symptoms, the patients may be assigned to groups according to the intensity of the symptoms to a new type of disease. As a consequence, it has been extensively studied in this literature.

Generally, two distinct problems can be considered. First one aims to attribute alternatives to groups that are unknown a priori. In this case, the problem is referred to a clustering problem and the groups are so-called ‘clusters’.

On the other hand, once the groups are defined a priori, with a central element or boundary alternatives, the problem of assigning an object to one of these groups is referred to a classification problem and these groups are so-called ‘classes’.

Traditional clustering uses attributes, which means that there is no preference order on scale of attributes and no order between clusters [30],[31]. On the other hand, multicriteria clustering uses at least criteria. In a special case of MC clustering, multicriteria ordered clustering (MCOC), there is also order on clusters [2],[40].

Multicriteria Decision Aid (MCDA) is a discipline aimed at supporting decision makers faced with making numerous and sometimes conflicting evaluations. MCDA aims at highlighting these conflicts and deriving a way to come to a compromise in a transparent process. Multicriteria decision aid will permit us to have another insight into these problems. Moreover, we will see that the groups can be ordered or not. In the context of multicriteria decision aid, some authors [5] have been interested in assigning objects to ordered groups.

The study of complex decision problems has been a subject of research and has a long history back to ancient philosophers. The multicriteria decision aiding support can be characterized as a set of methods that seek to clarify a problem in which alternatives are assessed by multiple criteria, which in most cases are conflicting [23],[32]. According to Marins and Cozendy [26], this approach does not present an ideal solution for problems, but among all possible, the most consistent with the scale of values and the method used.

The Decision making is a part of every day, it is present in many activities developed by man. Naturally, people face situations that require from them some kind of decision. In these situations multiple alternatives are presented and out of these, the decision that best satisfies the goal(s) in question should be selected.

Some authors claim that to decide, is to position yourself into the future. Gomes Araya and Carignano [23] define the decision making as the process of gathering information attaching importance, then searching for possible alternative solutions and then making a choice between alternatives. The process of decision making can be seen as simple tasks faced by humans. For example we may sort “to-do” tasks for the next day into three classes, such as “must do”, “wish to do” and “can do”.

However, there are many complex issues to be solved by people. For example, the choice of a country to go on vacation or a house to be bought. In focuses of this thesis there are complex problems, with alternatives evaluated on two or more conflicting criteria. Another source of difficulty when making decisions is that they must meet multiple objectives and their impacts cannot be clearly identified[33].

The aim of this work is to propose a method that helps a decision maker to obtain what we call “ordered clusters” (i.e. ordered groups of alternatives). This will be referred to as the “multicriteria ordered clustering” problem.

The structure of our work will be as follows: In chapter 2 basic definitions and survey of literature are given. Chapter 3 presents two outranking methods (Electre and Promethee) with examples selected here for more detailed study. Chapter 4 and 5 describe multicriteria decision aid methods, one classification method and two clustering methods in chapter 4 and one clustering method proposed by us in chapter 5. In chapter 6 the experimental results are discussed and proposed approach is compared to another clustering/classification methods. Chapter 7 presents the conclusion and future work.

Chapter 2

Basic definitions and survey of literature

2.1 Alternative

The identification of an alternative is a procedure that belongs obviously to the beginning of the process, as well as the verification of its feasibility. In many cases, the identification is immediate, but there are situations where it is essential to a priori process the alternatives, which can be quite complex. On the other hand, there is a class of problems where the alternatives are only implicitly defined as a combination of values of decision variables, respecting a set of constraints (equations and inequalities) that define feasibility [29].

From the standpoint of terminology, it is important to clarify that the term "alternative" is used here as synonymous with "option", "hypothesis", "possible solution" or " potential action”. From the formal point of view, each alternative in addition to its name will be characterized by criterion.

2.2 Criteria

The definition of evaluation criterion is a crucial point of the decision process, because it corresponds to the identification of aspects or points of view relevant to be taken into account for determining the preference of one alternative over another. A coherent family of criteria should be [29]:

-Exhaustive: all relevant points of view should be included.

-Consistent: If two alternatives A and B are equivalent except for one criterion k, and if this criterion a_k is better than b_k, then A should be considered globally at least as good as B.

-Non-redundant: if we eliminate one of the criteria, the above conditions are no longer satisfied.

In addition, it is desirable to provide the following additional properties:

- Readability: the number of criteria must be relatively low.
- Operability: the family of criteria must be accepted by interested decision makers.

Once is identified a coherent family of criteria, it is necessary to advance in the operation, setting the units in which criterion is measured and associated scale. There can be simple economic criteria, such as cost evaluated in euro, or more complicated criteria associated with concepts such as quality, risk, environmental or social impact, etc. Another way do defining criterion scale is with categories that correspond to an overall assessment of the degree of satisfaction of these criteria (for example, satisfaction of a criterion associated with the quality "Very High, High, Medium, Low, and Very Low"). In this case, the degrees should clearly characterize the aspects to take into account and the situations that correspond to each category to reduce the subjectivity of judging.

At this point, appears to us the concept of decision maker (DM), which is central to these problems. This is the person (sometimes a representative of an entity) responsible for final decision. On the one hand, the decision maker defines and specifies the criteria to consider, possibly with the support of experts. On the other hand, it is not possible to carry out the decision process without it incorporate the preferences of the decision maker.

2.3 Classification problem

Assigning a set of alternatives evaluated on a set of criteria into predefined classes is a problem that a decision maker faces many times in real life. Classification problems are commonly encountered in various application fields such as health care, finance, marketing, etc. In classification problem (also known as supervised learning problem), the classes are predefined and well-described, on the other hand in clustering problem (also known as unsupervised learning problem) there is no a priori information about classes [3]. One good example of this kind of classification problem is the medical diagnosis where a patient has to be assigned to a known pathology-class on the basis of a set of symptoms. To solve this problem, we have some good procedures such as k-nearest neighbor algorithm or the bayes classifier.

A central concept of the classification problem is a class. The class is a collection of alternatives that are more similar to each other than the alternatives in neighbor classes. When we deal with different classification methods the similarity measure between two alternatives and rules of assignment are subjects of discussion [3].There are two types of classification problems: nominal and ordinal. In a nominal classification problem, the classes are not ordered. In ordinal classification problem the classes are ordered according to some quality.

2.4. Clustering problem

There are situations where there may be no information about the groups and the purpose is then to extract a structure in the data set. For example, we can consider a marketing problem where the aim is to discover similar customer behaviors in the retail industry. The most common traditional clustering procedures are the k-means, hierarchical or finite mixture densities algorithms [9],[35],[36]. Multicriteria methods have also been extended to clustering problems with order between classes. For instance an Electre-like clustering procedure based on the L-values Kernels was proposed in [10].

2.5 Taxonomy of clustering procedures

In Figure 2.1 [2], we can see a summary of the clustering procedures. First criteria dependency is used to distinguish between classical clustering and multicriteria clustering. Then we can separate multicriteria clustering in two other different methods, non-relational multicriteria clustering and relational multicriteria clustering.

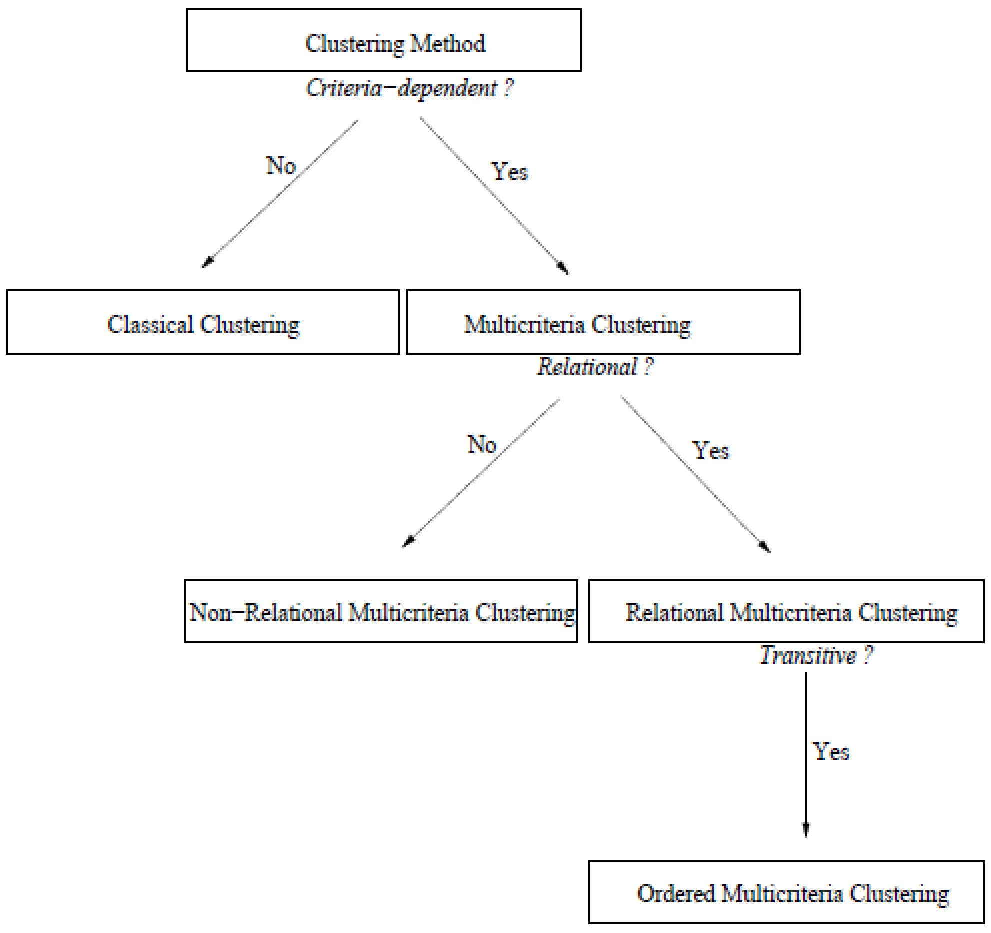


Figure 2.1 - Taxonomy of clustering procedures.

2.5.1 Criteria dependency

Some clustering procedures [1], [11] have been proposed in the multicriteria decision aid domain. These procedures use criteria instead of attributes that are commonly applied in traditional clustering methods. A criterion is an attribute that contains including additional information about direction for its values on the set of considered alternative. For example, a "price" is an attribute, but for a seller, a "sell price" is a criterion because it has the additional information that it should be maximized and for a buyer, a "buy price" is also a criterion and have the extra information that this value should be minimize.

2.5.2 Relational multicriteria clustering

A multicriteria clustering procedure is a criteria dependent procedure. The presence of a relation between clusters one of the points of interest in the criteria-dependant clustering procedures. Classical clustering procedures typically do not propose a preferential relation between the obtained clusters because they are not criteria-dependant.

The use of the preference criteria for solving classical clustering problems with no order between the clusters induces a loss of information and may be criticized, so a strategy is applied for the relational multicriteria clustering usually in this type of procedures. Firstly, to obtain the centroid that characterizes each cluster, a classical clustering algorithm is used. After that any multicriteria pair wise comparison procedure can be applied to the centroids in order to come up with “at least as good” relation on the clusters.

2.5.3 Ordered multicriteria clustering

This clustering procedure is a special case of relational multicriteria clustering. Ordered multicriteria clustering procedures have advantage over relational multicriteria clustering because they have a transitivity propriety that unambiguously implies an order on the clusters.

Ordering clusters can be useful when some hierarchy has to be discovered in the data. For example, we can consider a problem where the employees’ performance is being evaluated. Depending on the data, three clusters can be created: above average, average and below average performance.

Usually, his procedure combines the ideas of both the clustering and the ranking problematic. First, the data is clustered, and then the centroids are ranked using a multicriteria ranking procedure. Conversely, we can apply a ranking procedure on the data, and then an ordered partition compatible with that ranking can be built, effectively merging some alternatives in the cluster of the same rank.

2.6 Clustering problem vs ranking problem

The problem of ranking is closely related to the problem of ordered clustering. The ranking problem consists in partition the set of alternatives into partially or totally ordered classes with number of clusters close to the number of alternatives. The ordered multicriteria clustering procedure can be considered as a particular case of ranking problem. In fact, an ordered multicriteria clustering procedure, which partitions the alternatives into ordered classes, can very naturally be seen as just another rank procedure. Despite some similarities between these two problematic let us insist on some fundamental differences.

A rank procedure aims at discriminating the different alternatives, so this procedure tends to maximize the number of classes. The preferred case in a ranking procedure is to build a linear order whenever possible. In this case the number of alternatives and classes are almost the same.

On the other hand, a clustering procedure tends at discriminating different alternatives but at the same time tries to group similar alternatives. The first objective tends to maximize the number of classes and the second tries to minimize it, so the clustering solution can usually be seen as a compromise between these two objectives. The most common solution is to set a priori the number of clusters that is desired.

Chapter 3

Outranking methods

Bernard Roy [8], was the first to define the outranking relation as follows: *An outranking relation is a binary relation, defined on the set A | ai S aj if, given the information about the decision maker's preference, the evaluations of these actions and the nature of the problem, there is enough arguments to admit that action ai is at least as good as action aj , while there is no argument to deny this consideration.*

Some methods have been developed utilizing this idea for different decision making problems, such as selection of the best alternative(s), ranking and classification methods. In the next section we will present the two most famous approaches that utilizing outranking relation.

3.1 ELECTRE III method

Bernard Roy can be considered as the father of the family of the Electre methods, exploiting an outranking relation. The estimation of the outranking relations between pairs of alternatives is the basis for all methods from the ELECTRE family.

For the calculation of the outranking index, the decision maker needs to define a set of alternatives and a set of criteria on which these alternatives are evaluated. In addition, the Electre III [13],[19],[15],[38],[39] method requires the following information for each criterion g_k: indifference q_k, preference p_k and veto v_k thresholds, and weight w_k in addition cutting level λ parameter has to be predefined.

The indifference threshold q_k is the largest difference between two alternatives on the criterion g_k such that they remain indifferent to the decision maker. The preference threshold p_j defines the smallest difference between two alternatives such that on alternative is preferred to the other one on the criterion $g_{k.}$ The veto threshold v_k indicates the smallest difference between two alternatives on the same criterion g_k that shows incomparability of these two alternatives. The relation between the thresholds must be $v_k>p_k>q_k$. The weight w_k of a criterion g_k indicates the relative importance of each criterion. The cutting level λ shows the smallest value of the outranking index that is sufficient for considering and outranking situation between two alternatives.

The two conditions (concordance and discordance) are used to verify the outranking relation of this method. On one hand, the concordance condition requires that for the majority of criteria the alternative a_i is preferred over a_j, on the other hand the discordance demands the lack of strong opposition to the first condition in the minority of criteria. The partial indices are computed for each condition: concordance $C_k(a_i,a_j)$, and discordance $D_k(a_i,a_j)$. They allow to calculate the outranking index $S_k(a_i,a_j)$.

3.1.1 Electre III algorithm:

First, the partial concordance indices $C_k(a_i,a_j)$ are calculated for each criterion g_k. The criterion g_k has an increasing direction of preference because a maximization problem is under consideration.

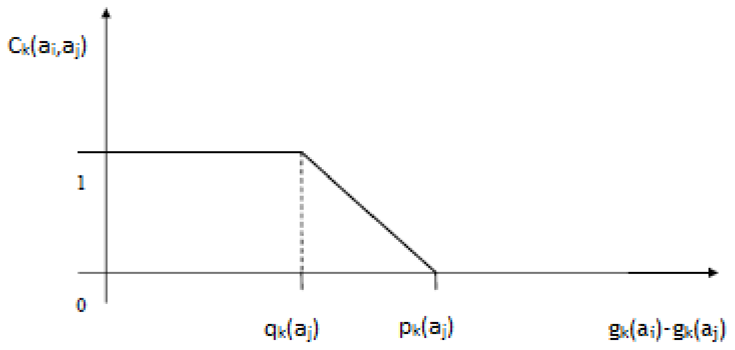


Figure 3.1 - Electre III partial concordance indices $C_k(a_i,a_j)$.

As we can see in Figure 3.1, the concordance indices $C_k(a_i,a_j)$ are calculated as follows:

$$C_k(a_i,a_j)=\begin{cases} 0, if\ g_k(a_j)-g_k(a_i)\geq p_k(a_j) \\ 1, if\ g_k(a_j)-g_k(a_i)<q_k(a_j) \\ \dfrac{p_k(a_j)-g_k(a_j)+g_k(a_i)}{p_k(a_i)-q_k(a_i)}, \\ if\ g_k(a_j)-p_k(a_j)<g_k(a_i)\leq g_k(a_j)-q_k(a_j). \end{cases}$$

At the next step the overall concordance index C(a_i,a_j) is defined as an aggregation of partial concordance indices, where n is the number of criteria:

$$C(a_i, a_j) = \frac{\sum_{k=1}^{n} w_k\, C_k(a_i, a_j)}{\sum_{k=1}^{n} w_k},$$

In the third step, the partial discordance indices D_k(a_i,a_j) are calculated for each criterion g_k according to the increasing direction of preference. If there is no information about the veto threshold, D_k(a_i,a_j)=0.bg g_k(a_i)-g_k(a_j)

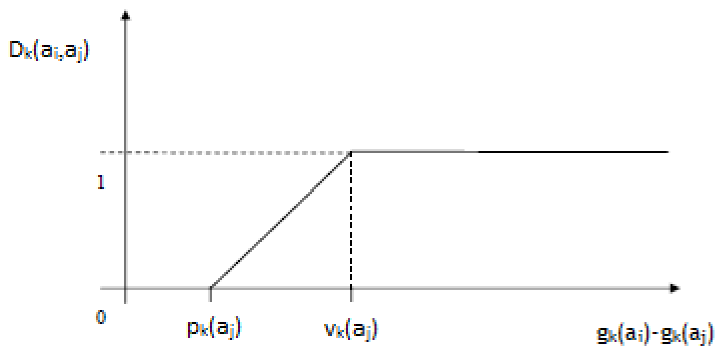


Figure 3.2 -ELECTRE III partial discordance indices D_k(a_i,a_j).

As we can see in Figure 3.2, the discordance indices D_k(a_i,a_j) are calculated as follows:

$$\mathrm{D_k}(a_i, a_j) = \begin{cases} 0, if\ g_k(a_j) - g_k(a_i) < p_k(a_j) \\ 1, if\ g_k(a_j) - g_k(a_j) \geq v_k(a_j) \\ \dfrac{g_k(a_j) - g_k(a_i) - p_k(a_j)}{v_k(a_j) - p_k(a_j)}, \\ if\ g_k(a_j) - v_k(a_j) < g_k(a_i) \leq g_k(a_j) - p_k(a_j) \end{cases}$$

In this step the outranking index S(a_i,a_j) can be calculate that shows outranking credibility of a_i over a_j assuming S(a_i,a_j) ∈ [0,1] as follows:

$$S(a_i, a_j) = C(a_i, a_j) * T_k(a_i, a_j),$$

Where k=1,…,n and $T_k(a_i, a_j) = \begin{cases} \prod_{k=1}^{n} \frac{1-D_k(a_i,a_j)}{1-C_k(a_i,a_j)}, & if\ D_k(a_i, a_j) > C_k(a_i, a_j) \\ 1 & , if\ D_k(a_i, a_j) \leq C_k(a_i, a_j) \end{cases}.$

In the final step the decision maker defines the value of the cutting level λ. Usually, the cutting level belong to the interval [0.5,1]. The minimum value of outranking indices accepted for outranking of one alternative over the other one is defined by this level. The cutting level is compared with the value of the outranking index. Based on this comparison, the preference situation between two alternatives is specified [19]:

If $S(a_i,a_j) \geq \lambda$ and $S(a_j,a_i) \geq \lambda$, then the alternative a_i and a_j are indifferent ($a_i I a_j$).

If $S(a_i,a_j) \geq \lambda$ and $S(a_j,a_i) < \lambda$, then the alternative x_i is strongly or weakly preferred to the alternative a_j ($a_i P a_j$ or $a_i J a_j$).

If $S(a_i,a_j) < \lambda$ and $S(a_j,a_i) \geq \lambda$, then the alternative a_j is strongly or weakly preferred to the alternative a_i ($a_j P a_i$ or $a_j J a_i$).

If $S(a_i,a_j) < \lambda$ and $S(a_j,a_i) < \lambda$, then the alternatives a_i and a_j are incomparable ($a_i J a_j$).

3.1.2 Example:

We will demonstrate an example of outranking indices for the data set with the following six alternatives evaluated on five criteria with their weights as basis to build the outranking index.

Table 3.1- Data set and weights for ELECTRE III example

Alternatives a_i	g_1	g_2	g_3	g_4	g_5
a_1	0.188	0.172	0.168	0.122	0.114
a_2	0.125	0.069	0.188	0.244	0.205
a_3	0.156	0.241	0.134	0.22	0.136
a_4	0.188	0.034	0.174	0.146	0.159
a_5	0.188	0.276	0.156	0.171	0.205
a_6	0.156	0.207	0.18	0.098	0.182
Weights	0.25	0.25	0.1	0.2	0.2

To construct the concordance matrix, we have fixed the following parameters: p_k=0.05, and q_k=0.01 for all criteria, although these thresholds are normally different for the different criteria. Using these values we obtain the vector for $C_k(a_1,a_2)$=[1.00 1.00 0.75 0.00 0.00], with the vectors $C_k(a_i,a_j)$ we calculate the following values for the concordance matrix:

$$C = \begin{pmatrix} 1.000 & 0.575 & 0.490 & 0.765 & 0.355 & 0.639 \\ 0.500 & 1.000 & 0.619 & 0.750 & 0.500 & 0.619 \\ 0.803 & 0.630 & 1.000 & 0.723 & 0.476 & 0.730 \\ 0.750 & 0.454 & 0.550 & 1.000 & 0.495 & 0.695 \\ 0.995 & 0.745 & 0.805 & 0.980 & 1.000 & 0.965 \\ 0.793 & 0.740 & 0.650 & 0.673 & 0.348 & 1.000 \end{pmatrix}$$

Consider now the computation of the single-criterion discordance index with v_k=1 and p_k=0.05, the vector for $D_k(a_1,a_2)$=[0.000 0.000 0.000 0.758 0.432]. These values permit us to calculate the credibility degree matrix S. The matrix S is thus as follows:

$$Outranking\ index\ S = \begin{pmatrix} 1.000 & 0.508 & 0.456 & 0.755 & 0.320 & 0.627 \\ 0.466 & 1.000 & 0.539 & 0.740 & 0.412 & 0.561 \\ 0.803 & 0.615 & 1.000 & 0.723 & 0.467 & 0.730 \\ 0.681 & 0.431 & 0.448 & 1.000 & 0.395 & 0.596 \\ 0.995 & 0.727 & 0.805 & 0.980 & 1.000 & 0.965 \\ 0.793 & 0.661 & 0.601 & 0.673 & 0.332 & 1.000 \end{pmatrix}$$

From the credibility matrix we can calculate de outranking index, but first we need to define the cutting level λ. In this example we will define two values for λ. Defining λ=0.6 we have the outranking relation matrix as follows:

$$outranking\ relational\ matrix = \begin{pmatrix} I & J & P^- & I & P^- & I \\ J & I & P^- & P^+ & P^- & P^- \\ P^+ & P^+ & I & P^+ & P^- & I \\ I & P^- & P^- & I & P^- & P^- \\ P^+ & P^+ & P^+ & P^+ & I & P^+ \\ I & P^+ & I & P^+ & P^- & I \end{pmatrix}$$

and defining λ=0.7 we have the outranking relation matrix as follows:

$$outranking\ relational\ matrix = \begin{pmatrix} I & J & P^- & P^+ & P^- & P^- \\ J & I & J & P^+ & P^- & J \\ P^+ & J & I & P^+ & P^- & P^+ \\ P^- & P^- & P^- & I & P^- & J \\ P^+ & P^+ & P^+ & P^+ & I & P^+ \\ P^+ & J & P^- & J & P^- & I \end{pmatrix}$$

Where P^+ means preference of first alternative when compared to the second one , I means indifference of first alternative when compared to the second one , J means incomparability of first alternative when compared to the second one and P^- means preference of second alternative when compared to the first one.

3.2 Promethee method

Like Electre methods, the Promethee methods are based on a pair wise comparison of the alternatives, leading to valued outranking relation. The Promethee method [18],[12],[15],[37] encompasses two phases: the construction of an outranking relation, aggregating the information about the alternatives and about the criteria, and the exploitation of that relation for decision aid.

At the construction phase of the outranking relation's, the preference degree is presented by a preference function $P_k(x)$. This function evaluates the preference of an action a_x when compared to a_j as a function of $x=g_k(a_i)- g_k(a_j)$.In a generalized point of view, the preference functions, when the value of x is negative, $P_k(x)$ is 0, for the remaining values of x, the function is non-decreasing with $P_k(x)$ varying between 0 and 1. Six preference functions are proposed and they are defined by at most two parameters. The outranking relation can be then represented by an oriented valued graph. The value of each arc is the multicriteria preference index $\pi(a_i,a_j)$, which is defined for all pair of alternatives. These indices may take any value between 0 and 1, and they define a fuzzy outranking relation. Considering $g_k(a_i)=a_i$ and $g_k(a_j)=a_{j,}$ we have the following functions:

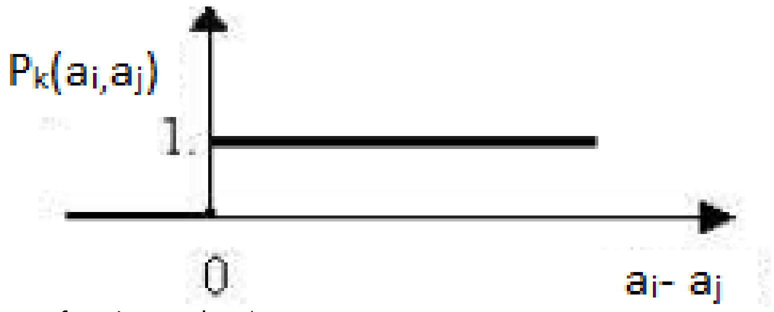


Figure 3.3 - Preference function number 1.

In this preference function, see (Figure 3.3), there are no parameters to be defined and the preference situation is resolved in favor of a_i if the difference between its comparing value on criterion $g_k(a_i)$ is bigger than 0:

a_i-a_j≤0, $P_k(a_{i,}a_j)$=0

a_i-a_j>0, $P_k(a_{i,}a_j)$=1

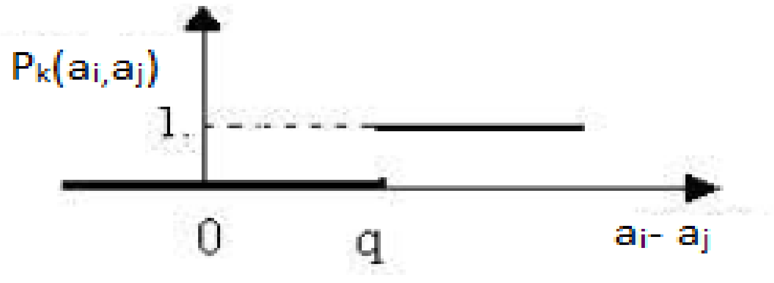


Figure 3.4 - Preference function number 2.

In this preference function, see (Figure 3.4), exists one indifference threshold (q) that must be defined:

a_i-a_j≤q, $P_k(a_i,a_j)$=0

a_i-a_j>q, $P_k(a_i,a_j)$=1

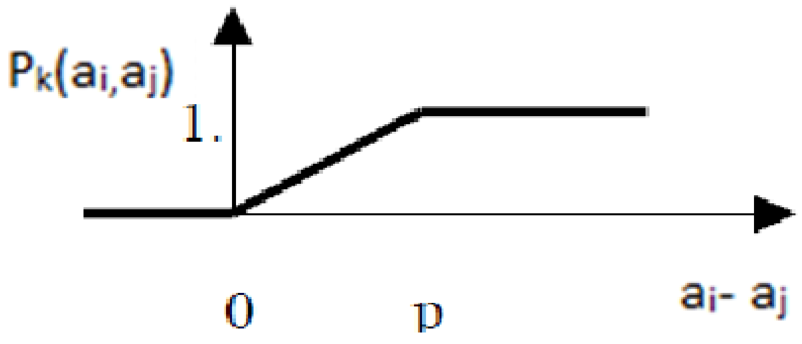


Figure 3.5 - Preference function number 3.

In this preference function, see (Figure 3.5), the preference is increasing until a preference threshold (p) is reached:

a_i-a_j>p, $P_k(a_i,a_j)$=1

a_i-a_j<0, $P_k(a_i,a_j)$=0

0≤ a_i-a_j ≤p, $P_k(a_i,a_j)$= (1/p)* (a_i-a_j)

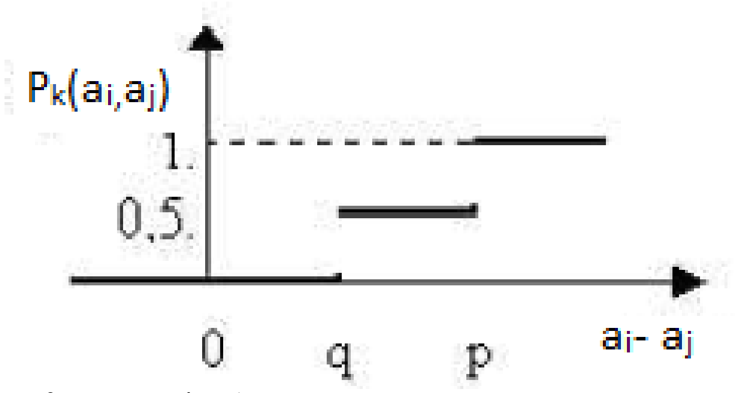


Figure 3.6 - Preference function number 4.

In this preference function, (Figure 3.6), there are 2 thresholds that must be fixed, indifference and preference thresholds (q and p respectively).The values locate between these thresholds get the average value:

a_i-a_j>p, $P_k(a_i,a_j)$=1

a_i-a_j≤q, $P_k(a_i,a_j)$=0

q< a_i-a_j ≤ p, $P_k(a_i,a_j)$= 0.5

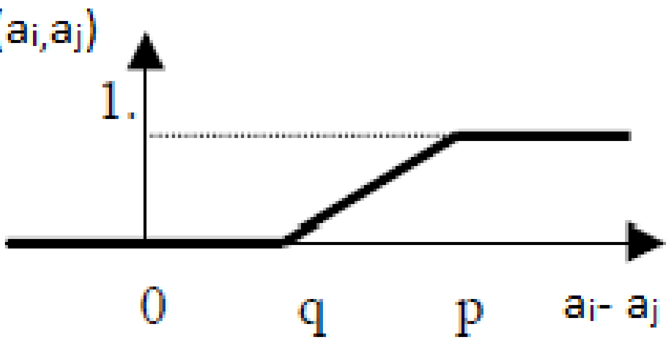


Figure 3.7 - Preference function number 5.

In this preference function, see (Figure 3.7), there are two thresholds that must be fixed, indifference and preference thresholds (q and p respectively). Similar to previous:

a_i-a_j>p, $P_k(a_i,a_j)$=1

a_i-a_j<q, $P_k(a_i,a_j)$=0

q≤ a_i-a_j ≤ p, $P_k(a_i,a_j)$=(a_i-a_j -q)/(p-q)

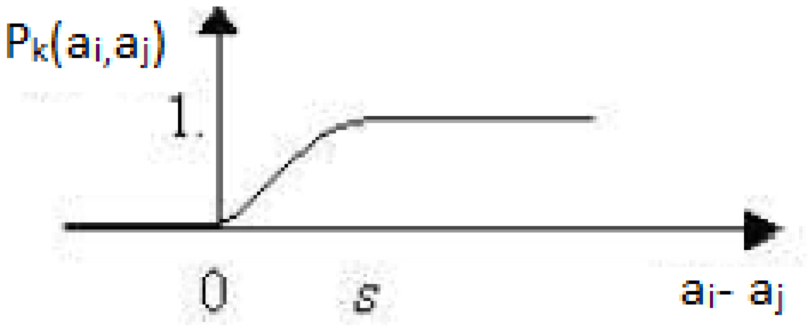


Figure 3.8 - Preference function number 6.

In this preference function, see (Figure 3.8), the preference increase following a normal distribution, the standard deviation must be defined:

$a_i - a_j \geq 0, P_k(a_i, a_j) = e^{\frac{-(a_i-a_j)^2}{2s^2}}$

$a_i - a_j < 0, P_k(a_i, a_j) = 0$

$a_i - a_j \to +\infty, P_k(a_i, a_j) \to 1$

3.2.1 Promethee algorithm

Choosing one of the preference functions, for pair wise comparison of alternatives, the preference matrix is calculated as follows, taking into account weights of criteria, w_k, where the relative importance of each alternative:

$$\pi(a_i, a_j) = \sum_{n=1}^{k} w_k P_k(a_i, a_j)$$

Aggregating a pair wise comparison of two alternatives on al criterion into a global preference degree, allows establishing pre-order on a set of alternatives. The main idea is to analyze how an alternative is preferred to all the other alternatives and inversely, how others are preferred to it. In another words, we will define the globally positive and negative outranking flow as follows:

$$\emptyset^+(a_i) = \sum_{a_j \in A} \pi(a_i, a_j)$$
$$\emptyset^-(a_i) = \sum_{a_j \in A} \pi(a_j, a_i)$$

The positive flow $\emptyset^+$ defines the strength of the alternative and the negative flow $\emptyset^-$ defines the weakness of the alternative.

The Promethee I method obtains a partial preorder with following relations: P^+,P^-,I and J, where P indicates preference, I indicates indifference and like in the other methods, J indicates incomparability relations. With the positive and negative flows, we can obtain the outranking relation matrix as follows:

a_i outranks a_j (a_iPa_j , $a_iP^+a_j$ or $a_jP^-a_i$), if $\begin{cases} \emptyset^+(a_i) > \emptyset^+(a_j) \text{ and } \emptyset^-(a_i) < \emptyset^-(a_j) \text{ or} \\ \emptyset^+(a_i) > \emptyset^+(a_j) \text{ and } \emptyset^-(a_i) = \emptyset^-(a_j) \text{ or} \\ \emptyset^+(a_i) = \emptyset^+(a_j) \text{ and } \emptyset^-(a_i) < \emptyset^-(a_j) \end{cases}$

a_i is indifferent to a_j (a_iIa_j), if $\emptyset^+(a_i) = \emptyset^+(a_j)$ and $\emptyset^-(a_i) = \emptyset^-(a_j)$,

a_i is incomparable to a_j (a_iJa_j), if none of the previous conditions is true.

The Promethee II method, on the other hand, produces a complete preorder with following relations: P^+,P^- and I Note that incomparability relation has no place in this method. The net flow $\emptyset(a_i) = \emptyset^+(a_i) - \emptyset^-(a_i)$ allows to evaluate the following situations:
a_i outranks a_j (a_iPa_j , $a_iP^+a_j$ or $a_jP^-a_i$), if $\emptyset(a_i) > \emptyset(a_j)$,
a_i is indifferent to a_j (a_iIa_j), if $\emptyset(a_i) = \emptyset(a_j)$.

3.2.2 Example:

We will consider the following six alternatives evaluated on five criteria with their weights as basis to build the outranking index (same as used in Electre III Example in Section 3.1.2).

Table 3.2- Data set and weights for Promethee example

Alternatives a_i	g_1	g_2	g_3	g_4	g_5
a_1	0.188	0.172	0.168	0.122	0.114
a_2	0.125	0.069	0.188	0.244	0.205
a_3	0.156	0.241	0.134	0.22	0.136
a_4	0.188	0.034	0.174	0.146	0.159
a_5	0.188	0.276	0.156	0.171	0.205
a_6	0.156	0.207	0.18	0.098	0.182
Weights	0.25	0.25	0.1	0.2	0.2

In the first step we need to choose one of the preference functions, and in this case we choose the most common one, the preference function 5. To construct the preference matrix, we have fixed the following parameters: p_k=0.05, and q_k=0.01 for all criteria, although these thresholds are normally different for the different criteria. Using these values we calculate the following values for the preference matrix:

$$\pi(a_i, a_j) = \begin{pmatrix} 0.000 & 0.500 & 0.198 & 0.250 & 0.005 & 0.207 \\ 0.425 & 0.000 & 0.370 & 0.546 & 0.255 & 0.265 \\ 0.510 & 0.381 & 0.000 & 0.450 & 0.195 & 0.350 \\ 0.245 & 0.250 & 0.278 & 0.000 & 0.023 & 0.328 \\ 0.645 & 0.500 & 0.524 & 0.505 & 0.000 & 0.653 \\ 0.361 & 0.381 & 0.270 & 0.350 & 0.035 & 0.000 \end{pmatrix}$$

Considering now the computation of the positive and negative flows, we have:

$$\emptyset^+(a_i) = [0.232 \;\; 0.372 \;\; 0.377 \;\; 0.224 \;\; 0.565 \;\; 0.273]$$

$$\emptyset^-(a_i) = [0.437 \;\; 0.403 \;\; 0.328 \;\; 0.413 \;\; 0.102 \;\; 0.273]$$

Having the flows, we can finally calculate the outranking relation matrix, and the result is:

$$outranking\ relational\ matrix = \begin{pmatrix} I & P^- & P^- & J & P^- & P^- \\ P^+ & I & P^- & P^+ & P^- & J \\ P^+ & P^+ & I & P^+ & P^- & P^+ \\ J & P^- & P^- & I & P^- & P^- \\ P^+ & P^+ & P^+ & P^+ & I & P^+ \\ P^+ & J & P^- & P^+ & P^- & I \end{pmatrix}$$

Where P^+ means preference of first alternative when compared to the second one , I means indifference of first alternative when compared to the second one , J means incomparability of first alternative when compared to the second one and P^- means preference of second alternative when compared to the first one.

Chapter 4

4. Classification and clustering

We will present in this chapter one algorithm for classification, Electre Tri. And two algorithms for clustering.

4.1 Electre Tri

Electre Tri is a method for ordinal classification. The estimation of the outranking relations between pairs of alternatives is the basis for all methods from the Electre family. The Electre Tri [19],[15] procedure assigns the alternatives A to classes L predefined by set of boundary alternatives B. The upper and lower bound alternatives are the limits for each class. The upper bound alternative of class l_{q-1} is the lower bound of the class l_q. The boundary alternatives can be moved to a neighbor class by changing values on at least one criterion.

For the calculation of the outranking index, the decision maker needs to define a set of alternatives to be classified and a set of criteria. The decision maker should also define the number of classes, their order and the upper and lower boundary alternatives for each class. In addition, the Electre Tri method required also the same information that Electre III, that was described earlier, for each criterion g_j: indifference q_j , preference p_j, veto v_j thresholds, weight w_j and cutting level λ should be selected.

There are two assignment procedures: optimistic and pessimistic and they consist of a comparison of outranking indices to the cutting level λ. The optimistic procedure starts with the comparison of an alternative to the upper bound of the lowest class: the outranking indices are computed and compared to the cutting level. The pessimistic procedure works in the same way but begins with the comparison of an alternative to the lower bound of the highest class. The decision maker can select one of the two assignment procedure or apply both of them. In order to assign the alternative a_i to the class l_{q+1} according to the optimistic procedure (or to the class l_p according the pessimistic one), the cutting level λ should be smaller than the value of the outranking index $S(a_i,b_q)$ and bigger than $S(a_i,b_{q+1})$. Next, the algorithm of the ELECTRE TRI method is discussed.

The algorithm of the Electre Tri method consists in two parts: construction of outranking relation and utilization of this relation for the assignment of alternative to classes. In the first part, the outranking relation a_iSb_q is constructed for each alternative a_i and each boundary alternative b_q. This first part the Electre III algorithm is used, and the outranking index is obtained.

In the second part, the outranking indices $S(a_i,b_q)$ are utilized for the classification of each alternative in the following way. The assignment procedure is selected by the decision maker: Either optimistic, pessimistic or both. Then, the outranking indices calculated for each pair (the alternative a_i, to be classified and each boundary alternative b_q) are compared to the cutting level λ with regards to the assignment procedures as it is shown in Figure 4.1 [19].

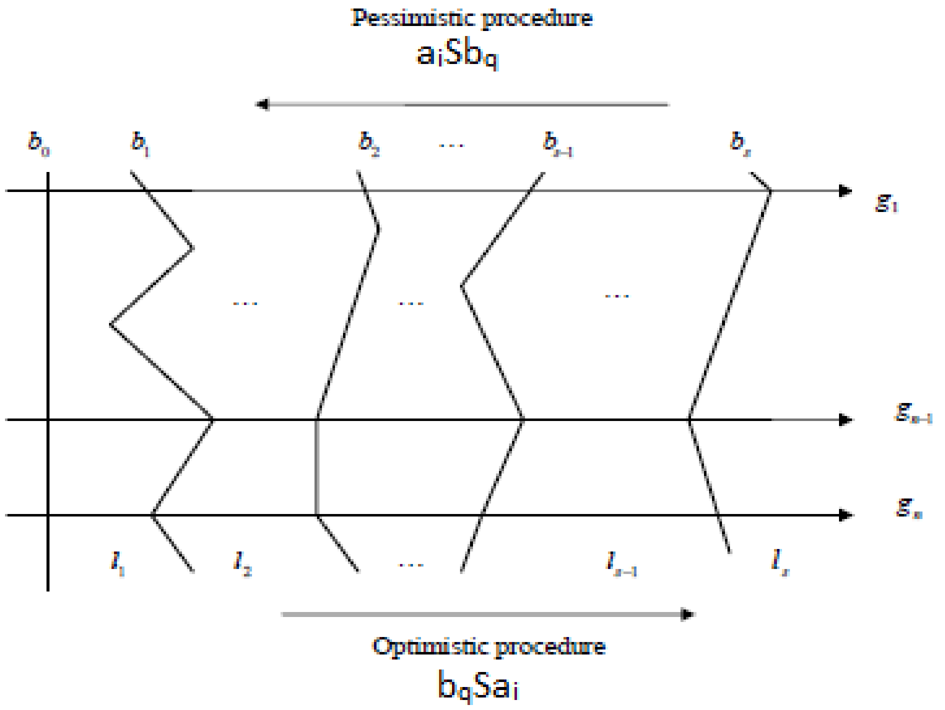


Figure 4.1 - Assignment procedure of the Electre TRI method.

In the pessimistic (or conjunctive) procedure, the comparison between the alternative a_i to the lower bound b_{q-1} of the highest class l_q (q=s,…1) is the starting point. The procedure analyzes classes in a decreasing order until a lower bound b_{q-1} that is outranked by the alternative a_i , a_iSb_{q-1}, is found. The outranking index $S(a_i,b_{q-1})$ is calculated, for the estimation of the outranking relation a_iSb_{q-1}. The outranking index between the alternative a_i

and the upper bound b_q of the class l_q: $S(a_i,b_q)$ is calculated to be classified. The alternative a_i is assigned to the class l_q , if $S(a_i,b_{q-1}) \geq \lambda$ and $S(a_i,b_q)<\lambda$.

On the other hand, in the optimistic (or disjunctive) procedure, the comparison between the alternative a_i to the upper bound b_q of the lowest class l_q (q=s,…1) is the starting point. The procedure continue to analyze classes in a increasing order until an upper bound b_q that is strictly preferred to the alternative a_i , $b_q P a_i$, is found. The outranking index $S(a_i,b_{q-1})$ between the alternative a_i to be classified and the lower bound b_{q-1} of the class l_q: $S(a_i,b_{q-1})$ is calculated. The alternative a_i is assigned to the class l_q if $S(a_i,b_{q-1}) \geq \lambda$ and $S(a_i,b_q)<\lambda$.

Classification according to one of the described produces is unambiguous. An alternative can be assigned to different classes if an assignment is considered according to both procedures simultaneously. For instance, in the optimistic procedure an alternate ve can be assigned to an upper class when compared to the assignment in the pessimistic procedure. This ambiguity must be solved with help of the decision maker or by changing the assignment procedure to earlier optimistic or pessimistic one.

4.2 Multicriteria clustering (extension of K-means)

This method is an extension of the k-means algorithm with a MCDA background [1]. The Multicriteria clustering (MC clustering) method utilizes criteria, and not attributes when compared to classic clustering approaches, however, alternatives are assigned to clusters that are not ordered. That is why we can say that it is a relational multicriteria clustering according taxonomy of clustering methods, see Figure 2.1. The starting point of this approach is k centroids that are randomly chosen. The alternative is assigned to a cluster if the distance to the centroid of this cluster is smallest when compared to distance to other clusters. This distance is calculated using a multicriteria preferences structure discussed next. After the assignment of all alternatives, the centroids for all clusters are recalculated and the process is repeated until there are no changes in all clusters. In this way all alternatives are assigned to k clusters.

This method compares alternatives using preference modeling that constructs so-called profile of each alternative. The comparison between alternatives a_j and $a_i \in A$, results on one of the following relations: Preference (P), Indifference (I) or Incomparability (J). Only one of these relations is true between each couple of given alternatives. This clustering model uses the idea that inside the same cluster, all the alternatives are similar; this means that these alternatives have the same preference relation to more or less the same alternatives.

To build the profiles of the alternatives could be considered outranking methods such as Promethee, Electre, etc. The profile of an alternative is only true for one of the follow relations to an alternative: J(a_i), P$^-$(a_i), I(a_i) and P$^+$(a_i). A profile contains four sets of alternatives each of which containing alternatives that are in the same relation with the studied alternatives (J, I, $^{P-}$ or $^{P+}$). For example, J(a_1) is a set of all alternatives that are incomparable with a_1.

$$J(a_i) = \{a_i \in A | a_i J a_j\} = P_1(a_i).$$
$$P^+(a_i) = \{a_i \in A | a_i P a_j\} = P_2(a_i).$$
$$I(a_i) = \{a_i \in A | a_i I a_j\} = P_3(a_i).$$
$$P^-(a_i) = \{a_i \in A | a_j P a_i\} = P_4(a_i).$$

4.2.1 Multicriteria distance

For multicriteria methods, the concept of distance used in traditional clustering algorithms does not seem to be well suited. That is why the MC distance between two alternatives a_i,a_j $\in$ A is calculated as difference between 1 and sum of intersections of profiles of these alternatives divided by the total number of alternatives.

$$d(a_i, a_j) = 1 - \frac{\sum_{k=1}^{4} |P_k(a_i) \cap (P_k(a_j)|}{n}$$

4.2.2 Construction of the centroids

In the first iteration, the k centroids are chosen randomly from the alternatives A, then the rest of alternatives is assigned to the clusters based on distance to the nearest centroid. When all alternatives are assigned to clusters, the centroids are recalculated and the alternatives are reassigned to the new clusters using the same method of calculating distance between the centroids and the alternatives.

The recalculation of the centroids is based on a voting procedure. The voting procedure starts from finding the profiles of the centroids in the same way as discussed above (for construct profiles of alternatives) but evaluating relations between centroid and each alternative belonging to this cluster.

To select centroid for each cluster, the most common profile for the cluster is selected. It does not have to be a real alternative, but a virtual one (to be exact only common profile is needed). This selection of common profile is called voting since each alternative belonging to the cluster “votes” or shows relation to the rest of alternatives [1].

$$a_j \in P_k(C_i) \leftrightarrow K = \arg max \sum_{a_{i_l}} 1_{\{a_j \in P_k(a_{i_l})\}}$$

Where $1_{\{A\}} = 1$ if condition A is true, 0 otherwise. If different values of k satisfy the previous condition, the final value will be randomly chosen.

4.3 Multicriteria ordered clustering with tabu heuristic

MC ordered clustering with tabu heuristic is criteria dependent method and has ordered clusters, so we can say that it is an ordered multicriteria clustering according to taxonomy presented in Figure 2.1. This is an ordered method, so all the clusters will be ranked. The ordered clustering problem [5] consists of detecting the ordered partition of a set of alternatives that is the most compatible with the information contained in the preference matrix.

Let $A=\{a_1,a_2...a_n\}$ be the set of alternatives evaluated according to a set of criteria $G=\{g_1,g_2...g_n\}$. A comparison of these alternatives lead to a preference matrix $\Pi=(\pi_{ij})$ with $\pi_{ij} \geq 0$ and $\pi_{ij}+\pi_{ji} \leq 1$,because the outranking method considered is Promethee, where π_{ij} represents the preference degree of alternative a_i when compared to alternative a_j. This matrix can be build using classical multicriteria methods such as Electre or Promethee.

An ordered partition of A in k clusters is noted $C= \{ C_1,C_2...C_z \}$ is defined as follows: $A= \cup C_i$, with $i=1,2,...,m$; $C_i \cap C_j = \emptyset$; $C_z \succ C_{z-1} \succ ... \succ C_1$, where the symbol $\succ$ denotes preference, that means that the alternatives that belong to the cluster C_z are better than the alternatives of C_{z-1} in this example.

It is believed that all alternatives belonging to the same cluster should be characterized by a relative small preference degree, also known as homogeneity (HI). Ideally, the preference degree is zero. If an alternative a_i is strongly preferred to a_j they cannot be in the same cluster that means that the clusters are not coherent and is evaluated with coherence index (CI). The symbols HI_C and CI_C are indicators of the partition P and characterize the violation or the satisfaction of two conditions.

Firstly, for each cluster, a homogeneity indicator will be defined as the maximum value of the preference degree between two alternatives belonging to the same cluster, $\max(\pi_{ij}, \pi_{ji})$. The smaller homogeneity indicator, the better the cluster C_i from homogeneity point of view. The homogeneity indicator related to a given partition P will be equal to the maximum homogeneity index of its component, $\max(HI_i)$ for $i=1,...,k$.

Secondly, a coherence indicator is used between each pair of clusters, $CI_{lm.}$ Let us assume that two alternatives are assigned to two distinct clusters, a_i belongs to cluster a cluster C_l and a_j belongs to C_m, and C_l is preferred to C_m. If the value of π_{ij} is higher than π_{ji} ,a_j must be assigned to a better cluster than a_j. The lower CI_{lm}, the better the coherence between the two ordered clusters. The coherence indicator between two clusters will be defined as max (π_{ij}), where a_i belongs to C_m, a_j belongs to C_l and C_l is preferred to C_m.

Finally, the quality of the partition P is measured using a fitness indicator, FT_C , which is the maximum value between its homogeneity and coherence index, max (HI_C, CI_C). The lower the FT_C , the better the partition.

4.3.1 Tabu meta-heuristic

The total number of ordered partitions C of a set with n elements is, much bigger than the total number of not ordered partitions. When n is rising, it is impossible to enumerate all the possible partitions. For example, when n is equal to 10, the number of ordered partitions is bigger than 100 millions. For that reason, a met- heuristic based on Tabu search approach has been developed in [5]. The purpose of this meta-heuristic is to find a partition with small optimal values for the indicators without studying all partitions. This meta-heuristic consists of two steps: split and merge processes. With these two steps, the clusters are updated at each iteration of Tabu search algorithm and the fitness indicator improves leading to a better solution. A cluster is splitted in two sub-clusters and then one of the two sub-clusters is merged with another existing cluster. The remaining sub-cluster is transformed in a new cluster, this way the number of clusters remains the same.

In the first step, the cluster C_i with the highest homogeneity is selected and then the cluster is splitted in 2 sub-clusters SC_{1i} and SC_{2i}. To split the cluster, the two alternatives, a_p and a_q, with highest preference indicators are selected and each one of them goes to the two different sub-clusters. The remaining alternatives are placed in a way that the homogeneity index of the formed sub-clusters has the smallest value. This means that the assignment rule for a new alternative following a_z is: if max (π_{zp}, π_{pz}) ≤max(π_{zq}, π_{zp}) then a_z is assigned to SC_{1i}, or in SC_{2i} otherwise.

Having two sub-clusters, one of them is merged with an existing cluster and the other one is promoted to a new cluster. The sub-cluster with smallest homogeneity factor is selected to be merged. That means that one existing cluster will have all the alternatives of its own cluster and the alternatives of the selected sub-cluster and the new cluster which, compared from remaining sub-cluster and its alternatives. The not chosen sub-cluster is promoted then to a new cluster.

Finally, the indices are calculated for new k clusters and are compared to find the cluster with the best coherence indicator. This mechanism is continued as long as there is improvement of the fitness indicator. When there is no improvement during a certain time, the process is stopped.

To avoid the cycling effect in Tabu search a tabu list is used. Tabu list contains the last T partitions (with T being a parameter of the algorithm which varies). The algorithm consists in either splitting the cluster randomly (with probability p) or splitting them in order to get two sub-clusters with the lowest homogeneity indicators. A distinctive feature of this approach is that the value of probability p varies from iteration to iteration of the Tabu search algorithm. If the generated partition is already in the Tabu list, p increases, if it does not appear yet in Tabu list, p decrease.

The first step of Tabu search meta-heuristic for finding best partition P consists in initializing the parameters (K, K_{max}, T, p^{+}_{step} p^{-}_{step}, p_{max}, p), generating a random partition P with k clusters for which FI_C is evaluated and defining the stopping criterion, which can be for example time or number of iterations. After that, for k=2 to k_{max} , while the stopping criterion is not true, a random value r, is generated and if r<p the cluster is splitted and one of the sub-clusters is merged with an existing cluster like it was described earlier, else the cluster is splitted randomly and then follows the merging process. All the permutations of k clusters are listed, the current partition is now the best of the list according to CI_C and the

best partition among the best and the current partitions are retained. At this step, if the current partition is in Tabu list, p is increased, otherwise p is decreased. This process is repeated until some stopping criterion is satisfied, for example, the number of iterations.

Chapter 5

The MCOC approach proposed in this work

Our focus is the MCDA ordered clustering, so we are going to propose a new approach to solve a MCDA ordered clustering problem in 5.1.

5.1 The MCOC approach proposed in this work

In this work, multicriteria ordered clustering approach based on split and merge processes similar to those that Y.De Smet and L. Montano Guzmán [1] and Nemery and Y. De Smet [5] is proposed. MCOC approach proposed in this work is criteria dependent use the order relation on set of clusters, so we can say that it is an ordered multicriteria clustering according to taxonomy presented in Figure 2.1.

The idea is to assign each alternative to a cluster based its relation to the rest of the alternatives. The relation of order between alternatives is very important in a MCOC (multicriteria ordered clustering), that is why preference or outranking relations could be used in this approach.

For a set of alternatives $A=\{a_1,a_2,\ldots a_n\}$,a set of criterion $G=\{g_1,g_2,\ldots g_n\}$ and a set of clusters $C=\{C_1,\ldots,C_k\}$ with C_{n-1} always worse than C_n , the outranking index is build considering the preference, indifference and veto thresholds (p_k, q_k and v_k) and the cutting level λ if the Electre III algorithm is used, or only the preference and indifference thresholds if Promethee is the applied.

In MCDA ordered method proposed by Nemery and Y. De Smet [5] the split and merge processes are repeated until some stopping condition is satisfied. The number of clusters maintains the same during all process like in the method proposed by Nemery and Y. De Smet

[5]. On the other hand, in the MCOC approach proposed in this work the starting point is one cluster with all alternatives belonging to it and then the cluster will be splitted until some stopping condition is satisfied and a list of ordered clusters is obtained.

To illustrate this MCOC approach, the following example is provided. The starting point for this method is the outranking index obtained by Electre III or Promethee I method:

$$outranking\ relational\ matrix = \begin{pmatrix} I & J & P^- & J & P^- & P^- \\ J & I & J & P^+ & P^- & J \\ P^+ & J & I & P^+ & P^- & P^+ \\ J & P^- & P^- & I & P^- & J \\ P^+ & P^+ & P^+ & P^+ & I & P^+ \\ P^+ & J & P^- & J & P^- & I \end{pmatrix}$$

5.1.1 Split procedure:

In the beginning of this process we have C_1= {a_1, a_2,…a_n}, let us remember that C_{n-1} is always worse than C_n. The idea of splitting is to divide one cluster into as many clusters as possible. By looking at the outranking relation matrix that provides ordered relations (preference, indifference and incomparability) between all pairs of alternatives, we can observe witch alternatives are better, indifferent of incomparable to the current one. That helps in splitting alternatives in ordered clusters.

In the first iteration of the splitting process, after comparing the first alternative (a_i, i=1) to the rest of alternatives (a_j, j>i), we can split all alternatives into 4 clusters: those that are better than the alternative a_1, those that are worse than the alternative a_1, those that are indifferent to the alternative a_1 and those that are incomparable to the alternative a_1.

We will consider the first line of the outranking relation matrix that shows the relation between a_1 and all the other alternatives. Only the alternatives belonging to the same cluster can be moved to another cluster. At the first iteration, all alternatives are in the same cluster and can be moved. Now, we will select the alternatives that are better than alternative a_1 (a_j P a_1) and create a new cluster for them (C_2). Similarly, alternatives that are worse than the current alternative should be allocated in a worse cluster. We can see that there is no (P^+) in the first line, that means that the result of this iteration all alternatives will be better or equally good to this alternative a_1. After the first iteration we will have ordered clusters C_1={a_1, a_2, a_4} and C_2={a_3, a_5, a_6} with C_2 better than C_1.

In the second iteration of the split process, for the alternative a_i , i=2, we will analyze alternatives a_j, j>i, and find out what alternatives are worst and what alternatives are better than a_2. We have ordered clusters C_1={a_1, a_2, a_4 } and C_2={a_3, a_5, a_6 }, so we can only move alternatives from C_1 because a_2 belongs to C_1. At this point we only need to know the outranking index for the current alternative a_2 and the not yet analyzed alternative a_4 , and we know this relation is a_2Pa_4, so we will split C_1 in 2 clusters, one with alternatives worse than a_2 and one with alternatives that are equally good to a_2. To maintain the order, the cluster C_1 will be splitted in C_1 and C_2 and all the other clusters that were better than C_1 in the beginning of this iteration will be increased by 1, because we are creating one new cluster by splitting C_1. After this iteration we will have the following order of clusters: C_1= {a_4}, C_2={a_1, a_2} and C_3={a_3, a_5, a_6 }.

In the third iteration of the splitting process, the alternative a_i , i=3 is compared to all alternatives that have not yet been analyzed, a_j, j>I and the information about alternatives that are worse than $a_{3,}$ and alternatives that are better than a_3 is obtained having: C_1={a_4} ,C_2={a_1, a_2} and C_3={a_3, a_5 a_6}. We can only move alternatives from C_3 because a_3 belongs to C_3. At this point we only need to know the outranking indices between a_3 and a_5. Evaluating this relation provides us with information about relations a_3Pa_6 and a_5Pa_3, so we will split C_3 in three clusters, one for the alternatives worse than a_3, one for the alternatives better than a_3 and one for the alternatives that are equally good to a_3. To maintain the order, the cluster C_3 will be splitted in C_3, C_4 and C_5. All the other clusters will keep the same order because all the other clusters are worse than the cluster C_5. After this iteration we will have following ordered clusters: C_1={ a_4} ,C_2={$a_{1,}$ a_2} ,C_3={a_6}, C_4={a_3} and C_5={a_5}.

In the following steps no changes are made in the order of clusters because the alternatives $a_{4,}$ a_5 and a_6 are the only alternatives in the corresponding clusters. We can conclude now the splitting process and we have the alternatives allocated to the maximum number of cluster that we can obtain (five alternatives are allocated to five ordered clusters). This is important information because we know now the preference order between all alternatives. Next the clusters are merged until the specified number of clusters is reached.

5.1.2 Merge procedure:

The merging process is based on the definition of homogeneity [5] and its objective is to merge the clusters until some stopping criterion is satisfied, for example the number of clusters defined a priori is reached. For the example introduced earlier in Section 5.3.1, after splitting process, the alternatives are allocated in five clusters: C_1={ a_4} ,C_2={$a_{1,}$ a_2} ,C_3={a_6}, C_4={a_3} and C_5={a_5}.

Assuming that a priori the number of clusters is defined as three, we have to merge some clusters. One way of merging clusters is suggested in [5] and is based on calculating the preference matrix. The preference relation between all pairs of alternatives can be calculated based on Electre Promethee methods for each criterion. For instance, here we calculated preference relation based on the Promethee I method as was already shown in Section 3.2.1.

Let us assume the overall preference matrix obtained for the example from [15, p.35] is:

$$\pi(a_i, a_j) = \begin{pmatrix} 0.000 & 0.500 & 0.195 & 0.250 & 0.005 & 0.208 \\ 0.425 & 0.000 & 0.370 & 0.546 & 0.255 & 0.264 \\ 0.510 & 0.381 & 0.000 & 0.450 & 0.195 & 0.350 \\ 0.245 & 0.250 & 0.278 & 0.000 & 0.020 & 0.328 \\ 0.645 & 0.500 & 0.524 & 0.505 & 0.000 & 0.653 \\ 0.361 & 0.381 & 0.270 & 0.315 & 0.035 & 0.000 \end{pmatrix}$$

Now we have all the data needed to proceed with the merging process. It is usually assumed [5] that alternatives belonging to the same cluster C_i, should be characterized be a relatively small preference degree. In the ideal case, the pairwise comparison of alternatives belonging to the same cluster, a_i and a_j belong to C_i, should lead to $\pi(a_i,a_j)= \pi(a_j,a_i)=0$. On the other hand, if a_i is strongly preferred to a_j, these two alternatives should be in different clusters. For these two alternatives, the cluster that a_i belongs to, should be better than the cluster that contains the alternative a_j.

In [5], the homogeneity indicator is introduced, which is equal to the maximum value of the preference degree between the alternatives belonging to the same cluster C_i. It is obvious that the smaller the homogeneity indicator is, the better is the cluster C_i from the homogeneity point of view. For example, for the cluster C_2, the preference degrees are: $\pi(a_1,a_2)=0.500$ and $\pi(a_1,a_2)=0.425$, so the homogeneity indicator is the maximum between these two values, 0.500.

In the merging process we can only merge two neighbor clusters, this guarantees that the order is maintained, that means that all the clusters are still ordered after the merging process. In the first iteration of this process, we will calculate the homogeneity indicator for all the neighbor clusters; in our case we will calculate the homogeneity indicator for C_1 and C_2, for C_2 and C_3, for C_3 and C_4 and for C_4 and C_5. These values are HI_C=[0.546 0.5 0.350 0.524]. The lowest value of this vector is 0.350, the homogeneity indicator between C_3 and C_4, these are the two clusters to be merged. After this iteration we have four ordered clusters $C_1=\{a_4\}$,$C_2=\{a_1,a_2\}$,$C_3=\{a_6,a_3\}$ and $C_4=\{a_5\}$.

In the following iterations the same process is repeated until some stopping criterion is reached, in this case the number of ordered clusters that we want to have, three. After the second iteration we reach our objective, three ordered clusters, and the result of clustering process is three following ordered clusters: $C_1=\{a_4\}$,$C_2=\{a_1,a_2,a_6,a_3\}$ and $C_3=\{a_5\}$.

Obviously, the number of final clusters must be smaller than the number of clusters obtained. If the number of final clusters that we want to have is equal to the number of cluster of splitting process, there is no need to perform the merge process.

Chapter 6

Experimental results and analysis

In this section the results of solving clustering problem on several data sets using the algorithm described in 5.3. Using as basic the Promethee I and the Electre III outranking procedures, will be presented and compared with results obtained using another methods from the multicriteria literature such as Electre Tri [15] and Electre tri-c [16]. Our approach was encoded in Java language, using the software Eclipse SDK (version: 3.6.2). The tests were done in a Core 2 Duo with 1.66 GHz processor and 1.0 Gb RAM.

6.1 Example 1

For the first example, a small data set with only two criteria will be presented. We will consider the following six alternatives evaluated on two criteria with equal weights that should be assigned to three ordered using our approach.

Table 6.1- Data set for the example used in Section 6.1

Alternatives a_i	g_1	g_2
a_1	0.172	0.122
a_2	0.069	0.244
a_3	0.241	0.220
a_4	0.034	0.146
a_5	0.276	0.171
a_6	0.207	0.098
Weights	0.5	0.5

The outranking method to be considered is Electre III. The parameters needed for this method such as indifference, preference and veto thresholds and cutting level λ are indicated in Table 6.1.

Table 6.2- Parameters for the example used in Section 6.1

Criterion g_k	q_k	p_k	v_k	λ
g_1	0.01	0.05	1	0.6
g_2	0.01	0.05	1	

At this point all needed information is known, that leads us to the first step of the algorithm, which is calculation of the outranking index. Following the Electre III algorithm we have the following outranking relation matrix:

$$outranking\ relational\ matrix = \begin{pmatrix} I & J & P^- & P^+ & P^- & I \\ J & I & P^- & P^+ & J & J \\ P^+ & P^+ & I & P^+ & P^+ & P^+ \\ P^- & P^- & P^- & I & P^- & J \\ P^+ & J & P^- & P^+ & I & P^+ \\ I & J & P^- & J & P^- & I \end{pmatrix}$$

Where P^+ means preference of first alternative when compared to the second one , I means indifference of first alternative when compared to the second one , J means incomparability of first alternative when compared to the second one and P^- means preference of second alternative when compared to the first one.

Now that we have the outranking relation matrix, we can apply the splitting algorithm. At the end of the split algorithm we have four ordered clusters: $C_1=\{a_4\}$, $C_2=\{a_1, a_2, a_6\}$, $C_3=\{a_5\}$ and $C_4=\{a_3\}$.

At this point we have four clusters, and we want to have only three, so we need to apply the merging algorithm. For the next step, we need the values of outranking index $S(a_i,a_j)$ that provides values of preference of each pair of alternatives. We have then the preference matrix as follows:

$$\pi(a_i a_j) = \begin{pmatrix} 0.000 & 0.500 & 0.000 & 0.500 & 0.000 & 0.175 \\ 0.500 & 0.000 & 0.175 & 0.812 & 0.500 & 0.500 \\ 1.000 & 0.500 & 0.000 & 1.000 & 0.488 & 0.800 \\ 0.175 & 0.000 & 0.000 & 0.000 & 0.000 & 0.475 \\ 0.986 & 0.500 & 0.313 & 0.688 & 0.000 & 1.000 \\ 0.313 & 0.500 & 0.000 & 0.500 & 0.000 & 0.000 \end{pmatrix}$$

As a result of applying merging algorithm to the set of clusters obtained after splitting algorithm we have three ordered clusters $C_1=\{a_4\}$, $C_2=\{a_1, a_2 , a_6\}$ and $C_3=\{a_5, a_3\}$.

In Figure 6.1 in the x axis we have the value of all alternatives on the criterion g_1 and in the y axis on g_2.

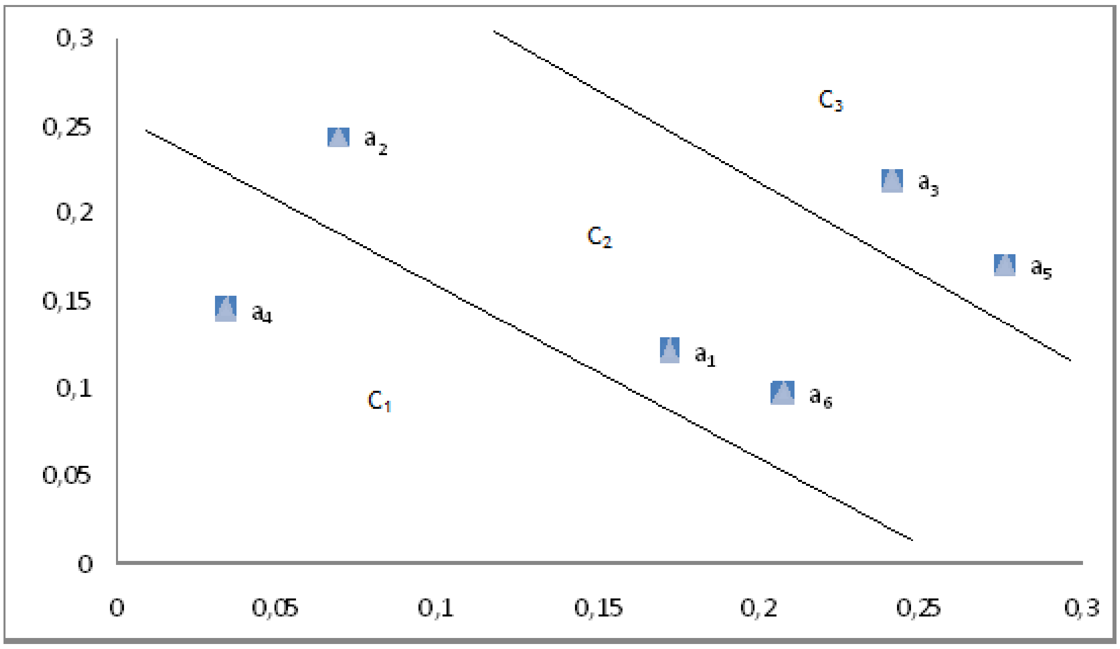


Figure 6.1 - Dissemination of clusters in two criteria space for the example 5.1

The clustering algorithm joins the alternatives in the same cluster that are equally good to each other. The weights of g_1 and g_2 are the same, and with the values given in Table 6.1 and Table 6.2 we have this three ordered clusters. We can easily see from the Figure 6.1 that C_3 is better than C_2 and C_2 is better than C_1.

Now we are going to compare the proposed algorithm with other clustering and classification algorithms from the multicriteria literature.

6.2 Example 2

We will present an example of application of our approach in the field of business failure risk. It concerns the assignment of firms to different risk categories. The example can be found in [15] and [28]. Each firm is evaluated on the basis of seven criteria. The number of considered clusters is five and they are:

•C_1: very high risk (the worst cluster)

•C_2: high risk

•C_3: medium risk

•C_4: low risk

•C_5: very low risk (best cluster)

The 7 criteria are the following:

- g_1 : earning before interest / total assets [to be maximized]
- g_2 : net income / net worth [to be maximized]
- g_3 : total liabilities / total assets [to be minimized]
- g_4 : interest expenses / sales [to be minimized]
- g_5 : general and administrative expenses / sales [to be minimized]
- g_6 : manager work experience [to be maximized]
- g_7 : market niche / position [to be maximized]

In [15] this problem has solved by Electre Tri optimistic and Electre Tri pessimistic method as a classification problem. The following parameters presented in Table 6.3 were used for the Electre Tri method.

Table 6.3 - parameter of alternatives that are used in example 6.3

Parameters	g_1	g_2	g_3	g_4	g_5	g_6	g_7
b_1	-10	-60	90	28	40	1	0
b_2	0	-40	75	23	32	2	2
b_3	8	-20	60	18	22	4	3
b_4	25	30	35	10	14	5	4
q_k	1	4	1	1	3	0	0
p_k	2	6	3	2	4	0	0
w_k	0.01	0.295	0.225	0.01	0.225	0.01	0.225

The Table 6.3 contains preference and indifferent thresholds and the weights for each criterion that are the parameters used in this case. The cutting level parameter used in Electre III is fixed at 0.85. The alternatives $b_1,\ldots,b_n$ denote the lower (upper) limits of the categories in Electre Tri.

Table 6.4 - Alternatives evaluated on criteria used in the example 6.2

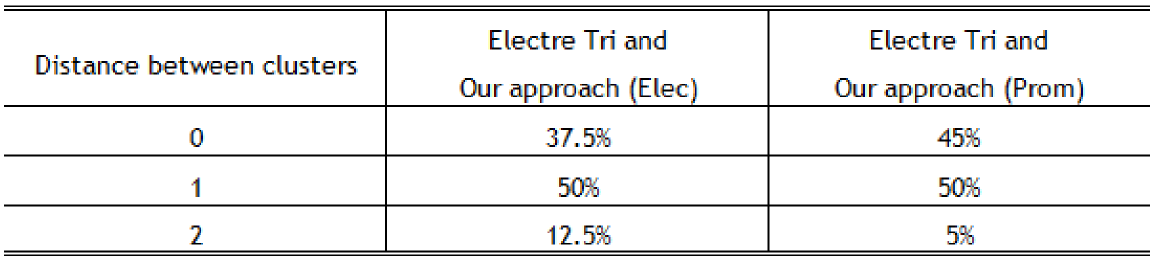
Alternatives	g_1	g_2	g_3	g_4	g_5	g_6	g_7
a_0	35.8	67	19.7	0	0	5	4
a_1	16.4	14.5	59.8	7.5	5.2	5	3
a_2	35.8	24	64.9	2.1	4.5	5	4
a_3	20.6	61.7	75.7	3.6	8	5	3
a_4	11.5	17.1	557.1	4.2	3.7	5	2
a_5	22.4	25.1	49.8	5	7.9	5	3
a_6	23.9	34.5	48.9	2.5	8	5	3
a_7	29.9	44	57.8	1.7	2.5	5	4
a_8	8.9	5.4	27.4	4.5	4.5	5	2
a_9	25.7	29.7	46.8	4.6	3.7	4	2
a_{10}	21.2	24.6	64.8	3.6	8	4	2
a_{11}	18.3	31.6	69.3	2.8	3	4	3
a_{12}	20.7	19.3	19.7	2.2	4	4	2
a_{13}	9.9	3.5	53.1	8.5	5.3	4	2
a_{14}	10.4	9.8	80.9	1.4	4.1	4	2
a_{15}	17.7	19.8	52.88	7.9	6.1	4	4
a_{16}	14.8	15.9	27.9	5.4	1.8	4	2
a_{17}	16	14.7	53.5	6.8	3.8	4	4
a_{18}	11.7	10	42.1	12.2	4.3	5	2
a_{19}	11	4.2	60.8	6.2	4.8	4	2
a_{20}	15.5	8.5	56.2	5.5	1.8	4	2
a_{21}	13.2	9.1	74.1	6.4	5	2	2
a_{22}	9.1	4.1	44.8	3.3	10.4	3	4
a_{23}	12.9	1.9	65	14	7.5	4	3
a_{24}	5.9	-27.7	77.4	16.6	12.7	3	2
a_{25}	16.9	12..4	60.1	5.6	5.6	3	2
a_{26}	16.7	13.1	73.5	11.9	4.1	2	2
a_{27}	14.6	9.7	59.5	5.6	5.6	2	2
a_{28}	5.1	4.9	28.9	2.5	46	2	2
a_{29}	24.4	22.3	32.8	3.3	5	3	4
a_{30}	29.5	8.6	41.8	5.2	6.4	2	3
a_{31}	7.3	-64.5	67.5	30.1	8.7	3	3
a_{32}	23.7	31.9	63.6	12.1	10.2	3	2
a_{33}	18.9	13.5	74.5	12	8.4	3	3
a_{34}	13.9	3.3	78.7	14.7	10.1	2	2
a_{35}	-13.3	-31.1	63	21.2	29.1	2	1
a_{36}	6.2	-3.2	46.1	4.8	10.5	2	1
a_{37}	4.8	-3.3	71.1	8.6	11.6	2	2
a_{38}	0.1	-9.6	42.5	12.9	12.4	1	1
a_{39}	13.6	9.1	76	17.1	10.3	1	1

The results obtained with our approach after merging, using Electre III and Promethee to build the outranking index, and Electre-tri [15] pessimist and optimist, are in Table 6.5.

Table 6.5 – Allocation of alternatives from the example 6.2 in ordered groups using Electre Tri pessimistic and optimistic, and our approaches using Electre III and Promethee as a basis

Alternatives	Electre-tri (pess)	Electre-tri (opt)	Our approach (Electre III)	Our approach(Prom)
a_0	C_5	C_5	C_5	C_5
a_1	C_4	C_5	C_2	C_4
a_2	C_3	C_5	C_2	C_4
a_3	C_3	C_5	C_2	C_4
a_4	C_3	C_5	C_2	C_3
a_5	C_4	C_5	C_4	C_4
a_6	C_4	C_5	C_4	C_4
a_7	C_4	C_5	C_4	C_4
a_8	C_3	C_5	C_3	C_3
a_9	C_3	C_5	C_4	C_4
a_{10}	C_3	C_5	C_2	C_3
a_{11}	C_3	C_5	C_3	C_4
a_{12}	C_3	C_5	C_4	C_4
a_{13}	C_3	C_5	C_2	C_2
a_{14}	C_2	C_5	C_1	C_1
a_{15}	C_4	C_5	C_4	C_4
a_{16}	C_3	C_5	C_4	C_4
a_{17}	C_4	C_5	C_4	C_4
a_{18}	C_3	C_5	C_3	C_4
a_{19}	C_3	C_5	C_3	C_2
a_{20}	C_3	C_5	C_2	C_3
a_{21}	C_3	C_5	C_2	C_1
a_{22}	C_4	C_4	C_2	C_3
a_{23}	C_3	C_5	C_3	C_2
a_{24}	C_2	C_4	C_2	C_1
a_{25}	C_3	C_5	C_1	C_3
a_{26}	C_3	C_5	C_2	C_2
a_{27}	C_3	C_5	C_2	C_3
a_{28}	C_1	C_5	C_2	C_2
a_{29}	C_4	C_5	C_2	C_4
a_{30}	C_4	C_5	C_4	C_4
a_{31}	C_2	C_5	C_3	C_2
a_{32}	C_3	C_5	C_2	C_3
a_{33}	C_3	C_5	C_2	C_3
a_{34}	C_2	C_5	C_1	C_1
a_{35}	C_2	C_3	C_1	C_1
a_{36}	C_2	C_4	C_2	C_1
a_{37}	C_3	C_4	C_1	C_1
a_{38}	C_2	C_4	C_2	C_1
a_{39}	C_2	C_5	C_1	C_1

Table 6.6 - Percentage of the allocation of alternatives from the example 6.2 in ordered groups using Electre Tri pessimistic and optimistic, and our approaches using Electre III and Promethee as a basis



Distance between clusters	Electre Tri and Our approach (Elec)	Electre Tri and Our approach (Prom)
0	37.5%	45%
1	50%	50%
2	12.5%	5%

Since the parameters used in Electre Tri, Electre III and Promethee I, are similar, the results obtained for Electre Tri pessimistic procedure and our approach are quite similar. There are no assignments made by our approach, with both Electre III and Promethee), is out of range defined by Electre tri pessimistic procedure.

We have thus presented the results for a business failure risk problem obtained with two different methods. Electre Tri which is a classification method, uses lower and upper bounds given, whereas our clustering approach does not use any information about clusters, uses the similarity to clusters. Let us remark that is not always obvious how to construct these clusters. It is not easy to define appropriate parameters for the two methods like for example cutting level, thresholds, and weights.

The Electre Tri is a classification method that uses information about classes, such as boundary alternatives, to assign the set of alternatives to classes. In case of absence of information about classes our method could be applied and give results similar to Electre Tri.

6.3 Example 3

Next we will compare our approach to the results of another multicriteria classification algorithm, called Electre Tri-C [16].

Consider fifteen potential actions, denoted as $a_1,\dots,a_{15}$ and evaluated on a coherent family of seven criteria, denoted $G=\{g_1,\dots,g_7\}$ (all the criteria are in increasing preference direction and should be maximized), when taking into account the preferences of the decision maker (see Table 6.7). The importance of each criterion is defined by the weight. The veto thresholds are not considered in this case.

Table 6.7- Criteria and parameters for the example 6.3

Parameters	g_1	g_2	g_3	g_4	g_5	g_6	g_7
q_k	4	10	10	2	2	0	0
p_k	8	15	15	4	4	1	1
w_k	0.20	0.15	0.10	0.10	0.10	0.15	0.20

Table 6.8 - Potential alternatives for the example 6.3

alternatives	g_1	g_2	g_3	g_4	g_5	g_6	g_7
a_1	16	15	40	12	15	5	3
a_2	45	92	58	16	16	5	5
a_3	21	62	24	16	12	5	3
a_4	21	25	50	10	12	3	5
a_5	4	12	15	4	2	2	1
a_6	5	30	60	10	15	1	2
a_7	6	25	25	4	16	4	5
a_8	40	80	60	16	12	4	5
a_9	10	20	30	8	8	2	1
a_{10}	21	19	80	18	16	4	2
a_{11}	10	4	47	11	15	4	2
a_{12}	45	85	85	15	15	5	5
a_{13}	15	16	72	15	18	4	2
a_{14}	18	20	47	12	14	4	4
a_{15}	35	70	60	10	10	3	3

Table 6.9 - Characteristic alternatives for the example 6.3

C_k	b_h	g_1	g_2	g_3	g_4	g_5	g_6	g_7
	b_0	0	0	0	0	0	0	0
C_1	b_1	5	10	20	5	5	1	1
C_2	b_2	15	30	40	10	10	2	2
C_3	b_3	25	50	60	10	10	3	3
C_4	b_4	35	70	60	10	10	4	4
C_5	b_5	45	90	80	15	15	5	5
	b_6	50	100	100	20	20	6	6

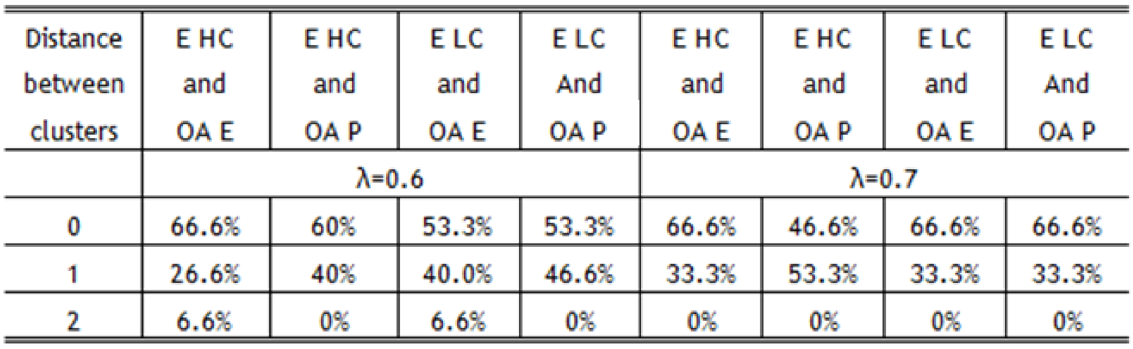
The objective of Electre Tri-C is to assign the alternatives, see (Table 6.8), to a set of five categories defined by a set of so-called characteristic alternatives that are basically centroids of clusters see, (Table 6.9).

We are going to compare the results of our approach, using Electre III and Promethee I outranking methods, with the results obtained in [16] by the Electre Tri-C method. This time, we are going to use two different values of the cutting level λ. For Electre III and Electre Tri-C the values used for λ are 0.60 and 0.70. The comparative results of our approach using Electre III and Promethee I outranking methods and Electre Tri C are provided in Table 6.10.

Table 6.10 - Electre Tri-C and our approach results for example 6.3.

Alternatives	ET-C HC	ET-C LC	OA E	ET-C HC	ET-C LC	OA E	OA P
	λ=0.6			λ=0.7			
a_1	C_3	C_2	C_2	C_3	C_2	C_3	C_3
a_2	C_5	C_5	C_5	C_5	C_5	C_5	C_5
a_3	C_3	C_3	C_3	C_3	C_3	C_4	C_4
a_4	C_3	C_3	C_2	C_3	C_3	C_3	C_2
a_5	C_1	C_1	C_2	C_1	C_1	C_1	C_1
a_6	C_2	C_2	C_2	C_2	C_2	C_2	C_2
a_7	C_2	C_3	C_2	C_3	C_2	C_2	C_2
a_8	C_4	C_5	C_5	C_5	C_4	C_5	C_5
a_9	C_1	C_2	C_1	C_2	C_1	C_1	C_1
a_{10}	C_3	C_3	C_3	C_3	C_3	C_3	C_4
a_{11}	C_2	C_2	C_2	C_2	C_2	C_2	C_2
a_{12}	C_5	C_5	C_5	C_5	C_5	C_5	C_5
a_{13}	C_2	C_2	C_2	C_3	C_2	C_3	C_2
a_{14}	C_4	C_4	C_2	C_4	C_3	C_3	C_3
a_{15}	C_3	C_4	C_4	C_4	C_3	C_4	C_3

Table 6.11 - Electre Tri-C and our approach results for example 6.3 in percentages.



Distance between clusters	E HC and OA E	E HC and OA P	E LC and OA E	E LC And OA P	E HC and OA E	E HC and OA P	E LC and OA E	E LC And OA P
	λ=0.6				λ=0.7			
0	66.6%	60%	53.3%	53.3%	66.6%	46.6%	66.6%	66.6%
1	26.6%	40%	40.0%	46.6%	33.3%	53.3%	33.3%	33.3%
2	6.6%	0%	6.6%	0%	0%	0%	0%	0%

In Table 6.10, ET-C HC means Electre Tri-C higher category, ET-C LC means Electre Tri-C lowest category, OA E means our approach using Electre III outranking method and OA P means our approach using Promethee outranking method.

Once again we have very similar results for our approach and Electre Tri C. Despite the fact that our approach does not use information about clusters like characterizing alternatives given in advance in Electre Tri C, it provides similar results. The difference of the clusters attributed to the alternatives is bigger than one only in one case (for a_{14}, the difference for our approach and Electre Tri-C using λ=0.6 is two clusters/classes), showing a very good performance of our approach.

For both Electre III used in our approach and Electre Tri-C the testes were done with two different values of the cutting level, 0.60 and 0.70. We can note small change in results when the value of the clustering level increases. We can see that in our approach using Electre III with λ= 0.6 and λ=0.7 only four alternatives are in different clusters and for Electre Tri-C for the highest category only two alternative are in different clusters and for the lowest category there is only one alternative that is not in the same cluster. Construction of these clusters is not always obvious; defining appropriate parameters for two methods is not easy but we can see that our approach gives very similar results.

Chapter 7

Conclusions and future research

In this paper we have proposed an approach to solve the multicriteria ordered clustering problem. In this kind of problems, similar alternatives have to be grouped in ordered clusters. The difficulty of this problem is the fact that the clusters are not defined a priori.

Different multicriteria decision aiding methods have been presented to assign a set of alternatives into clusters. In this work these approaches were discussed and an approach for one type of the clustering problems was suggested, in particular, the approach to solve ordered clustering problem was introduced.

Our approach has, successfully, been applied on datasets for which, the obtained results have been compared to the results obtained by earlier developed methods represented in multicriteria literature. In particular the clustering results obtained by our approach on the business failure risk problem with 40 alternatives evaluated on 7 criteria and on other hypothetical example with 15 alternatives evaluated on 7 criteria are coherent with the results obtained with Electre Tri and Electre Tri-C. This permits us to hope to tackle and solve similar problems of this type that appearing in real life.

Although the results seem to be encouraging, future attention should be paid to analysis of problems with a larger set of alternatives evaluated in a big set of criteria, study stability in face of different labels of the input alternatives and make it interactive. One way to work with large data set is to first select a representative data set and cluster this data set. This could help to simplify the problem. After this step we can classify the rest of alternatives in classification problem, by comparing to the centroids obtained.

References

[1] Y.De Smet and L. Montano Guzmán. Towards multicriteria clustering: An extension of the k-means algorithm. European Journal of Operational Research, 158(2):390-398, oct 2004.

[2] Cailloux, O., C. Lamboray, Ph. Nemery , A taxonomy of clustering procedures, Proceedings of the 66th Meeting of the EWG on MCDA, Marrakech, Maroc, 2007.

[3] Yevseyeva, I., K. Miettinen, P. Salminen, and R. Lahdelma, SMAA-Classification: A new method for nominal classification, Helsinki School of Economics, Working Paper, 2007.

[4] E. Fernandez, J. Navarro, S. Bernal, Handling multicriteria preferences in cluster analysis, Eur. J. Oper. Res. 202 (2010) 819–827.

[5] Ph. Nemery and Y. De Smet. Multicriteria ordered clustering. Technical Report TR/SMG/2005-003, Universit´e Libre de Bruxelles/SMG, 2005.

[6] A. Valls Mateu, CLUSDM: a multiple criteria decision making method for heterogeneous data sets, Polytechnic University of Catalonia. 2002.

[7] L. Montano-Guzman, FuzzyMeasures and Integrals in the MCDA Sorting Problematic: Methodology and Application to the Diagnosis of Firms, PhD thesis, Univerisite Libre de Bruxelles, 2002.

[8] B. Roy. Crit`eres multiples et mod´elisation des pr´ef´erences : l'apport des relations de surclassement. Revue d'Economie Politique, 1974.

[9] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. ACM Computing Surveys, 31(3):264–323, sep 1999.

[10] R. Bisdorff. Electre-like clustering from a pairwise fuzzy proximity index. European Journal of Operational Research, 138:320–331, 2002.

[11] J.R. Figueira, Y. De Smet, B. Mareschal, and J.-P. Brans. Mcda methods for sorting and clustering problems: Promethee tri and promethee cluster. Technical Report IS-MG 2004/02, Universite Libre de Bruxelles/SMG, 2004.

[12] Dias, L. C., J. P. Costa, J. N. Clímaco, "A Parallel Implementation of the PROMETHEE Method", European Journal of Operational Research, Vol. 104, Nº 3, pp. 521-531, 1998

[13] Almeida-Dias, J., Figueira, and Roy, B., ELECTRE TRI-C: a multiple criteria sorting method based on characteristic reference actions. Cahier du lamsade 274, may 2008

[14] Nemery, P., On the use of multicriteria ranking methods in sorting problems. PhD Thesis. Université Libre de Bruxelles, 2008-2009

[15] Nemery, P., Multicriteria Clustering. Master Thesis. Université Libre de Bruxelles, 2005-2006.

[16] Almeida-Dias, J., Figueira, J.R. and Roy, B., ELECTRE TRI-C: a multiple criteria sorting method based on characteristic reference actions. European Journal of Operational Research. v204. 565-580, 2010.

[17] V. Mousseau, J. Figueira, L. C. Dias, C. Gomes da Silva, and J. N. Clímaco. Resolving inconsistencies among constraints on the parameters of an MCDA model. European Journal of Operational Research, 147:72-93, 2003.

[18] Bastos, L. N. V., Almeida, A. T. - Utilização do Método Promethee II na Análise das Propostas de Preços em um Processo de Licitação. In: ENEGEP (Encontro Nacional de Engenharia de Produção), 2002, Curitiba.

[19] Yevseyeva I., Solving classification problems with multicriteria decision aiding approaches. Phd Theis. University of Jyvaskyla,2007

[20] Hyde, K., Maier, H., Colby, C., Incorporating uncertainty in the PROMETHEE MCDA method. Journal of Multi-Criteria Decision Analysis 12, 245–259, 2003.

[21] Ph. Nemery and C. Lamboray. FlowSort: a flow-based sorting method with limiting and central profiles. TOP, 16:90–113, 2008.

[22] Chou, W. C.; Lin, W. T.; Lin, C. Y., (2007); Application of fuzzy theory and PROMETHEE technique to evaluate suitable ecotechnology method: A case study in Shihmen Reservoir Watershed, Taiwan; Ecol. Eng.,; 31 (4), 269-280

[23] GOMES, L. F. M. A.; GOMES, C. F. S. & ALMEIDA, A. T. Tomada de decisão Gerencial: Enfoque Multicritério. Rio de Janeiro: Atlas, 2002.

[24] CHIAVENATO, Idalberto. Introdução à teoria geral da administração. 5ª Edição, Rio de Janeiro: Campus, 1999.

[25] GOMES, L. F. M. A.; ARAYA, M. C. G. & CARIGNANO, C. Tomada de decisões em cenários complexos. São Paulo: Pioneira, 2004.

[26] MARINS, Cristiano S. & COZENDEY, Manaara I. A metodologia de multicritério como ferramenta para tomada de decisões gerenciais: um estudo de caso. In: 25º Encontro Nacional de Engenharia de Produção (ENEGEP). Anais. Porto Alegre / RS, 2005.

[27] Helmann, Kurtt Schamne; Marçal, Rui Francisco Martins. Multiple criteria method to maintenance management decision support: application of electre I method to process critic equipments election. Campus Ponta Grossa - Paraná – Brasil, Revista Gestão Industrial p. 123-133, 2007

[28] L. Dias, V. Mousseau, J. Figueira, and J. Climaco. An aggregation/disaggregation approach to obtain robust conclusions with electre tri. European journal of Operational Research 138:332-348, 2002.

[29] Matos, Manuel. Ajuda multicriteria à decisão – Introdução. FEUP, 2005

[30] A. Sharma, C.W. Omlin. Performance Comparison of Particle Swarm Optimization with Traditional Clustering Algorithms used in Self-Organizing Map. World Academy of Science, Engineering and Technology 51, 2009.

[31] A. J. Hartugan, Clustering algorithms. New York: J. Wiley, 1975.

[32] B. Mirkin, Mathematical Classification and Clustering. Dordrecht: Kluwer Academic Publishers, 1996.

[33] B. Everitt. Cluster Analysis. Heinemann Educational, 1974.

[34] B. Everitt. Cluster Analysis (3rd ed.). Edward Arnold, 1993.

[35] L. Kaufman and P. Rousseeuw. Finding Groups in Data : An introduction to Cluster Analysis. John Wiley and Sons, New York, 1990.

[36] R. Steuer. Multiple criteria optimization : theory, computation and application. Wiley, New York, 1986.

[37] Ph. Vincke. Multicriteria Decision Aid. J. Wiley, 1992.

[38] B. Roy. Critéres multiples et modélisation des préférences : l'apport des relations de surclassement. Revue d'Economie Politique, 1974.

[39] B. Roy. Méthodologie multicritére d'aide à la décision. Econmica, Paris, 1985.

[40] Richard O. Duda, Peter E. Hart, David G. Stork. Pattern Classification (2nd edition) 2000.

Appendix

Appendix A

In this appendix we present the source code, made in Java programming language, for the approach proposed by us in Section 5.3 using as basis Promethee I outranking method.

```
public class proposed_approach_using_Promethee_I {

        public static double[][] partialpref (double x [][],double z [][], double  p[], double
q[],int c){
                // calculation of partial preference

                double partpref [][]= new double[x.length][z.length];
                int i=0;
                int j=0;

                while(i<x.length){
                        j=0;

                        while(j<z.length){

                                if(z[i][c]-x[j][c]<=q[c] || z[i][c]<=x[j][c])partpref[i][j]=0;
                                if(z[i][c]-x[j][c]> p[c])partpref[i][j]=1;
                                if((z[i][c]-p[c]) <= x[j][c] && x[j][c] < (z[i][c]-q[c]
))partpref[i][j]=((z[i][c]-x[j][c]- q[c])/(p[c]-q[c]));
                                j++;
                        }
                        i++;
                }
                return partpref;
        }

        public static double[][] overallpref (double x [][],double z [][], double  p[], double
q[], double w []){
                //calculation of overall preference

                double partpref[][] = new double[x.length][z.length];
                double prefmat[][] = new double[x.length][z.length];
                int c=0;

                while(c<x[0].length)
```

```
                        {
                        if (c==0){
                                partpref=partialpref(x,z,p,q,c);
                                int i=0,j=0;
                                while (i<partpref.length){
                                        j=0;
                                        while(j<partpref.length){
                                                prefmat[i][j]=w[c]*partpref[i][j];
                                                j++;
                                        }
                                i++;
                                }
                        }
                        else if (c>0 && c<x[0].length){
                                partpref=partialpref(x,z,p,q,c);
                                int i=0,j=0;
                                while (i<partpref.length){
                                        j=0;
                                        while(j<partpref.length){

        prefmat[i][j]=prefmat[i][j]+w[c]*partpref[i][j];
                                                j++;
                                        }
                                i++;
                                }
                        }
                        c++;
                }

                return prefmat;
        }

        public static double[] flowplus (double x [][],double z [][], double  p[], double  q[],
double w []){
                //calculation of positive flow

                double flow[] = new double[x.length];
                double overallpref[][] = new double[x.length][x.length];
                overallpref=overallpref(x,x,p,q,w);

                int n=0;
                while (n<x.length) {flow[n]=0;n++;}

                int i=0;
                while(i<x.length) {

                        int j=0;
                        while (j<z.length){

                                flow[i]=flow[i]+overallpref[i][j];

                                j++;
                        }

                        i++;
                }

                return flow;
        }
```

```
	public static double[] flowminus (double x [][],double z [][], double  p[], double  q[],
double w []){
		//calculation of negative flow

		double flow[] = new double[z.length];
		double overallpref[][] = new double[x.length][x.length];
		overallpref=overallpref(x,x,p,q,w);

		int n=0;
		while (n<x.length) {flow[n]=0;n++;}

		int i=0;
		while(i<x.length) {

			int j=0;
			while (j<z.length){

				flow[i]=flow[i]+overallpref[j][i];

				j++;
			}

			i++;
		}

		return flow;
	}

	public static double[] flow (double x [][],double z [][], double  p[], double  q[],
double w []){
		//calculation of the overall flow

		double flow[] = new double[z.length];
		double flowp[] = new double[z.length];
		double flowm[] = new double[z.length];

		flowp=flowplus(x,z,p,q,w);
		flowm=flowminus(x,z,p,q,w);

		int i=0;
		while(i<x.length) {
			flow[i]=flowp[i]-flowm[i];
			i++;
		}

		return flow;
	}


	public static int[][] pji (double x [][],double z [][], double  p[], double  q[], double w
[]){
		// calculation of  the outranking relation matrix
		int pji[][] = new int[x.length][z.length];
		double flow[] = new double[z.length];
		double flowp[] = new double[z.length];
		double flowm[] = new double[z.length];

		flowp=flowplus(x,z,p,q,w);
		flowm=flowminus(x,z,p,q,w);
```

```
                flow=flow(x,z,p,q,w);

                int i=0;
                while(i<flow.length) {

                        int j=0;
                        while (j<flow.length){

                                if (flowp[i]==flowp[j] && flowm[i]==flowm[j])  {pji[i][j]=2 ;
pji[j][i]=2;}   //2 means indifferent
                                else if ((flowp[i]>flowp[j] && flowm[i]<flowm[j]) ||
                                                (flowp[i]>flowp[j]  && flowm[i]==flowm[j])||
                                                (flowp[i]==flowp[j] && flowm[i]<flowm[j]))
                                                {pji[i][j]=4;                              //4
means that the alternative i is better than j
                                                 pji[j][i]=1;}                             //1
means that the alternative j is better than i
                                else if(pji[j][i]==0){pji[i][j]=3 ; pji[j][i]=3;}    //3 means
incomparable
                                j++;
                        }
                        i++;
                }

                return pji;
        }



        public static int [] split (double x [][],double z [][], double  p[], double  q[],double w
[],int c){
                // split procedure of the proposed approach

                int pji[][] = new int[x.length][z.length];
                int newsplit[] = new int[x.length];
                int split[] = new int[x.length];
                int i=0,j=0,n=0;
                pji=pji(x,x,p,q,w);
                i=0;

                while (n<x.length) {split[n]=1;n++;}
                n=0;
                while (n<x.length) {newsplit[n]=1;n++;}

                while (i<x.length){
                        j=0;
                        n=0;
                        while(j<x.length){
                                if(j>i){
                                        if(pji[i][j]==1 ){
                                                if (split [i]==split[j]){
                                                        n=0;
                                                        while (n<x.length){
                                                                if (split[n]>split[j]){

        newsplit[n]=split[n]+1;
                                                                }
                                                        n++;}
                                                        newsplit[j]=split[j]+1;
                                                }
                                        }
                                }

                        j++;
                        }
                        n=0;
                        while (n<x.length) {split[n]=newsplit[n];n++;}
                        //printvector(split);

                        j=0;
                        while(j<x.length){
                                if(j>i){
                                        if(pji[i][j]==4 ){
                                                if (split [i]==split[j]){
                                                        n=0;
                                                        while (n<x.length){
                                                                if (split[n]>split[j]
)newsplit[n]=split[n]+1;
                                                                else if (
split[n]<split[j])newsplit[n]=split[n];
                                                                else if ( split[n]==split[i] &&
pji[i][n]!=4 && n>=i )newsplit[n]=split[n]+1;
                                                                else if ( split[n]==split[i] &&
pji[i][n]!=4 && n>i)newsplit[n]=split[n];
                                                        n++;
                                                        }
                                                }
                                        }
                                }
                        j++;
                        }
                        n=0;
                        while (n<x.length) {split[n]=newsplit[n];n++;}
                        //printvector(split);
                i++;
                }
                return split;
        }



        public static double homogeneity (double x [][],double z [][], double  p[], double
q[],double w [],int c, int i, int j){
                // calculation of homogeneity index for all pairs of actions
                double homogeneity;
                double pref[][]= new double [x.length][x.length];

                pref=overallpref(x,x,p,q,w);
                homogeneity=pref[i][j];

                return homogeneity;
        }

        public static int max(int[] t) {
           int maximum = t[0];   // start with the first value
           for (int i=1; i<t.length; i++) {
              if (t[i] > maximum) {
                 maximum = t[i];   // new maximum
              }
        }
        return maximum;
    }


    public static int posmin(double[] t) {   // calculation of the position in the minimum
of a vector double
        double min = t[0];
        int minpos = 0;
        for (int i=1; i<t.length; i++) {
            if (t[i] < min) {
                min = t[i];
                minpos=i;
                }
        }
        return minpos;
    }


    public static int [] merge (double x [][],double z [][], double  p[], double  q[],double
w [],int c, int k){
                // merge procedure of the proposed approach

                int merge []= new int [x.length];
                double homogeneity;
                double maxhomo;
                int i,j,t,y,a;
                merge=split (x,x,p,q,w,c);
                homogeneity=0;
                int m;

                while(max(merge)>k){
                        m=1;
                        double memmax[]=new double[max(merge)-1];
                        a=0;
                        while(a<max(merge)-1){
                                int count=0;
                                int mem [];

                                i=0;
                                while(i<merge.length){
                                        if(merge[i]==m || merge[i]==m+1){ count++;}
                                        i++;
                                 }

                                mem=new int[count];
                                i=0;
                                j=0;

                                while(i<merge.length){
                                        if(merge[i]==m || merge[i]==m+1){ mem[j]=i;j++;}
                                        i++;
                                }
                                //printvector(mem);

                                t=0;
                                maxhomo=0;
                                while(t<mem.length){
                                        y=0;
                                        homogeneity=0;
```

```
                                                while(y<mem.length){
                                                                homogeneity=homogeneity
(x,x,p,q,w,c,mem[t],mem[y]);
                                                                if(maxhomo<homogeneity)
maxhomo=homogeneity;
                                                                y++;
                                                }
                                                t++;
                                        }

                                memmax[m-1]=maxhomo;
                                a++;
                                m++;
                                }

                                int n=0;
                                while (n<x.length){
                                        if (merge[n]>posmin(memmax)+1){
                                                merge[n]=merge[n]-1;
                                        }
                                n++;
                                }
                                //printvector(merge);
                        }
                                return merge;
                }


                public static void printmatrix(double[][] array) {  //print matrix double
                    int rowSize = array.length;
                    int columnSize = array[0].length;

                    for(int i = 0; i < rowSize; i++) {
                                System.out.print("[");
                        for(int j = 0; j < columnSize; j++) {
                                System.out.print(" " + array[i][j]);
                                }
                                System.out.println(" ]");
                        }
                                    System.out.println();
                }
                public static void printmatrixint(int[][] array) {  //print matrix int
                    int rowSize = array.length;
                    int columnSize = array[0].length;
                    for(int i = 0; i < rowSize; i++) {
                        System.out.print("[");
                        for(int j = 0; j < columnSize; j++) {
                            System.out.print(" " + array[i][j]);
                            }
                        System.out.println(" ]");
                        }
                            System.out.println();
                 }

                public static void printvector(int[] array) { // print vector int
                    int rowSize = array.length;
                    System.out.print("[");
                    for(int i = 0; i < rowSize; i++) {
```

```
                System.out.print(" " + array[i]);
                }
        System.out.println(" ]");
        System.out.println();
     }

    public static void printvectord(double[] array) { //print vector double
        int rowSize = array.length;
        System.out.print("[");
        for(int i = 0; i < rowSize; i++) {

                System.out.print(" " + array[i]);

                }
        System.out.println(" ]");
        System.out.println();
     }


    public static void main(String[] args) {


    double x [][]={ {0.188,0.172,0.168,0.122,0.114},{0.125,0.069,0.188,0.244,0.205},
                        {0.156,0.241,0.134,0.22,0.136},{0.188,0.034,0.174,0.146,0.159},
                        {0.188,0.276,0.156,0.171,0.205},{0.156,0.207,0.18,0.098,0.182} };

    double p[] ={0.05,0.05,0.05,0.05,0.05}; double q[] ={0.01,0.01,0.01,0.01,0.01};  int
k=3;
    double w []= {0.25,0.25,0.1,0.2,0.2};

    double x2
[][]={{16,15,40,12,15,5,3},{45,92,85,16,16,5,5},{21,62,24,16,12,5,3},{21,25,50,10,12,3,5},{4,
12,15,4,2,2,1},{5,30,60,10,15,1,2},{6,25,25,4,16,4,5},{40,80,60,16,12,4,5},{10,20,30,8,8,2,1}
,{21,19,80,18,16,4,2 },{  10,4,47,11,15,4,2}  ,  {45,85,85,15,15,5,5}   ,{15,16,72,15,18,4,2}
,{18,20,47,12,14,4,4}  , {35,70,60,10,10,3,3}};

    //double q[]={4,10,10,2,2,0,0};double p[]={8,15,15,4,4,1,1}; int k=5;
    ///double w[]={0.20,0.15,0.1,0.1,0.1,0.15,0.2};

    double x3 [][]= {{  35.8 ,67, -19.7 ,0 ,0, 5, 4} , {16.4 ,14.5 ,-59.8 ,-7.5 ,-5.2 ,5 ,3 },
{35.8 ,24 ,-64.9 ,-2.1,- 4.5, 5 ,4 }, {20.6 ,61.7 ,-75.7 ,-3.6 ,-8 ,5 ,3 }, {11.5 ,17.1, -57.1 ,-4.2
,-3.7, 5, 2},  {22.4 ,25.1 ,-49.8 ,-5 ,-7.9, 5 ,3 }, {23.9 ,34.5 ,-48.9 ,-2.5 ,-8 ,5, 3 },{29.9, 44, -
57.8 ,-1.7,- 2.5 ,5 ,4 },  {8.7 ,5.4, -27.4, -4.5, -4.5 ,5 ,2 },{25.7, 29.7, -46.8,- 4.6,- 3.7, 4, 2}
,{ 21.2 ,24.6 ,-64.8 ,-3.6 ,-8 ,4 ,2 },{ 18.3 ,31.6 ,-69.3 ,-2.8 ,-3 ,4 ,3 }, {20.7 ,19.3 ,-19.7 ,-2.2
,-4 ,4 ,2 }, {9.9, 3.5 ,-53.1 ,-8.5 ,-5.3 ,4,2 }, {10.4, 9.3 ,-80.9, -1.4 ,-4.1 ,4 ,2 },{17.7 ,19.8 ,-
52.8 ,-7.9 ,-6.1 ,4 ,4 },   {14.8 ,15.9 ,-27.9 ,-5.4 ,-1.8 ,4 ,2 }, {16, 14.7 ,-53.5 ,-6.8 ,-3.8 ,4, 4
}, {11.7 ,10 ,-42.1 ,-12.2,- 4.3 ,5 ,2 },    {11 ,4.2 ,-60.8 ,-6.2 ,-4.8 ,4 ,2 }, {15.5 ,8.5 ,-56.2 ,-
5.5 ,-1.8, 4 ,2}, {13.2 ,9.1 ,-74.1 ,-6.4 ,-5 ,2 ,2 }, {9.1 ,4.1 ,-44.8 ,-3.3, -10.4 ,3 ,4},  {12.9
,1.9 ,-65 ,-14 ,-7.5, 4 ,3 }, {5.9, -27.7 ,-77.4 ,-16.6 ,-12.7 ,3 ,2 }, {16.9 ,12.4 ,-60.1, -5.6 ,-5.6
,3 ,2 }, {16.7 ,13.1 ,-73.5 ,-11.9 ,-4.1 ,2 ,2 }, {14.6 ,9.7 ,-59.5 ,-6.7 ,-5.6 ,2 ,2 }, {5.1 ,4.9 ,-
28.9 ,-2.5 ,-46 ,2 ,2 },    {24.4 ,22.3 ,-32.8 ,-3.3 ,-5 ,3 ,4 }, {29.5 ,8.6 ,-41.8 ,-5.2 ,-6.4, 2, 3
}, {7.3 ,-64.5 ,-67.5 ,-30.1, -8.7, 3, 3} , {23.7 ,31.9 ,-63.6, -12.1 ,-10.2 ,3 ,2}, {18.9 ,13.5 ,-
74.5 ,-12 ,-8.4 ,3 ,3 }, {13.9 ,3.3 ,-78.7, -14.7, -10.1 ,2, 2} ,{-13.3 ,-31.1 ,-63 ,-21.2 ,-29.1 ,2
,1}, {6.2 ,-3.2 ,-46.1 ,-4.8 ,-10.5 ,2 ,1 }, {4.8 ,-3.3 ,-71.1 ,-8.6 ,-11.6 ,2 ,2 }, {0.1 ,-9.6 ,-42.5
,-12.9 ,-12.4 ,1 ,1}, {13.6 ,9.1 ,-76, -17.1 ,-10.3 ,1 ,1 }};

    //double q[]={1,4,1,1,3,0,0};double p[]={2,6,3,2,4,0,0}; int k=5;
    //double w []= {0.01,0.295,0.225,0.01,0.225,0.01,0.225};
```

```
        double s[][] = new double[x.length][x.length];
        int pij[][] = new int[x.length][x.length];

        double flow[]=new double[x.length];
        double flowp[]=new double [x.length];
        double flowm[]=new double [x.length];

        double prefmat[][] = new double[x.length][x.length];
        double partialprefmat[][] = new double[x.length][x.length];
        int c=0;
        int split []=new int [x.length];
        int merge []=new int [x.length];
        double homogeneity;



        //partialprefmat=partialpref(x,x,p,q,c);printmatrix(partialprefmat);

        //prefmat=overallpref(x,x,p,q,w);printmatrix(prefmat);

        //pij=pji(x,x,p,q,w); printmatrixint(pij);

        //split=split (x,x,p,q,w,c );printvector(split);

        //homogeneity=homogeneity (x,x,p,q,w,c ,0,1); System.out.print(homogeneity);

        //flow=flow(x,x,p,q,w); printvectord(flow);

        //flowm=flowminus(x,x,p,q,w); printvectord(flowm);

        //flowp=flowplus(x,x,p,q,w); printvectord(flowp);

        merge=merge(x,x,p,q,w,c,k);printvector(merge);

        }
}
```

Appendix B

In this appendix we present the source code, made in Java programming language, for the approach proposed by us in Section 5.3 using as basis Electre III outranking method.

```
public class proposed_approach_using_Electre_III {

        public static double[][] partialconc (double x [][],double z [][], double  p [], double  q
[], double v [],int c){
                // calculation of partial concordance

                double matrix [][]= new double[x.length][z.length];
                int i=0;
                int j=0;

                while(i<x.length){
                        j=0;

                        while(j<z.length){

                                if(z[j][c]-x[i][c]>p[c])matrix[i][j]=0;
                                if(z[j][c]-x[i][c]<=q[c])matrix[i][j]=1;
                                if((z[j][c]-p[c]) <= x[i][c] && x[i][c] < (z[j][c]-q[c]
))matrix[i][j]=((p[c]-z[j][c]+x[i][c])/(p[c]-q[c]));

                                j++;
                        }i++;
                }
                return matrix;
        }

        public static double[][] overallconc (double x [][],double z [][], double  p[], double
q[], double v[],double w []){
                // calculation of overall concordance

                double cpart[][] = new double[x.length][z.length];
                double coverall[][] = new double[x.length][z.length];
                int c=0;

                while(c<x[0].length)
                        {
                        if (c==0){
                                cpart=partialconc(x,z,p,q,v,c);
                                int i=0,j=0;
                                while (i<cpart.length){
                                        j=0;
                                        while(j<cpart.length){
                                                coverall[i][j]=w[c]*cpart[i][j];
                                                j++;
                                        }
                                i++;
                                }
                        }
                        else if (c>0 && c<x[0].length){
```

```
					cpart=partialconc(x,z,p,q,v,c);
					int i=0,j=0;
					while (i<cpart.length){
						j=0;
						while(j<cpart.length){
							coverall[i][j]=coverall[i][j]+w[c]*cpart[i][j];
							j++;
						}
					i++;
					}
				}
				c++;
			}

			return coverall;
		}


		public static double[][] partialdisc (double x [][],double z [][], double  p[], double
q[], double v[],int c){
			//calculation of partial discordance

			double matrix [][]= new double[x.length][z.length];
			int i=0;
			int j=0;

			while(i<x.length){
				j=0;

				while(j<z.length){

					if(z[j][c]-x[i][c]<=p[c])matrix[i][j]=0;
					if(z[j][c]-x[i][c]> v[c])matrix[i][j]=1;
					if((z[j][c]-v[c]) <= x[i][c] && x[i][c] < (z[j][c]-p[c]
))matrix[i][j]=((z[j][c]-x[i][c]- p[c])/(v[c]-p[c]));
					j++;
				}
				i++;
			}
			return matrix;
		}

		public static double[][] outindex (double x [][],double z [][], double  p[], double  q[],
double v[],double w []){
		//calculation of the outranking index matrix
			double s [][]= new double[x.length][z.length];

			double cpart[][] = new double[x.length][z.length];
			double coverall[][] = new double[x.length][z.length];
			double partialdisc[][] = new double[x.length][z.length];
			int c=0;
			int i=0;
			while(i<cpart.length) {

				int j=0;
				while (j<cpart.length){
					c=0;
					coverall=overallconc(x,z,p,q,v,w);
```

```
                                        while(c<x[0].length){
                                                cpart=partialconc(x,z,p,q,v,c);
                                                partialdisc=partialdisc(x,z,p,q,v,c);
                                                if (partialdisc[i][j]>cpart[i][j])
coverall[i][j]=coverall[i][j]*(1-partialdisc[i][j])/(1-cpart[i][j]);
                                                else coverall[i][j]=coverall[i][j];
                                                c++;
                                        }

                                        s[i][j]=coverall[i][j];
                                        j++;
                                }

                                i++;
                        }


                        return s;
                }




                public static int[][] pji (double x [][],double z [][], double  p[], double  q[], double
v[],int c, double lambda, double w[]){
                //calculation of outranking relation matrix

                        double s [][]= new double[x.length][z.length];
                        s=outindex(x,x,p,q,v,w);

                        int pji[][] = new int[x.length][z.length];

                        int i=0;
                        while(i<s.length) {

                                int j=0;
                                while (j<s.length){

                                        if (s[i][j]>=lambda && s[j][i]>= lambda)  {pji[i][j]=2 ;
pji[j][i]=2;}   //2 means indifferent
                                        else if (s[i][j]<lambda && s[j][i]>= lambda) pji[i][j]=1 ;
                                //1 means that the alternative j is better than i
                                        else if (s[i][j]>=lambda && s[j][i]< lambda) pji[i][j]=4 ;
                                //4 means that the alternative i is better than j
                                        else if (s[i][j]<lambda && s[j][i]< lambda) {pji[i][j]=3 ;
pji[j][i]=3;}  //3 means incomparable
                                        //if (s[i][j]==1 && s[j][i]==1) pji[i][j]=0 ; //just a test
                                        j++;
                                }

                                i++;
                        }



                        return pji;
                }

                public static double[][] partialpref (double x [][],double z [][], double  p[], double
q[], double v[],int c){
```

```
                //calculation of partial preference

                double partpref [][]= new double[x.length][z.length];
                int i=0;
                int j=0;

                while(i<x.length){
                        j=0;


                        while(j<z.length){

                                //if(z[j][c]-x[i][c]<=q[c])partpref[i][j]=0;
                                //if(z[j][c]-x[i][c]> p[c])partpref[i][j]=1;
                                //if((z[j][c]-p[c]) <= x[i][c] && x[i][c] < (z[j][c]-q[c]
))partpref[i][j]=((z[j][c]-x[i][c]- q[c])/(p[c]-q[c]));

                                if(z[i][c]-x[j][c]<=q[c] || z[i][c]<=x[j][c])partpref[i][j]=0;
                                if(z[i][c]-x[j][c]> p[c])partpref[i][j]=1;
                                if((z[i][c]-p[c]) <= x[j][c] && x[j][c] < (z[i][c]-q[c]
))partpref[i][j]=((z[i][c]-x[j][c]- q[c])/(p[c]-q[c]));
                                j++;
                        }
                        i++;
                }
                return partpref;
        }

        public static double[][] overallpref (double x [][],double z [][], double  p[], double
q[], double v[],double w []){
                //calculation of overall preference

                double partpref[][] = new double[x.length][z.length];
                double prefmat[][] = new double[x.length][z.length];
                int c=0;

                while(c<x[0].length)
                        {
                        if (c==0){
                                partpref=partialpref(x,z,p,q,v,c);
                                int i=0,j=0;
                                while (i<partpref.length){
                                        j=0;
                                        while(j<partpref.length){
                                                prefmat[i][j]=w[c]*partpref[i][j];
                                                j++;
                                        }
                                i++;
                                }
                        }
                        else if (c>0 && c<x[0].length){
                                partpref=partialpref(x,z,p,q,v,c);
                                int i=0,j=0;
                                while (i<partpref.length){
                                        j=0;
                                        while(j<partpref.length){

        prefmat[i][j]=prefmat[i][j]+w[c]*partpref[i][j];
                                                j++;
                                        }
                                i++;
                                }
                        }
                        c++;
                }

                return prefmat;
        }



        public static int [] split (double x [][],double z [][], double  p[], double  q[], double
v[],double w [],int c,  double lambda){
        // split procedure of the proposed approach

                int pji[][] = new int[x.length][z.length];
                int newsplit[] = new int[x.length];
                int split[] = new int[x.length];
                int i=0,j=0,n=0;
                pji=pji(x,x,p,q,v,c,lambda,w);
                i=0;

                while (n<x.length) {split[n]=1;n++;}
                n=0;
                while (n<x.length) {newsplit[n]=1;n++;}

                while (i<x.length){
                        j=0;

                        while(j<x.length){
                                if(j>i){
                                        if(pji[i][j]==1 ){
                                                if (split [i]==split[j]){
                                                        n=0;
                                                        while (n<x.length){
                                                                if (split[n]>split[j]){

        newsplit[n]=split[n]+1;
                                                                }
                                                        n++;}
                                                        newsplit[j]=split[j]+1;
                                                }
                                        }
                                }

                        j++;
                        }
                        n=0;
                        while (n<x.length) {split[n]=newsplit[n];n++;}
                        //printvector(split);
                        j=0;

                        while(j<x.length){
                                if(j>i){
                                        if(pji[i][j]==4 ){
                                                if (split [i]==split[j]){
                                                        n=0;
                                                        while (n<x.length){
```

```
                                                                                    if (split[n]>split[j]
)newsplit[n]=split[n]+1;
                                                                                    else if (
split[n]<split[j])newsplit[n]=split[n];
                                                                                    else if ( split[n]==split[i] &&
pji[i][n]!=4 && n>=i )newsplit[n]=split[n]+1;
                                                                                    else if ( split[n]==split[i] &&
pji[i][n]!=4 && n>i)newsplit[n]=split[n];
                                                                            n++;
                                                                            }
                                                                    }
                                                            }
                                                    }
                                            j++;
                                            }
                                            n=0;
                                            while (n<x.length) {split[n]=newsplit[n];n++;}
                                            //printvector(split);
                                    i++;
                                    }
                                    return split;
                            }


                            public static double homogeneity (double x [][],double z [][], double  p[], double  q[],
double v[],double w [],int c,  double lambda, int i, int j){
                                    // calculation of homogeneity index for all pairs of actions

                                    double homogeneity;
                                    double pref[][]= new double [x.length][x.length];

                                    pref=overallpref(x,x,p,q,v,w);
                                    homogeneity=pref[i][j];

                                    return homogeneity;
                            }

                            public static int max(int[] t) { // get the maximum of one vector
                                int maximum = t[0];   // start with the first value
                                for (int i=1; i<t.length; i++) {
                                    if (t[i] > maximum) {
                                        maximum = t[i];   // new maximum
                                    }
                                }
                                return maximum;
                            }


                            public static int posmin(double[] t) {
                                double min = t[0];
                                int minpos = 0;
                                for (int i=1; i<t.length; i++) {
                                    if (t[i] < min) {
                                        min = t[i];
                                        minpos=i;
                                        }
                                }
                                return minpos;
```

```
        }


        public static int [] merge (double x [][],double z [][], double  p[], double  q[], double
v[],double w [],int c,  double lambda, int k){
                // merge procedure of the proposed approach

                int merge []= new int [x.length];
                double homogeneity;
                double maxhomo;
                int i,j,t,y,a;
                merge=split (x,x,p,q,v,w,c, lambda);
                homogeneity=0;
                int m;

                while(max(merge)>k){
                        m=1;
                        double memmax[]=new double[max(merge)-1];
                        a=0;
                        while(a<max(merge)-1){
                                int count=0;
                                int mem [];

                                i=0;
                                while(i<merge.length){
                                        if(merge[i]==m || merge[i]==m+1){ count++;}
                                        i++;
                                 }
                                mem=new int[count];
                                i=0;
                                j=0;

                                while(i<merge.length){
                                        if(merge[i]==m || merge[i]==m+1){ mem[j]=i;j++;}
                                        i++;
                                }
                                //printvector(mem);
                                t=0;
                                maxhomo=0;


                                while(t<mem.length){
                                        y=0;
                                        homogeneity=0;
                                        while(y<mem.length){

                                                homogeneity=homogeneity (x,x,p,q,v,w,c,
lambda,mem[t],mem[y]);
                                                if(maxhomo<homogeneity)
maxhomo=homogeneity;
                                                y++;
                                        }
                                        t++;
                                }

                        memmax[m-1]=maxhomo;
                        a++;
                        m++;
                        }
                        int n=0;
                        while (n<x.length){
                                    if (merge[n]>posmin(memmax)+1){
                                                merge[n]=merge[n]-1;
                                    }
                        n++;
                        }
                        //printvector(merge);

            }
                        return merge;
      }

      public static void printmatrix(double[][] array) {  //print matrix double
          int rowSize = array.length;
          int columnSize = array[0].length;

          for(int i = 0; i < rowSize; i++) {
                        System.out.print("[");
                for(int j = 0; j < columnSize; j++) {
                        System.out.print(" " + array[i][j]);
                        }
                        System.out.println(" ]");
                }
                            System.out.println();
      }
      public static void printmatrixint(int[][] array) {  //print matrix int
          int rowSize = array.length;
          int columnSize = array[0].length;
          for(int i = 0; i < rowSize; i++) {
                System.out.print("[");
                for(int j = 0; j < columnSize; j++) {
                    System.out.print(" " + array[i][j]);
                    }
                System.out.println(" ]");
                }
                    System.out.println();
       }

      public static void printvector(int[] array) { // print vector int
          int rowSize = array.length;
          System.out.print("[");
          for(int i = 0; i < rowSize; i++) {

                System.out.print(" " + array[i]);
                }
          System.out.println(" ]");
          System.out.println();
       }

      public static void printvectord(double[] array) { //print vector double
          int rowSize = array.length;
          System.out.print("[");
          for(int i = 0; i < rowSize; i++) {

                System.out.print(" " + array[i]);

                }
          System.out.println(" ]");
```

```
        System.out.println();
      }


      public static void main(String[] args) {


      double x [][]={ {0.188,0.172,0.168,0.122,0.114},{0.125,0.069,0.188,0.244,0.205},
{0.156,0.241,0.134,0.22,0.136},{0.188,0.034,0.174,0.146,0.159},
{0.188,0.276,0.156,0.171,0.205},{0.156,0.207,0.18,0.098,0.182} };

      double p[] ={0.05,0.05,0.05,0.05,0.05}; double q[] ={0.01,0.01,0.01,0.01,0.01};
double v[] ={99,99,99,99,99,99,99};  int k=3;
      double w []= {0.25,0.25,0.1,0.2,0.2};

      double x2
[][]={{16,15,40,12,15,5,3},{45,92,85,16,16,5,5},{21,62,24,16,12,5,3},{21,25,50,10,12,3,5},{4,
12,15,4,2,2,1},{5,30,60,10,15,1,2},{6,25,25,4,16,4,5},{40,80,60,16,12,4,5},{10,20,30,8,8,2,1}
,{21,19,80,18,16,4,2 },{   10,4,47,11,15,4,2}  ,   {45,85,85,15,15,5,5}    ,{15,16,72,15,18,4,2}
,{18,20,47,12,14,4,4}  , {35,70,60,10,10,3,3}};

      //double q[]={4,10,10,2,2,0,0};double p[]={8,15,15,4,4,1,1}; double
v[]={99,99,99,99,99,99,99}; int k=5;
      ///double w[]={0.20,0.15,0.1,0.1,0.1,0.15,0.2};

      double x3 [][]= {{  35.8 ,67, -19.7 ,0 ,0, 5, 4} ,{16.4 ,14.5 ,-59.8 ,-7.5 ,-5.2 ,5 ,3 },
{35.8 ,24 ,-64.9 ,-2.1,- 4.5, 5 ,4 }, {20.6 ,61.7 ,-75.7 ,-3.6 ,-8 ,5 ,3 }, {11.5 ,17.1, -57.1 ,-4.2
,-3.7, 5, 2},  {22.4 ,25.1 ,-49.8 ,-5 ,-7.9, 5 ,3 },{23.9 ,34.5 ,-48.9 ,-2.5 ,-8 ,5, 3 },{29.9, 44, -
57.8 ,-1.7,- 2.5 ,5 ,4 },{8.7 ,5.4, -27.4, -4.5, -4.5 ,5 ,2 },{25.7, 29.7, -46.8,- 4.6,- 3.7, 4, 2} ,{
21.2 ,24.6 ,-64.8 ,-3.6 ,-8 ,4 ,2 },{ 18.3 ,31.6 ,-69.3 ,-2.8 ,-3 ,4 ,3 }, {20.7 ,19.3 ,-19.7 ,-2.2 ,-
4 ,4 ,2 }, {9.9, 3.5 ,-53.1 ,-8.5 ,-5.3 ,4,2 }, {10.4, 9.3 ,-80.9, -1.4 ,-4.1 ,4 ,2 },{17.7 ,19.8 ,-
52.8 ,-7.9 ,-6.1 ,4 ,4 },   {14.8 ,15.9 ,-27.9 ,-5.4 ,-1.8 ,4 ,2 }, {16, 14.7 ,-53.5 ,-6.8 ,-3.8 ,4, 4
}, {11.7 ,10 ,-42.1 ,-12.2,- 4.3 ,5 ,2 },    {11 ,4.2 ,-60.8 ,-6.2 ,-4.8 ,4 ,2 }, {15.5 ,8.5 ,-56.2 ,-
5.5 ,-1.8, 4 ,2}, {13.2 ,9.1 ,-74.1 ,-6.4 ,-5 ,2 ,2 }, {9.1 ,4.1 ,-44.8 ,-3.3, -10.4 ,3 ,4},  {12.9
,1.9 ,-65 ,-14 ,-7.5, 4 ,3 }, {5.9, -27.7 ,-77.4 ,-16.6 ,-12.7 ,3 ,2 }, {16.9 ,12.4 ,-60.1, -5.6 ,-5.6
,3 ,2 }, {16.7 ,13.1 ,-73.5 ,-11.9 ,-4.1 ,2 ,2 }, {14.6 ,9.7 ,-59.5 ,-6.7 ,-5.6 ,2 ,2 }, {5.1 ,4.9 ,-
28.9 ,-2.5 ,-46 ,2 ,2 },{24.4 ,22.3 ,-32.8 ,-3.3 ,-5 ,3 ,4 }, {29.5 ,8.6 ,-41.8 ,-5.2 ,-6.4, 2, 3 },
{7.3 ,-64.5 ,-67.5,-30.1, -8.7, 3, 3} , {23.7 ,31.9 ,-63.6, -12.1 ,-10.2 ,3 ,2},  {18.9 ,13.5 ,-74.5
,-12 ,-8.4 ,3 ,3 }, {13.9 ,3.3 ,-78.7, -14.7, -10.1 ,2, 2} ,{-13.3 ,-31.1 ,-63 ,-21.2 ,-29.1 ,2 ,1},
{6.2 ,-3.2 ,-46.1 ,-4.8 ,-10.5 ,2 ,1 }, {4.8 ,-3.3 ,-71.1 ,-8.6 ,-11.6 ,2 ,2 }, {0.1 ,-9.6 ,-42.5 ,-
12.9 ,-12.4 ,1 ,1}, {13.6 ,9.1 ,-76, -17.1 ,-10.3 ,1 ,1 }};

      //double q[]={1,4,1,1,3,0,0};double p[]={2,6,3,2,4,0,0}; double v[]={2,6,3,2,4,0,0};
int k=5;
      //double w []= {0.01,0.295,0.225,0.01,0.225,0.01,0.225};

      double x4 [][]={ {0.172,0.122},{0.0690,0.244},
                       {0.241,0.22},{0.034,0.146},
                       {0.276,0.171},{0.207,0.098} };
      //double w []= {0.5,0.5};
      //double p[] ={0.05,0.05}; double q[] ={0.01,0.01}; double v[] ={1,1}; int k=3;

      double s[][] = new double[x.length][x.length];
      int pij[][] = new int[x.length][x.length];


      double prefmat[][] = new double[x.length][x.length];
      double partialprefmat[][] = new double[x.length][x.length];
      int c=0;
      int split []=new int [x.length];
```

```
        int merge []=new int [x.length];
        double homogeneity;



        double lambda=0.6;

        double coverall[][] = new double[x.length][x.length];
        double partdisc[][] = new double[x.length][x.length];
        double partconc[][] = new double[x.length][x.length];
        double s[][] = new double[x.length][x.length];
        int pij[][] = new int[x.length][x.length];

        double prefmat[][] = new double[x.length][x.length];
        double partialprefmat[][] = new double[x.length][x.length];

        int split []=new int [x.length];
        int merge []=new int [x.length];
        double homogeneity;

        //coverall=overallconc(x,x,p,q,v,w);printmatrix(coverall);

        //partconc=partialconc(x,x,p,q,v,c); printmatrix(partconc);

        //partdisc=partialdisc(x,x,p,q,v,c);printmatrix(partdisc);

        //s=outindex(x,x,p,q,v,w);printmatrix(s);

        //pij=pji(x,x,p,q,v,c,lambda,w); printmatrixint(pij);

        //partialprefmat=partialpref(x,x,p,q,v,c);printmatrix(partialprefmat);

        //prefmat=overallpref(x,x,p,q,v,w);printmatrix(prefmat);

        //split=split (x,x,p,q,v,w,c, lambda);printvector(split);

        //homogeneity=homogeneity (x,x,p,q,v,w,c, lambda,0,1);
System.out.print(homogeneity);

        merge=merge(x,x,p,q,v,w,c, lambda,k);printvector(merge);



        }
    }
```