FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

REDUCING THE NUMBER OF MEMBERSHIP FUNCTIONS IN LINGUISTIC VARIABLES

Margarida Santos Mattos Marques Gomes

Dissertation presented at Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia in fulfilment of the requirements for the Masters degree in Mathematics and Applications, specialization in Actuarial Sciences, Statistics and Operations Research

Supervisor: Paula Alexandra da Costa Amaral Jorge
Co-supervisor: Rita Almeida Ribeiro

Lisboa
2009

Acknowledgments

I would like to thank Professor Rita Almeida Ribeiro for inviting me to work at CA3/Uninova in the context of the MODI project and for accepting to co-supervise this thesis. The work developed during this project was the basis of this thesis. Thank you for introducing me to the fuzzy world.

I want to express my gratitude to Professor Paula Amaral, for her excellent guidance and important contribution to overcome some of the difficulties encountered through this thesis.

To all my family and friends, especially my friends at CA3, thank you for the emotional support that gave me strength to finish the thesis.

Abstract

The purpose of this thesis was to develop algorithms to reduce the number of membership functions in a fuzzy linguistic variable. Groups of similar membership functions to be merged were found using clustering algorithms. By “summarizing” the information given by a similar group of membership functions into a new membership function we obtain a smaller set of membership functions representing the same concept as the initial linguistic variable.

The complexity of clustering problems makes it difficult for exact methods to solve them in practical time. Heuristic methods were therefore used to find good quality solutions. A Scatter Search clustering algorithm was implemented in Matlab and compared to a variation of the K-Means algorithm. Computational results on two data sets are discussed.

A case study with linguistic variables belonging to a fuzzy inference system automatically constructed from data collected by sensors while drilling in different scenarios is also studied. With these systems already constructed, the task was to reduce the number of membership functions in its linguistic variables without losing performance. A hierarchical clustering algorithm relying on performance measures for the inference system was implemented in Matlab. It was possible not only to simplify the inference system by reducing the number of membership functions in each linguistic variable but also to improve its performance.

Resumo

O objectivo desta tese era desenvolver algoritmos para reduzir o número de funções de pertença numa variável linguística. Foram usados algoritmos de agrupamento ou *clustering* para encontrar grupos de funções de pertença semelhantes. Concentrando a informação dada por um grupo de funções de pertença semelhantes numa nova função de pertença obtém-se um conjunto mais reduzido de funções de pertença que representam o mesmo conceito que a variável linguística original.

Dada a complexidade computacional dos problemas de agrupamento, métodos exactos para a resolução de problemas de programação inteira apenas conseguem encontrar uma solução óptima em tempo útil para pequenas instâncias. Assim, foram usados métodos heurísticos para encontrar boas soluções. Foi implementado em Matlab um algoritmo do tipo *Scatter Search* e este foi comparado com uma variante do algoritmo *K-Means*. São apresentados resultados computacionais para dois casos de estudo.

É também apresentado um caso de estudo em que as variáveis linguísticas pertencem a um sistema de inferência previamente construído a partir de dados recolhidos por sensores. O objectivo era reduzir o número de funções de pertença das suas variáveis linguísticas sem comprometer o desempenho do sistema. Foi implementado em Matlab um algoritmo de agrupamento hierárquico que tem em conta medidas de desempenho do sistema de inferência. Para além de ter sido possível simplificar o sistema, a redução do número de funções de pertença levou a um aumento do desempenho do próprio sistema, através da remoção de alguma redundância existente no sistema inicial.

Table of Contents

List of Figures

List of Tables

Introduction

In human reasoning many concepts are not crisp in the sense of being completely true or false, instead they can be interpreted in a more qualitative way. In everyday life we use concepts like tall, small, fast, slow, good, bad … that are difficult to translate numerically. Classical logic and inference have been insufficient to deal with these apparently vague concepts. Although humans reason with these concepts in a natural way on a daily basis, our search for scientific knowledge has lead us to address the problem of representing these concepts in a more systematic and precise way. As Engelbrecht [Engelbrecht 2002] states, “In a sense, fuzzy sets and logic allow the modelling of common sense”.

Since 1965, when Zadeh first formalized the concept of fuzzy set [Zadeh 1965], the field of fuzzy logic and approximate reasoning has attracted the interest of the scientific community. Fuzzy set theory and fuzzy logic concepts have been applied in almost all fields, from decision making to engineering [Costa, Gloria et al. 1997; Ross 2004], from medicine [Adlassnig 1986] to pattern recognition and clustering [Nakashima, Ishibuchi et al. 1998].

In engineering, fuzzy logic has been used, for instance, in monitoring and classification applications [Isermann 1998; Ribeiro 2006]. The main goal when constructing a fuzzy monitoring system is to develop a fuzzy inference system (FIS) [Lee 1990a; Lee 1990b] to monitor certain variables and warn decisors (or an automatic system) when variables behaviour is not correct, so that they can intervene. For the development of monitoring systems, in general, a formal and precise mathematical understanding of the underlying process is usually needed. These mathematical models may become too complex to formalize or to implement, reducing the advantage of an automatic and independent system over a human expert. Once again, fuzzy knowledge can be used to overcome this problem, modelling complex systems by mimicking human thinking.

In decision making, for instance, the advantages of using fuzzy logic is even more evident. In many cases the processes behind a decision are too complex to be defined through a precise classical mathematical model and the underlying preferences and choices of decision makers have many uncertainties and are better represented through a fuzzy number. Although crisp decision models do exist, more and more papers and books propose the use of fuzzy sets and fuzzy models to deal with the underlying uncertainty [Anoop Kumar and Moskowitz 1991; Lai and Hwang 1994; Ribeiro 1996; Ross 2004].

The main idea when choosing a fuzzy model over a classical one is to obtain models that are less complex and easy to interpret. The trade off between interpretability and precision must be studied for each application. To achieve such interpretability, it is desirable that the linguistic variables in a fuzzy model [Zadeh 1975] are as intuitive as possible. This in addition to a search for computationally efficient models motivated the research of this master thesis. When linguistic variables are constructed directly from expert knowledge its interpretability is usually clearer. This is not the case when an automatic procedure is used to create the membership functions of a certain linguistic variable or when membership functions represent a single sample from a large data base. As an example consider a fuzzy set used to represent an agent preference between two alternatives and suppose the number of agents involved in the process to be modelled is considerably large.

The purpose of this thesis is to develop algorithms to reduce the number of membership functions in a linguistic variable. The problem of reducing the amount of data to be analysed, while maintaining as most information as possible from the original data, is not exclusive from fuzzy domains. Large crisp data sets often have to be clustered to become treatable [Hartigan 1975; Murtagh 1983; Everitt, Landau et al. 2001; Gan, Ma et al. 2007]. Clustering data corresponds to finding natural groups of data that represent similar objects. The same approach can be used to reduce the number of membership functions in linguistic variables. We start by identifying clusters of similar membership functions. If each cluster of membership functions can be “summarized” into a new membership function, we obtain a new and smaller set of membership functions that approximately represents the same concept as the initial linguistic variable. This will be the basic approach that will be developed during this thesis. The problem of reducing the number of membership functions in linguistic variables will be formulated as a clustering problem. Resulting clusters of membership functions will be merged in a way of “summarizing” the information contained in the original membership functions.

In Chapter 1 theoretical background that is needed to understand following development is presented. An introduction to fuzzy logic and fuzzy inference systems is described. Similarity measures and merging methods that will be used to reduce the number of membership functions in linguistic variables are also introduced.

Since, as stated before, the problem of reducing the number of membership functions in a linguistic variable can be stated as a clustering problem, Chapter 2 will present different approaches to the clustering problem in statistics and optimization and the state of the art. Also, some possible formulations to the clustering problem will be discussed.

The complexity of clustering problems makes it difficult for exact methods to solve them in practical time. Exact methods can only find an optimal solution in a reasonable amount of time for very small data sets, especially if the number of clusters is unknown. However, before deciding for heuristic methods, it is important to use exact methods to better understand the complexity of the problem at hands. Since it was never the purpose of this thesis to solve these problems through exact methods, Chapter 3 gives only a brief introduction to some of the exact methods used for combinatorial and integer programming.

When finding optimal solutions through exact procedures is too time consuming, it is still usually possible to find good quality solutions in a reasonable amount of time, using heuristic methods that take advantage of the problem structure to achieve good solutions (not necessarily optimal) in less computational time. Both a heuristic and a metaheuristic to solve the automatic clustering problem were implemented in Matlab. Chapter 4 describes these algorithms and presents computational results on two case studies. In both case studies several linguistic variables are pruned. These linguistic variables could later be used in a fuzzy inference system or any other fuzzy model. The model would be constructed taking into account the already clustered membership functions instead of the original ones.

Chapter 5 introduces another case study. This case study has different characteristics from those used in Chapter 5. In this case study linguistic variables belong to an already existing fuzzy inference system. Instead of using the algorithms from Chapter 4, a heuristic relying on measures of performance of the inference system is used. The work presented in this chapter was developed within the scope of project “MODI – Simulation of a Knowledge Enabled Monitoring and Diagnosis Tool for ExoMars Pasteur Payloads”[CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006], a CA3 – UNINOVA project for the European Space Agency [ESA] Aurora programme [ESA 2008]. In this project two inference systems were constructed: one for monitoring exploratory drilling processes and another capable of detecting the type of terrain being drilled. These systems were automatically constructed using data collected from sensors while drilling in different scenarios. With these systems already constructed, the task was to reduce the number of membership functions in its linguistic variables without losing performance. This project was on the origin of the development of the ideas presented in this thesis. The contribution to this project can also be found in [Gomes, Santos et al. 2008]. This paper summarizes the main results obtained when reducing the number of membership functions of MODI’s linguistic variables and was presented at the Eight International Conference on Application of Fuzzy Systems and Soft Computing (ICAFS-2008) in September 2008 in Helsinki, Finland.

Finally, Chapter 6 presents the conclusions of this thesis and some guidelines for future work.

Chapter 1. Preliminaries

In this Chapter we present the background on fuzzy set theory necessary to understand the results presented later.

Sections 1.1 and 1.2 introduce the main concepts of fuzzy logic and fuzzy inference systems. Formal definitions of the concepts of linguistic variable, fuzzy set, membership function and the most used operations on fuzzy sets are given. A description of the structure of a fuzzy inference system and of its underlying modules is also presented.

In section 1.3, analytical and R^p representations of some of the most common types of membership functions - triangular, trapezoidal and Gaussian membership functions – are introduced.

The notion of similarity or proximity between membership functions will be the main idea underneath the algorithms for reducing the number of membership functions in linguistic variables. Section 1.4 describes these concepts and presents the measures of proximity of fuzzy sets that will be used. After identifying the most similar membership functions, these will be merged to give rise to a new set of membership functions simultaneously as small and as representative of the original linguistic variable as possible. Section 1.5 presents some membership functions merging methods.

1.1 Fuzzy Logic

In crisp logic, if we want to categorize a group of individuals as tall, medium or small, we have to distribute those individuals into two disjoint sets, as in Figure 1.1, by a crisp rule. For instance, if the height of an individual is above 1.75m, the individual is tall, if the height is bellow 1.60m, the individual is short and otherwise the individual is medium.

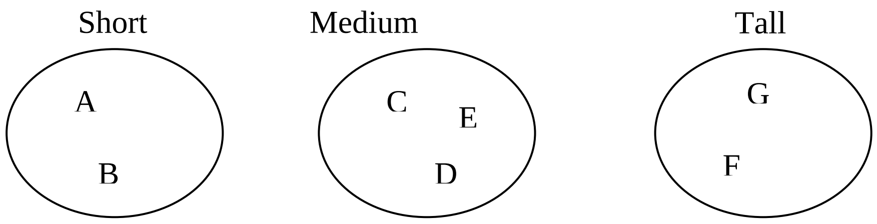


Figure 1.1: Concepts Short, Medium and Tall represented by Crisp Sets

This does not accurately represent human reasoning. In our mind, the frontier between these sets is not as well defined as in Figure 1.1. These concepts are better represented by fuzzy sets [Zadeh 1965], as in Figure 1.2. This representation allows for an individual to be considered simultaneously short and medium or medium and tall, with different degrees of membership. The definition of fuzzy set is given bellow.

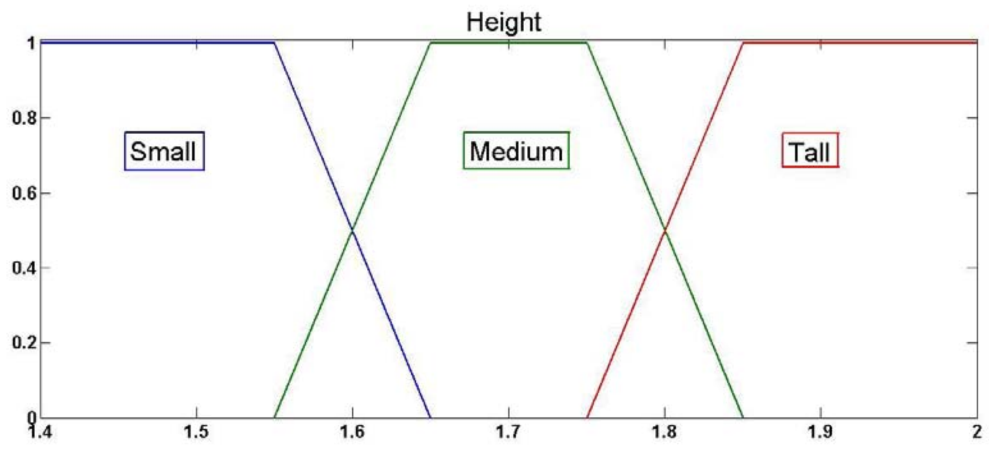


Figure 1.2: Concepts Short, Medium and Tall represented by Fuzzy Sets

Definition 1.1 [Zimmermann 1990] - If X is a collection of objects denoted generically by x then a fuzzy set $\tilde{A}$ in X is a set of ordered pairs:

$$\tilde{A} = \{(x, \mu_A(x)) : x \in X\} \tag{1.1}$$

where $\mu_{\tilde{A}}(x)$ is called the membership function or grade of membership of x in $\tilde{A}$ which maps X to the membership space M. The range of the membership function is a subset of nonnegative real numbers whose supremum is finite. Usually M is the real interval $[0,1]$.

♦

The representation of some common types of membership functions will be further presented in the next section.

Zadeh [Zadeh 1975] defines a linguistic variable as a quintuple $(x, T(x), U, G, M)$ in which x is the name of the variable; $T(x)$ is the term set of x, that is, the collection of its linguistic values; U is a universe of discourse; G is a syntactic rule which generates the terms in $T(x)$; and M is a semantic rule which associates with each linguistic value $T(x)$ its meaning, $M(X)$, where $M(X)$ denotes a subset of U.

The fuzzy sets in Figure 1.2 represent a linguistic variable Height.

T-norms and t-conorms generalize the idea of intersection and union of sets to fuzzy set theory.

Definition 1.2 [Klir and Yuan 1995] – A t-norm is a function $t:[0,1]\times[0,1]\to[0,1]$ satisfying the following properties:

Boundary Condition: $t(a,1)=a$ (1.2)

Monotonicity: $t(a,b)\le t(a,c) \quad \text{if} \quad b\le c$ (1.3)

Commutativity: $t(a,b)=t(b,a)$ (1.4)

Associativity: $t(a,t(b,c))=t(t(a,b),c)$ (1.5)

♦

Definition 1.3 [Klir and Yuan 1995] – A t-conorm or s-norm is a function $u:[0,1]\times[0,1]\to[0,1]$ satisfying the following conditions:

Boundary Condition: $u(a,0)=a$ (1.6)

Monotonicity: $u(a,b) \leq u(a,c) \quad \text{if} \quad b \leq c$ (1.7)

Commutativity: $u(a,b) = u(b,a)$ (1.8)

Associativity: $u(a,u(b,c)) = u(u(a,b),c)$ (1.9)

♦

The fuzzy minimum and the fuzzy maximum, defined bellow, are the most used t-norms and t-conorms. Examples of these operators can be found in Figure 1.3 and Figure 1.4, respectively.

Definition 1.4 [Klir and Yuan 1995] – Given two fuzzy sets A and B, their standard intersection, $A \cap B$, and standard union, $A \cup B$, also known as fuzzy minimum and fuzzy maximum, are defined for all $x \in X$ by the equations:

$$(A \cap B)(x) = \min[A(x), B(x)] \tag{1.10}$$

$$(A \cup B)(x) = \max[A(x), B(x)] \tag{1.11}$$

♦

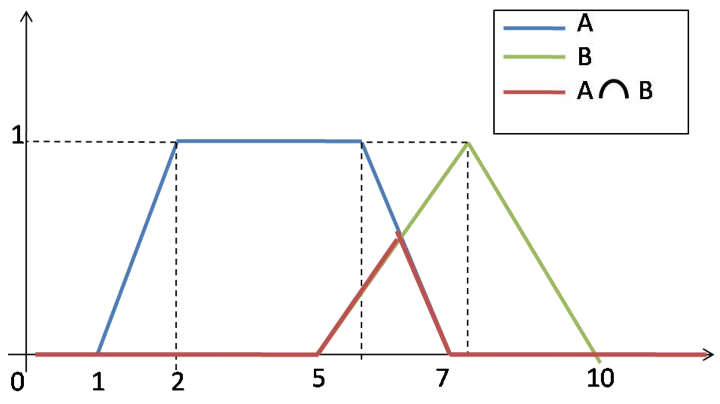


Figure 1.3: Fuzzy min

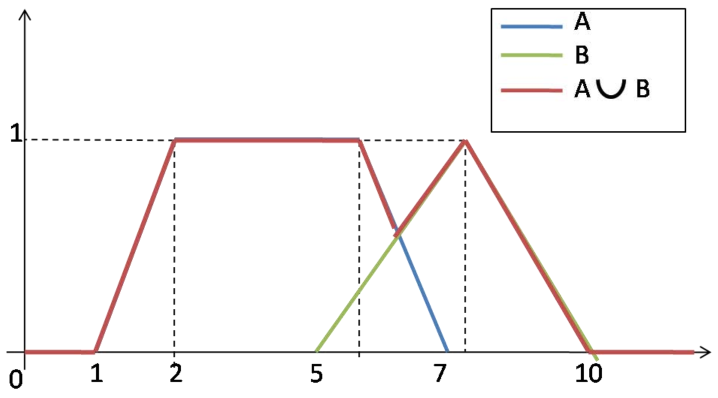


Figure 1.4: Fuzzy max

To generalize the concept of negation, complement operators are used. The membership function of a fuzzy set A represents, for each x in its universe of discourse, the degree to which x belongs to A. The membership functions of the complement of A represents the degree to which x does not belong to A.

Definition 1.5 [Klir and Yuan 1995] – A complement of a fuzzy set A is specified by a function $c:[0,1]\rightarrow[0,1]$ satisfying the following properties [Klir and Yuan 1995]:

Boundary Conditions: $c(0)=1; \quad c(1)=0$ (1.12)

Monotonicity: $c(a)\geq c(b) \quad \text{if} \quad a\leq b$ (1.13)

♦

The standard complement is defined bellow and exemplified in Figure 1.5.

Definition 1.6 [Klir and Yuan 1995] - The standard complement, $\overline{A}$, of a fuzzy set A with respect to the universal set X is defined for all $x\in X$ by the equation:

$$\overline{A}(x)=1-A(x) \tag{1.14}$$

♦

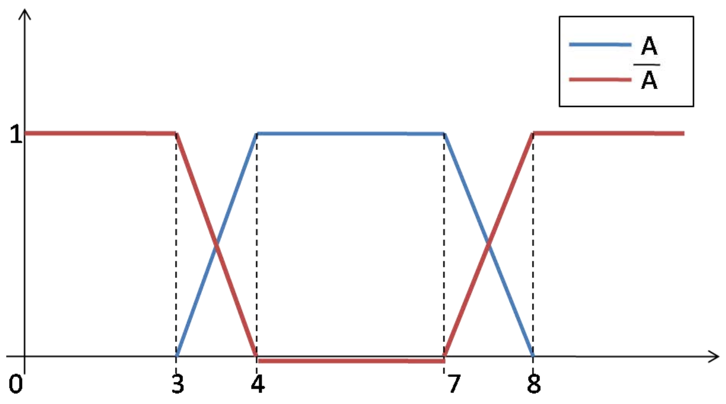
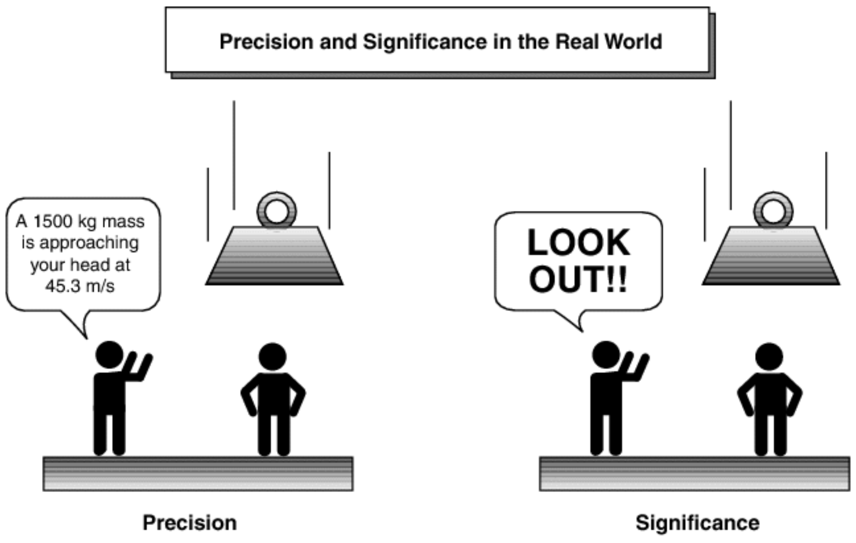


Figure 1.5: Standard fuzzy complement

1.2 Fuzzy Inference Systems

A fuzzy inference system is composed of fuzzy *if-then* rules relating different fuzzy sets, which are stored in a knowledge-base, and an inference engine that performs approximate reasoning [Ross 2004]. As mentioned before, one of the main advantages of inference systems [Ross 2004] is the ability to build models that mimic human reasoning and are relatively simple and easy to interpret. These models might be less accurate than classical and more formal ones but when dealing with real world applications interpretability, significance and computational efficiency can overcome some lack of accuracy, as depicted in Figure 1.6, taken from [Mathworks].


Precision and Significance in the Real World
A 1500 kg mass is approaching your head at 45.3 m/s
LOOK OUT!!
Precision
Significance
Figure 1.6: Precision vs. Significance in the Real World [Mathworks]

There are two main kinds of fuzzy inference systems, Mamdani and Sugeno [Lee 1990a; Lee 1990b]. The knowledge base of a Mamdani inference system contains rules where both the antecedents and the consequents are fuzzy sets. Sugeno inference systems, on the other hand, use rules with fuzzy antecedents and crisp consequents. In this thesis only Mamdani inference systems will be used but the ideas and algorithms developed can also be used in Sugeno inference systems.

Fuzzy if-then rules used in Mamdani inference systems are expressions of the type [Ross 2004]:

"if x is A then y is B"

where A and B are fuzzy sets, *"x is A"* is called the antecedent and *"y is B"* is called the consequent of the rule.

The antecedent part of the rule can have multiple parts connected by fuzzy operators, typically t-norms and t-conorms giving meaning to the linguistic expressions "and" and "or" respectively. The consequent can have multiple parts representing distinct conclusions that can be inferred from the given antecedent. The firing level or firing strength of the rule is the degree to which the antecedent part of the rule is satisfied.

To determine the outcome of fuzzy if-then rules given the crisp inputs, we need to fuzzify the inputs, apply the fuzzy operators that connect the multiple parts of the antecedent (if needed) to find the firing level of the rule and use an implication operator to apply the firing level to the consequent part (or parts) [Lee 1990a; Lee 1990b]. The output of the rule is a fuzzy set (or fuzzy sets). These concepts are better explained through an example. The following example in Figure 1.7 is taken from Matlab Fuzzy Logic Toolbox documentation [Mathworks].

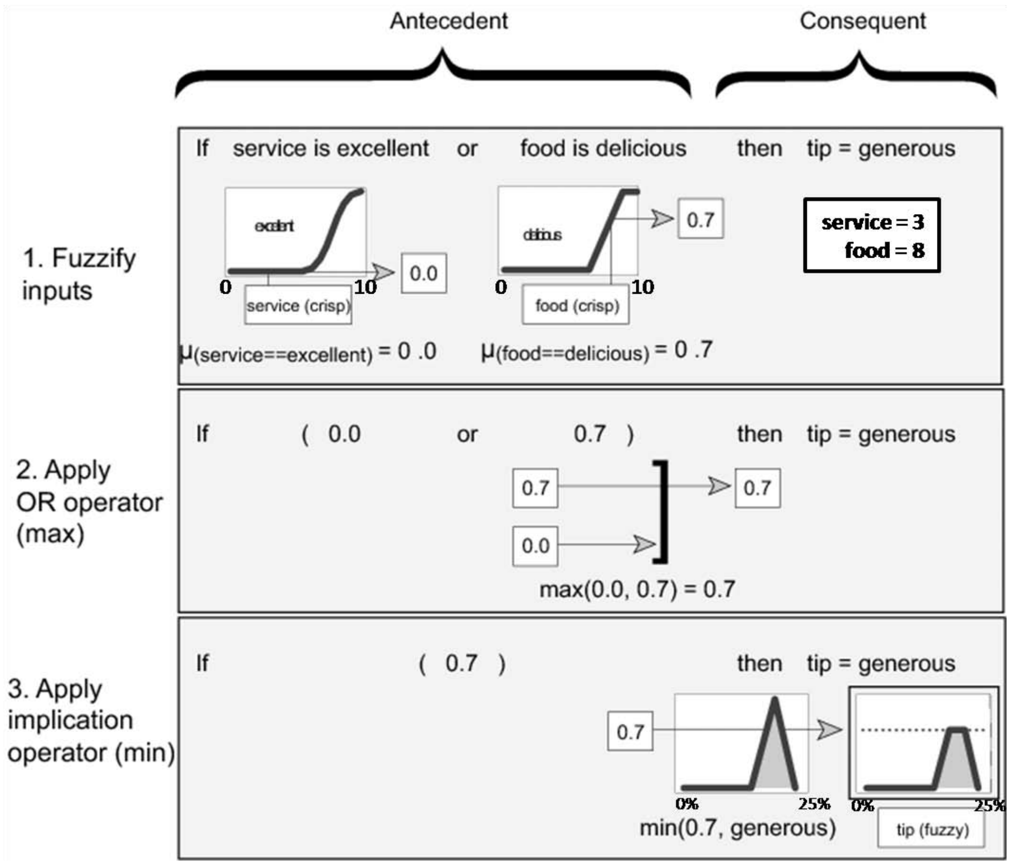

Antecedent
Consequent
If service is excellent or food is delicious then tip = generous
1. Fuzzify inputs
excellent
0
10
service (crisp)
0.0
delicious
food (crisp)
0.7
service = 3
food = 8
μ(service==excellent) = 0 .0
μ(food==delicious) = 0 .7
If (0.0 or 0.7) then tip = generous
2. Apply OR operator (max)
0.7
0.0
0.7
max(0.0, 0.7) = 0.7
If (0.7) then tip = generous
3. Apply implication operator (min)
0.7
0%
25%
min(0.7, generous)
tip (fuzzy)

Figure 1.7: Example of fuzzy if-then rule [Mathworks]

Given crisps values for the service and food quality the correspondent degrees of membership in the antecedent are computed and combined through the OR operator to give the rule firing level. For instance, if we consider service=3 and food=8, the degrees of membership in excellent (for service) and delicious (for food) are 0 and 0.7, respectively, and the firing level of the rule is given by $\max(0,0.7) = 0.7$. The implication operator is then applied taking into account this firing level to obtain the fuzzy set representing the output of the rule.

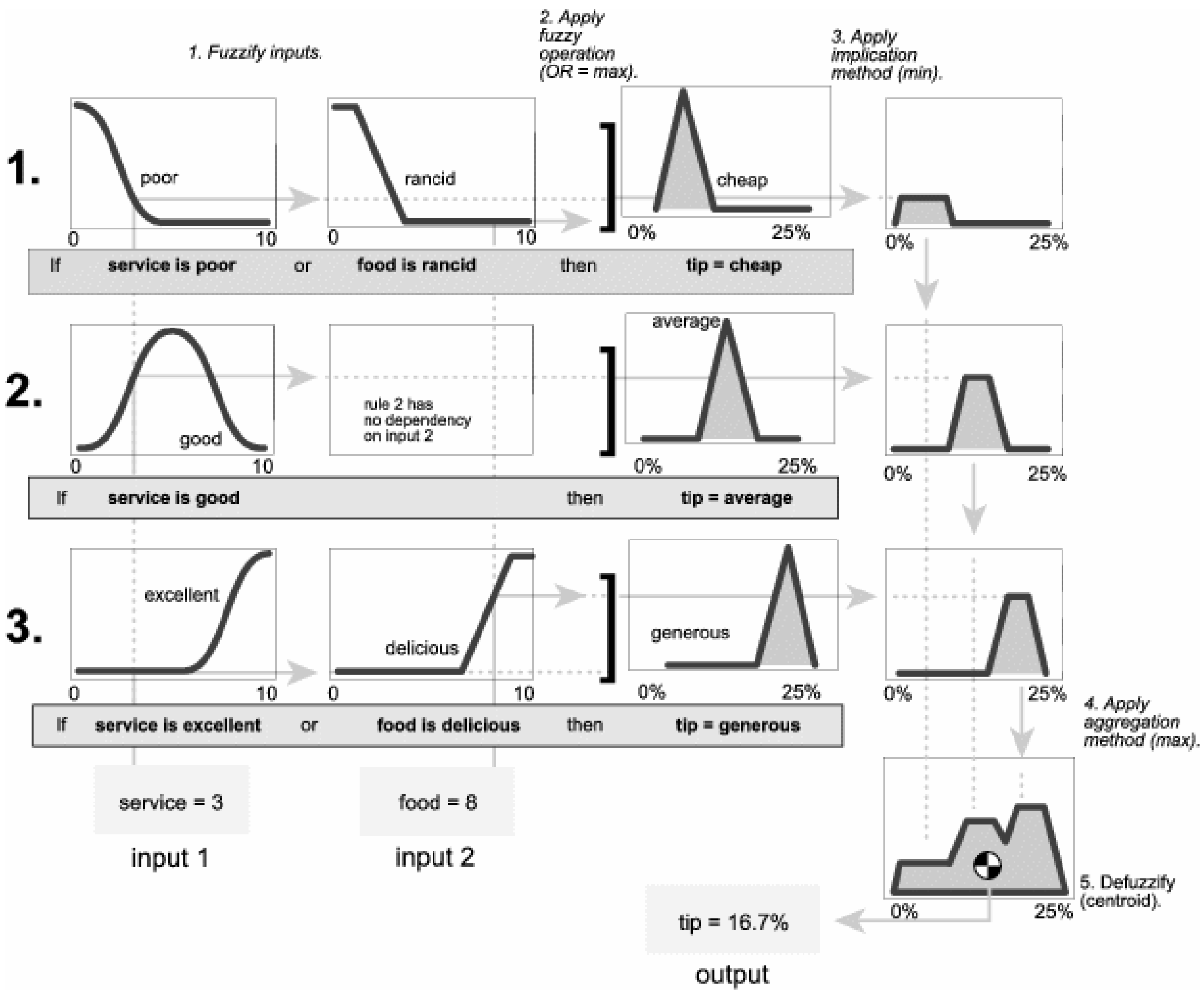

1. Fuzzify inputs.
2. Apply fuzzy operation (OR = max).
3. Apply implication method (min).
1.
poor
rancid
cheap
0
10
0%
25%
If service is poor or food is rancid then tip = cheap
2.
good
rule 2 has no dependency on input 2
average
If service is good then tip = average
3.
excellent
delicious
generous
If service is excellent or food is delicious then tip = generous
4. Apply aggregation method (max).
service = 3
input 1
food = 8
input 2
5. Defuzzify (centroid).
tip = 16.7%
output

Figure 1.8: Example of fuzzy inference system [Mathworks]

For each rule in the knowledge base the previously described steps are performed and the resulting fuzzy sets are aggregated through an appropriate operator (usually standard fuzzy maximum) to obtain a new fuzzy set representing the output of the system. This fuzzy set is then defuzzified to obtain a crisp value for the inference. Several defuzzification methods can be used, e.g. the centroid [Lee 1990a; Lee 1990b]. Continuing with the tipping example from Fuzzy Logic Toolbox documentation [Mathworks], Figure 1.8 shows a possible inference system with three rules and the necessary steps to determine the tip to be given crisp values for the service and food quality. In Figure 1.8, the three first fuzzy sets on the right represent the output of each rule after implication, using the same input values as before, i.e., service=3 and food=8. Aggregating these three fuzzy sets, the fuzzy set in the bottom right of Figure 1.8 is obtained. In this example the centre of area or centroid defuzzification method, defined by (1.15), is used and a tip of 16.7% is recommended.

Definition 1.7 [Klir and Yuan 1995] - Consider a fuzzy set A with membership function $\mu_A : X \to [0,1]$. The centre of area or centroid defuzzification method returns the value $d_{CA}(A)$ within X for which the area underneath the graph of membership function μ_A is divided into two equal subareas. This value is given by the following expression:

$$d_{CA}(A) = \frac{\int_X \mu_A(x) \cdot x \, dx}{\int_X \mu_A(x) \, dx} \tag{1.15}$$

♦

1.3 Representation of Membership Functions

Some types of membership functions can be mapped to R^p, where p is the number of parameters of that family of membership functions and each dimension represents a different parameter. In this section both analytical and R^p representations of some of the most common types of membership functions are presented.

1.3.1 Triangular Membership Functions

Definition 1.8 - A triangular membership function is given by the analytical expression:

$$\mu(a,b,c,x) = \begin{cases} \dfrac{x-a}{b-a} & , \quad a \le x \le b \\ \dfrac{c-x}{c-b} & , \quad b \le x \le c \\ 0 & , \quad \text{otherwise} \end{cases} \tag{1.16}$$

where a, b and c correspond to the x-axis coordinates of the vertices of the triangle, as in Figure 1.9.

♦

There are several possibilities for mapping these membership functions into R^p, $p = 2, 3$. For instance, for $p = 3$ we can consider a vector with the x-axis coordinates of the vertices of the triangle, (a,b,c), or a vector $(b, \varepsilon_L, \varepsilon_R)$ where $\varepsilon_L = b - a$ and $\varepsilon_R = c - b$ represent its left and right spreads, respectively. This way we define a mapping between the family of triangular membership functions and R^3. If we only consider symmetrical membership functions, i.e., if $\varepsilon_L = \varepsilon_R = \varepsilon$, we can use a pair (b, ε) to represent a membership function of this family. In this way the mapping can be done in R^2.

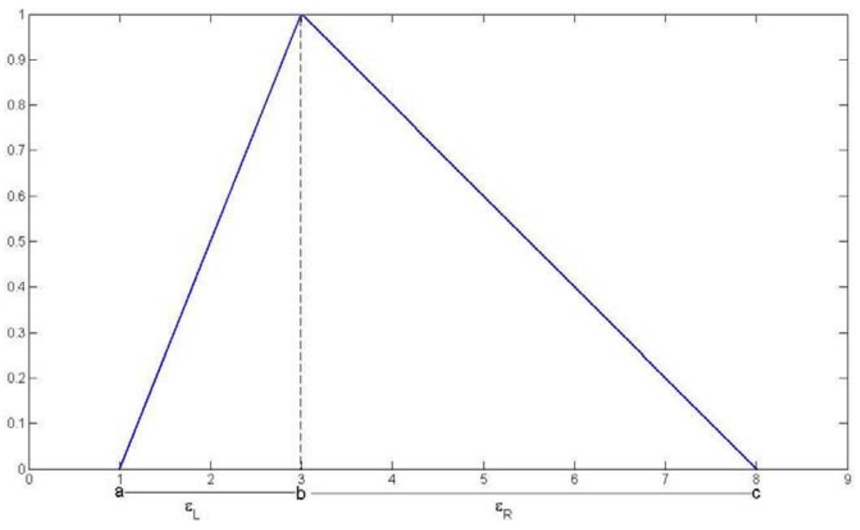


Figure 1.9: Triangular membership function $(a,b,c) = (1,3,8)$

1.3.2 Trapezoidal Membership Functions

Definition 1.9 - A trapezoidal membership function is given by the analytical expression:

$$\mu(a,b,c,d,x)=\begin{cases}1 & , \quad b \le x \le c \\ \dfrac{x-a}{b-a} & , \quad a \le x \le b \\ \dfrac{d-x}{d-c} & , \quad c \le x \le d \\ 0 & , \quad \text{otherwise}\end{cases} \tag{1.17}$$

where a,b,c and d correspond to the x-axis coordinates of the vertices of the trapezoid, as in Figure 1.10.

♦

Similarly to the case of triangular membership functions, we can now map the family of trapezoidal membership functions to R^4 and R^3 (symmetric trapezoidal). We can consider a vector with the x-axis coordinates of the vertices of the trapezoidal, (a,b,c,d), to map this family of membership functions to R^4 and if we only consider symmetrical membership functions, i.e., if $\frac{a+d}{2}=\frac{b+c}{2}$, we can use a vector (m,ε,δ), where $m=\frac{a+d}{2}=\frac{b+c}{2}$, $\varepsilon=c-b$ and $\delta=d-a$, to represent a membership function of this family.

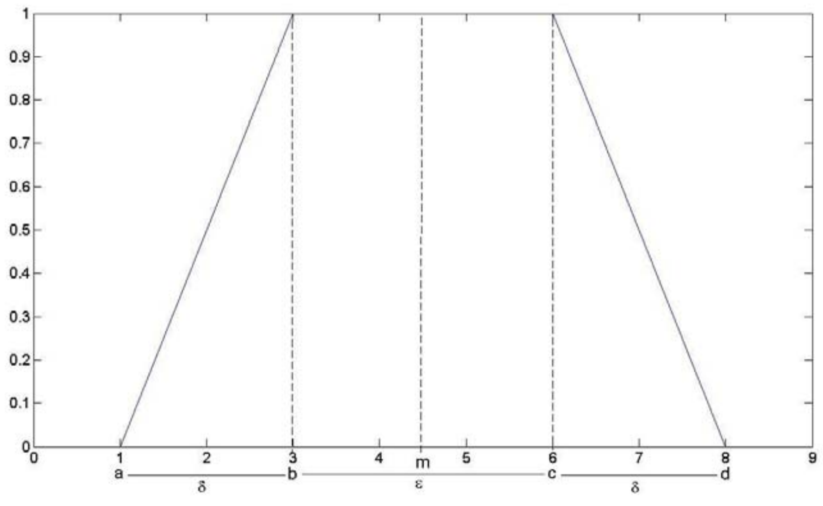


Figure 1.10: Symmetrical trapezoidal membership function $(a,b,c,d) = (1,3,6,8)$

1.3.3 Gaussian Membership Functions

Definition 1.10 – A Gaussian membership function is given by the analytical expression:

$$e^{-\sigma(x-\mu)^2} \tag{1.18}$$

where μ and σ are the mean and spread of the Gaussian function.

♦

The mapping of this family of membership functions to R^2 is straightforward and is given by the pair (μ, σ).

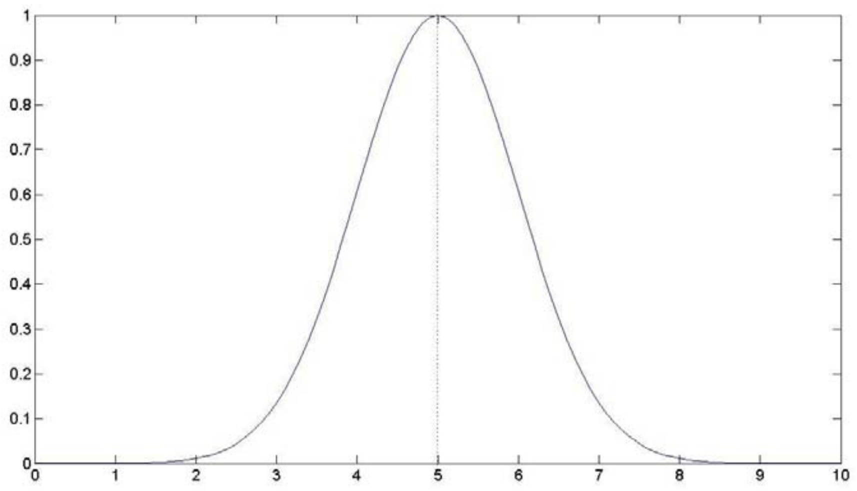


Figure 1.11: Gaussian membership function $(\mu, \sigma) = (5,1)$

1.4 Proximity Measures between Membership Functions

As stated in the introduction of this Chapter, the notion of similarity or proximity between membership functions will be the main idea underneath the algorithms for reducing the number of membership functions in linguistic variables. When faced with the problem of reducing the number of terms in linguistic variables, we intuitively think of joining or merging membership functions that are somehow similar. For crisp data sets, a similar idea is the foundation of cluster analysis. Clusters are groups of objects that are similar according to some proximity measure [Hartigan 1975]. The problem presented in this thesis can then be approached as a clustering problem where the objects are membership functions and suitable proximity measures are used.

In general, similarity measures between membership functions or fuzzy sets can be classified as geometric or set-theoretical [Miyamoto 1990]. Geometric measures are based on distance-measures and represent proximity between fuzzy sets. Set-theoretical similarity measures, based on operations such as union and intersection, translate the degree to which two fuzzy sets are equal and are not influenced by scaling and ordering of the domain.

One of the most used set-theoretical similarity measures, the fuzzy Jaccard index or Jaccard similarity measure [Miyamoto 1990], is defined by:

$$S_J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1.19}$$

where $|C| = \int_U \mu_C(x)\,dx$ and $(\cap, \cup)$ is a pair of fuzzy t-norms $(\cap)$ and t-conorms $(\cup)$.

An overview of some similarity measures for comparing fuzzy sets can be found in [Chen, Yeh et al. 1995]. The Jaccard similarity measure will be used in the algorithms presented in Chapter 5, but other similarity measures could also be used. For instance, since in Chapter 5 only trapezoidal membership functions are used, the following two similarity measures used for comparing trapezoidal fuzzy sets could be considered.

The first one can be calculated by the following expression [Chen 1996]:

$$S_C(A,B) = 1 - \frac{\sum_{i=1}^{4} |a_i - b_i|}{4} \tag{1.20}$$

where $A = (a_1, a_2, a_3, a_4)$ and $B = (b_1, b_2, b_3, b_4)$.

The second one was proposed by Shi-Jay Chen and Shyi-Ming Chen [Chen and Chen 2008] and is given by:

$$S_{SCGM}(A,B) = \left[1 - \frac{\sum_{i=1}^{4} |a_i - b_i|}{4} \right] \times (1 - |x_A^* - x_B^*|)^{B(S_A, S_B)} \times \frac{\min(y_A^*, y_B^*)}{\max(y_A^*, y_B^*)} \tag{1.21}$$

where

$$B(S_A, S_B) = \begin{cases} 1, & \text{if } S_A + S_B > 0 \\ 0, & \text{if } S_A + S_B = 0 \end{cases} \tag{1.22}$$

$$S_A = a_4 - a_1 \tag{1.23}$$

$$S_B = b_4 - b_1 \tag{1.24}$$

and (x_A^*, y_A^*) and (x_B^*, y_B^*) are the centre of gravity points of A and B, respectively. These points can be easily determined by the simple centre of gravity method (SCGM) [Chen and Chen 2008], using the following expressions:

$$y_A^* = \begin{cases} \dfrac{\dfrac{a_3 - a_2}{a_4 - a_1} + 2}{6}, \; if \;\; a_1 \neq a_4 \\ \dfrac{1}{2}, \; if \;\; a_1 = a_4 \end{cases} \tag{1.25}$$

$$x_A^* = \frac{y_A^*(a_3 + a_2) + (a_4 + a_1)(1 - y_A^*)}{2} \tag{1.26}$$

In the previous section it was shown how the most used families of membership functions can be mapped to R^p. By mapping a membership function to R^p the problem to be addressed becomes equivalent to finding clusters given a data set in R^p, provided that we are considering linguistic variables where all membership functions belong to the same family, which is usually the case. Therefore, the proximity measures used for comparing objects in R^p can also be used to compare membership functions of the same family. For instance, the Euclidean Distance given by (1.27) can be used to compare two membership functions of the same family, $A = (a_1, \ldots, a_p)$ and $B = (b_1, \ldots, b_p)$, represented in R^p. This will be done in the algorithms presented in Chapter 4 where the problem of reducing the number of membership functions in linguistic variables will be approached by clustering the vectors of parameters representing the membership functions.

$$D(A,B) = \sqrt{\sum_{i=1}^{p} (a_i - b_i)^2} \tag{1.27}$$

1.5 Merging Membership Functions

In this section we discuss some methods on how to merge membership functions to reduce the number of membership functions in a linguistic variable, by using the concept of similarity. This section is not intended as an overview of the possible methods for merging membership functions since these methods could vary according to several factors: the type of membership functions being merged, the algorithms in use, the context of the problem, among others.

Membership functions of the types referred in section 1.2 will be considered, since these are the most used ones. Also, throughout this thesis, it will be assumed that all membership functions of a certain linguistic variable to be pruned share the same type (either triangular or trapezoidal) and that the merging of two membership functions should yield a new membership function of the same type as the original ones. This simplification does not change the nature of the problem and the algorithms that will be use to solve it are as general as possible. If one of these conditions fails we only have to redefine the way two membership functions are merged but the algorithms still apply.

1.5.1 Merging Trapezoidal Membership Functions

Given two trapezoidal membership functions $A=(a_1,a_2,a_3,a_4)$ and $B=(b_1,b_2,b_3,b_4)$, merging them using the method proposed in [Setnes, Babuska et al. 1998] gives a new trapezoidal membership function $C=(c_1,c_2,c_3,c_4)$ where:

$$c_1 = \min(a_1, b_1) \tag{1.28}$$

$$c_2 = \lambda_2 a_2 + (1-\lambda_2) b_2 \tag{1.29}$$

$$c_3 = \lambda_3 a_3 + (1-\lambda_3) b_3 \tag{1.30}$$

$$c_4 = \max(a_4, b_4) \tag{1.31}$$

The parameters λ_2 and λ_3 belong to the interval $[0,1]$. These parameters allow weighting the importance of A and B in the final membership C. In subsequent chapters this operator will be used with $\lambda_2 = \lambda_3 = 0.5$. See for instance Figure 1.12, which shows the trapezoidal membership functions $A = (1,2,4,6)$ and $B = (2,3,5,7)$ combined into $C = (1,2.5,4.5,7)$. Notice that (1.28) and (1.31) guarantee that the same "coverage" as A and B, i.e., points with positive membership in either A or B will still have positive membership in C. This might be crucial for some applications.

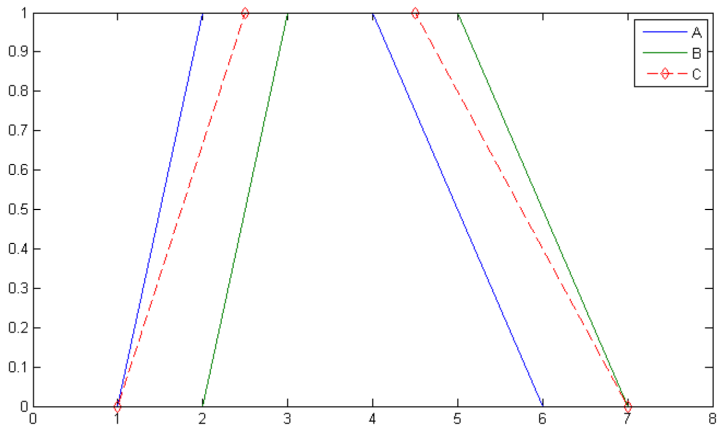


Figure 1.12: Merging trapezoidal membership functions $A = (1,2,4,6)$ and $B = (2,3,5,7)$ into $C = (1,2.5,4.5,7)$.

In the previous merging method only two membership functions are merged at a time. When merging more than two membership functions at a time, a generalization of this method was used. Given n trapezoidal membership functions $T^i = (a_i, b_i, c_i, d_i)$, $i = 1,\ldots,n$, these will be simultaneously merged into a membership function $T = (a,b,c,d)$, $i = 1,\ldots,n$ where:

$$a = \min_{i=1,\ldots,n} a_i \tag{1.32}$$

$$b = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{1.33}$$

$$c = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{1.34}$$

$$d = \max_{i=1,\ldots,n} d_i \tag{1.35}$$

1.5.2 Merging Triangular Membership Functions

It is straightforward to adapt the previous methodology to the case of triangular membership functions. Considering that a triangular membership function is a trapezoidal membership function with $b_i = c_i$ Given n triangular membership functions $S^i = (a_i, b_i, d_i),\ \ i = 1,\ldots,n$, these will be simultaneously merged into a membership function $S = (a,b,d),\ \ i = 1,\ldots,n$ where:

$$a = \min_{i=1,\ldots,n} a_i \tag{1.36}$$

$$b = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{1.37}$$

$$d = \max_{i=1,\ldots,n} d_i \tag{1.38}$$

1.5.3 Merging Gaussian Membership Functions

In [Song, Marks et al. 1993] the fusion of two Gaussian membership functions with parameters (μ_1, σ_1) and (μ_2, σ_2) is a Gaussian membership function with parameters (μ, σ) defined by the following equations. See for instance Figure 1.13.

$$\mu = \frac{\mu_1\sigma_1 + \mu_2\sigma_2}{\sigma_1 + \sigma_2} \tag{1.39}$$

$$\sigma^2 = \frac{\sigma_1^3 + \sigma_2^3}{\sigma_1 + \sigma_2} \tag{1.40}$$

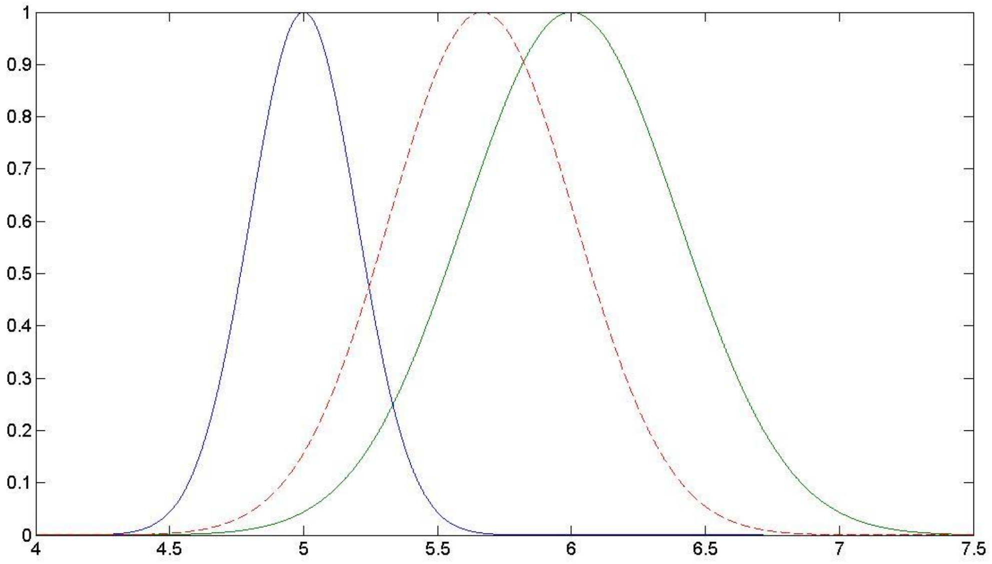


Figure 1.13: Merging Gaussian membership functions $(\mu_1, \sigma_1) = (5, 0.2)$ and $(\mu_2, \sigma_2) = (6, 0.4)$ into $(\mu, \sigma) \approx (5.667, 0.3464)$

We can extend this method by defining the merge of n Gaussian membership functions with parameters $(\mu_i, \sigma_i), \quad i = 1, \ldots, n$ as by the pair (μ, σ), where:

$$\mu = \frac{\sum_{i=1}^{n} \mu_i \sigma_i}{\sum_{i=1}^{n} \sigma_i} \tag{1.41}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n} \sigma_i^3}{\sum_{i=1}^{n} \sigma_i} \tag{1.42}$$

1.6 Summary

In this Chapter the concepts of fuzzy set theory necessary to understand the work presented in this thesis were introduced. The concepts in sections 1.1 and 1.2

are the basic concepts of fuzzy logic and inference systems. Analytical and R^p representations of three of the most used families of membership functions are given in section 1.3. In this thesis it will be seen how to reduce the number of terms in a linguistic variable by merging similar membership functions. Sections 1.4 and 1.5 present the proximity measures between membership functions that will be used in later chapters and the methods for merging membership functions.

Chapter 2. A Clustering Problem Approach

As stated in the introduction, the problem of reducing the number of membership functions in linguistic variables can be formulated as a clustering problem. We need to identify groups of similar membership functions and merge them. As a result, we should obtain a smaller set of membership functions capable of approximately represent the initial linguistic variable.

In section 2.1 the clustering problem will be introduced. Section 2.2 will present the *state of the art* and finally in section 2.3 some integer programming formulations for the clustering problem will be given.

2.1 The Clustering Problem

There is no uniform formal definition for data clustering. The task of defining the meaning of clustering have been pointed out as a difficult one by several authors [Everitt, Landau et al. 2001; Estivill-Castro 2002]. In [Gan, Ma et al. 2007] the following informal definition can be found:

“Data clustering (or just clustering), also called cluster analysis, segmentation analysis, taxonomy analysis, or unsupervised classification, is a method of creating groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct.”

As can be seen in Figure 2.1, two main types of clustering problems exist: hard clustering and fuzzy clustering [Gan, Ma et al. 2007]. In hard clustering problems an object or record has to belong to one and only one cluster, that is, a partition of the data into mutually exclusive groups is obtained. Fuzzy clustering problems on the other hand, allow an object to belong to several clusters, with different degrees of membership. In this thesis the problem of reducing the number of membership functions in a linguistic variable will be formulated as a hard clustering problem. Therefore the term *clustering* will be used instead of *hard clustering*. Approaching this problem as a fuzzy clustering problem by allowing membership functions to belong to more than one cluster and defining appropriate membership function merging techniques is a possibility to be studied in the future.

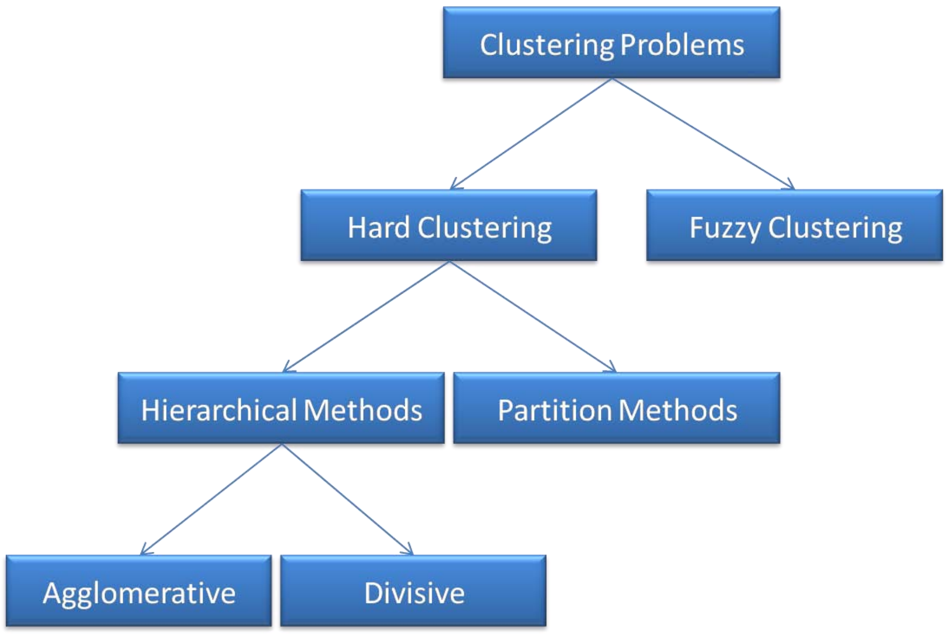


Figure 2.1: Diagram of Clustering Algorithms [Gan, Ma et al. 2007]

2.2 State of the Art

As stated in the previous section, there is no unique definition of clustering. Appropriate criteria for clustering have to be chosen for each application, according to the type of groups to be found in data. This partially explains the existing diversity of clustering algorithms [Estivill-Castro 2002]. Given this diversity, this *state of the art* will not be exhaustive in describing all the existing methods. Some more extensive reviews can be found in [Sokal and Sneath 1963; Hartigan 1975; Rijsbergen 1979; Jain and Dubes 1988; Kaufman and Rousseeuw 1990; Jain, Murty et al. 1999; Everitt, Landau et al. 2001; Engelbrecht 2002; Mirkin 2005; Gan, Ma et al. 2007].

As depicted in Figure 2.1, conventional (hard) clustering algorithms can be divided into two categories, according to the type of structures they return.

Hierarchical methods return a hierarchy or set of nested partitions while partition methods return a single partition of the data.

The next subsections will present the main ideas of hierarchical methods (section 2.2.1), classical partition methods (section 2.2.2), graph based methods (section 2.2.3), metaheuristics (section 2.2.4) and other clustering methods (section 2.2.5).

2.2.1 Hierarchical Methods

As stated before, hierarchical methods return a hierarchy or set of nested partitions, as depicted in Figure 2.2. Agglomerative hierarchical algorithms start with each data point in a different cluster and proceed by merging clusters, according to some criterion, until there is only one cluster containing all data points in the data set. Divisive hierarchical algorithms start with one cluster containing all data points and proceed by splitting clusters until each data point is in a different cluster.

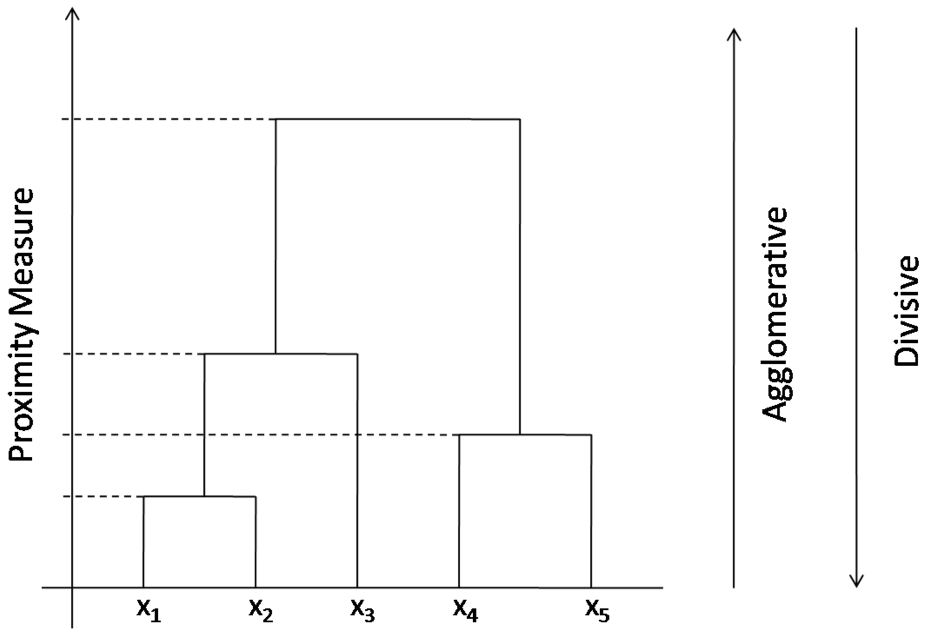


Figure 2.2: Dendogram

An hierarchical agglomerative clustering algorithm consists of the following steps [Jain, Murty et al. 1999]:

1. Compute the proximity matrix containing the distance between each pair of data points. Treat each data point as a cluster;
2. Find the most similar pair of clusters using the proximity matrix and merge them into one cluster;
3. Update the proximity matrix to reflect the merging operation in 2;
4. If all data points are in one cluster, stop. Otherwise, go to step 2.

Different algorithms can be developed according to the way the proximity measure is updated in step 3. The most used are the single-link, complete link and Ward's methods [Jain, Murty et al. 1999].

The single-link method, also known as nearest neighbour method and minimum method, was first introduced by [Florek, Lukaszewicz et al. 1951] and then independently by [McQuitty 1957] and [Sneath 1957]. Let C_1 and C_2 be two clusters and $d(\cdot,\cdot)$ a distance measure between two points. In the single-link method, the distance between C_1 and C_2, also referred to as linkage function, is given by:

$$D(C_1, C_2) = \min_{x \in C_1,\, y \in C_2} d(x, y) \tag{2.1}$$

The complete-link [King 1967], also known as farthest neighbour method, updates the proximity measure using the following expression, using the same notation as in (2.1).

$$D(C_1, C_2) = \max_{x \in C_1,\, y \in C_2} d(x, y) \tag{2.2}$$

The Ward's method [Ward Jr. 1963; Ward Jr. and Hook 1963], also known as minimum-variance method, aims at forming partitions $P_n, \cdots, P_1$ of the original data minimizing the loss of information, quantified in terms of the error sum of squares (ESS) criterion, associated with each merge. Consider a partition of the data into K clusters $C_1, \cdots, C_K$. The information loss is represented by:

$$ESS = \sum_{i=1}^{K} ESS(C_i) \tag{2.3}$$

where

$$ESS(C) = \sum_{\underline{x} \in C} (\underline{x} - \mu(C))(\underline{x} - \mu(C))^T \tag{2.4}$$

and

$$\mu(X) = \frac{1}{|C|} \sum_{\underline{x} \in X} \underline{x} \tag{2.5}$$

At each step of the Ward's method the two clusters whose fusion results in the minimum increase in loss of information are merged. The linkage function is computed as the increase in ESS after merging two clusters, i.e.:

$$D(C_1, C_2) = ESS(C_1 C_2) - ESS(C_1) - ESS(C_2) \tag{2.6}$$

where $C_1 C_2$ denotes the cluster resulting from merging C_1 and C_2.

Other linkage functions are described in [Hartigan 1975; Everitt, Landau et al. 2001; Gan, Ma et al. 2007]. In [Kuiper and Fisher 1975] a comparison of several hierarchical clustering algorithms is done using the Monte Carlo method.

As stated before, divisive hierarchical algorithms proceed the opposite way of the agglomerative algorithms. We start with one cluster containing all data points and proceed by splitting clusters until each data point is in a different cluster. Since given a cluster C there are $2^{|C|-1} - 1$ nontrivial ways of splitting it into two subclusters, it is not feasible to enumerate all the possible divisions of a cluster to find the optimal division, except for small clusters [Edwards and Cavalli-Sforza 1965]. Several divisive hierarchical clustering algorithms can therefore be designed considering different criteria for choosing the cluster to be split and different methods for splitting clusters. Examples of divisive algorithms can be found in [Edwards and Cavalli-Sforza 1965; Spath 1980; Kaufman and Rousseeuw 1990].

To illustrate divisive hierarchical clustering algorithms we will consider the DIANA (DIvisive ANAlysis) algorithm proposed by [Kaufman and Rousseeuw 1990].

For a given distance measure $d(\cdot,\cdot)$, the diameter of a cluster C is given by:

$$Diam(C) = \max_{x,y\in C} d(x,y) \tag{2.7}$$

Denote the average dissimilarity from a point x to the points in a set S by $D(x,S)$, i.e.,

$$D(x,S) = \frac{1}{|S|}\sum_{y\in S} d(x,y) \tag{2.8}$$

In each step of the DIANA algorithm, the cluster with largest diameter, C ($|C| \geq 2$), is split into two subclusters, A and B. These subclusters are determined by the following procedure:

1. Do $A = C$ and $B = \{\ \}$;
2. Do $z = \arg\max\{D(x, A\setminus\{x\}),\ x \in A\}$;
3. Move point z from A to B, i.e., $A \leftarrow A\setminus\{z\}$ and $B \leftarrow B \cup \{z\}$;
4. Do $z = \arg\max\{D(x, A\setminus\{x\}) - D(x,B),\ x \in A\}$;
5. If $D(z, A\setminus\{z\}) - D(z,B) > 0$ then move point z from A to B, i.e., $A \leftarrow A\setminus\{z\}$ and $B \leftarrow B \cup \{z\}$, and return to 4. Otherwise stop the procedure, returning A and B.

The procedure starts by considering $A = C$ and $B = \{\ \}$, i.e., all points belong to subcluster A. Then the point with highest dissimilarity is moved from subcluster A to B. The procedure continues by moving points from A to B whenever their average dissimilarity to B is smaller than the average dissimilarity to the rest of the points in A.

Generally, hierarchical methods have a complexity of $O(n^2)$ for memory space and $O(n^3)$ for CPU time [Hartigan 1975; Murtagh 1983], n being the number of points to be clustered. Therefore, they become impractical for large data sets.

2.2.2 Classical Partition Clustering Methods

Unlike hierarchical methods, partition methods create a single partition of the data points.

The most known partition method is the K-Means algorithm [McQueen 1967]. This algorithm is a centre-based method. Each cluster is represented by a centre and the corresponding clusters have convex shapes. The algorithm starts by choosing initial K cluster centres from the original data. After the initialization, a partition of the data is determined by assigning each point to the cluster with closest centre. After this assignment the centroids of each cluster are calculated according to the following expression:

$$c_i = \frac{1}{|C_i|}\sum_{\underline{x}\in C_i}\underline{x}\,, \quad i = 1,\cdots,K \tag{2.9}$$

where c_i is the centre of cluster C_i.

Then the points are reassigned to the clusters regarding the closeness to the centroids. Again, the centroids are recalculated and the algorithm proceeds in the same way until some stopping criterion is met. Usually the algorithm will proceed until the cluster centroid and partition no longer change or until a predefined number of iterations is reached. This way the K-Means algorithm is a heuristic method that tries to minimize the sum of squared distances from each point to its cluster centre. The number of clusters K is determined by the user *a priori.* In practice, if the user can not identify the correct number of clusters, the algorithm is run for a certain range for the number of clusters, i.e. $K \in \{K_{\min},\cdots,K_{\max}\}$, and the best configuration found, according to some criterion, is chosen.

Many variations of the original K-Means algorithm have been developed. Some try to improve the efficiency of the algorithm by reducing the computational effort demanded by the algorithm [Tapas, David et al. 2002]. Others differ from the original algorithm in the way the initial cluster centres are chosen, as is the case of the algorithm presented in [David and Sergei 2007] called K-Means++ that will be further discussed in section 4.1. Some allow merging or splitting clusters according to centres distances or cluster within variance [Ball and Hall 1965].

Another widely used partition method is the Expectation Maximization Algorithm (EM) [Dempster, Laird et al. 1977], a model based clustering algorithm. In model based clustering it is assumed that the data comes from a certain mixture of distributions $f(x)=\sum_{k=1}^{K} p_k\, f(x,a_k)$ $\left(p_k \geq 0,\ \ \sum_{k=1}^{K} p_k = 1\right)$, with each component $f(x,a_k)$ representing a different cluster, where $f(x,a_k)$ is a family of density functions over x and a_k is the parameter vector that identifies a particular density from that family. Model based clustering algorithms try to optimize the fit between the data and the proposed model.

To estimate the individual cluster parameter the EM algorithm uses the maximum likelihood approach. The logarithm of the likelihood of the observed data given by (2.10) is maximized under the assumption that the data comes from a mixture of distributions.

$$L=\log\left\{\prod_{i=1}^{N}\sum_{k=1}^{K} p_k\, f\left(y_i,a_k\right)\right\} \tag{2.10}$$

Maximization of (2.10) can be reformulated as the maximization of (2.11).

$$L=\sum_{i=1}^{N}\sum_{k=1}^{K} g_{ik}\log p_k+\sum_{i=1}^{N}\sum_{k=1}^{K} g_{ik}\log f\left(y_i\, fa_k\right)-\sum_{i=1}^{N}\sum_{k=1}^{K} g_{ik}\log g_{ik} \tag{2.11}$$

where

$$g_{ik}=\frac{p_k\, f\left(y_i,a_k\right)}{\sum_{j=1}^{K} p_k\, f\left(y_j,a_k\right)} \tag{2.12}$$

and K and N are the number of clusters and data points, respectively.

The EM algorithm can then be summarized in the following way [Mirkin 2005]:

1. Start with any initial values of the parameters p_k, a_k and g_{ik}, $i = 1,\cdots,N$, $k = 1,\cdots,K$;
2. (E-step) Given p_k and a_k estimate g_{ik};
3. (M-step) Given g_{ik} find p_k and a_k maximizing (2.11);
4. Repeat steps 2 and 3 until there is no change in the parameter values (or the absolute difference is below some previously defined threshold).

2.2.3 Graph Based Methods

The relationship between graph theory and the clustering problem has been discussed by [Wirth, Estabrook et al. 1966; Jardine and Sibson 1968; Gower and Ross 1969; Hubert 1974; Hansen and Delattre 1978], among other authors. Algorithms that take advantage of the graph theoretical properties of data are called graph based methods.

The single-link and complete-link hierarchical methods discussed in section 2.2.1 can be approached from a graph theoretical view. More computationally efficient algorithms for single and complete link hierarchical methods than the ones already presented are described in [Gower and Ross 1969; Hansen and Delattre 1978; Jain and Dubes 1988].

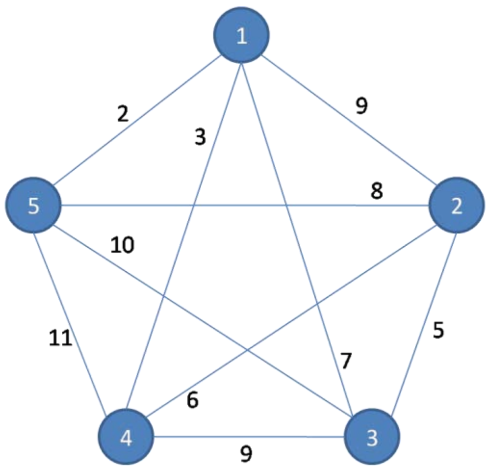
A minimum spanning tree (MST) of a connected, undirected, weighted graph is a subgraph that connects all its edges without cycles (tree) with minimum weight. Several methods for finding a minimum spanning tree of a graph have been developed [Kruskal 1956; Prim 1957]. In [Jain and Dubes 1988] the following algorithm for the single-link method using a minimum spanning tree is given, where the data is represented by a complete weighted graph $G = (V, E, W)$, V being the vertices of the graph representing the objects or data points to be clustered, E being the set of edges connecting all pairs of vertices and W being the weights of the edges representing the distance between two points:

1. Begin with each object in its own cluster and find the MST of G;
2. Merge the two clusters connected by the MST edge with smallest weight to define the next clustering;
3. Replace the weight of the edge selected in 2 by a weight larger than the largest proximity;
4. Repeat steps 2 and 3 until all objects are in one cluster.

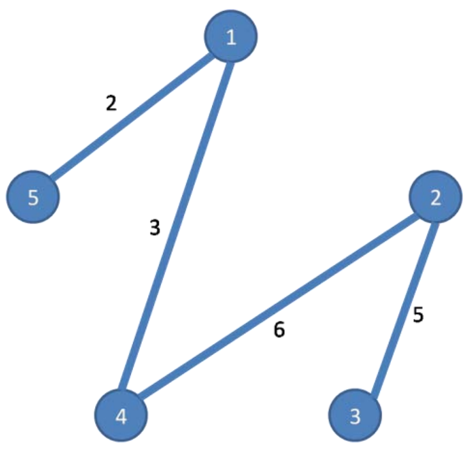
Figure 2.3 presents an example of this procedure. The information in the distance matrix D serves as a basis for the construction of the graph in Figure 2.3 (b). Figure 2.3 (c) depicts a possible minimum spanning tree for this graph. Merging the clusters corresponding to connected vertices in the MST from the smallest to the largest edge weight gives the dendogram in Figure 2.3 (d).

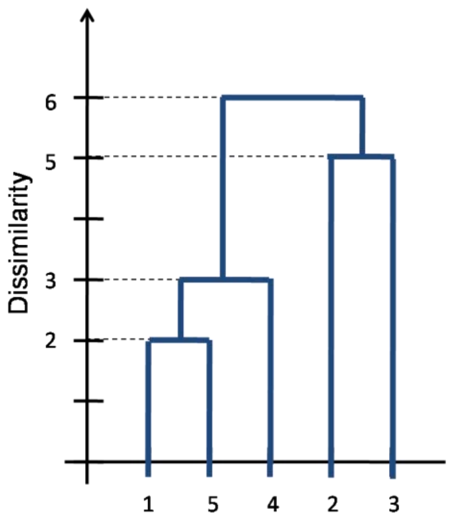
$$D = \begin{array}{ccccc|c} 1 & 2 & 3 & 4 & 5 & \\ \hline 0 & & & & & 1 \\ 9 & 0 & & & & 2 \\ 7 & 5 & 0 & & & 3 \\ 3 & 6 & 9 & 0 & & 4 \\ 2 & 8 & 10 & 11 & 0 & 5 \end{array}$$

(a)



(b)





(c) (d)

Figure 2.3: Example of single-link method using a MST: (a) distance matrix; (b) weighted graph; (c) MST; (d) Dendogram

Just as the single-link method can be approached using a minimum spanning tree, the complete-link method can be approached using node colouring theory [Hansen and Delattre 1978]. Other graph based methods for clustering data are reviewed in [Gan, Ma et al. 2007].

2.2.4 Metaheuristics

Heuristic approaches consist on a search strategy starting from a given feasible or unfeasible solution or set, an iterative process designed to favour the improvement of the solutions regarding feasibility and value and a stopping criterion. In [Colin 1993], the following definition of heuristic is given:

Definition 2.1 – A heuristic is a technique which seeks good (i.e. near-optimal) solution at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is.

♦

The most classical clustering methods in statistics and data mining, namely hierarchical clustering methods and partitioning methods, like K-Means [Gan, Ma et al. 2007], are heuristic. They take advantage of the problem structure to find good solutions but they cannot guarantee optimality.

The most basic heuristic methods may be trapped at local optima. Although it is possible that this local optimum is also the global optimum in general this will not be the case. To overcome this deficiency more sophisticated and elaborated heuristics incorporate techniques to increase the search space and escape local optima. With this purpose, in recent decades more algorithms that use information regarding the search process itself have been developed. These methods are designated as metaheuristics. In [Hillier and Lieberman 2005], the following definition of metaheuristics in given.

Definition 2.2 – A metaheuristic is a general kind of solution method that orchestrates the interaction between local improvement procedures and higher level strategies to create a process that is capable of escaping from local optima and performing a robust search of a feasible solution.

♦

Among the most well-known metaheuristics we have Simulated Annealing, Genetic Algorithms and Tabu Search.

Simulated Annealing, proposed by [Kirkpatrick, Gelatt et al. 1983], mimics the process of healing and cooling of material. At each iteration of the algorithm we move from the current solution to a neighbour solution, similarly to what happens in a descent heuristic for minimization. However, instead of moving always in the direction of improvement, worse solutions are accepted with a probability that depends on the magnitude of increase of the cost function (in a minimization problem) and on a parameter representing the temperature of the system. This parameter is decreased during the algorithm, simulating the cooling of material, until the temperature is close enough to zero. Following thermodynamics rules, at high temperatures the probability of accepting a randomly generated neighbor solution is higher. As the temperature decreases, this probability of acceptance also decreases. Application of the Simulated Annealing algorithm to the clustering problem can be found in [Brown and Huntley 1990; McErlean, Bell et al. 1990; Shokri and Alsultan 1991].

Genetic Algorithms [Holland 1975] are population based methods and are inspired in Charles Darwin theory of evolution. During the algorithm, a population consisting of a usually large set of solutions (chromosomes) is evolved through crossover and mutation operators. Pairs of solutions (parents) are chosen randomly to serve as input for the crossover operator that will generate one or more children. Fittest members are more likely to become parents, thus the next generation tends to be more fitted than the current one, following the natural selection and the principle of survival of the fittest. Additionally, with a typically small probability, mutation of one or more genes (variables) of a chromosome occurs. Through the natural selection process, at the end of the algorithm we expect a population of good quality solutions. Genetic Algorithms have been widely used on the clustering problem. A variety of papers on this subject have been published, for instance [Jiang and Ma 1996; Maulik and Bandyopadhyay 2000; Cheng, Lee et al. 2002; Gautam and Chaudhuri 2004; Jimenez, Cuevas et al. 2007; Petra 2007].

Unlike the two previous metaheuristics, Tabu Search [Glover 1986; Glover and Laguna 1997] is a deterministic process. The keyword in Tabu Search is “memory”. Tabu Search uses different structures of memory – long term and short term memory - to control the search process. In this way it is possible to avoid search cycles, conduct the search to domains of the solution space that would otherwise be skipped, concentrate the search around good quality solutions and avoid getting stuck at local optima. By concentrating the search around good solutions, usually called elite solutions, we are intensifying the search process. On the other hand, by moving to solutions somehow distant to the ones already visited, to avoid local optima, we are diversifying the search process. Efficiency of the Tabu Search Algorithm widely depends on a good balance between these two opposite strategies – intensification and diversification. Just as the previous metaheuristics, Tabu Search has also been applied to the clustering problem [Joyce and Michael 2000; Sung and Jin 2000; Yongguo, Zhang et al. 2008].

In this thesis a Scatter Search algorithm [Glover 1977] will be implemented. In a Scatter Search algorithm a reference set of both good quality and diverse solutions chosen from a larger original set of solutions is sequentially updated to produce better solutions. The algorithm implements both diversification and intensification search strategies to achieve a more intelligent search. Scatter Search algorithms were already applied to the clustering problem in [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006]. The scatter search algorithm that was implemented is based on the algorithms presented in these two papers. The algorithm is presented in detail in section 4.2.

2.2.5 Other Methods

Density-based or grid-based clustering methods are useful for finding arbitrarily shaped clusters consisting of denser regions than their surroundings in large multidimensional spaces. As pointed out in [Gan, Ma et al. 2007], “the grid-based clustering approach differs from the conventional clustering algorithms in that it is concerned not with the data points but with the value space that surrounds the data points”. The main idea of a density-based cluster is that for each point of a cluster the density of points in its ε-neighbourhood, for some $\varepsilon > 0$, has to exceed some threshold [Ester, Kriegel et al. 1996]. The most well-known density-based algorithm, proposed by [Ester, Kriegel et al. 1996], is called DBSCAN.

For high dimensional data it is hard to find good clusters using conventional clustering algorithms. Dimension reduction or feature selection techniques can be used before performing clustering, thus reducing the dimensionality of the data to be clustered. However, these approaches imply a loss of information and consequently the clusters obtained may not fully reflect the original structure of a given data set [Gan, Ma et al. 2007]. The goal of subspace clustering or projected clustering is to find clusters embedded in subspaces of the original data space with their own associated dimensions. The first subspace clustering algorithm, CLIQUE, was proposed by [Agrawal, Gehrke et al. 1998]. Other subspace clustering algorithms were proposed by [Agrawal, Gehrke et al. 1998; Aggarwal and Yu 2000; Procopiuc, Jones et al. 2002], among others. In this thesis we are clustering data points representing the parameters of membership functions belonging to a certain family of membership functions, typically Triangular, Trapezoidal or Gaussian membership functions. Since these families of membership functions can be described using a small number of parameters, the dimensionality of the data involved is low. Therefore, the methodology for subspace clustering will not be further described. Details on some of these algorithms can be found in [Gan, Ma et al. 2007].

2.3 Formulations in Integer Programming

In this section some formulations of the clustering problem to be solved are given. In these formulations only binary and integer variables will be used. The problem consists of clustering n fuzzy sets into k clusters, $1 \leq k \leq n$. The number of clusters is not known *a priori*. In all formulations d_{ij} denotes the distance between fuzzy sets i and j. If the fuzzy sets are represented in R^p, the Euclidean Distance defined by (1.27) or other distance for comparing objects in R^p can be used. It is also possible to use distance measures based on similarity measures for comparing fuzzy sets. The formulations presented are as general as possible and do not assume any particular distance measure.

2.3.1 A Binary Linear Programming Formulation - I

This first formulation is a linear programming formulation using only binary variables.

$$Min \quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^2 \sum_{k=1}^{n} y_{ijk} + \alpha \sum_{k=1}^{n} z_k \qquad (2.13)$$

$s.t.$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad , \quad i \in \{1, \cdots, n\} \tag{2.14}$$

$$x_{ik} + x_{jk} - 1 \leq y_{ijk} \quad , \quad i, j, k \in \{1, \cdots, n\} \tag{2.15}$$

$$x_{ik} + x_{jk} \geq 2y_{ijk} \quad , \quad i, j, k \in \{1, \cdots, n\} \tag{2.16}$$

$$x_{ik} \geq y_{ijk} \quad , \quad i, j, k \in \{1, \cdots, n\} \tag{2.17}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} y_{ijk} \leq M\, z_k \quad , \quad k \in \{1, \cdots, n\} \tag{2.18}$$

$$x_{ik} y_{ijk}, z_k \in \{0,1\} \quad , \quad i \in \{1, \cdots, n\}, \quad j \in \{1, \cdots, n\}, \quad k \in \{1, \cdots, n\} \tag{2.19}$$

As can be seen by the integrality conditions (2.19), x_{ik} , y_{ijk} and z_k are binary variables. Variable x_{ik} takes value 1 if and only if point i is in cluster k , y_{ijk} equals 1 if and only if points i and j belong to cluster k and z_k takes value 1 if and only if cluster k is not empty. Notice that $y_{ijk} = x_{ik} \cdot x_{jk}$ and $y_{iik} = x_{ik}$.

One of the most used criteria for clustering is to minimize the sum of squared distances (or equivalently the mean of squared distances) of data points belonging to the same cluster. However, if the number of clusters is not defined *a priori*, this yields an optimal solution where each data point forms a different cluster, with an optimal value of zero. Therefore the objective function has to account for the number of clusters formed. Since $\sum_{k=1}^{n} y_{ijk} = 1$ if points i and j belong to the same cluster and $\sum_{k=1}^{n} y_{ijk} = 0$ otherwise, $\frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^2 \sum_{k=1}^{n} y_{ijk}$ is the mean of squared distances of all pair of points belonging to the same cluster. The number of non-empty clusters is given by $\sum_{k=1}^{n} z_k$ and the parameter $\alpha > 0$ is not only used to control the importance given to both objectives – minimization of mean of squared distances and minimization of the number of clusters – but also to deal with the difference in scales present in the objective function.

Equations (2.14) ensure that each point belongs to exactly one cluster. Equations (2.15) translate that if point i belongs to cluster k ($x_{ik}=1$) and point j belongs to cluster j ($x_{jk}=1$), then both clusters belong to cluster k ($y_{ijk}=1$). The reciprocal is ensured by minimization of the objective function but can also be expressed by equations (2.16) or by equations (2.17). Equations (2.18), where M is a large constant, allow identifying if the clusters are empty or not. If $y_{ijk}=1$ for some i and j then $z_k=1$, i.e., the cluster is not empty. Minimization of the objective function guarantees that $z_k=0$ whenever cluster k is empty.

2.3.2 A Binary Linear Programming Formulation - II

This formulation is another linear programming formulation using only binary variables. In the previous formulation the x_{ik} variables are redundant, since $x_{ik}=y_{iik}$. Also, since $y_{ijk}=y_{jik}\ \ \forall\, i,j\in\{1,\cdots,n\}$, it is possible to further reduce the number of variables in the formulation by considering only variables y_{ijk} for $i\in\{1,\cdots,n\}$ and $j\in\{i,\cdots,n\}$.

$$Min\quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n}d_{ij}^{2}\sum_{k=1}^{n}y_{ijk}+\alpha\sum_{k=1}^{n}z_k \tag{2.20}$$

s.t.

$$\sum_{k=1}^{n}y_{iik}=1\quad ,\quad i\in\{1,\cdots,n\} \tag{2.21}$$

$$y_{iik}+y_{jjk}-1\le y_{ijk}\quad ,\quad i\in\{1,\cdots,n\},\quad j\in\{i+1,\cdots,n\},\quad k\in\{1,\cdots,n\} \tag{2.22}$$

$$y_{ijk}\le z_k\quad ,\quad i\in\{1,\cdots,n\},\quad j\in\{i,\cdots,n\},\quad k\in\{1,\cdots,n\} \tag{2.23}$$

$$y_{ijk},z_k\in\{0,1\}\quad ,\quad i\in\{1,\cdots,n\},\quad j\in\{i,\cdots,n\},\quad k\in\{1,\cdots,n\} \tag{2.24}$$

In this formulation y_{ijk} are binary variables, as can be seen by the integrality conditions (2.24), taking value 1 if and only if points i and j belong to cluster k and z_k is a binary variable that takes value 1 if and only if cluster k is not empty. Notice that $y_{iik} = 1$ if and only if point i belongs to cluster k.

The objective function in (2.20) was already explained in the previous formulation. Equations (2.21) ensure that each point belongs to exactly one cluster, as was the case for equations (2.14). Equations (2.22), similarly to equations (2.15), translate that if point i belongs to cluster k ($y_{iik} = 1$) and point j belongs to cluster j ($y_{jjk} = 1$), then both clusters belong to cluster k ($y_{ijk} = 1$). The reciprocal is ensured by minimization of the objective function. Equations (2.23) allow identifying if the clusters are empty or not. If $y_{ijk} = 1$ then $z_k = 1$, i.e., the cluster is not empty. Minimization of the objective function guarantees that $z_k = 0$ whenever cluster k is empty.

Additional valid inequalities, i.e., constraints that are satisfied by all admissible solutions, can be considered. The following inequalities are just some of the possible valid inequalities that can be used.

$$y_{iik} \geq y_{ijk} \quad , \quad i \in \{1, \cdots, n\}, \quad j \in \{i+1, \cdots, n\}, \quad k \in \{1, \cdots, n\} \tag{2.25}$$

$$y_{iik} + y_{jjk} \geq 2 y_{ijk} \quad , \quad i \in \{1, \cdots, n\}, \quad j \in \{i+1, \cdots, n\}, \quad k \in \{1, \cdots, n\} \tag{2.26}$$

$$\sum_{i=1}^{n} \sum_{j=i}^{n} y_{ijk} \leq M \, z_k \quad , \quad k \in \{1, \cdots, n\} \tag{2.27}$$

Equations (2.25), similarly to equations (2.17) express that if both points i and j are in cluster k, then point i is in cluster k. Equations (2.26) can be immediately obtained from equations (2.25). Just like equations (2.23), Equations (2.27) allow to identify if the clusters are empty or not. These equations could replace equations (2.23), as in the case of the previous formulation.

2.3.3 A Formulation using precedence

In the previous formulation, we do not take advantage of the fact that membership functions have their domain in R. Consider the linguistic variable in Figure 2.4. In the previous formulation, membership functions A_1 and A_3 can belong to the same cluster even if A_2 does not belong to this cluster. Intuitively this should not happen. The space of admissible solutions can be reduced if we consider an ordering of the membership functions.

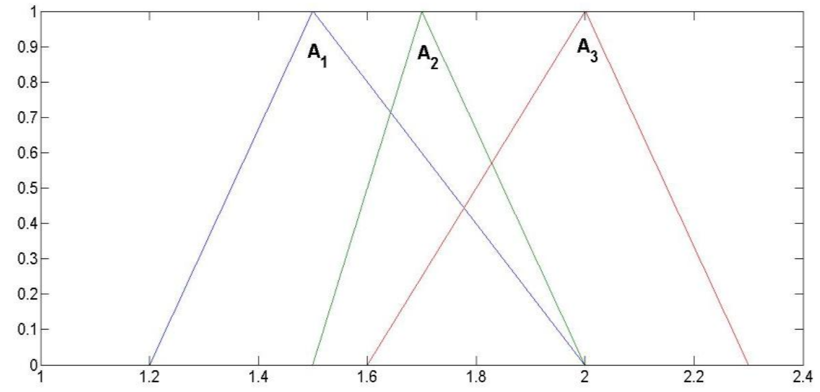


Figure 2.4: Example of a Linguistic Variable with three fuzzy sets

Consider that an ordering of the membership functions to be clustered $A_1 \preceq A_2 \preceq \cdots \preceq A_n$ exists. Then we can formulate the problem if the following way.

$$Min \quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^2 x_{ij} + \alpha\, z_n \tag{2.28}$$

s.t.

$$z_1 = 1 \tag{2.29}$$

$$z_{i+1} - z_i \le 1 \quad , \quad i \in \{1,\cdots,n-1\} \tag{2.30}$$

$$z_{i+1} - z_i \ge 0 \quad , \quad i \in \{1,\cdots,n-1\} \tag{2.31}$$

$$z_j - z_i \geq 1 - x_{ij} \quad , \quad i \in \{1, \cdots, n-1\} \quad , \quad j \in \{i+1, \cdots, n\} \tag{2.32}$$

$$z_j - z_i \leq M(1 - x_{ij}) \quad , \quad i \in \{1, \cdots, n-1\} \quad , \quad j \in \{i+1, \cdots, n\} \tag{2.33}$$

$$z_i \in \{1, \cdots, n\} \quad , \quad i \in \{1, \cdots, n\} \tag{2.34}$$

$$x_{ij} \in \{0,1\} \quad , \quad i \in \{1, \cdots, n-1\} \quad , \quad j \in \{i+1, \cdots, n\} \tag{2.35}$$

where z_i is the number of the cluster that contains membership function A_i, $i \in \{1, \cdots, n\}$, x_{ij} is a binary variable that takes value 1 if and only if membership functions A_i and A_j belong to the same cluster, $i \in \{1, \cdots, n-1\}$, $j \in \{i+1, \cdots, n\}$, and M is an arbitrarily large constant.

The equality in (2.29) guarantees that the first membership function is always in the first cluster. Since $z_i, i \in \{1, \cdots, n\}$ are integers, inequalities (2.30) and (2.31) state that two consecutive membership functions A_i and A_{i+1}, $i \in \{1, \cdots, n\}$ are in the same cluster ($z_{i+1} = z_i$) or A_{i+1} is in the cluster immediately after the cluster that contains A_i ($z_{i+1} = z_i + 1$). Equations (2.32) and (2.33) make the correspondence between the two groups of variables. Membership functions A_i and A_j, $i, j \in \{1, \cdots, n\}$ belong to the same cluster ($x_{ij} = 1$) if and only if they have the same cluster number ($z_i = z_j$).

The objective function has the same meaning as the one in (2.20).

This formulation assumes that an ordering of the fuzzy sets exists. There are several methods for ordering fuzzy sets [Shu-Jen and Hwang 1992]. However, this ordering is not unique. It varies according to the method used. Therefore, an optimal solution to the previous formulation is only optimal for that particular ordering and not for the problem itself.

2.3.4 Quadratic Formulation

The previous formulations were all linear formulations. It is also possible to formulate this problem as a quadratic integer programming problem. Although the problem is easy to formulate with a quadratic objective function, quadratic problems are usually more difficult to solve then linear ones.

$$Min \quad \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^{2} x_{ik} x_{jk} + \alpha \sum_{k=1}^{n} c_k \tag{2.36}$$

$s.t.$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad , \quad i \in \{1,\cdots,n\} \tag{2.37}$$

$$\sum_{i=1}^{n} x_{ik} \le Mc_k \quad , \quad k \in \{1,\cdots,n\} \tag{2.38}$$

$$\sum_{i=1}^{n} x_{ik} \ge c_k \quad , \quad k \in \{1,\cdots,n\} \tag{2.39}$$

$$x_{ik} \in \{0,1\} \quad , \quad i \in \{1,\cdots,n\} \quad , \quad j \in \{1,\cdots,n\} \tag{2.40}$$

$$c_k \in \{0,1\} \quad , \quad k \in \{1,\cdots,n\} \tag{2.41}$$

where x_{ij} is a binary variable that takes value 1 if and only if membership function i is in cluster k, $i,k \in \{1,\cdots,n\}$, c_k is a binary value that takes value 1 if and only if cluster k is not empty, $k \in \{1,\cdots,n\}$, and M is an arbitrarily large constant.

Equations (2.37) state that each membership function is in exactly one cluster. Equations (2.38) and (2.39) are equivalent to $\sum_{i=1}^{n} x_{ik} = 0 \Leftrightarrow c_k = 0$, $k \in \{1,\cdots,n\}$. By identifying if the clusters are empty or not it is possible to get the number of non-empty clusters to be used in the objective function.

2.4 Summary

The problem of reducing the number of membership functions in linguistic variables can be formulated as a clustering problem, as explained before. Therefore, this chapter started by introducing the clustering problem and the *state of the art* in this area (sections 2.1 and 2.2) and proceeded by discussing some integer programming formulations to the clustering problem (section 2.3).

Chapter 3. Exact Methods

The initial purpose of the work in this thesis was not to solve the reduction of membership functions through exact methods. The complexity of clustering problems is one of the main reasons why exact methods are in general not efficient and so finding an optimal solution in a reasonable amount of time will most likely only be possible for small data sets, particularly if the number of clusters is unknown. Nevertheless it seemed important to explain, even briefly, how to approach the problem if a global optimal solution is intended. Therefore, this chapter presents only a brief introduction to some of the exact methods used for combinatorial and integer programming.

Finding an optimal solution of a discrete optimization problem is in general difficult and known methods are not efficient for large instances. The complexity that characterizes these NP-Hard problems has the consequence that the computational implementation of exact algorithms is in general too heavy in terms of memory and too time-consuming for large problems. Partial enumeration methods, like Branch-and-Bound [Land and Doig 1960] or Branch-and-Cut [Wolsey 1998], are examples of such algorithms. The dimension of the instances above which is no more practical to apply an exact method varies according to the problems under study. This is one reason why exact methods should always be, at least, tested before switching to a heuristic approach. Cluster problems are among those problems for which a dimensionality above 40 variables makes the application of exact methods almost impractical [Lourenço 1995].

3.1 Branch-and-Bound

The Branch-and-Bound algorithm [Land and Doig 1960] is a *divide and conquer* technique that implicitly enumerates all feasible solutions of an integer (or mixed integer) linear programming problem. The three main aspects of this algorithm are the branching, fathoming or pruning and bounding strategies used. The original problem is divided into smaller problems by the branching strategy, usually represented by a solution tree. The bounding strategy tries to update the lower and upper bound on the optimal value of the objective function, L and U, by solving the linear relaxations of the integer problems considered, providing information that allows pruning some of the branches of the solution tree.

Consider the integer linear programming maximization problem defined by (3.1).

$$\begin{aligned} &Max \quad Z = cx \\ &s.t. \quad\; Ax \le b \\ &\qquad\quad x = (x_1, \cdots, x_n) \ge 0 \quad \text{and integers} \end{aligned} \tag{3.1}$$

and its linear relaxation

$$\begin{aligned} &Max \quad Z = cx \\ &s.t. \quad\; Ax \le b \\ &\qquad\quad x = (x_1, \cdots, x_n) \ge 0 \end{aligned} \tag{3.2}$$

To initialize the upper bound U the linear relaxation (3.2) at the root node is solved through a linear programming method. The lower bound is set to $L = -\infty$.

If the optimal solution x^* of the linear relaxation problem is integer, i.e., if $x_1, \cdots, x_n$ are integers, then this is also the optimal solution of the integer problem. Otherwise, a branching variable x_j is chosen among the basic variables that have non-integer values in this solution and two sub-problems are considered by adding the constraints $x_j \le \lfloor x_j^* \rfloor$ and $x_j \ge \lfloor x_j^* \rfloor + 1$ to (3.2), where $\lfloor a \rfloor$ stands for the largest integer smaller or equal to *a*.

The bounding strategy is applied for each new sub-problem. If an integer optimal solution for one of the sub-problems is found, we may try to update L because this solution is a feasible solution of the original problem. If z_{sub} denotes the objective function value of such solution we have $L = \max\{L, z_{sub}\}$.

The pruning strategy allows reducing the number of nodes in the solution tree that need to be explicitly visited. If a sub-problem satisfies one of the following conditions – pruning by optimality, pruning by bound or pruning by infeasibility

[Wolsey 1998] - the corresponding node will node be branched. These conditions are presented below:

1. Pruning by optimality – an integer optimal solution to the sub-problem was found;
2. Pruning by bound – $z_{sub} < L$, i.e., solutions found by branching this node will always be worse than a feasible known solution whose value is equal to the lower bound;
3. Pruning by infeasibility – the sub-problem (and thus all possible branches of this node) is infeasible.

The branching, bounding and pruning steps are iteratively applied to each sub-problem until there are no remaining non-pruned sub-problems or until $L = U = Z^*$. In this case either an optimal solution was found or the problem is infeasible. It is also common to stop the algorithm when the amplitude of the interval $[L, U]$ is small, where the concept small is given by considering an error measure and threshold for this error, but in this case optimality is not guaranteed.

3.2 Branch-and-Cut

The Branch-and-Cut algorithm [Wolsey 1998] is a hybrid of Branch-and-Bound and cutting plane algorithms. A cutting plane for an integer programming problem is a valid inequality, i.e., a constraint that is satisfied by all admissible solutions, that reduces the admissible region of the linear programming relaxation.

Several implementations of this algorithm exist. Basically, cutting planes are generated during the Branch-and-Bound algorithm. The goal is to find better bounds in each node in order to reduce the number of nodes to be visited. As stated in [Wolsey 1998], "though this may seem to be a minor difference, in practice there is a change of philosophy". Instead of quickly solving the node problems, emphasis is given to improving the formulation at each node.

Other than generating cutting planes, additional strategies can be used to improve the formulation at each node. Some of these strategies consist of fixing variables to the only possible value that can take part in an optimal solution or eliminating redundant constraints. The efficiency of a Branch-and-Cut algorithm depends on a good implementation of such strategies. Knowing when to include or eliminate constraints is a major aspect of this algorithm. Although general implementations exist, to solve a specific (and more complex) problem an implementation that takes advantage of the underlying problem structure should be developed.

3.3 Branch-and-Price

The Branch-and-Price algorithm [Barnhart, Johnson et al. 1998] is another variation of Branch-and-Bound. Just like in Branch-and-Cut, emphasis is given to the strategies employed in each node to obtain better solutions or better bounds. However, instead of using cutting planes (row generation) to improve the formulations at each node, column generation methods are used.

This algorithm is especially suited for solving problems with a large number of variables. The basic idea is that in many problems most of the variables will have a zero value in the optimal solution. By using column generation, a master problem corresponding to the original problem but where only a subset of variables is considered can be more efficiently solved. To identify which columns should enter the master problem, subproblems based on the dual linear programming problem, called pricing problems, are solved. This allows choosing variables with positive (negative) reduced cost in the minimization (maximization) problem that should therefore enter the master problem. When no such variables exist and the integrality conditions are not satisfied, branching is performed as in the original Branch-and-Bound algorithm.

3.4 Computational Results

To better understand the dimension of the problem and the difficulty of using exact methods for clustering some computational experiments were done. These experiments were done using data from the case study that will be presented in

Chapter 5. The formulation presented in section 2.3.1 was implemented in GAMS and latter run on CPLEX. In these experiments the distance between two membership functions i and j, d_{ij}, was chosen to be $d_{ij} = 1 - s_{ij}$, where s_{ij} is the Jaccard Similarity given by (1.19) using the fuzzy minimum and fuzzy maximum, given by (1.10) and (1.11), as intersection and union operators.

All experiments were done in Pentium(R) 4 CPU 2.6 GHz, 504 MB of RAM.

First we considered linguistic variables with 12 membership functions each. The results are summarized in Table 3.1. Instead of running CPLEX until an optimal solution was found (and proved to be optimal) a threshold of 10% for the relative gap between the lower and upper bounds on the objective function was used as a stopping criterion. As can be seen in Table 3.1, CPLEX took less than 2 minutes – 40.41 seconds in average – to stop. Given these results we ran CPLEX for linguistic variables with 54 membership functions to see if exact methods could still be used to solve these problems in a reasonable amount of time. However, for these problems CPLEX stopped because lack of memory, without returning an optimal solution. These results show what was already expected by the combinatorial nature of clustering problems: exact methods can only deal with very small data sets.

	Solution	Best Possible	Absolute Gap	Relative Gap	Elapsed Time (sec.)	Number of Clusters
Rotation Current	0.767505	0.690776	0.076729	0.099972	95.063	8
Rotation Voltage	0.898916	0.809514	0.089402	0.099455	33.672	8
Rotation Speed	0.768401	0.694777	0.073624	0.095815	19.313	7
Thrust	0.939391	0.845645	0.093746	0.099794	41.172	10
Torque	0.75501	0.680006	0.075003	0.099341	45.922	7
Translational Voltage	0.58426	0.527426	0.056835	0.097276	30.219	5
Translational Current	0.555146	0.501321	0.053825	0.096956	26.531	4
Translational Speed	0.684707	0.616784	0.067923	0.099199	31.422	7

Table 3.1: Computational Results

3.5 Summary

In this Chapter some exact methods for solving integer problems were briefly described. These methods consist of a set of strategies to methodically examine the search space of an integer or mixed integer problem without having to implicitly enumerate all possible solutions. Even though these methods allow to optimally solve many problems that by explicit enumeration could not be solved in a reasonable amount of time, for a wide class of combinatorial problems the search for an optimal solution is still too time-consuming. When this is the case, heuristic methods such as the ones described in Chapter 4 can provide good quality solutions with less computational effort without guaranteeing optimality.

Chapter 4. Heuristic Methods Based on Local Search

In real applications, the dimension and complexity of combinatorial and integer problems and the need to find good solutions in useful time have lead to the development of algorithms that take advantage of the problem structure to achieve good solutions (not necessarily optimal). Computational implementations of these algorithms, contrary to exact methods, are quite efficient regarding time and memory. Whenever the application of global optimization methods is not advisable, it is still usually possible to find good quality solutions by using heuristic methods.

As was pointed out in section 2.2.4, the most classical clustering methods in statistics and data mining are heuristic and can therefore be trapped at local optima. For this reason, a variety of metaheuristics have been applied to the clustering problem.

In this thesis a Scatter Search algorithm was implemented. Although this metaheuristic is not as well-known as the metaheuristics described in section 2.2.4, it already proved to be efficient at finding good quality solutions for many problems. Scatter Search has been applied to find solutions to the nodes graph coloring problem [Jean-Philippe and Jin-Kao 2002], to vehicle routing problems [Russell and Chiang 2006], to clustering problems [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006], among many other applications.

Both a heuristic and a metaheuristic to solve the automatic clustering problem were implemented in Matlab. Section 4.1 describes a heuristic approach called K-Means++ [David and Sergei 2007]. Section 4.2 describes the general Scatter Search algorithm and the details of this particular implementation. In Section 4.3 computational results on two case studies are presented in order to compare these two implementations.

4.1 A heuristic approach: K-means ++

One of the most used algorithms for clustering data is the K-Means algorithm [McQueen 1967], already described in section 2.2.2. The algorithm starts by choosing initial K cluster centres from the original data X. After the initialization, a partition of the data is determined by assigning each point to the cluster with closest centre. After this assignment the centroids of each cluster are calculated and the points are reassigned to the clusters regarding the closeness to the centroids. Again the centroids are recalculated and the algorithm proceeds in the same way until some stopping criterion is met.

A variation of this method, called K-Means++ [David and Sergei 2007], was implemented in Matlab for finding a feasible solution of the clustering problem. This method differs from the original K-Means algorithm in the way the initial clusters are chosen. Sections 4.1.1 and 4.1.2 describe the K-Means++ algorithm used for partitioning n points into K clusters. Section 4.1.3 discusses how to choose the correct number of clusters by evaluation of cluster validity indexes.

4.1.1 Initialization

The K-Means algorithm starts by choosing K cluster centres from the original data to be clustered. Usually the cluster centres are chosen uniformly at random from the original data, i.e., they are chosen with equal probabilities. The K-Means++ [David and Sergei 2007] differs from the original K-Means algorithm in the way the initial cluster centres are chosen. Cluster centres are still chosen randomly, but they are not chosen uniformly. After the first cluster centre is chosen uniformly at random from the original data, the remaining $K-1$ centres are chosen proportionally to their distance to the centres already chosen, a method referred in [David and Sergei 2007] by “D^2 weighting”. The empirical reasoning of this rule is to diversify the location centres within the set of points to allow for a better assignment of points to clusters.

Let $D(x)$ denote the shortest distance from a point $x \in X$ to the closest centre already chosen. In this work we used the Euclidean distance and so $D(x)$ is given by:

$$D(x) = \min_{i \in S} \{ d(x, c_i) \} \tag{4.1}$$

where S is the set of all already chosen cluster centres and $d(\cdot,\cdot)$ is the Euclidean distance.

Then the necessary steps to perform the cluster centres initialization are the following:

1. Choose an initial centre uniformly at random from X, i.e., $p_i = \frac{1}{n}$, $i = 1, \cdots, n$, where p_i denotes the probability of choosing x_i.
2. Choose the next centre randomly according to the probability distribution $p_i = \frac{D(x_i)^2}{\sum_{j=1}^{n} D(x_j)^2}$, $i = 1, \cdots, n$.
3. Repeat step 2 until K centres have been chosen.

We should notice that the probability of choosing a point is proportional to the distance to the closest already chosen centre. So the further away the point is the likely it is that it will be chosen as a new centre. After the K initial cluster centres are chosen, the algorithm proceeds as the original K-Means. The iterative procedure is described in the next section.

4.1.2 Iterative Procedure

Now that the initial cluster centres are chosen, the remaining points are assigned to its closest cluster and the cluster centroids are updated. This procedure is repeated until a stopping criterion is met. The steps of this procedure, after the initialization phase, are summarized below.

1. Assign each point to the closest centre.
2. Update cluster centres by recalculating the cluster centres according to (2.9).
3. Repeat steps 1 and 2 until the centres no longer change.

In addition, a maximum number of iterations could be used as a stopping criterion. However, due to the rapid convergence of the algorithm, in the computational experiments that will be presented in section 4.3, it was not necessary to prematurely stop the algorithm.

4.1.3 Choosing the number of clusters

The previous algorithm partitions n data points into K clusters (or less, because empty cluster might be formed). However, when the number of clusters is not known *a priori*, the correct number of clusters has to be estimated. Usually this is done by running the algorithm for a range of values for K and choosing the best partition according to some cluster validity index.

In [Ujjwal and Sanghamitra 2002] several validity indexes for clustering algorithms are compared using different clustering algorithms. The experiments conducted by the authors lead them to the following conclusion: “Compared to the other considered validity indexes, I is found to be more consistent and reliable in indicating the correct number of clusters”. This index is defined by equation (4.2).

$$I(K) = \left(\frac{1}{K} \times \frac{E_1}{E_K} \times D_K \right)^p \tag{4.2}$$

where
$$E_K = \sum_{k=1}^{K} \sum_{j=1}^{n} u_{kj} \left\| x_j - c_k \right\| \tag{4.3}$$

and
$$D_K = \max_{i,j=1,\cdots,K} \left\| c_i - c_j \right\| \tag{4.4}$$

In these equations $X = \{x_1, \cdots, x_n\}$ is the data to be clustered and $U = \lfloor u_{kj} \rfloor_{K \times n}$ is a partition binary matrix representing a possible clustering of the data into K disjoint clusters, i.e., $u_{kj} = 1$ if and only if x_j is in the kth cluster. The centroid of cluster k is denoted by c_k. To find the correct number of clusters we chose the value of K which maximizes $I(K)$.

Analyzing $I(K)$ we see that as the error E_K decreases, the factor $\frac{E_1}{E_K}$ increases. It is always possible to obtain a partition with zero error by considering n clusters, each consisting of a single data point. To balance the error with the number of cluster the factor $\frac{1}{K}$ is introduced. As the number of clusters decreases, this factor increases. To achieve well separated clusters, the factor D_K should be large, that is, the maximum distance between two cluster centres should be large. The previous considerations intuitively justify that $I(K)$ should be maximized.

4.2 Scatter Search

Scatter Search [Glover 1977] has some similarities to Tabu Search and Genetic Algorithms. The use of memory is one of the main features of Tabu Search and is usually present in Scatter Search. Such as Genetic Algorithms, Scatter Search is an evolutionary algorithm. While in Genetic Algorithms an usually large population is evolved through crossover and mutation operations, in Scatter Search instead of a population it is used a smaller reference set (composed of good quality solutions and diverse solutions) and it plays the most important role in the algorithm.

Essentially, Scatter Search operates on a small set of solutions, the reference set, and consists on the application of the following methods, which can be implemented in different ways, according to the problem at hand:

1. A Diversification Generation (DG) method to produce a collection of diverse trial solutions from which the initial reference set is built;
2. An Improvement (Imp) method to enhance the quality of trial solutions;
3. A Reference Set Update (RSU) method responsible for constructing a reference set of both high quality solutions and diverse solutions from the collection of solutions obtained by the diversification generation method and of updating this reference set when new solutions are created during the algorithm;

4. A Subset Generation (SG) method that, in each iteration of the algorithm, creates a collection of subsets of solutions belonging to the reference set, such that the solutions in each subset are to be combined through the solution combination method;
5. A Solution Combination (SC) method that takes a subset of solutions given by the subset generation method and generates one or more new trial solutions.

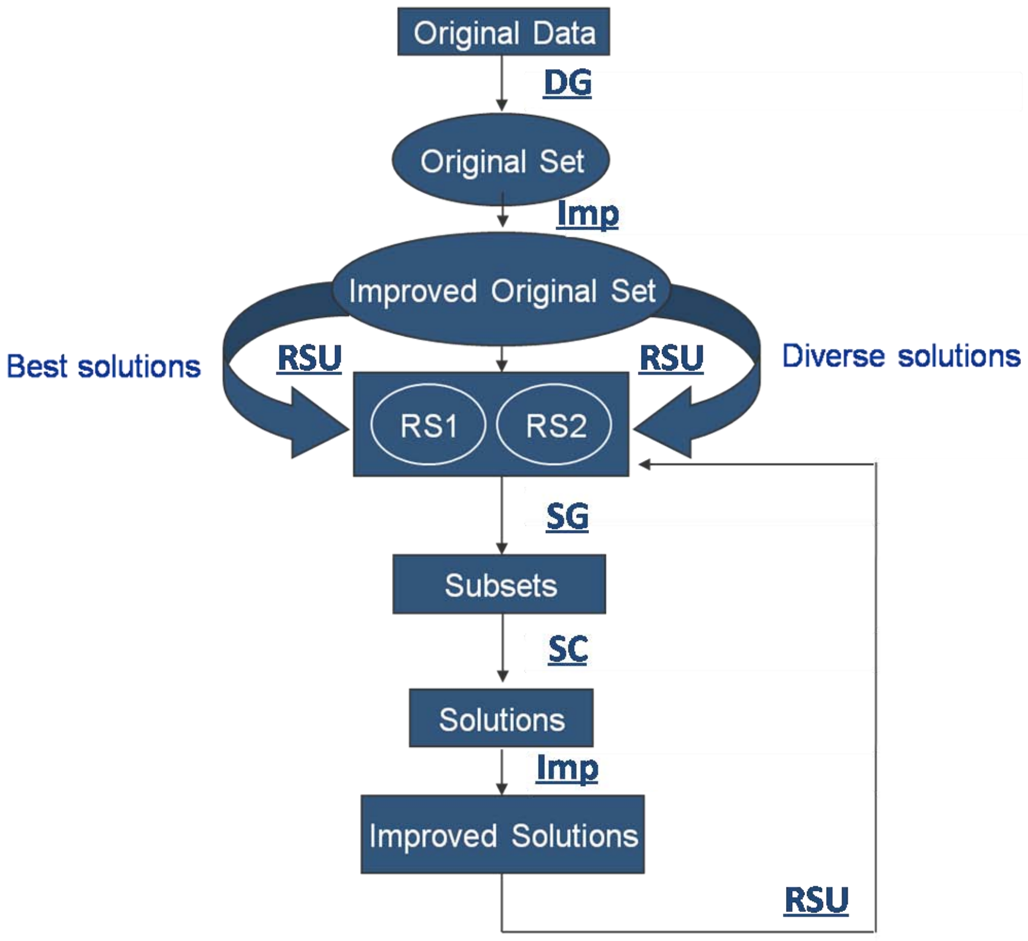


Figure 4.1: Scatter Search Algorithm

The way the previous methods operate is summarized in Figure 4.1. The original set is created by the Diversification Generation Method (DG). Each solution in this set is then improved by the Improvement Method (Imp) before the Reference Set Update Method (RSU) constructs the reference set, selecting the best quality solutions in the original set, along with diverse solutions. The subsets of solutions to be combined through the Solution Combination Method (SC) are chosen by the Subset Generation Method (SG). The solutions resulting from this combination are improved before the Reference Set Update Method (RSU) updates the reference set. The process continues until some stopping criterion is met.

The scatter search algorithm that was implemented is based on the algorithms in [Pacheco 2005] and [Abdule-Wahab, Monmarché et al. 2006]. The next sections describe the implementations of each of the five methods mentioned above.

4.2.1 Fitness Function

In [Pacheco 2005], quality of solutions was measured by the sum of squared distances from each point to the centroid of its cluster. This measure cannot be used in the automatic clustering problem, where the number of clusters is not defined *a priori.* Using this measure when the number of clusters can vary, yields the construction of as many clusters as the number of points to be clustered, giving a sum of the squared distances from each point to the centroid of its cluster (the point itself) of zero. Therefore, a different quality measure was used. The validity index I described in section 4.1.3 and defined by equations (4.2) - (4.4) was used as fitness function, to be maximized.

4.2.2 Diversification Generation Method

The diversification generation method used was proposed by Pacheco [Pacheco 2005], based on GRASP – Greedy Randomized Adaptive Search Procedure. However in this work the number of clusters is predefined by the user. To achieve an automatic clustering procedure, as it is aimed in our work, the correct number of clusters should be determined by the algorithm. Therefore, before creating a new solution with the diversification generation method, it is necessary to generate a random number of clusters, i.e., an integer K between 1 and $K_{\max}$, where $K_{\max}$ is the maximum number of clusters allowed.

Given the number of clusters K to be considered for the generation of a solution, the cluster centres $S = \{c_1, \cdots, c_K\}$ are randomly chosen from the data set X in the following way [Pacheco 2005]:

1. Find x_{j^*}, the farthest point from the centroid of X and do $c_1 = x_{j^*}$ and $S = \{c_1\}$. Set $h = 2$.
2. Fix $0 \le \alpha \le 1$ and while $|S| < K$ do:
    a. Determine $\Delta_j = \min\{\|x_j - c_l\| : c_l \in S\}, \forall x_j \in X \setminus S$
    b. Determine $\Delta_{\max} = \max\{\Delta_j : x_j \in X \setminus S\}$
    c. Do $L = \{x_j : \Delta_j \ge \alpha\, \Delta_{\max}\}$
    d. Choose $x_{j^*} \in L$ uniformly at random and do $c_h = x_{j^*}$, $S = S \cup \{c_h\}$ and $h = h + 1$.

If $\alpha = 0$, the cluster centres are chosen completely at random from the original data X. If $\alpha = 1$ the process is deterministic if there is only one point in L, and so the only farthest point from the centres already chosen will enter S. Therefore, generally speaking, the parameter α controls the level of randomization of the process.

A memory structure is used to avoid repetition of centres and consequently of solutions. The number of times that a point x_j is selected as a centre is stored in $freq(j)$ and the values of Δ_j in subsequent iterations are modified according to equation (4.6), where:

$$freq_{\max} = \max\{freq(j) : \forall j\} \tag{4.5}$$

and β controls the importance of memory in the diversification generation method.

$$\Delta'_j = \Delta_j - \beta \Delta_{\max} \frac{freq(j)}{freq_{\max}} \tag{4.6}$$

Equation (4.6) decreases the value of Δ_j proportionally to $freq(j)$ and so the possibility of inclusion of x_j in L also diminishes.

After the clusters centres are defined, the remaining points are assigned to these clusters. This is done with the goal of minimizing the sum of squared distances from each point to its cluster centre. When the number of clusters is not predetermined, minimizing the sum of squared distances from each point to its cluster centre yields a solution where each point is a centre itself and we have as many clusters as points. However, since for a particular solution to be generated the number of clusters is previously determined, the remaining points can be assigned in order to minimize this measure, as in [Pacheco 2005], in the following way:

1. Let A be the set of unsigned points, i.e., $A = X \setminus S$.
2. For each point $x_j \in A$ and each cluster $C_i\ , i = 1, \cdots, K$ determine Γ_{ij} given by

$$\Gamma_{ij} = \frac{|C_i|}{|C_i| + 1} \left\| c_i - x_j \right\|^2 \tag{4.7}$$

   where c_i is the centroid of C_i.
3. Calculate $\Gamma_{i^* j^*} = \min\left\{\Gamma_{ij} : x_j \notin A, i = 1, \cdots, K\right\}$.
4. Assign x_{j^*} to C_{i^*} and set $A = A \setminus \left\{x_{j^*}\right\}$.
5. If $|A| \neq 0$ return to 2, else stop.

The formula in (4.7) gives the increase in terms of sum of squared distances from each point to its cluster centre when point x_j is assigned to cluster C_i. Steps 1 through 5 define a greedy heuristic for assigning the remaining points to the clusters whose centres were previously chosen.

This algorithm is used to generate OS_{size} initial solutions, called the original set that will serve as basis for constructing the reference set.

4.2.3 Improvement Method

The improvement method is used to enhance the quality of the solutions generated both during the diversification phase and after the combination of two solutions. In this thesis it was chosen to implement the improvement method presented in [Abdule-Wahab, Monmarché et al. 2006], based on the K-Means algorithm [Gan, Ma et al. 2007]. The following steps are taken a number of times equal to *MaxIterImp,* where *MaxIterImp* is a parameter to be chosen by the user.

1. For each point $x_j \in X$ do:
   a. For each cluster $C_i \ ,i = 1,\cdots,K$ determine v_{ij} given by
   $$v_{ij} = \frac{|C_i|}{|C_i|+1}\left\|c_i - x_j\right\|^2 - \frac{|C_l|}{|C_l|+1}\left\|c_l - x_j\right\|^2 \tag{4.8}$$
   where x_j currently belongs to cluster C_l and c_i and c_l are the centroids of C_i and C_l, respectively.
   b. Determine $a = \arg\min_{j=1,\cdots,K,\, j\neq l}\{v_{ij}\}$.
   c. If $a < 0$ reassign x_j to C_a.
2. Compute the fitness of the new solution obtained.
3. If the fitness of the new solution, given by equation (4.2) is better than the original solution, replace the original solution by the new solution.

The formula in (4.8) is given by Spath [Spath 1980] to simplify the K-Means algorithm and approximates the increase in terms of sum of squared distances from each point to its cluster centre when point x_i is moved from cluster C_l to cluster C_j.

4.2.4 Reference Set Update Method

The reference set, RS , is composed of b_1 high quality solutions and b_2 diverse solutions. To construct the initial reference set, the first b_1 best solutions are inserted in the reference set, where the quality of solutions is given by the fitness function presented in section 4.2.1. Next, b_2 solutions are added one by one to reference set according to its diversity. In this work it was used the diversity measure proposed by Pacheco[Pacheco 2005]. Let $dif(\lambda, \lambda')$ be the number of assignments in solution λ that are different from the assignments in solution λ'. For instance, suppose that we have an instance with 5 objects and 4 clusters. Consider two solutions, $y_1 = (1,1,2,3,2)$ and $y_2 = (1,2,2,4,3)$, where the i-element of the vector corresponds to the cluster to which object i is assigned. In this case $dif(\lambda, \lambda')$ is equal to 3.

Iteratively, it is chosen to enter the reference set, the solution that maximizes:

$$\delta_{\min}(\lambda) = \min\{dif(\lambda, \lambda') : \lambda' \in RS\} \tag{4.9}$$

In this implementation the reference set is only updated when better quality solutions are found. Other implementations [Abdule-Wahab, Monmarché et al. 2006] also update the reference set according to the measure of diversity, reinforcing the diversification strategy.

4.2.5 Subset Generation Method

In each iteration of the algorithm, the subsets of solutions from the reference set that will be latter combined by the solution combination method consist of pairs of solutions. The collection of subsets created through this method is composed of all pairs of solutions from the reference set (i, j), $i = 1, \cdots, b_1 + b_2 - 1$, $j = 2, \cdots, b_1 + b_2$, $i < j$. Supposing that we have 3 solutions, we should consider the following subsets (1,2), (1,3), (2,3).

Each element of the collection of subsets generated by this method serves as an input to the solution combination method that generates one or more trial solutions that, after being enhanced by the improvement method in section 4.2.3, can enter the reference set as described in section 4.2.4.

4.2.6 Solution Combination Method

To combine two solutions it was implemented the path relinking strategy described in [Pacheco 2005]. The idea of path relinking is that, in the “path” between two good quality solutions other good quality solutions should exist. A “path” is a series of simple movements that lead from one solution to another. In this case we can start on one solution and move points from one cluster to another until the second solution is reached. For instance, consider two solutions, $y_1 = (1,1,2,3,2)$ and $y_2 = (1,2,2,4,3)$ to be combined. A path between these solutions could be given by: $(1,1,2,3,2) - (1,2,2,3,2) - (1,2,2,4,2) - (1,2,2,4,3)$, where in each movement the first point assigned differently than in y_2 is assigned as in y_2. Solutions in this path can be chosen as trial solutions.

As in [Pacheco 2005], given two solutions, the number of trial solutions that will be generated through the solution combination method varies. If the two solutions to be combined were chosen from the b_1 high quality solutions in the reference set, three trial solutions will be generated. If the two solutions were chosen from the b_2 diverse solutions in the reference set, only one solution will be generated. Otherwise, two solutions will be created. These solutions are randomly chosen from the solutions in the path.

4.2.7 The Final Algorithm

After the basic methods of the scatter search algorithm have been described, we may now describe the final algorithm.

The algorithm starts by generating the original set through the diversification method. The best quality solutions and most diverse solutions are chosen to form the reference set before the iterative part of the algorithm starts. In each iteration, the subset generation method forms all subsets consisting of pairs of solutions from the reference set. These pairs of solutions are then combined through the solution combination method and the generated trial solutions are improved through the improvement method. The reference set update method is then responsible for deciding if any of the generated solutions should replace one of the solutions in the reference set. The iterative procedure continues until there are no new elements in the reference set. The best solution is then returned.

4.3 Computational Results

In this section computational results on two case studies will be presented. In both case studies real world data sets are used. Fuzzyfication of the data is needed before using the previously described algorithms. The reason for fuzzyfying data comes either from a way to deal with different source of uncertainty or to categorize numerical data. Depending on the application that it will be given to the fuzzified data, it may be needed to reduce the number of membership functions to simplify the system that will use this functions. The idea is to reduce the number of terms in each linguistic variable by merging membership functions in the same cluster. These linguistic variables could then be used, for instance in a fuzzy inference system or other fuzzy model. All experiments were conducted in a Intel Core2 Duo, CPU 2.2 GHz, 2 GB of RAM.

4.3.1 Wisconsin Diagnostic Breast Cancer Data Set

The data used in this section is taken from [Asuncion 2007]. Wisconsin Diagnostic Breast Cancer (WDBC) data set contains 569 samples of data describing characteristics of the cell nuclei present in digitalized images of a fine needle aspirate (FNA) of a breast mass. Ten real-valued features were computed for each cell nucleus [Asuncion 2007]:

a) Radius (mean of distances from centre to points on the perimeter)
b) Texture (standard deviation of gray-scale values)
c) Perimeter
d) Area
e) Smoothness (local variation in radius lengths)

f) Compactness
g) Concavity (severity of concave portions of the contour)
h) Concave Points (number of concave portions of the contour)
i) Symmetry
j) Fractal dimension (“coastline approximation” -1)

For each of these features, the mean, standard error and mean of the three largest values (“worst”) of these features were computed for each image. Therefore, each sample has the following 32 attributes:

1. ID number
2. Diagnosis (M = malignant, B = benign)
3. Mean Radius
4. Mean Texture
5. Mean Perimeter
6. Mean Area
7. Mean Smoothness
8. Mean Compactness
9. Mean Concavity
10. Mean Concave Points
11. Mean Symmetry
12. Mean Fractal dimension
13. Radius Standard Deviation
14. Texture Standard Deviation
15. Perimeter Standard Deviation
16. Area Standard Deviation
17. Smoothness Standard Deviation
18. Compactness Standard Deviation
19. Concavity Standard Deviation
20. Concave Points Standard Deviation
21. Symmetry Standard Deviation

22. Fractal dimension Standard Deviation
23. Worst Radius
24. Worst Texture
25. Worst Perimeter
26. Worst Area
27. Worst Smoothness
28. Worst Compactness
29. Worst Concavity
30. Worst Concave Points
31. Worst Symmetry
32. Worst Fractal dimension

In this thesis only features 3 through 22 where used. The mean radius can be used as a measure of the cell nuclei radius. The radius standard deviation gives a measure of the error associated with this measure. For this reason it is advisable to fuzzify the data as a way to deal with uncertainty. To do so we represent each sample's radius by a symmetric triangular membership function $(\mu, 2\sigma)$, as described in section 1.3.1 with μ equal to the mean radius. The width of the triangular membership functions was chosen to be 4σ because in a random variable following a normal distribution, approximately 95% of the samples are expected to belong to the interval $[\mu - 2\sigma, \mu + 2\sigma]$. The same fuzzification scheme, and with the same reasoning, was used for the rest of the features, resulting in 10 linguistic variables with 569 membership functions each, depicted in Figure 4.2 through Figure 4.11. The objective is to reduce the number of membership functions in each linguistic variable using the algorithms described in Chapter 4. The resulting linguistic variables could then be used in a fuzzy inference system or other fuzzy model to diagnose the type of cancer. The construction of such model is outside the scope of this thesis.

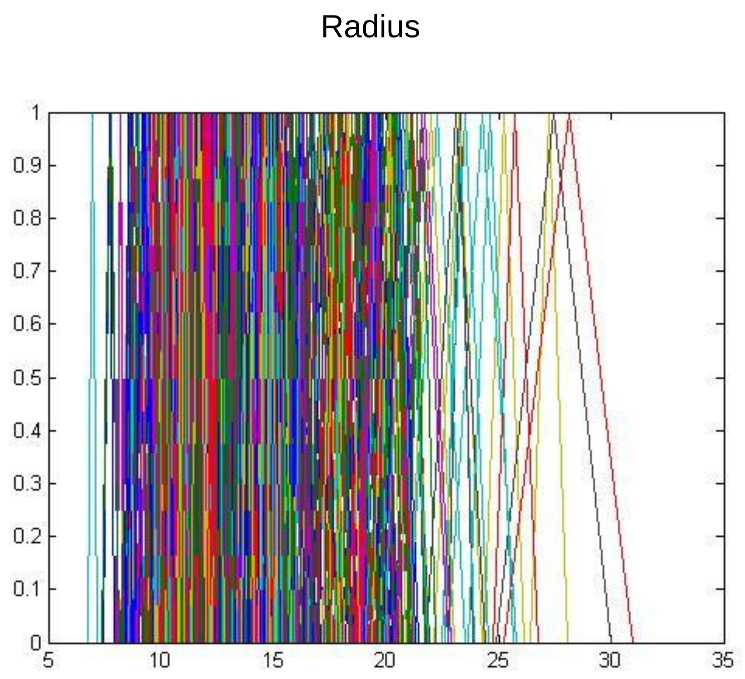


Figure 4.2: Linguistic Variable Radius

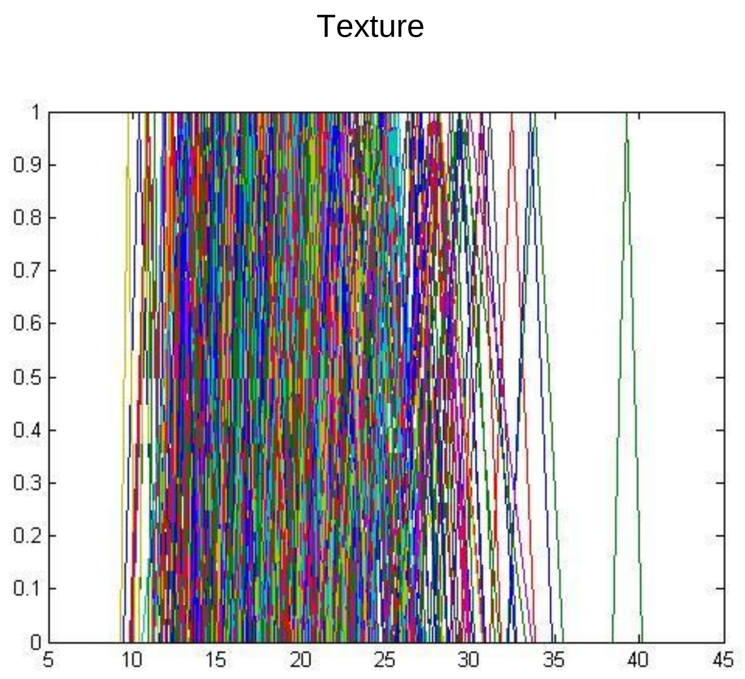


Figure 4.3: Linguistic Variable Texture

Perimeter

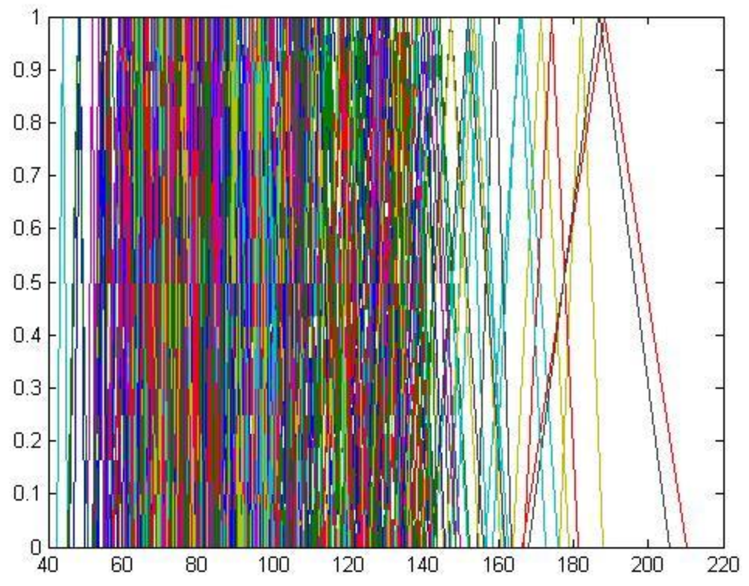


Figure 4.4: Linguistic Variable Perimeter

Area

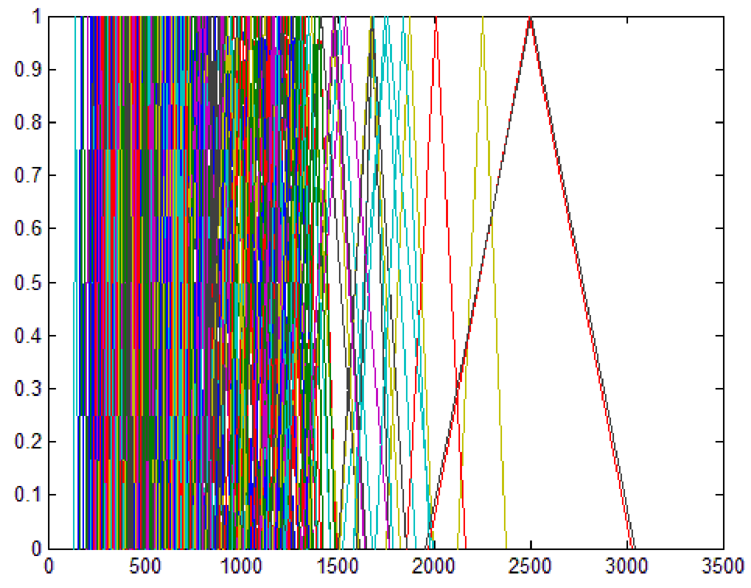


Figure 4.5: Linguistic Variable Area

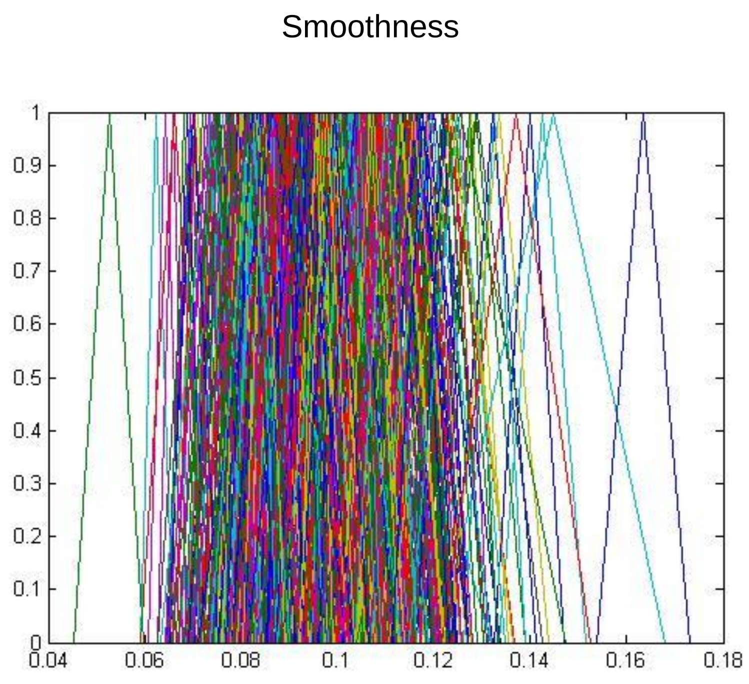


Figure 4.6: Linguistic Variable Smoothness

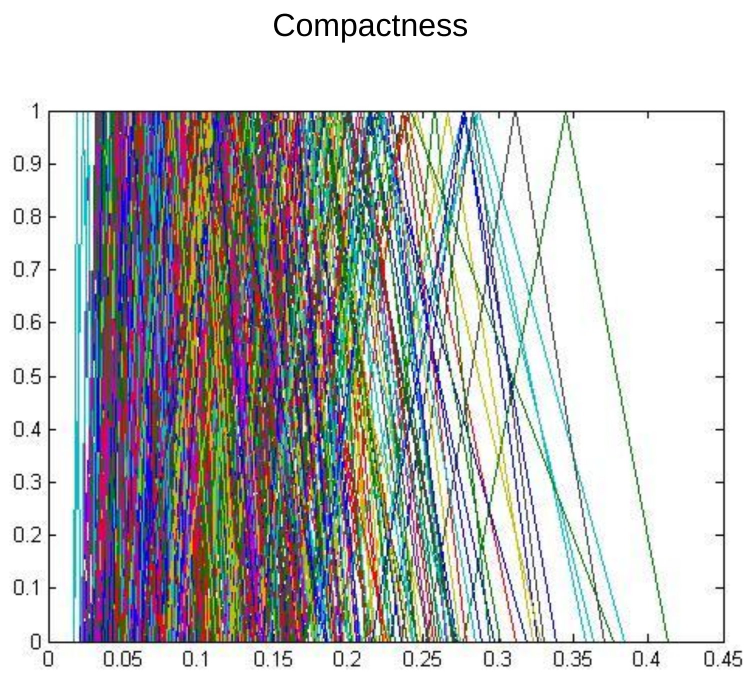


Figure 4.7: Linguistic Variable Compactness

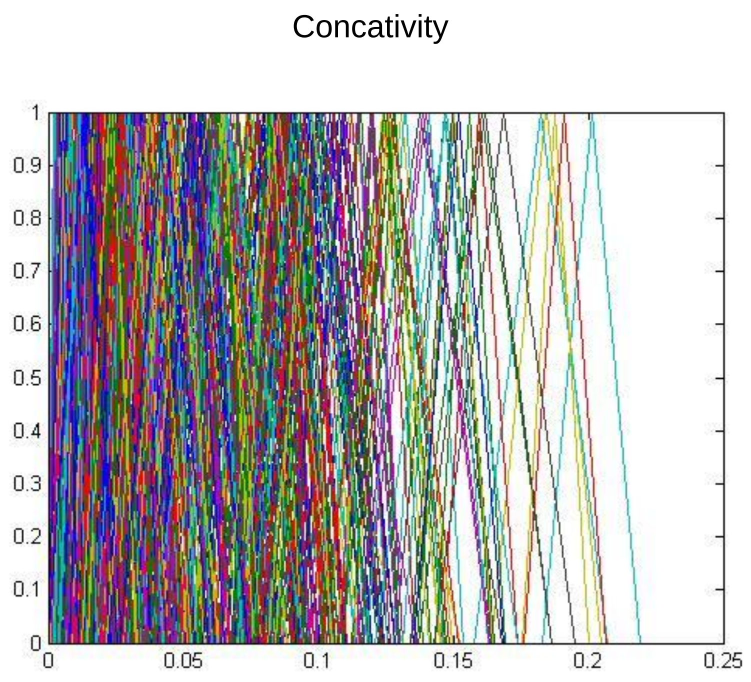


Figure 4.8: Linguistic Variable Concativity

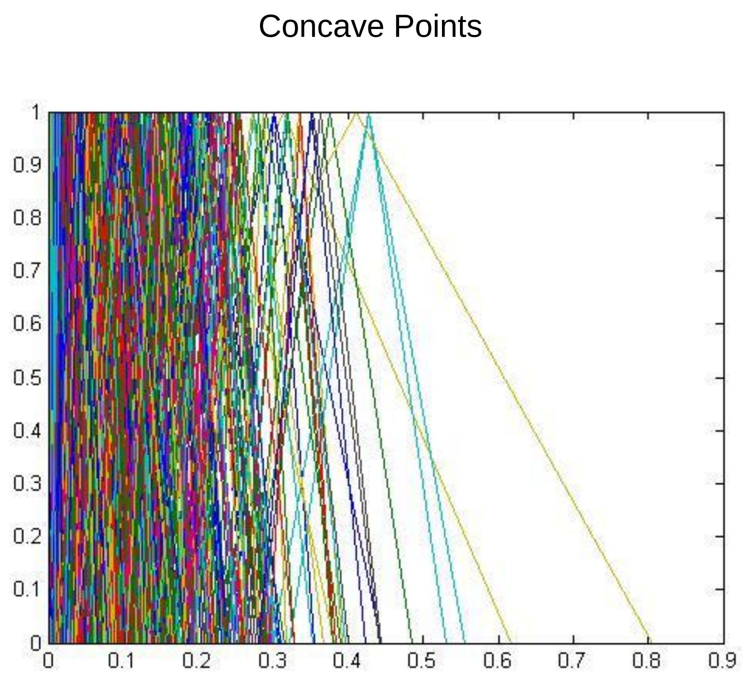


Figure 4.9: Linguistic Variable Concave Points

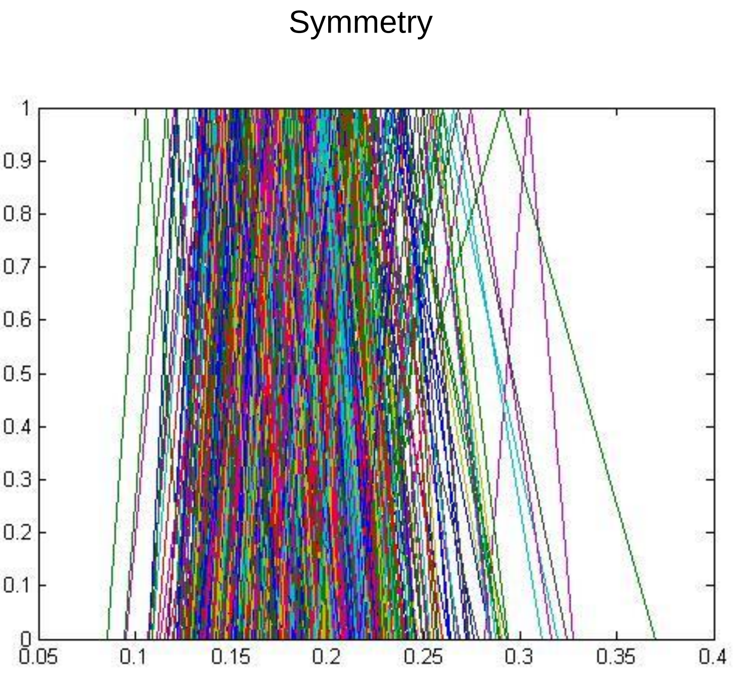


Figure 4.10: Linguistic Variable Symmetry

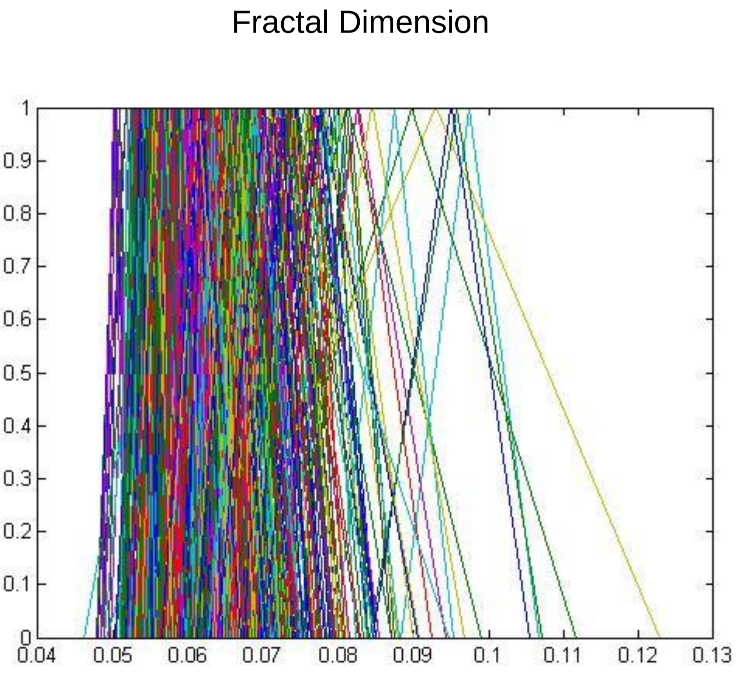


Figure 4.11: Linguistic Variable Fractal Dimension

4.3.1.1 Computational results

The K-Means++ algorithm, described in section 4.1, was applied to each linguistic variable previously presented. The number of clusters was estimated by running the algorithm for $1 \leq K \leq 568 = n-1$ and choosing the iteration that maximizes the evaluation measure I given in (4.2). For $K = n$ the evaluation measure I is not defined since $E_n = 0$, as can be seen in (4.3). Figure 4.12 and Figure 4.14 show the evolution of I using K-Means++ for $1 \leq K \leq 568$, for linguistic variables Radius and Texture. For all other linguistic variables in this case study the overall behaviour is the same. It seems that I increases with K. However, looking at Figure 4.13 and Figure 4.15 it is possible to see that this is not always that case.

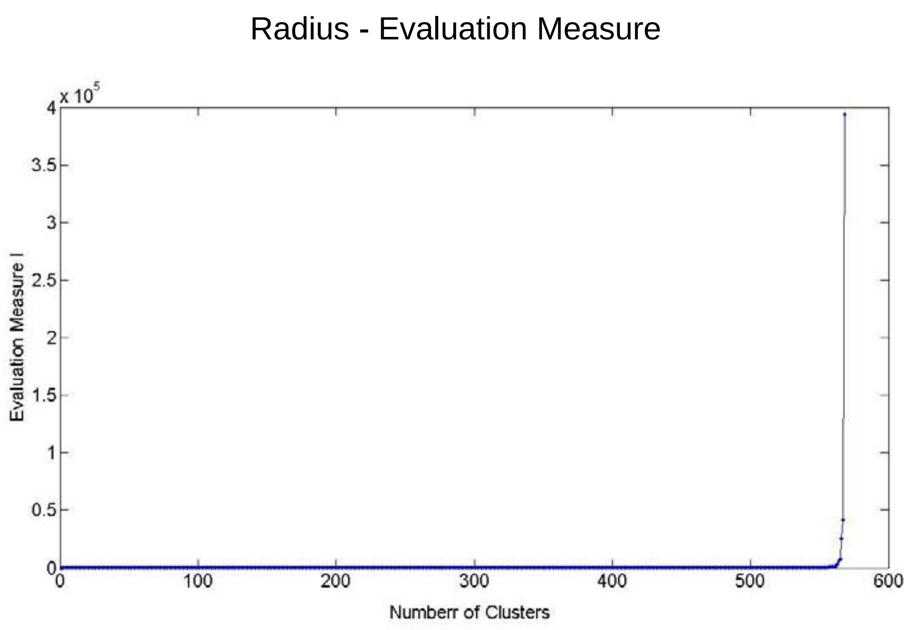


Figure 4.12: Evaluation measure I using K-means++ for $1 \leq K \leq 568$, linguistic variable Radius

Radius - Evaluation Measure – zoom

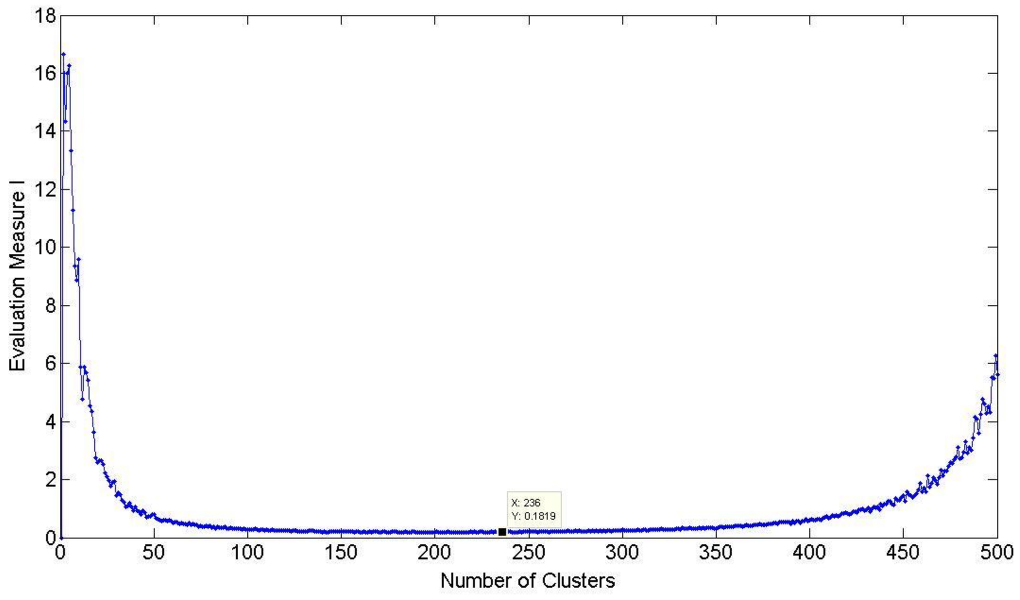


Figure 4.13: Evaluation measure I using K-means++ for $1 \le K \le 500$ (zoom in of previous plot) $1 \le K \le 568$, linguistic variable Radius

Texture - Evaluation Measure

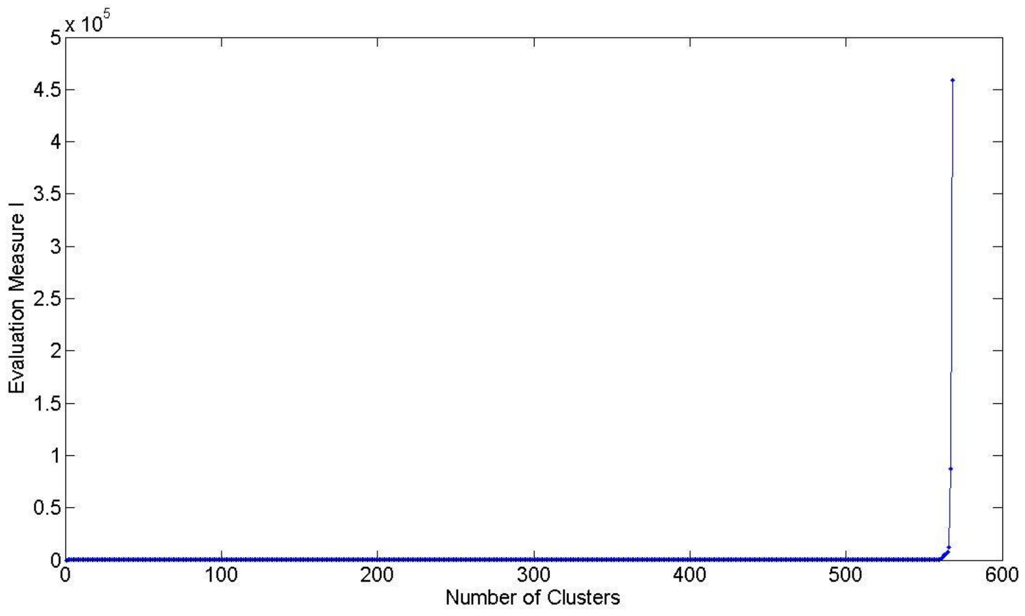


Figure 4.14: Evaluation measure I using K-means++ for $1 \le K \le 568$, linguistic variable Texture

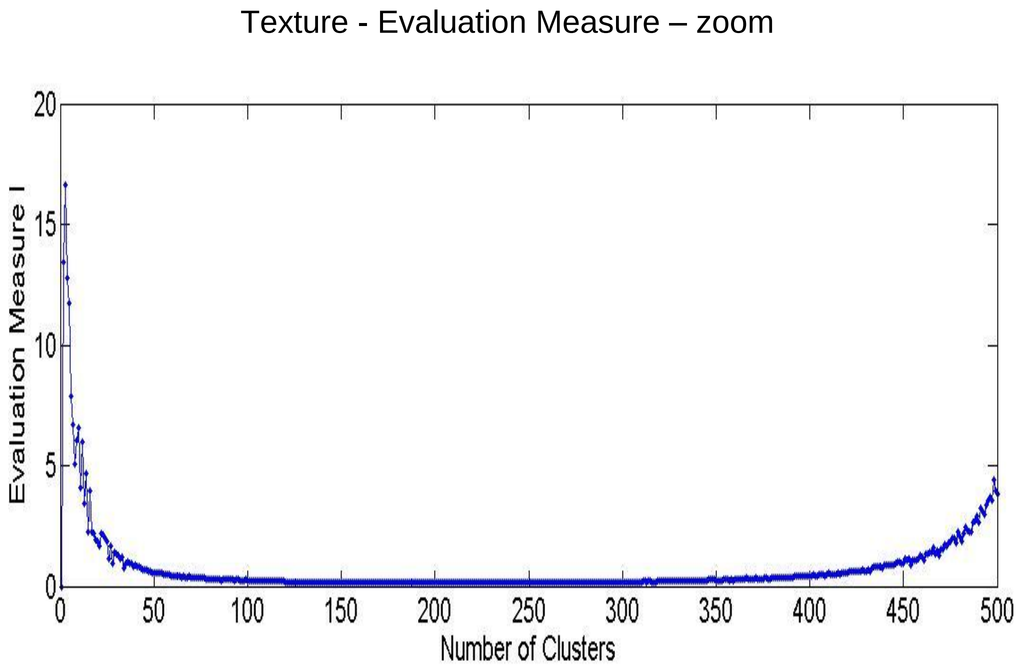


Figure 4.15: Evaluation measure I using K-means++ for $1 \leq K \leq 500$ (zoom in of previous plot), linguistic variable Texture

Running the Scatter Search algorithm from section 4.2 with $K_{\max} = n-1$ yield solutions with $n-1$ clusters. This is easily explained by the behavior of I depicted in Figure 4.12 and Figure 4.14. Given this results it was necessary to redefine the maximum number of clusters allowed, $K_{\max}$. Taking into account Figure 4.13 and Figure 4.15, it was chosen $K_{\max} = 100$. Therefore we are interested in finding a partition of the data into less that 101 clusters that maximizes the cluster validity index I.

The Scatter Search algorithm was run with several values for the parameters α, β, *MaxIterImp* and b_1. To reduce the number of parameters to be analyzed, the number of good quality solutions and of diverse solutions to be included in the reference set was chosen to be equal ($b_1 = b_2$) and the original set was chosen to be 10 times the size of the reference set ($OS_{size} = 10*(b_1 + b_2)$). This last choice is recommended in [Martí, Laguna et al. 1997; Abdule-Wahab, Monmarché et al. 2006]. The algorithm was run until no new elements entered the reference set. Since random numbers are used during the algorithm, for each combination of values of the parameters 5 experiments were run. Only 5 runs of each experiment is clearly not enough to take any statistically valid conclusions, the purpose of these experiments was only to see how results were influenced by different choices of the several parameters involved.

Only the results relative to the linguistic variable Radius will be discussed in more detail. Similar conclusions were found for the rest of the linguistic variables in this case study. Final results will be presented for all linguistic variables.

The first experiments were conducted with no improvement method (i.e. $MaxIterImp = 0$) and all possible combinations of $\alpha \in \{0; 0.5; 0.8; 1\}$, $\beta \in \{0; 0.5\}$ and $b_1 \in \{2; 5\}$, with the purpose of analysing the influence of the parameter α in the algorithm. In section 4.2.2 it is stated that the parameter α controls the level of randomization used when choosing cluster centres. When $\alpha = 0$, the cluster centres are chosen completely at random from the original data X. In Figure 4.16 it is clear that this randomness affects the standard deviation of the fitness of solutions returned by the algorithm. When $\alpha = 0$ the standard deviation of results is much higher than for larger values of α. This standard deviation means that it is likely to achieve very good results but also very bad results, a characteristic that is undesirable in an algorithm. In fact, although in terms of the best result found for each 5 experiments a choice of $\alpha = 0$ seems to produce good results (Figure 4.17), the same does not happen in terms of average results (Figure 4.18). Given this results, no further experiments were done with $\alpha = 0$.

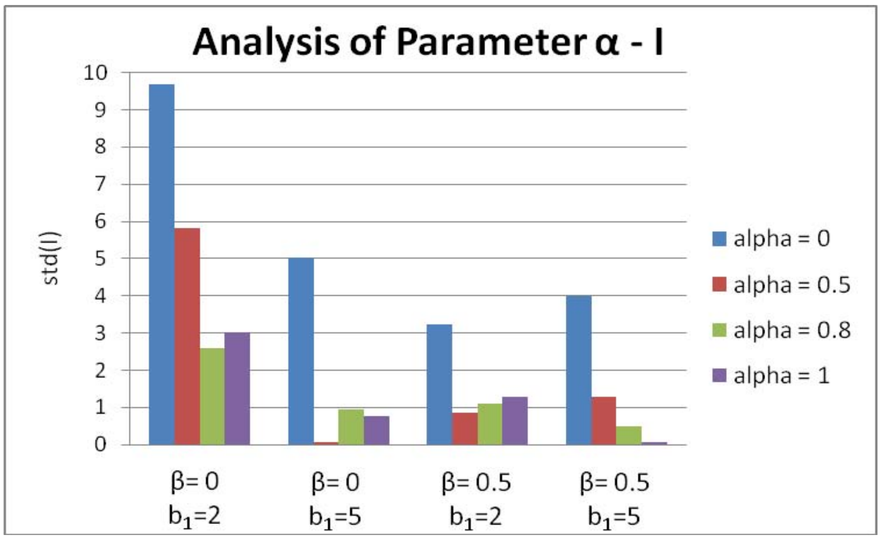


Figure 4.16: Influence of α in fitness function I standard deviation

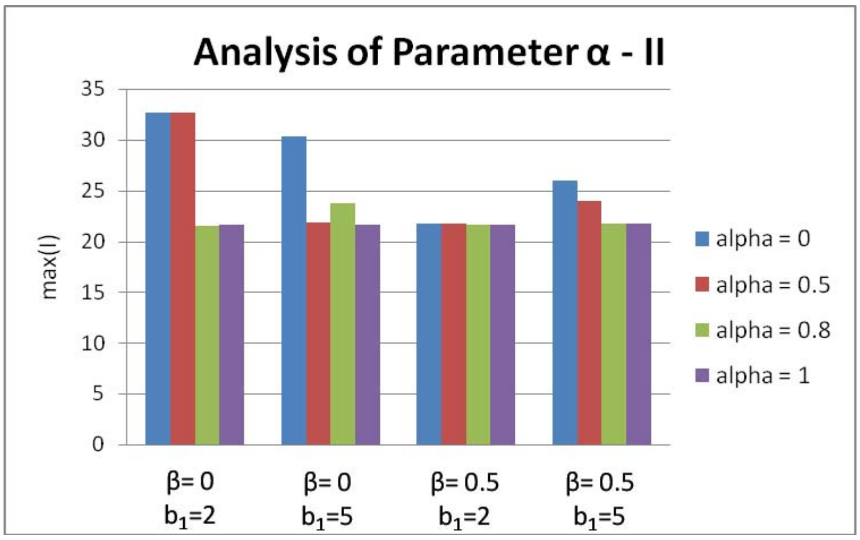


Figure 4.17: Influence of α in best results obtained for fitness function I

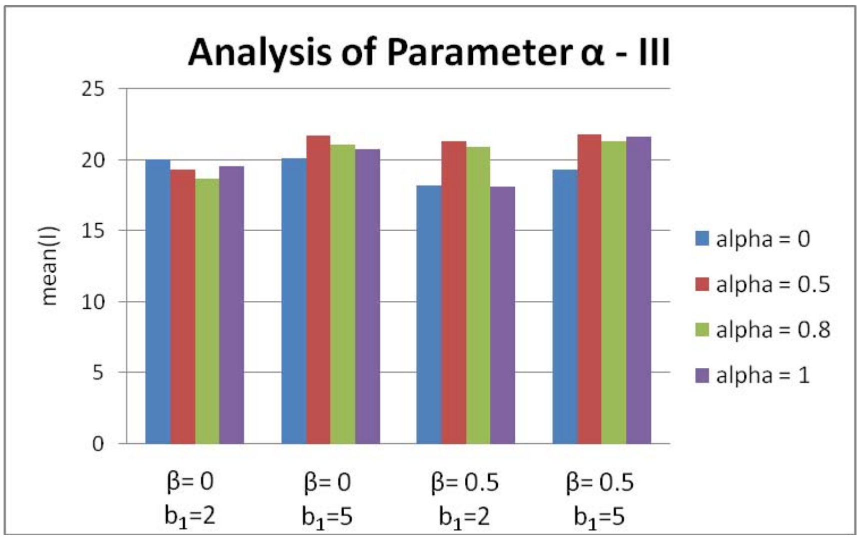


Figure 4.18: Influence of α in mean results for fitness function I

The parameter β controls the weight given to the memory in the process of choosing the cluster centres when creating initial solutions, as given by (4.6). Notice that it does not make sense to study the importance of this parameter for values of α very close to zero since the choice of the cluster centres is close to random. In this case, although the quantities Δ_j are replaced by Δ'_j, this does not significantly change the set L from which the cluster centres are chosen. As can be seen in Figure 4.19, there seems to be an improvement in terms of average results in using this memory during the creation of the original set. Therefore, no further experiments for $\beta = 0$ will be discussed.

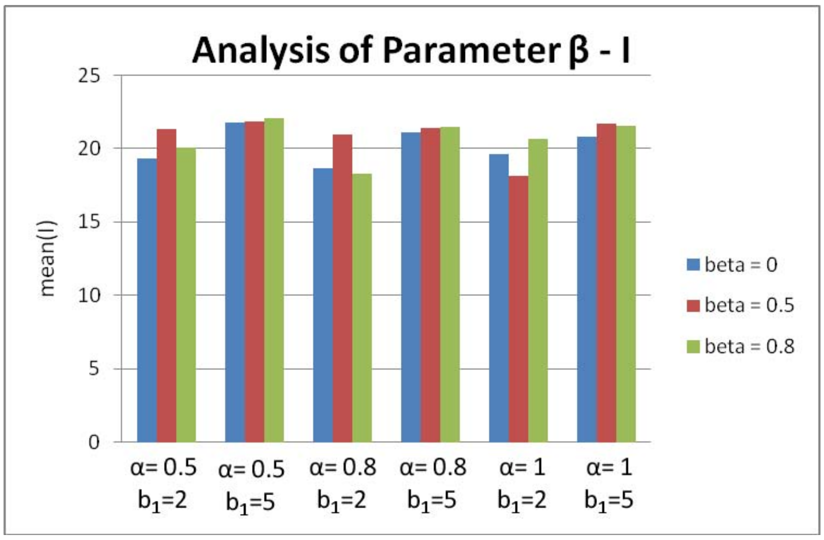


Figure 4.19: Influence of β in mean results for fitness function I

By considering a larger reference set we expect to obtain better results but worse computational efficiency. Not only does it take more time to generate the original set (since the size of the original set was chosen to be 10 times the size of the reference set), but also the number of subsets of solutions to be combined increases exponentially. In Figure 4.20 we can see that increasing the size of the reference set does indeed produce better quality results, but this improvement is achieved at a computational cost, as can be seen in Figure 4.21. However, in this case study, we are considering an increase from an execution time of around 1 minute to around 3.5 minutes. Due to these low execution times, we can afford to consider a larger reference set to obtain better solutions. In other applications where the number of membership functions to merge is higher, this increase in execution time could be unaffordable. Reference sets are typically small. Only two small values for b_1 were considered, $b_1 = 2$ and $b_1 = 5$. This is due to the way the size of original set is related to this parameter. Since it was chosen that $b_1 = b_2$ and $OS_{size} = 10 * (b_1 + b_2)$, we are considering original sets with 40 and 100 solutions. Larger original sets would mean that the initial diversification achieved would be such that a very good quality solution was probably already found in this first step of the algorithm.

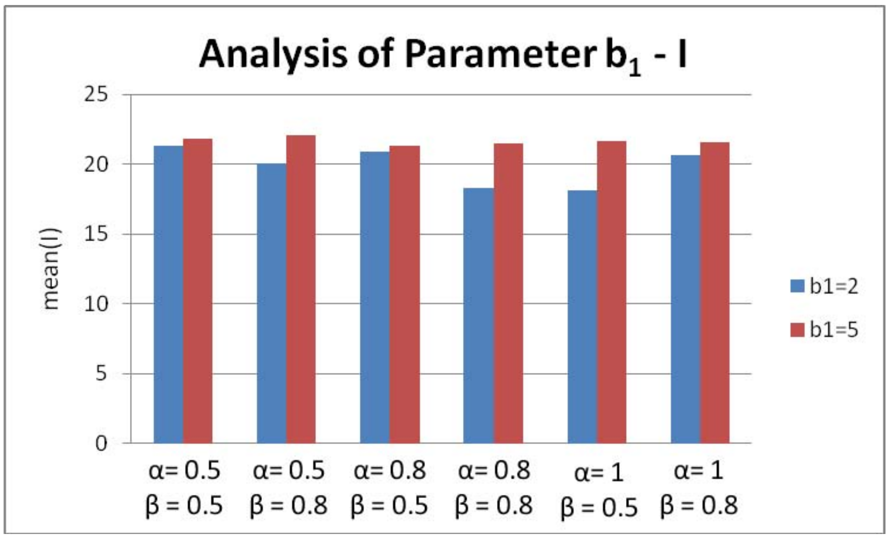


Figure 4.20: Influence of b_1 in mean results for fitness function I

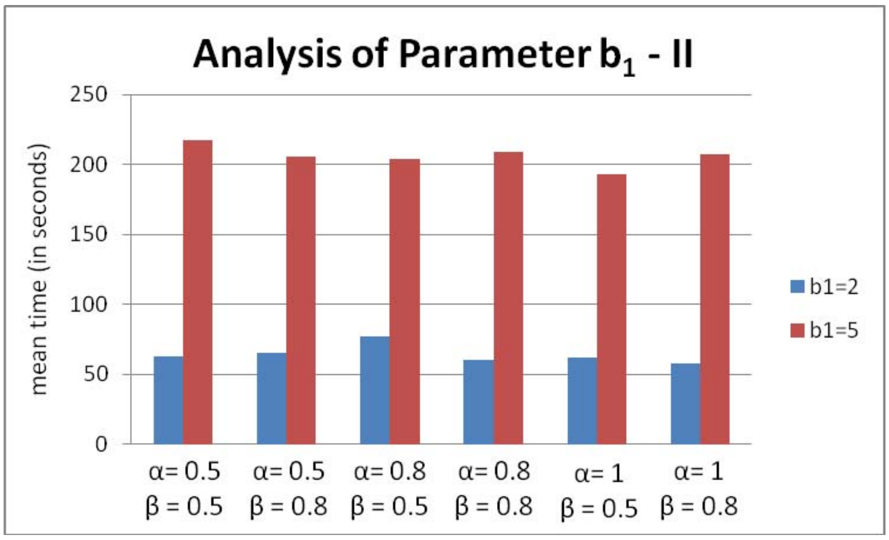


Figure 4.21: Influence of b_1 in execution time

Until this point experiments were done without the Improvement Method. This way the influence of the parameters being analyzed in the solutions obtained was clearer. The computational cost, in terms of execution time (Figure 4.23), of using the Improvement Method should lead to an improvement in the quality of the solutions obtained by the algorithm. However, this was not always the case, as can be seen in

Figure 4.22. The increase in execution time when using *MaxIterImp* $= 5$ is considerably high and does not result in a significant improvement in the quality of solutions. Considering *MaxIterImp* $= 2$ the increase in terms of execution time is not high (from approximately 1 minute to approximately 3 minutes) but only improves the average quality of solutions in some of the sets of experiences made. This result was unexpected. Since only 5 experiments per each choice of parameters were made, these results could be explained by the weak estimate of the actual mean values. Since the increase in execution time for using *MaxIterImp* $= 2$ is not too high, this value will be considered for this parameter. Also, the two first combinations of parameters seem to give consistently better results than the rest. From these two sets of experiences, the second presents slightly better average results. Therefore, in the results presented in Table 4.1 and Figure 4.24 through Figure 4.33 were obtained with $\alpha = 0.5$, $\beta = 0.8$, $b_1 = 5$ and *MaxIterImp* $= 2$. Figure 4.24 through Figure 4.33 show, for both algorithms, a scatter plot of the centre values versus the width of the triangular membership and the final configuration of the linguistic variable.

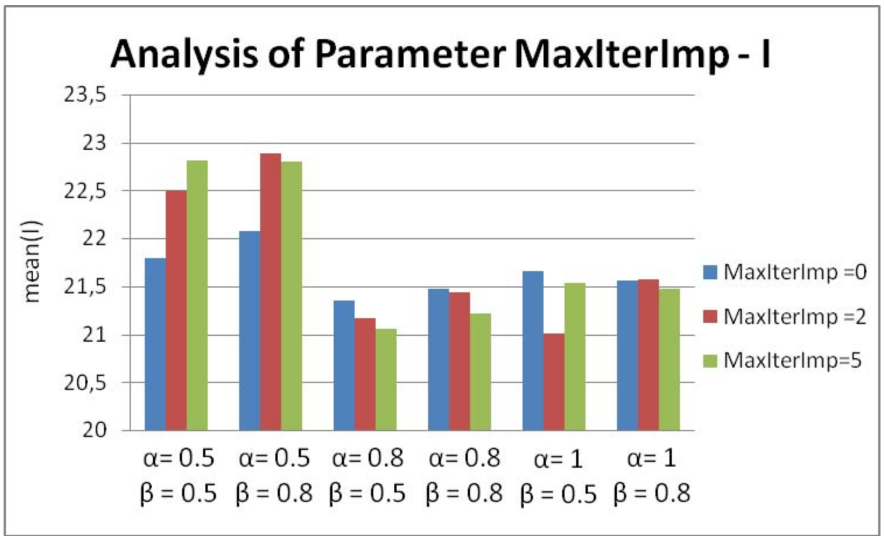


Figure 4.22: Influence of Improvement Method in mean results for fitness function I

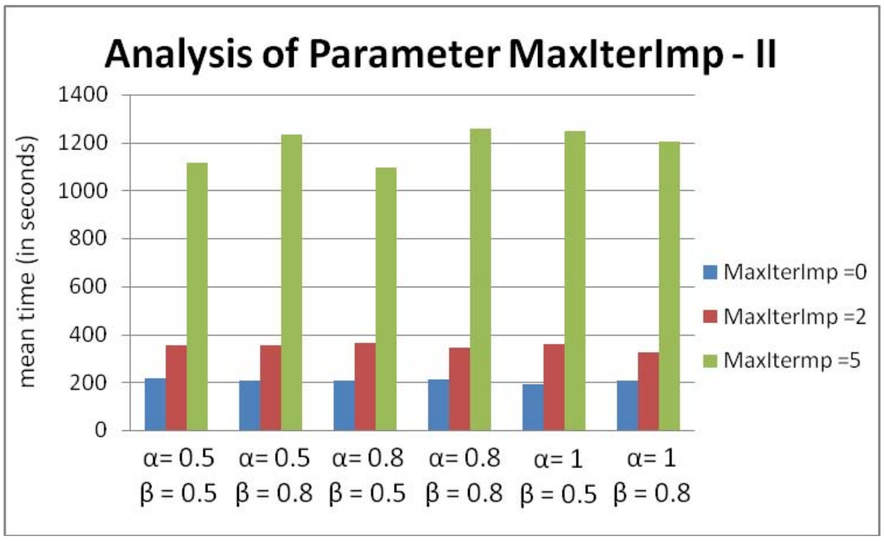


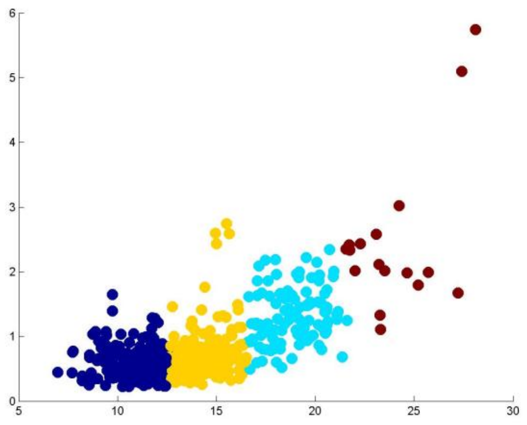
Figure 4.23: Influence of Improvement Method in execution time

To better understand what happens during the algorithm a study of how the reference set evolves was conducted. Surprisingly, for all experiments conducted the reference set was only updated a few times after its initial construction and the best solution found was always generated during the first part of the algorithm. This result was not expected. Unfortunately, the second part of the Scatter Search algorithm is not producing good solutions. Still, Table 4.1 shows that the Scatter Search returned better results than K-Means++ for almost all variables. The first part of the algorithm is sufficient to obtain better quality solutions than the K-Means++. The computational time of the Scatter Search was expected to be much higher than the computational time of the K-Means++. This did not happen only because the Scatter Search algorithm stopped after the first iteration of the second part of the algorithm.

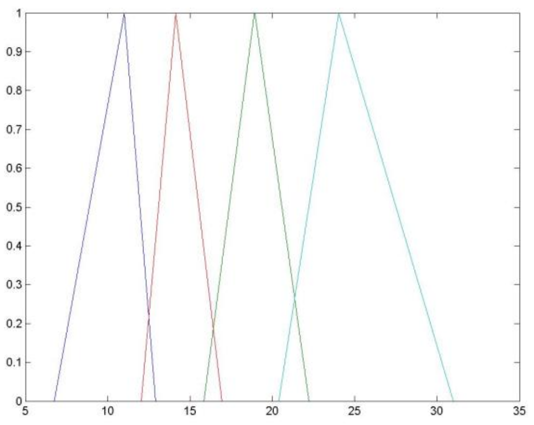
	K-Means++			Scatter Search		
	Time (sec.)	Nr. Clusters	I	Time (sec.)	Nr. Clusters	I
Radius	426,3904	4	18,9609	352,8277	3	26,9636
Texture	398,9221	3	16,8796	344,1798	5	30,18262
Perimeter	463,4574	7	830,33	371,4513	3	1286,17
Area	416,6392	6	426500,7	394,1495	3	668728,5
Smoothness	419,2019	3	0,000143	413,9573	4	0,0002044
Compactness	407,7729	3	0,004632	308,5648	3	0,004547
Concativity	418,1248	3	0,080241	305,7193	3	0,080673
Concave Points	405,9813	3	0,002273	470,995	3	0,002743
Symmetry	411,8899	3	0,000921	672,4182	3	0,000932
Fractal Dimension	413,8989	4	0,000167	679,9634	3	0,000255

Table 4.1: K-Means++ vs Scatter Search (best results)

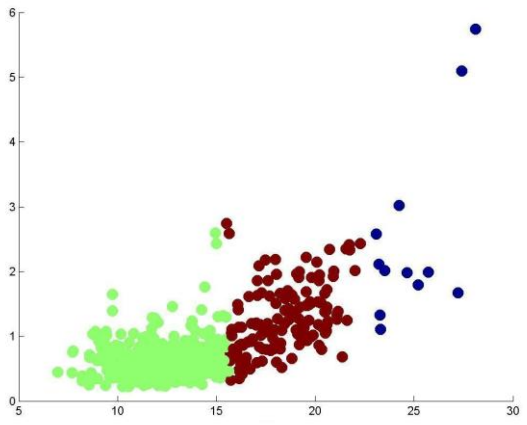
Radius



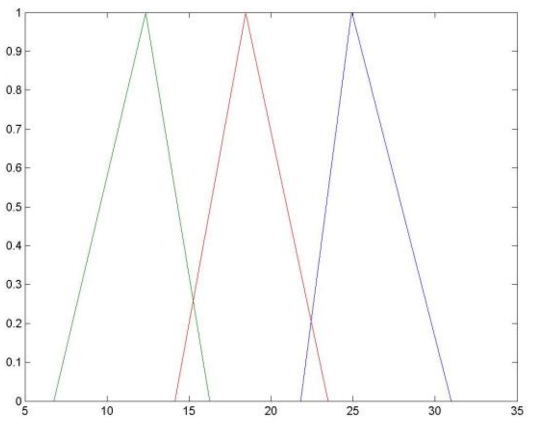
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



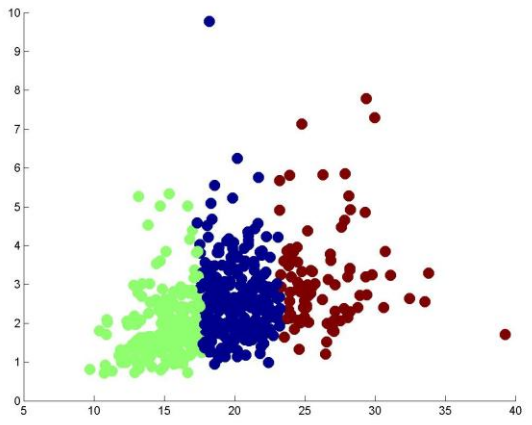
(c) Clusters with Scatter Search

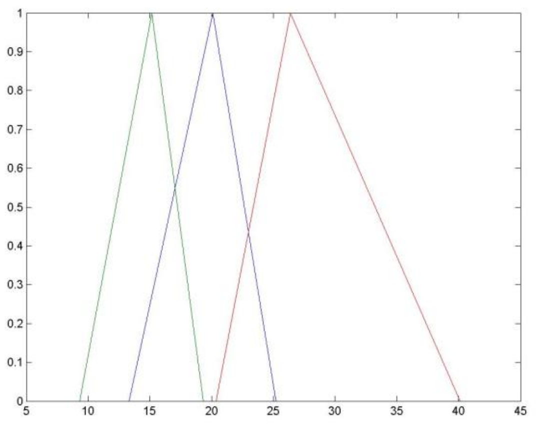


(d) Linguistic Variable after Scatter Search

Figure 4.24: Best Results for Linguistic Variable Radius

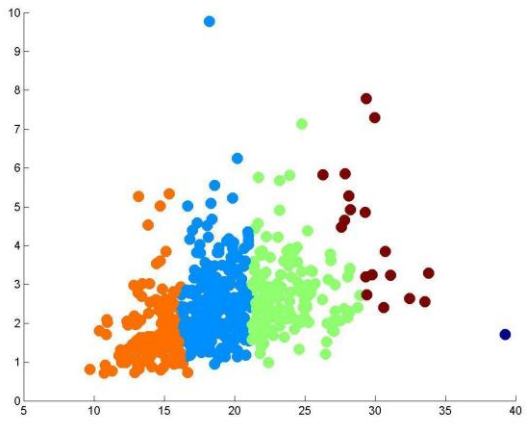
Texture

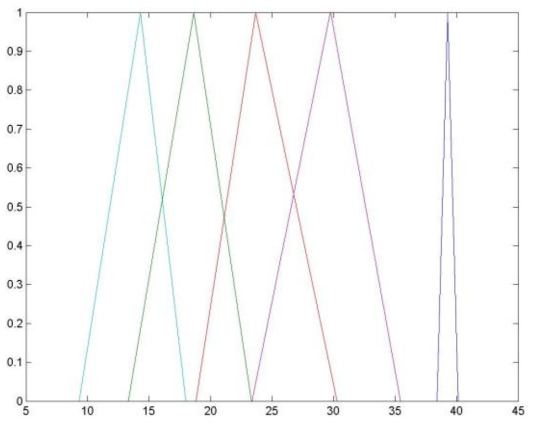




(a) Clusters with K-Means++

(b) Linguistic Variable after K-Means++



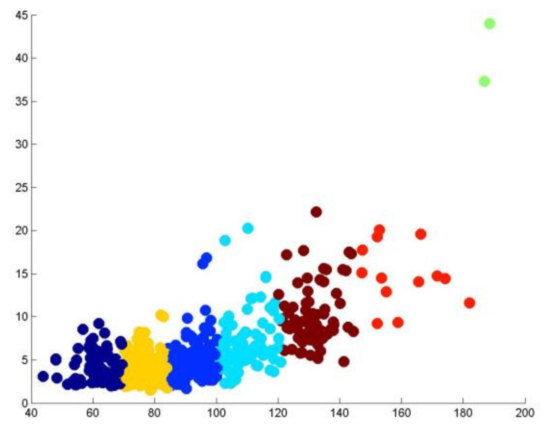


(c) Clusters with Scatter Search

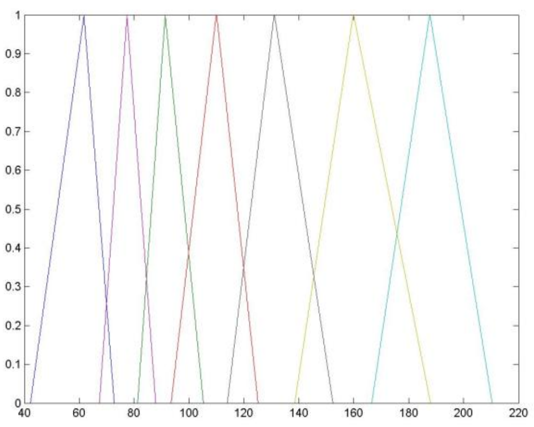
(d) Linguistic Variable after Scatter Search

Figure 4.25: Best Results for Linguistic Variable Texture

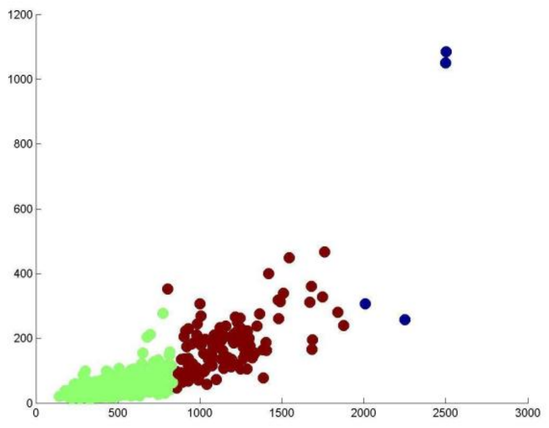
Perimeter



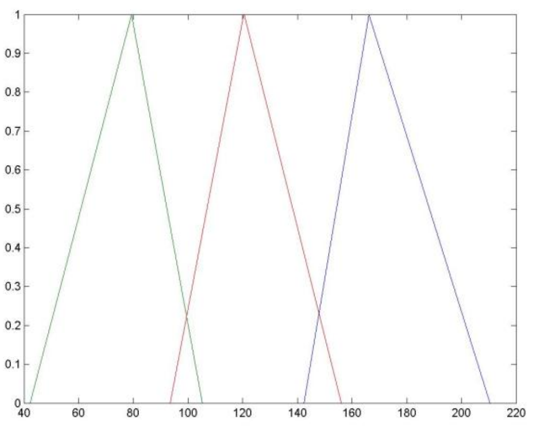
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



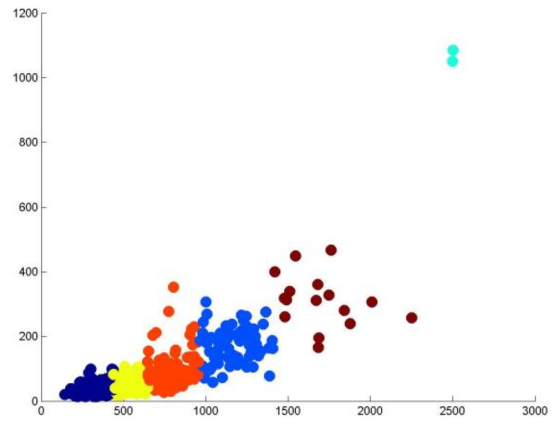
(c) Clusters with Scatter Search



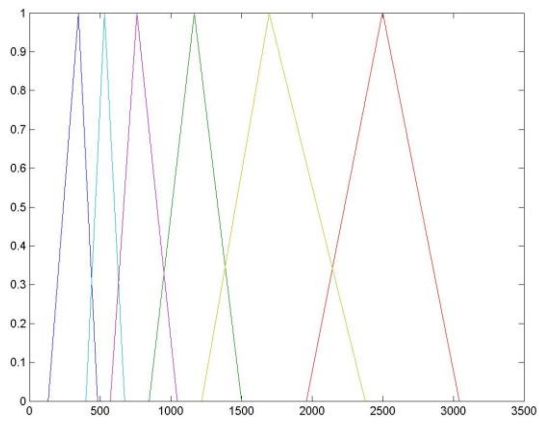
(d) Linguistic Variable after Scatter Search

Figure 4.26: Best Results for Linguistic Variable Perimeter

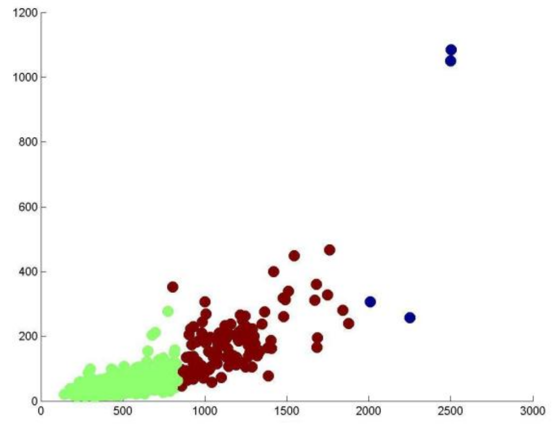
Area



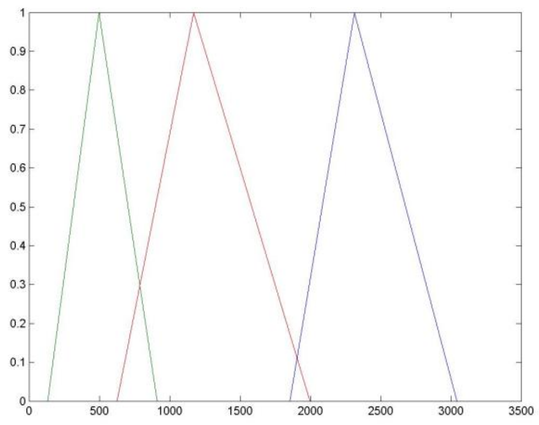
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



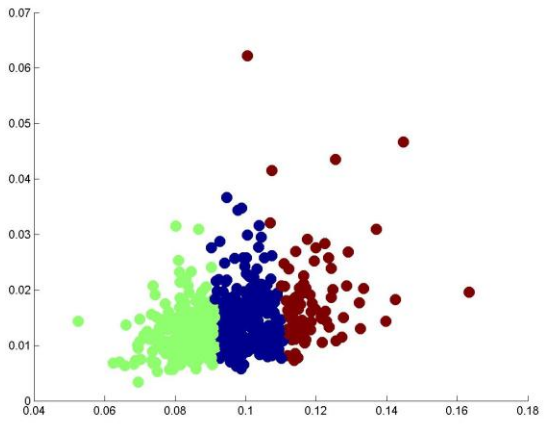
(c) Clusters with Scatter Search



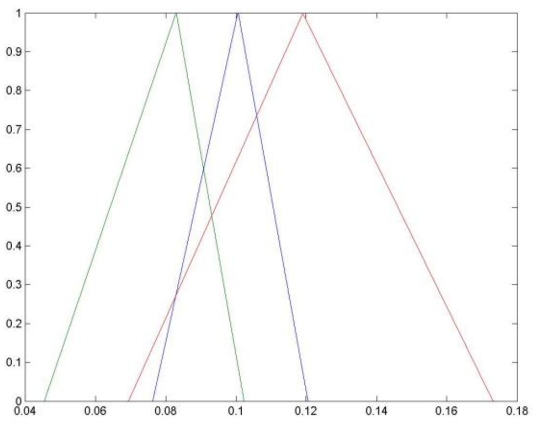
(d) Linguistic Variable after Scatter Search

Figure 4.27: Best Results for Linguistic Variable Area

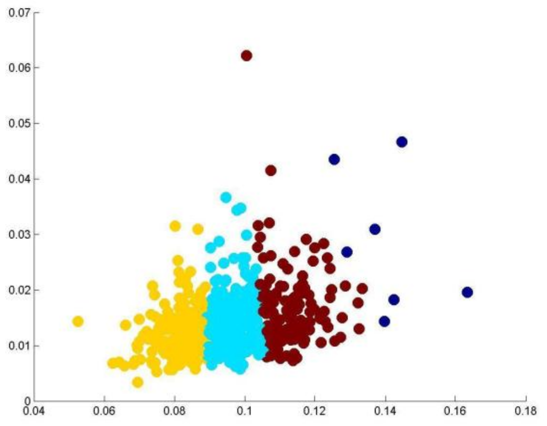
Smoothness



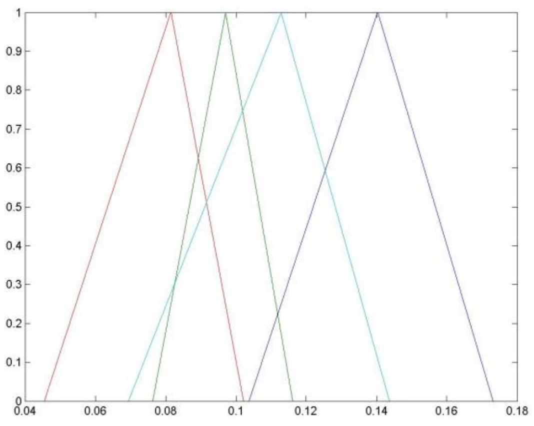
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



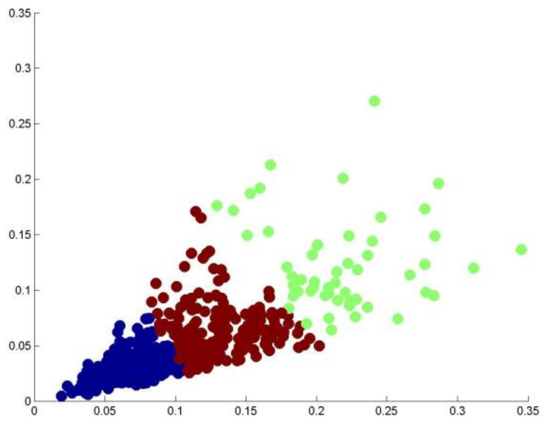
(c) Clusters with Scatter Search



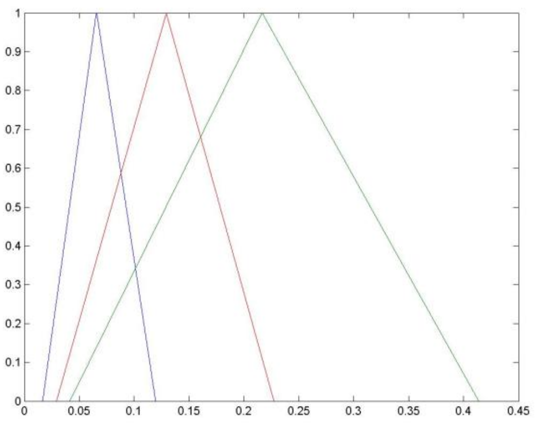
(d) Linguistic Variable after Scatter Search

Figure 4.28: Best Results for Linguistic Variable Smoothness

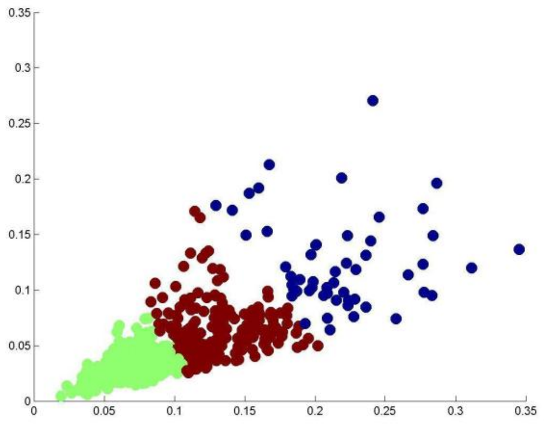
Compactness



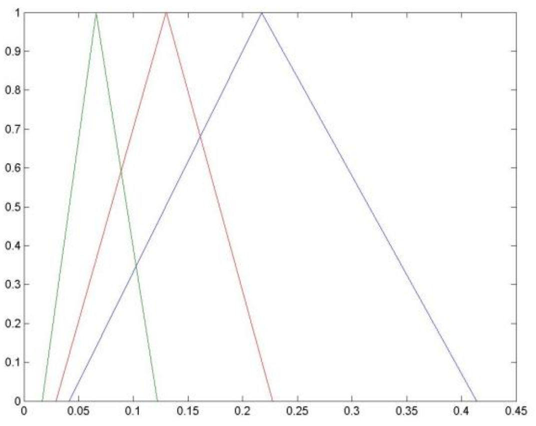
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



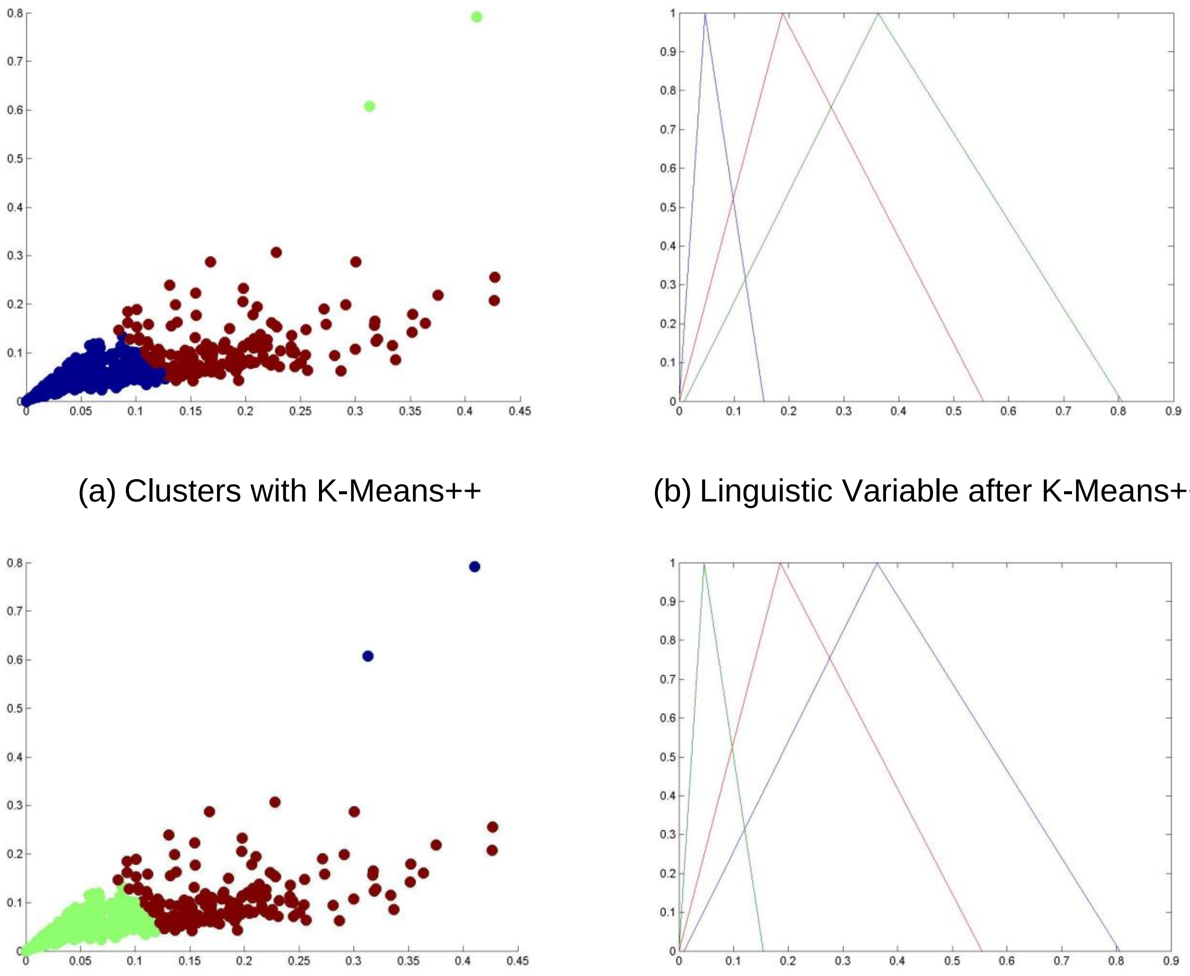
(c) Clusters with Scatter Search



(d) Linguistic Variable after Scatter Search

Figure 4.29: Best Results for Linguistic Variable Compactness

Concativity



(a) Clusters with K-Means++

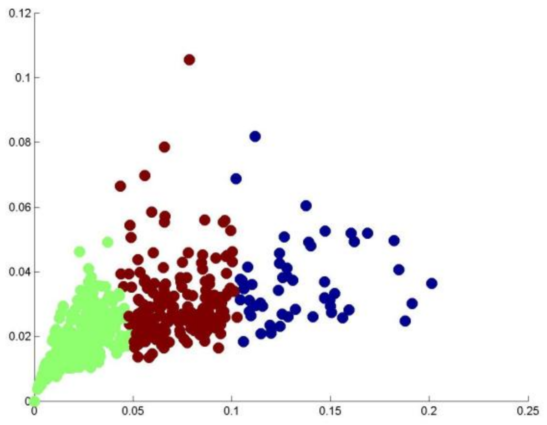
(b) Linguistic Variable after K-Means++

(c) Clusters with Scatter Search

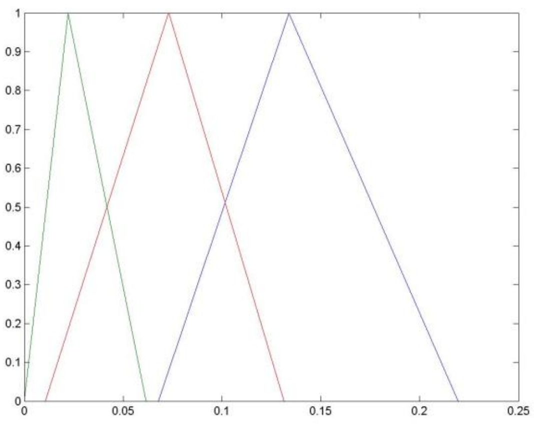
(d) Linguistic Variable after Scatter Search

Figure 4.30: Best Results for Linguistic Variable Concativity

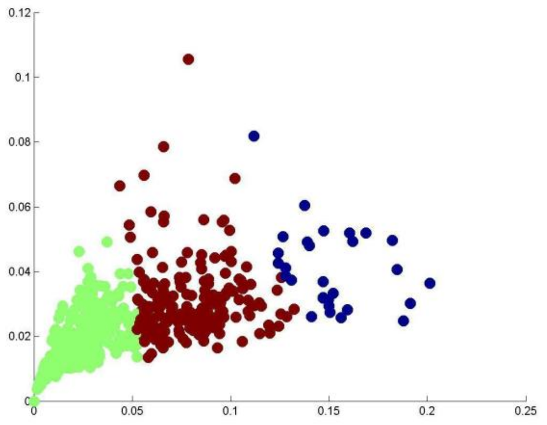
Concave Points



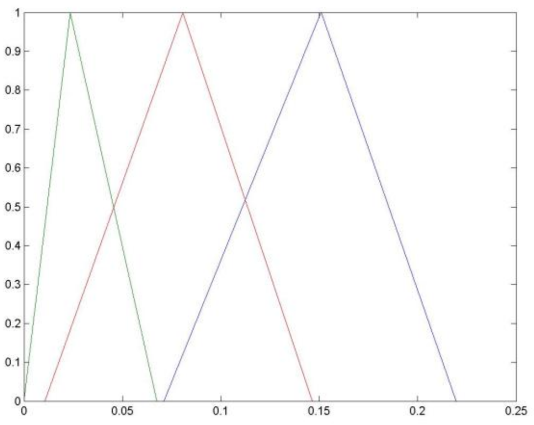
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



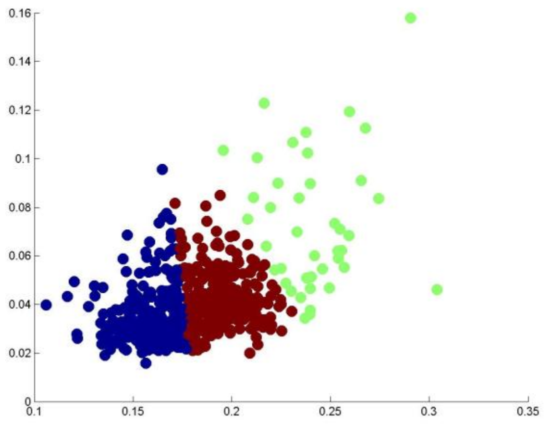
(c) Clusters with Scatter Search



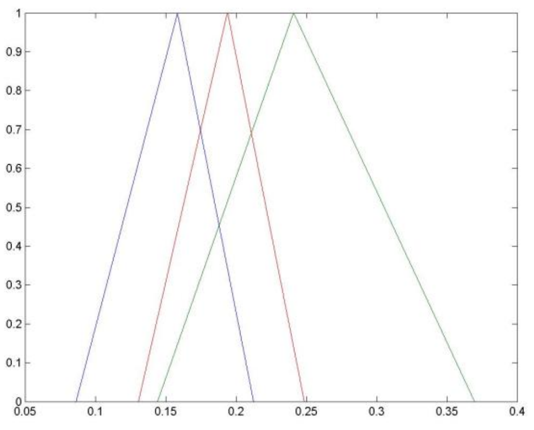
(d) Linguistic Variable after Scatter Search

Figure 4.31: Best Results for Linguistic Variable Concave Points

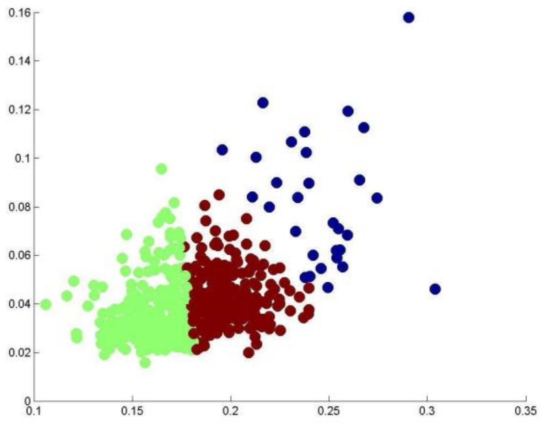
Symmetry



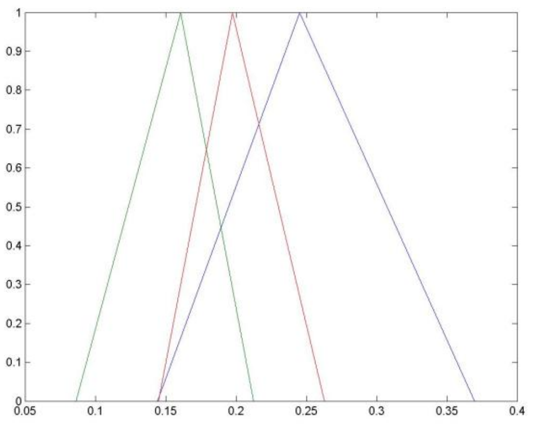
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



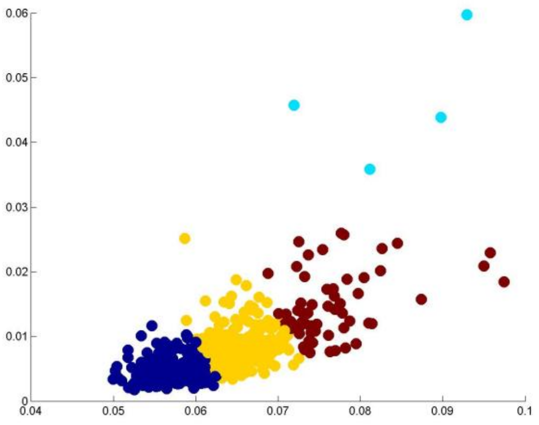
(c) Clusters with Scatter Search



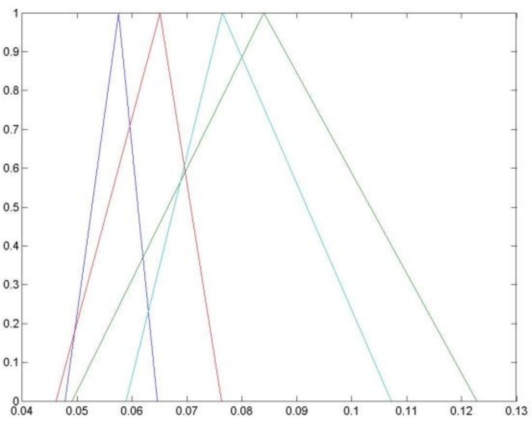
(d) Linguistic Variable after Scatter Search

Figure 4.32: Best Results for Linguistic Variable Symmetry

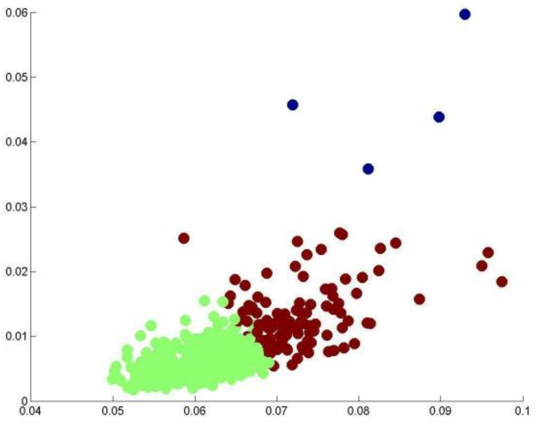
Fractal Dimension



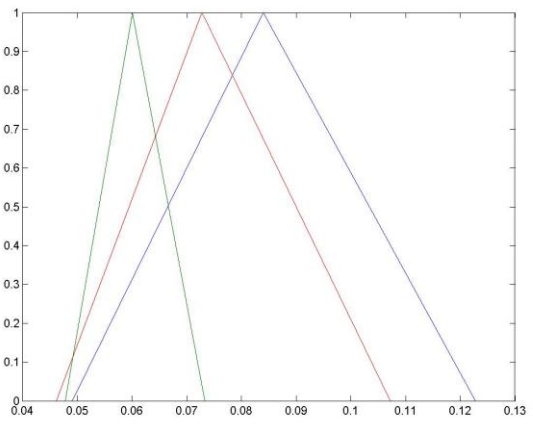
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



(c) Clusters with Scatter Search



(d) Linguistic Variable after Scatter Search

Figure 4.33: Best Results for Linguistic Variable Fractal Dimension

4.3.2 Credit Approval Data Set

The data used in this section is taken from [Asuncion 2007]. This data concerns credit approval information. Credit approval information is usually prone to uncertainty. On one hand, attributes like annual income are usually average information rather than absolute information. On the other hand, misinformation from the credit candidates, for instance undeclared income, provides additional uncertainty to the values presented. Therefore, it is natural to use fuzzy models when constructing automatic credit approval applications. Credit Approval (CA) data set contains 690 samples (665 after removing missing data) of data concerning credit approval information. Unfortunately, for confidentiality purposes, all attribute names and values have been changed to meaningless symbols. Each sample is composed of features A1 through A16. Ahead each variable its possible values are presented [Asuncion 2007]:

1. A1 – b, a
2. **A2 – continuous**
3. **A3 – continuous**
4. A4 - u, y, l, t
5. A5 – g, p, gg
6. A6 – c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff
7. A7 – v, h, bb, j, n, z, dd, ff, o
8. **A8 – continuous**
9. A9 - t, f
10. A10 – t, f
11. **A11 – continuous**
12. A12 – t, f
13. A13 – g, p, s
14. **A14 – continuous**
15. **A15 – continuous**
16. A16 - +,- (class attribute)

Only continuous attributes will be considered because other attributes are already categorized and the number of categories is already small. Once again, symmetrical triangular membership functions will be used. Since there is no information on the attributes and no additional information on accuracy of the data, triangular membership functions $(v_{ij}, 2\sigma_j)$ where used, where v_{ij} is the value of attribute j for sample i and σ_j is the standard deviation of attribute j. Of course all membership functions from the same linguistic variable, representing an attribute, will have the same width. In reality, the width of the membership functions would vary according to additional information collected from experts or from credit candidates. The six linguistic variables in this case study are represented in Figure 4.34 through Figure 4.39.

A2

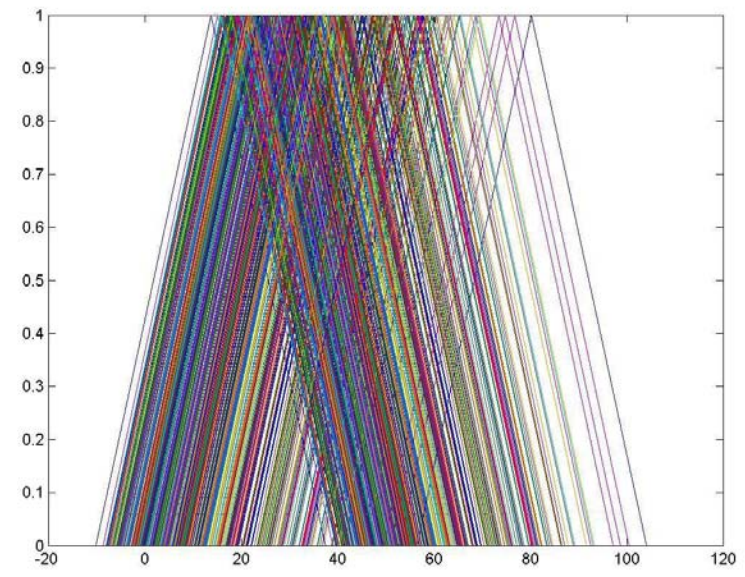


Figure 4.34: Linguistic Variable A2

A3

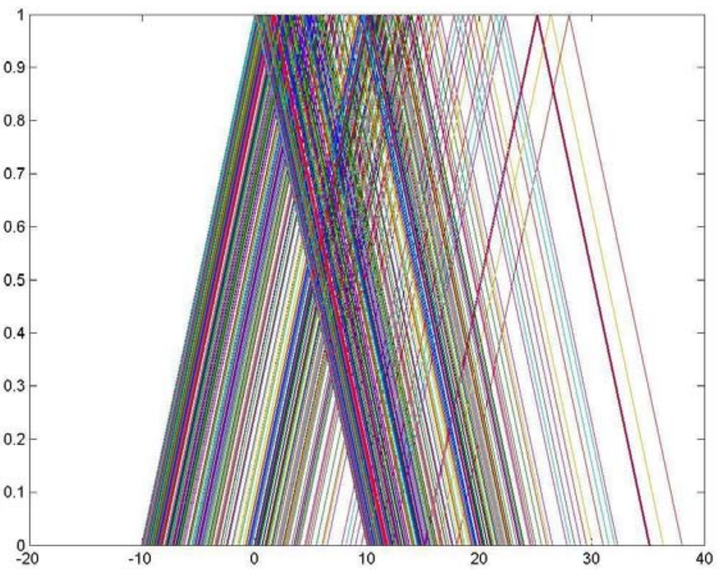


Figure 4.35: Linguistic Variable A3

A8

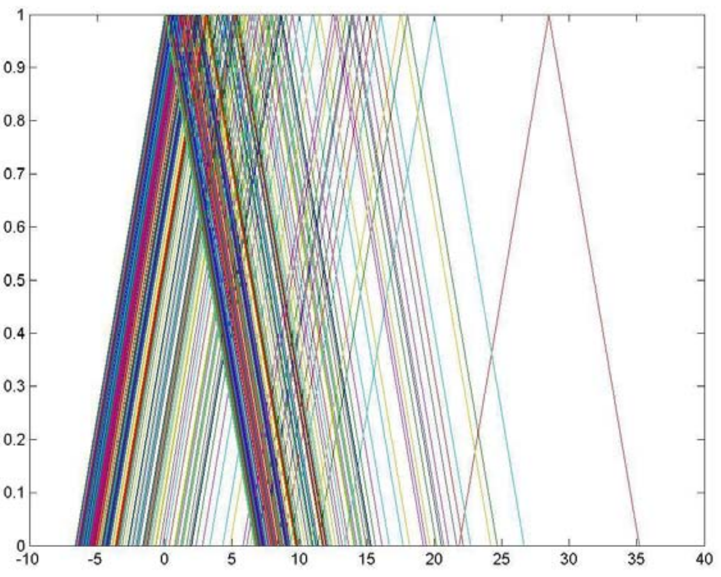


Figure 4.36: Linguistic Variable A8

A11

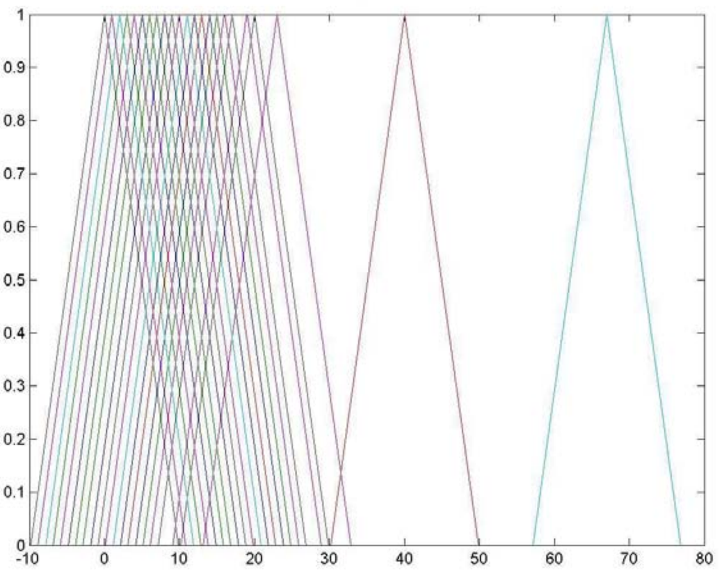


Figure 4.37: Linguistic Variable A11

A14

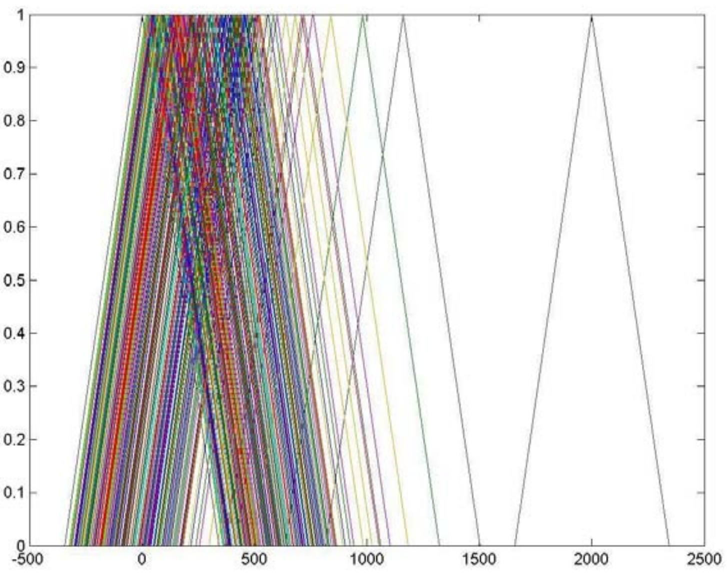


Figure 4.38: Linguistic Variable A14

A15

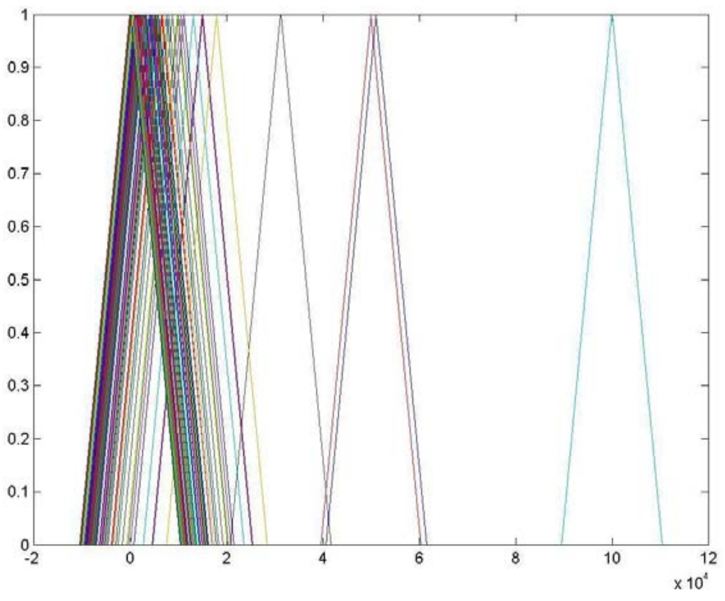


Figure 4.39: Linguistic Variable A15

4.3.2.1Computational Results

The same study conducted for the linguistic variables in the previous case study was conducted for the six linguistic variables in this case study. The K-Means++ algorithm, described in section 4.1, was applied to each of the linguistic variables.

Figure 4.40 and Figure 4.42 show the evolution of I using K-Means++ for $1 \leq K \leq 665$, for linguistic variables A2 and A3. For all other linguistic variables in this case study the overall behaviour is the same. This is not the same behaviour as in the previous case study. Now there is a very sudden improvement in the quality of solutions for a certain number of clusters. However, this sudden improvement is achieved only for high number of clusters, between 200 and 350 clusters for the linguistic variables in this case study. This is a high number of membership functions in a linguistic variable. It is desirable to achieve a greater reduction in the number of membership functions to improve its interpretability. Therefore, as in the previous case study, it was chosen $K_{\max} = 100$. It is left for future work to investigate on procedures to estimate this parameter. Figure 4.41 and Figure 4.43 show a zoom of the previous plots, for $1 \le K \le 150$.

A2 - Evaluation Measure

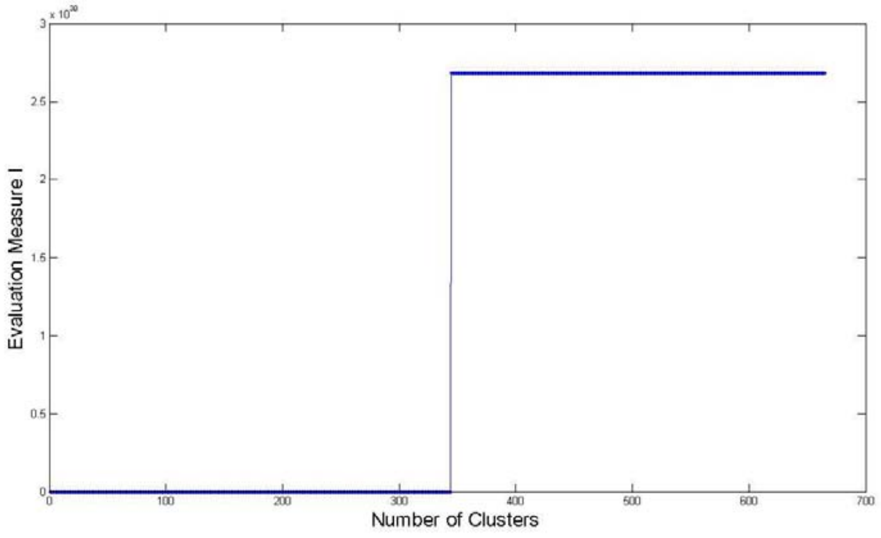


Figure 4.40: Evaluation measure I using K-means++ for $1 \le K \le 665$, linguistic variable A2

A2 - Evaluation Measure – zoom

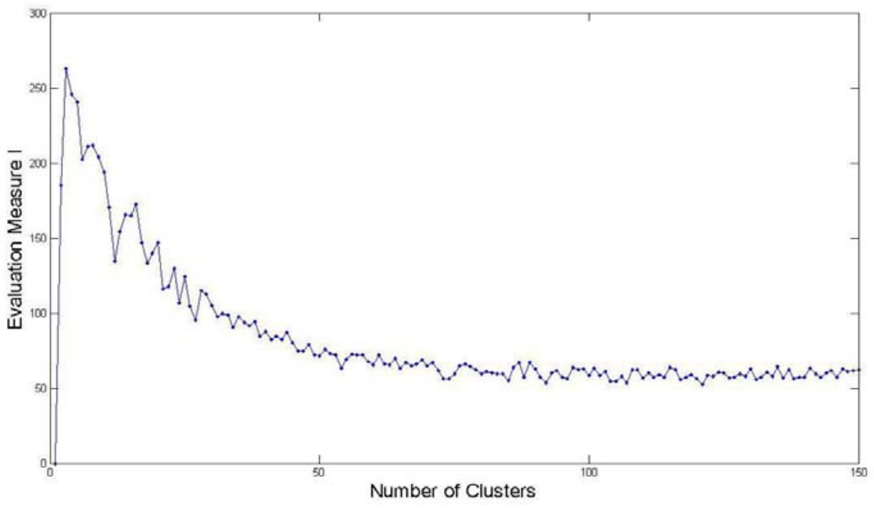


Figure 4.41: Evaluation measure I using K-means++ for $1 \le K \le 150$ (zoom in of previous plot), linguistic variable A2

A3 - Evaluation Measure

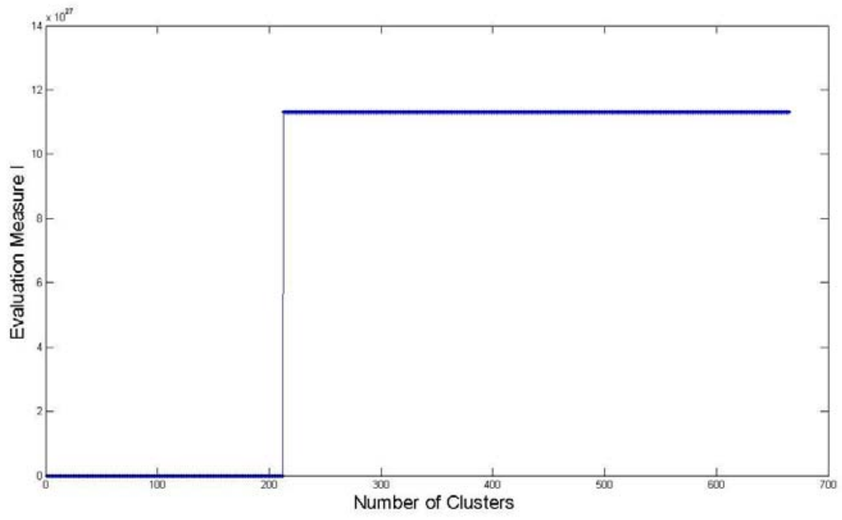


Figure 4.42: Evaluation measure I using K-means++ for $1 \leq K \leq 665$, linguistic variable A3

A3 - Evaluation Measure – zoom

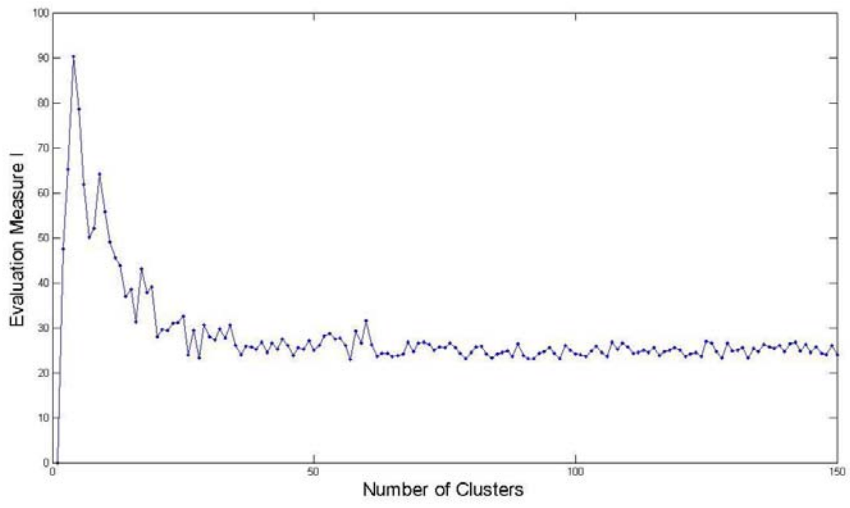


Figure 4.43: Evaluation measure I using K-means++ for $1 \leq K \leq 150$ (zoom in of previous plot), linguistic variable A3

The Scatter Search algorithm was run with the same values for the parameters α, β, *MaxIterImp* and b_1 as in the previous case study, considering 5 experiments for each choice of parameters and with the same stopping criterion as before. Once again, the number of good quality solutions and of diverse solutions to be included in the reference set was chosen to be equal ($b_1 = b_2$) and the original set was chosen to be 10 times the size of the reference set ($OS_{size} = 10*(b_1 + b_2)$), as in [Martí, Laguna et al. 1997; Abdule-Wahab, Monmarché et al. 2006].

Only the results relative to the linguistic variable A2 will be discussed in more detail.

To study the influence of α in the results, experiments were conducted with no improvement method (i.e. *MaxIterImp* $= 0$) and all possible combinations of $\alpha \in \{0; 0.5; 0.8; 1\}$, $\beta \in \{0; 0.5\}$ and $b_1 \in \{2; 5\}$. When $\alpha = 0$ the standard deviation of results is much higher than for other values of α (Figure 4.44), explaining why $\alpha = 0$ was responsible for the best results (Figure 4.45) but does not give competitive results in average (Figure 4.46). Due to the referred high standard deviation in the results, no further experiments were done with $\alpha = 0$.

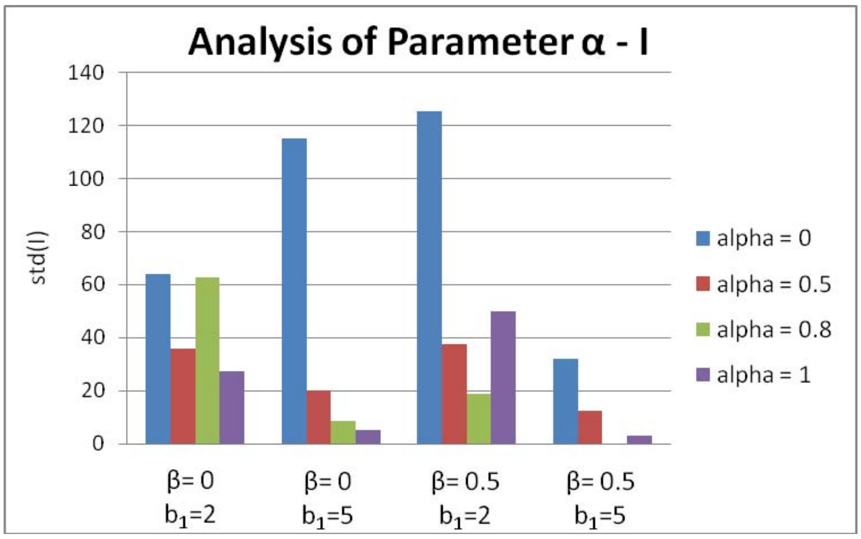


Figure 4.44: Influence of α in fitness function I standard deviation

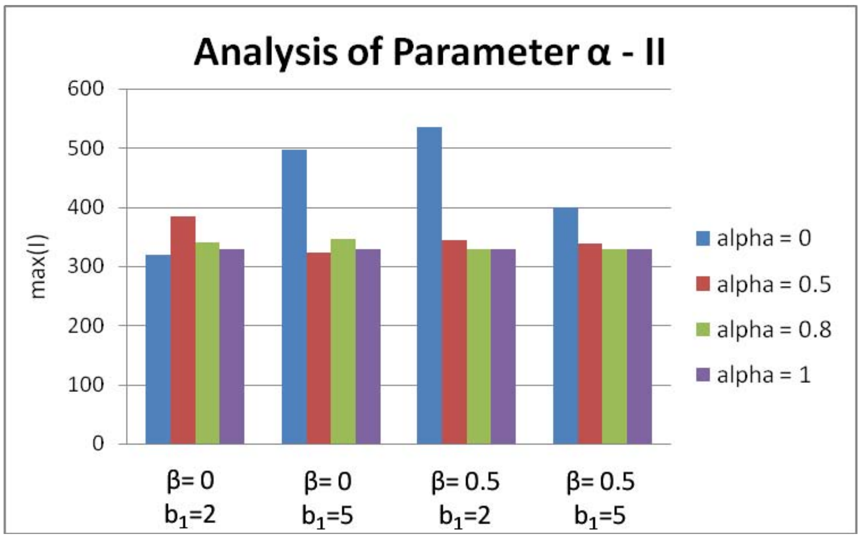


Figure 4.45: Influence of α in best results obtained for fitness function I

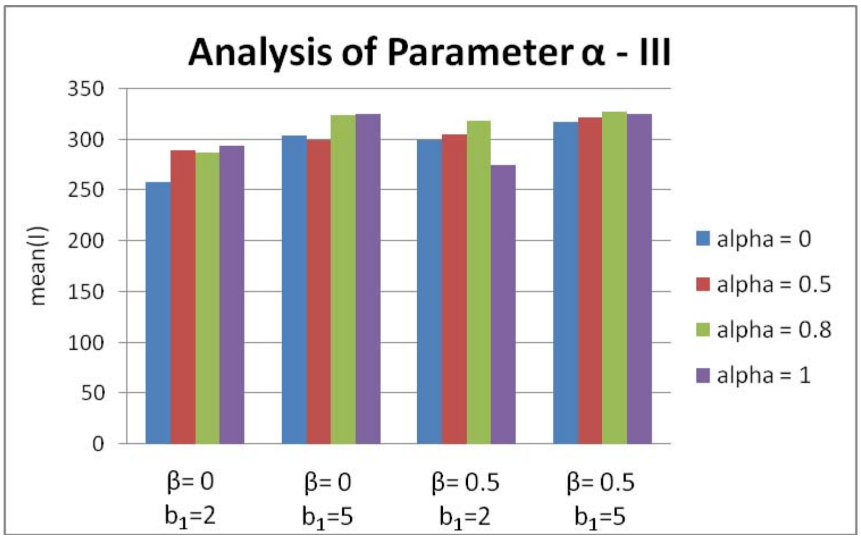


Figure 4.46: Influence of α in mean results for fitness function I

In Figure 4.47 the use of memory in the Diversification Generation method is clear since using $\beta = 0$ gives worse results in average. Therefore, no further experiments for $\beta = 0$ will be discussed.

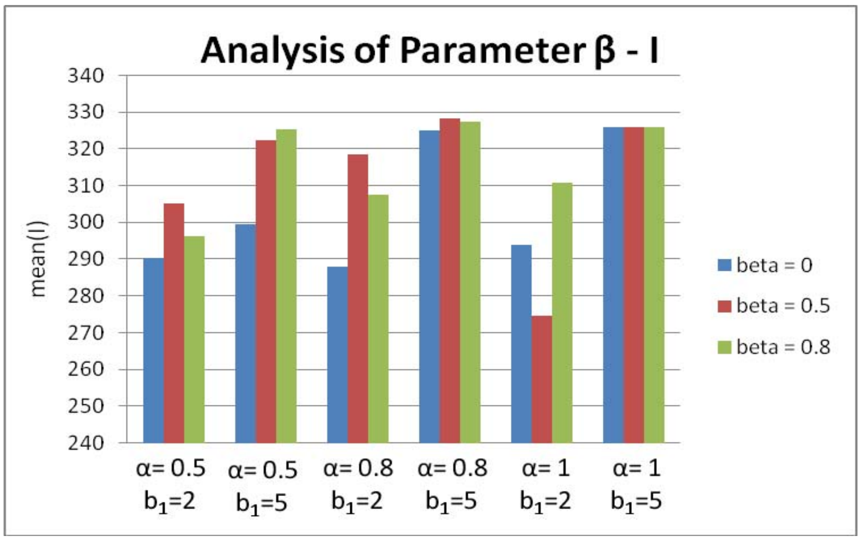


Figure 4.47: Influence of β in mean results for fitness function I

Increasing the size of the reference set produces better quality results, as can be seen in Figure 4.48. Increasing the number of solutions in the reference set from 4 to 10 ($b_1 = 2$ to $b_1 = 5$.) resulted in an increase in the execution time (Figure 4.49) by a factor between 1.6 and 2.9. Still, all experiments were run in less than 10 minutes. It is still affordable to consider a larger reference set to improve the overall quality of solutions.

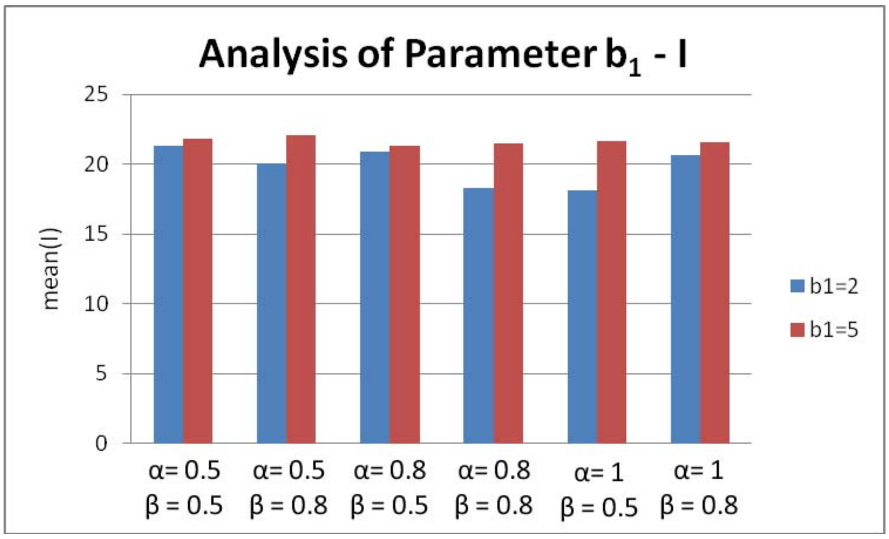


Figure 4.48: Influence of b_1 in mean results for fitness function I

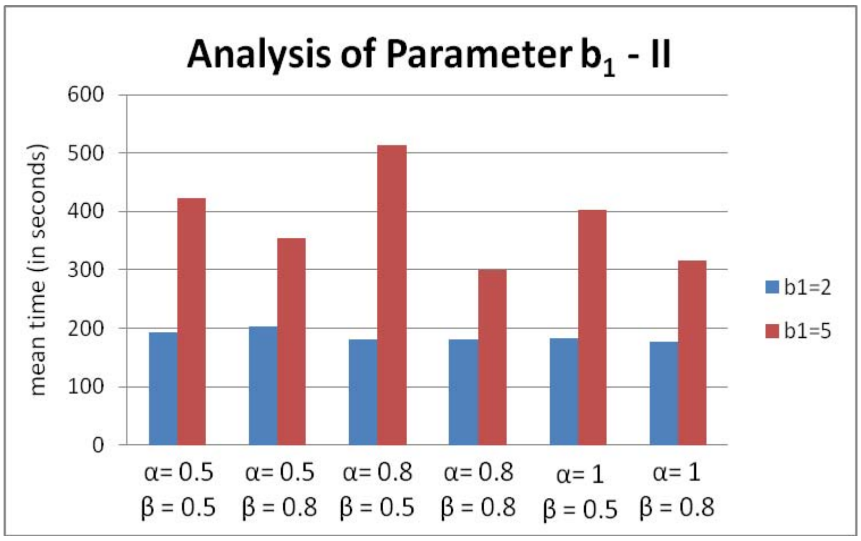


Figure 4.49: Influence of b_1 in execution time

Just as before, the increase in execution time when using *MaxIterImp*$=5$ is considerably high (Figure 4.51) and does not result in a significant improvement in the quality of solutions in most of the cases (Figure 4.50). Since the increase in execution time for using *MaxIterImp*$=2$ is not too high (from approximately 6 minute to approximately 10 minutes), this value will be considered for this parameter. Just as in the previous case study, the choice of parameters $\alpha = 0.5$, $\beta = 0.8$, $b_1 = 5$ and *MaxIterImp*$=2$ gives better results, in average, and were used to obtain the results summarized in Table 4.2 and Figure 4.52 through Figure 4.57.

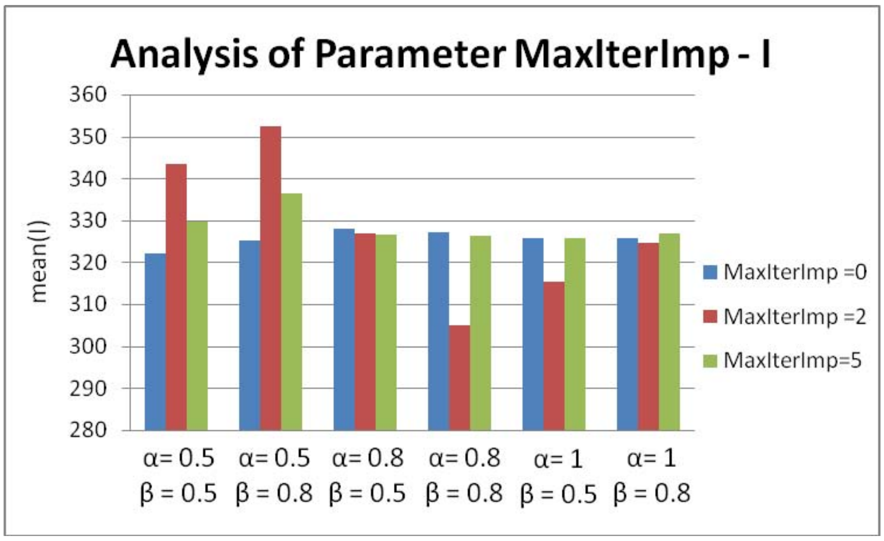


Figure 4.50: Influence of Improvement Method in mean results for fitness function I

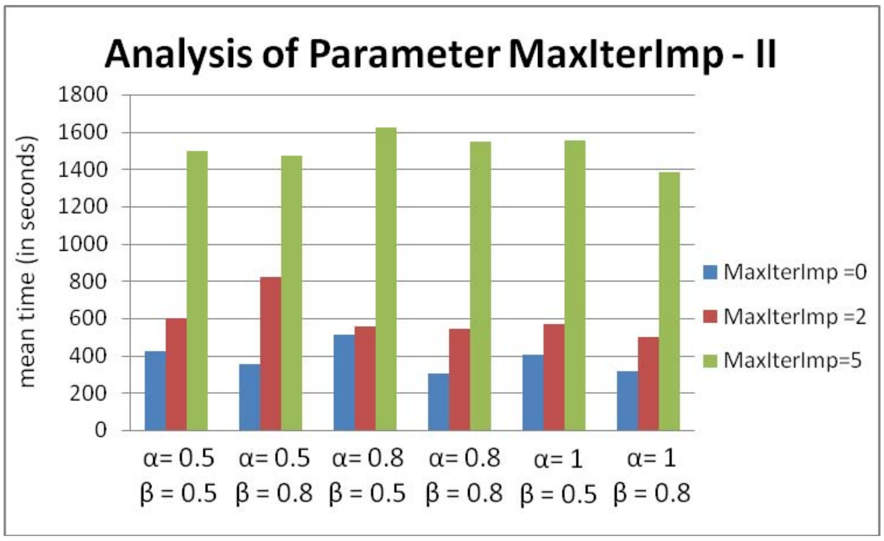


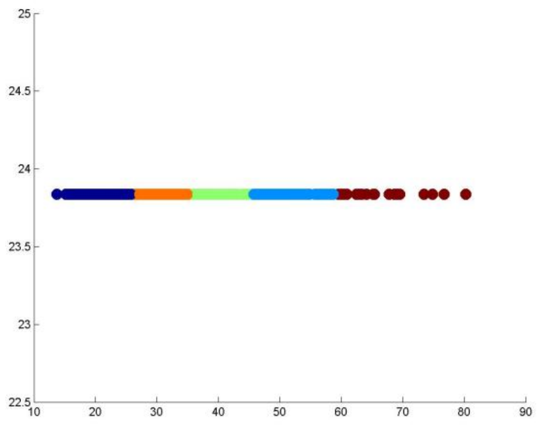
Figure 4.51: Influence of Improvement Method in execution time

Once again, the best solution found by the algorithm was always generated by the diversification generation method. The results reported in Table 4.2 were achieved only by the first stage of the algorithm, which still was sufficient to produce better results than the K-Means++ algorithm in most variables. Notice that both algorithms returned the same solution (except for cluster numbering) with 23 clusters for linguistic variable A11. This can be explained with information about this linguistic variable. Although A11 is continuous, there are only 23 different values for this variable. The solution returned by the algorithm corresponds to the 23 different membership functions corresponding to the 23 crisp values. This result show that the two algorithms are not giving “fake” low numbers of clusters (until now the number of clusters varied from 3 to 6) but are indeed capable of estimating a good number of clusters.

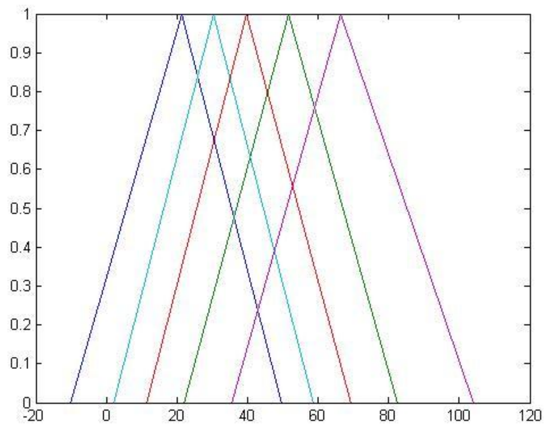
	K-Means++			Scatter Search		
	Time (sec.)	Nr. Clusters	I	Time (sec.)	Nr. Clusters	I
A2	854.5374	5	266.0796	821.3367	3	384.8384
A3	862.5715	4	90.18299	794.977	4	96.4486
A8	810.7072	6	105.4446	1277.734	6	94.76789
A11	735.1273	23	6.97E+28	1112.77	23	6.97E+28
A14	1134.509	6	346890.2	1812.912	4	402917.4
A15	1062.561	6	5.51E+09	2031.943	6	5.57E+09

Table 4.2: K-Means++ vs Scatter Search (best results)

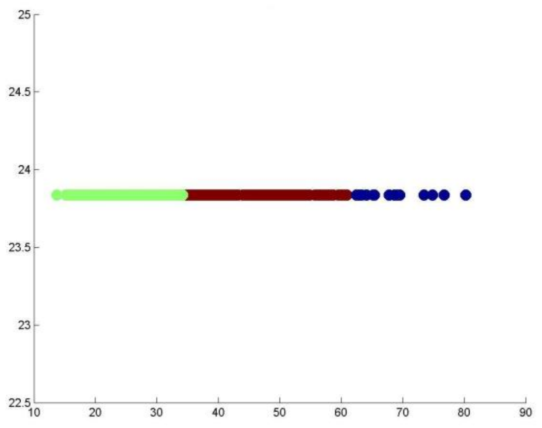
A2



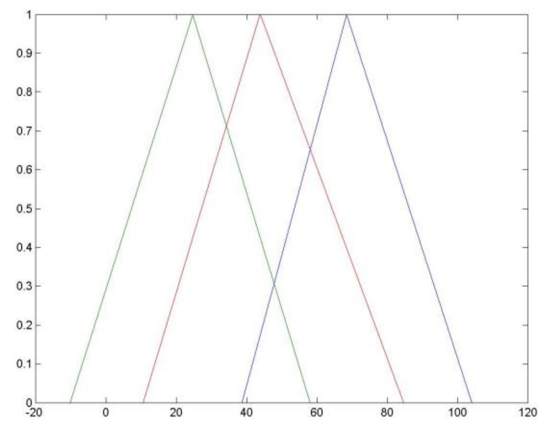
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



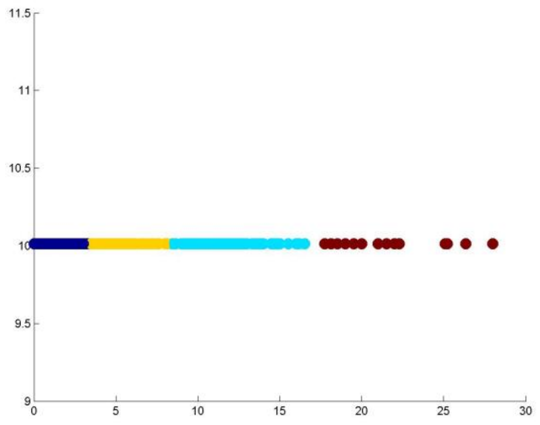
(c) Clusters with Scatter Search



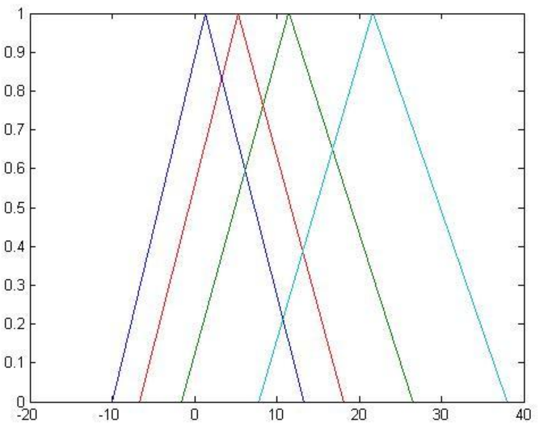
(d) Linguistic Variable after Scatter Search

Figure 4.52: Best Results for Linguistic Variable A2

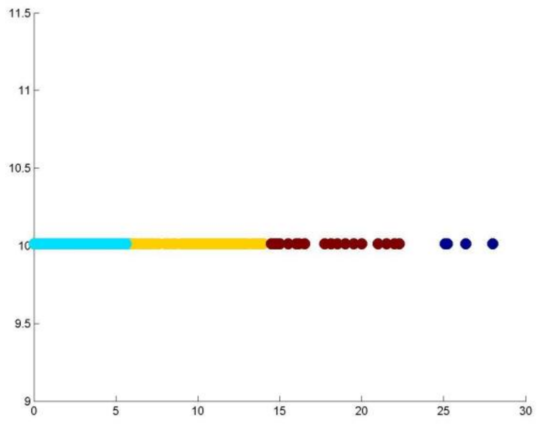
A3



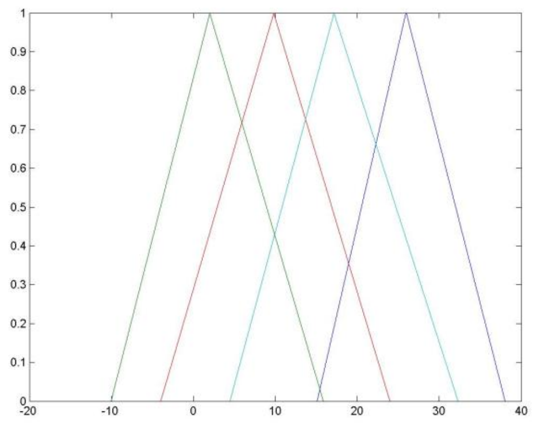
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



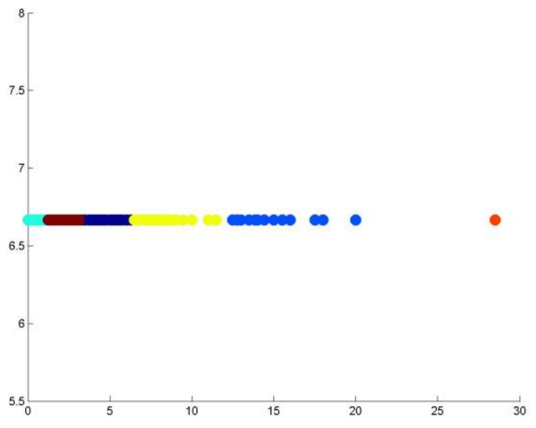
(c) Clusters with Scatter Search



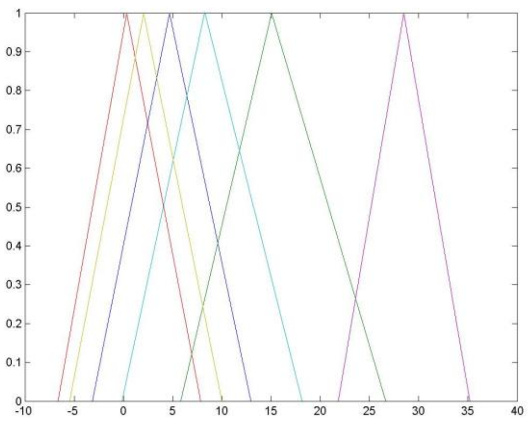
(d) Linguistic Variable after Scatter Search

Figure 4.53: Best Results for Linguistic Variable A3

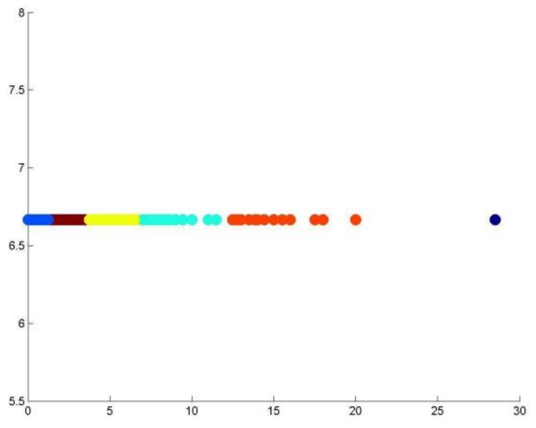
A8



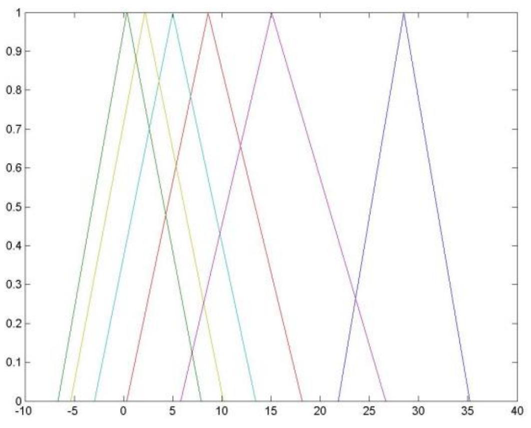
(a) Cluters with K-Means++



(b) Linguistic Variable after K-Means++



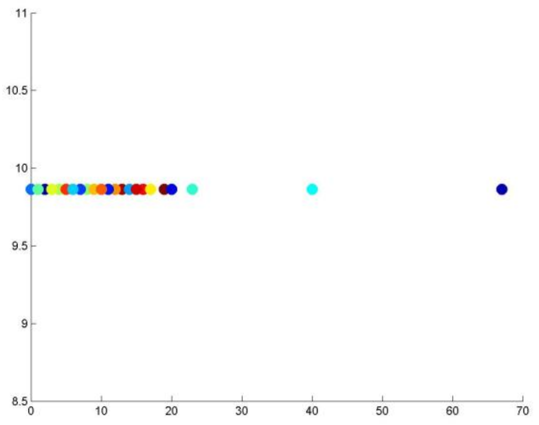
(c) Clusters with Scatter Search



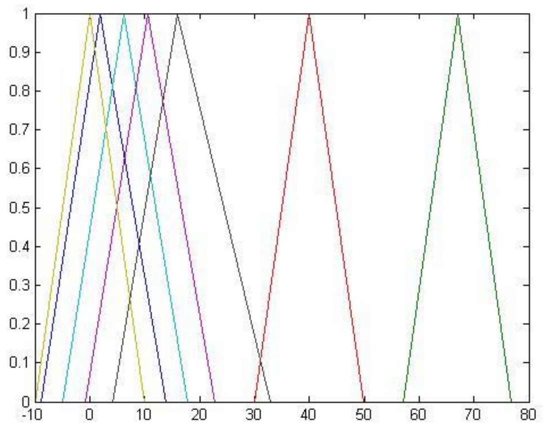
(d) Linguistic Variable after Scatter Search

Figure 4.54: Best Results for Linguistic Variable A8

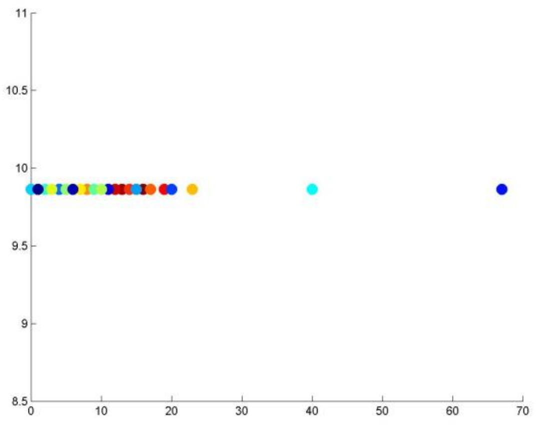
A11



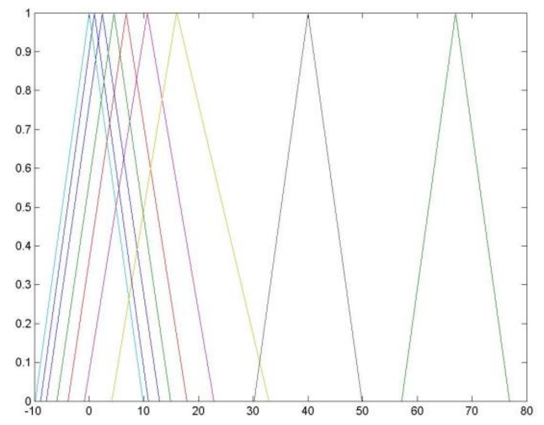
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



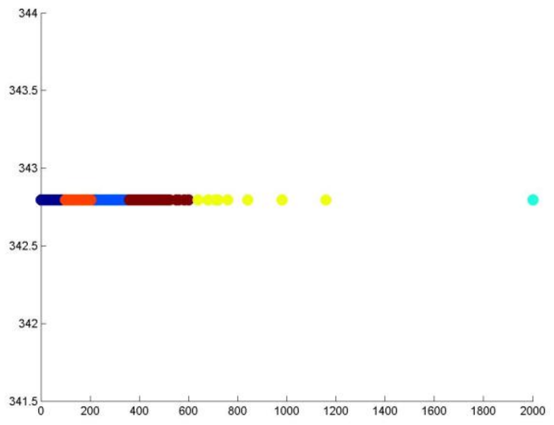
(c) Clusters with Scatter Search

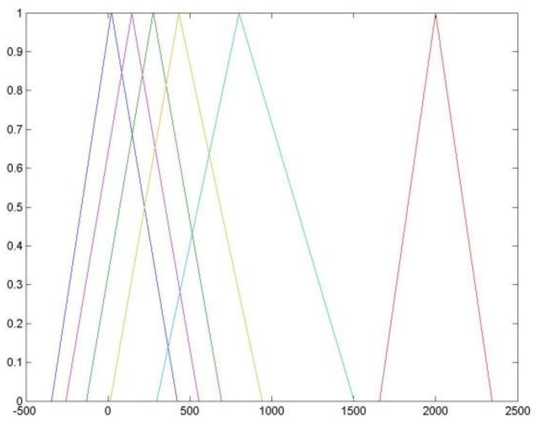


(d) Linguistic Variable after Scatter Search

Figure 4.55: Best Results for Linguistic Variable A11

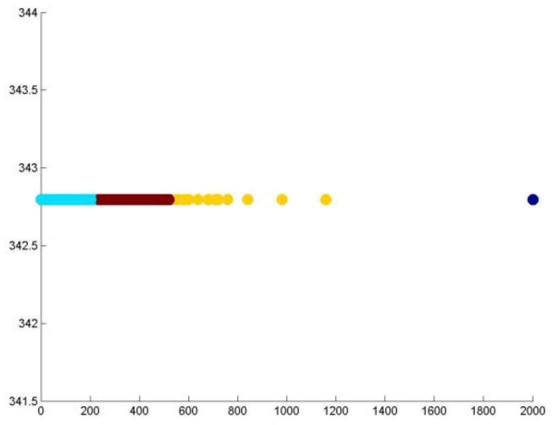
A14

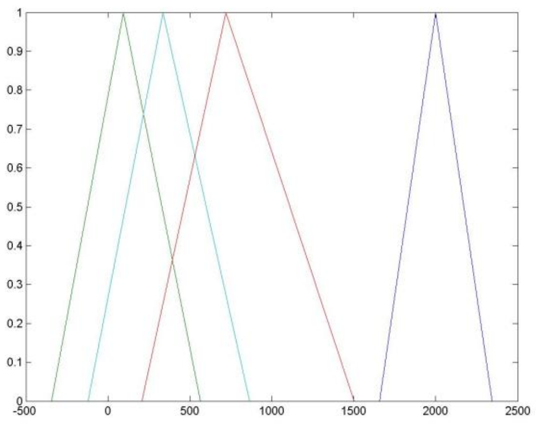




(a) Clusters with K-Means++

(b) Linguistic Variable after K-Means++



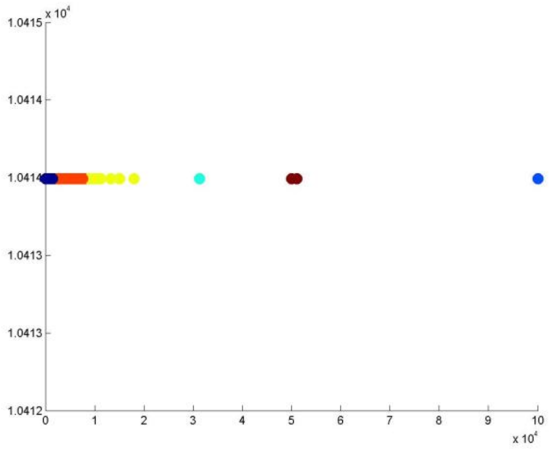


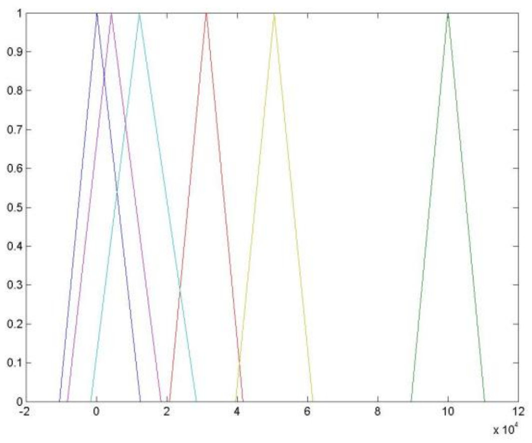
(c) Clusters with Scatter Search

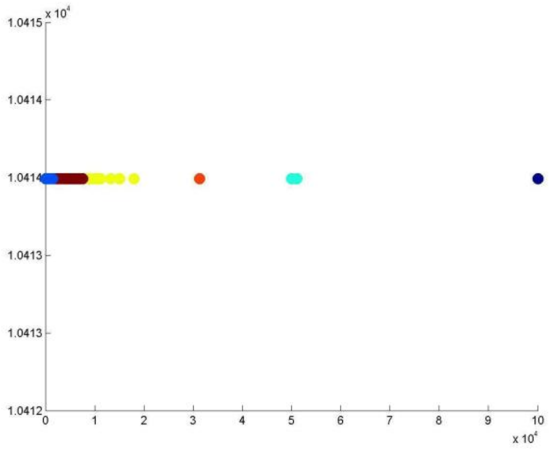
(d) Linguistic Variable after Scatter Search

Figure 4.56: Best Results for Linguistic Variable A14

A15


(a) Clusters with K-Means++


(b) Linguistic Variable after K-Means++


(c) Clusters with Scatter Search

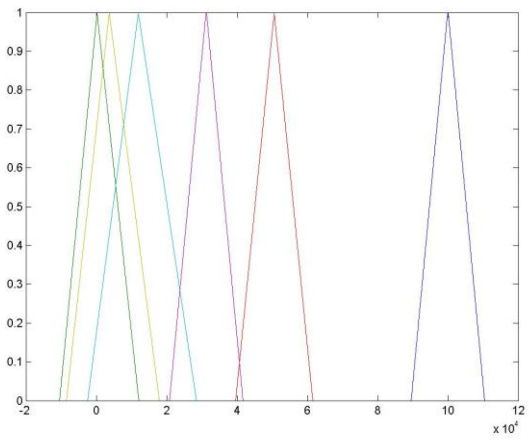

(d) Linguistic Variable after Scatter Search

Figure 4.57: Best Results for Linguistic Variable A15

4.4 Summary

In this Chapter an heuristic and a metaheuristic for clustering were described. The first is the K-Means++ algorithm [David and Sergei 2007], a variation of the K-Means algorithm [Gan, Ma et al. 2007]. The second is a Scatter Search algorithm based on the work of [Pacheco 2005] and [Abdule-Wahab, Monmarché et al. 2006]. These two algorithms where implemented in Matlab.

The computational results were not as expected. The second phase of the Scatter Search algorithm was not able to produce good quality solutions. However, the first part of the algorithm was sufficient to obtain better results than the ones given by the K-Means++ algorithm. Both methods achieved a high reduction in the number of membership functions in each linguistic variable.

Chapter 5. Case Study: a Fuzzy Inference System

In the previous chapter, computational results for two case studies were presented. In both cases the problem consisted in reducing the number of membership functions in linguistic variables that could later be used in the construction of an inference system. Each membership function represented a sample or an individual and merging similar membership functions could be seen as finding groups of individuals with similar characteristics. The rule base system that could be constructed afterwards would take into account the final morphology of the linguistic variables. The case study presented in this section is of a different nature. In this case, the goal is to reduce the number of membership functions in linguistic variables of an already existing inference system [Gomes, Santos et al. 2008]. The aim is to reduce the complexity of the inference system while maintaining its structure, without losing too much performance.

In 2001, the European Space Agency [ESA] launched the Aurora Programme whose main goal is the robotic and human exploration of the solar system [ESA 2008]. ExoMars, one of the missions under this programme, will require the drilling and sampling of Martian rocks [ESA 2008]. The case study here presented has been developed at CA3 - UNINOVA [CA3 2006] in the scope of this programme.

In section 5.1 the case study will be described. Section 5.2 discusses the heuristics used for reducing the number of membership functions and in section 5.3 the computational results for the heuristics applied to this case study will be presented and discussed.

5.1 Overview of the case study: MODI

During project “MODI- Simulation of a Knowledge Enabled Monitoring and Diagnosis Tool for ExoMars Pasteur Payloads” [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006; Santos, Martins et al. 2008] two fuzzy inference systems were developed: one for an alarm system for detecting faulty behaviours during drilling in Mars and other for recognition of terrain hardness types. These inference systems were created automatically from signals first generated by a simulator and later in the project acquired from a drilling station developed during the project. Pictures of this drilling station prototype constructed as proof of concept during this project and a simulated image of the rover that might use this technology can be found in Figure 5.1 and Figure 5.2, respectively.

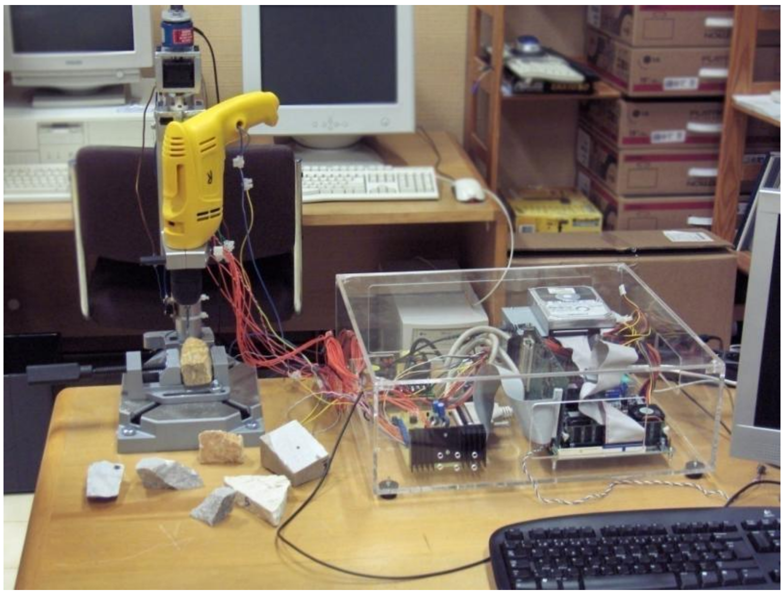


Figure 5.1: MODI drill station

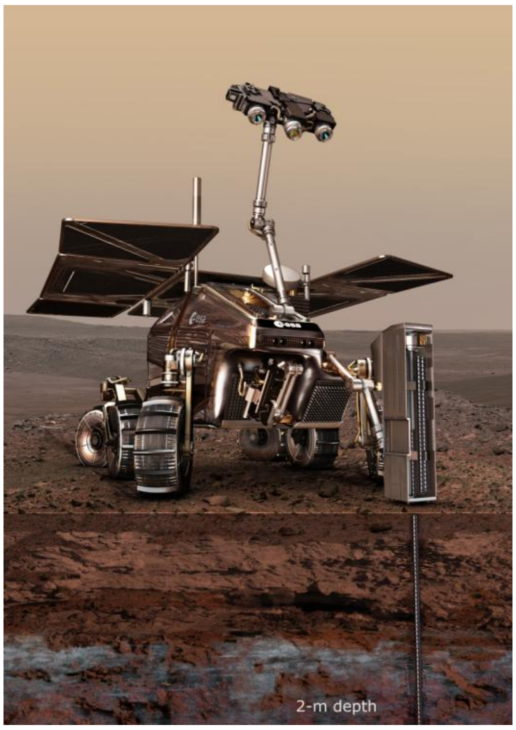


Figure 5.2: ExoMars Rover (courtesy of ESA [ESA 2008])

The inference systems developed included two types of input variables: set points and sensor variables. The set points are variables whose values are pre-defined by the user to study the behaviour of the drill while drilling in different types of materials (rocks). Given the values for these two set points, the drilling process would start and sensors installed in the drill would measure the rest of the variables in our model.

As the project evolved, the sensors available increased. These sensors were able to measure rotational and translational currents and voltages, thrust, among other measures.

All linguistic variables that were created in the MODI project are trapezoidal membership functions, except the ones that describe the set points that are either triangular membership functions or singletons [Santos, Fonseca et al. 2008]. In the MODI project the linguistic variables (except the set points that are pre-defined by the user) were constructed automatically, using sensor data collected during a learning phase [Santos, Fonseca et al. 2008]. During the learning phase, drills in different types of terrain hardness, using different values for the set points, were performed. Each combination of values for the set points and terrain type defined a sub-scenario in our model. Each linguistic variable represents a different sensor and each term in a linguistic variable refers to a different sub-scenario. Trapezoidal membership functions for each sub-scenario and sensor were constructed taking into account the mean and standard deviation of the corresponding signal.

If we consider a drill with d sensors and a set of tests consisting in drilling in t different types of terrain with all possible combinations of sp_1 values for set point 1 and sp_2 values for set point 2, the model will have $d \times t \times sp_1 \times sp_2$ membership functions, excluding set points and output variables. As the number of sub-scenarios or number of sensors increases, so does the complexity of the inference system.

The output of the terrain recognition inference system is the terrain hardness for the t scenarios defined (T) and the certainty level of that classification [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006]. An example of a rule used for scenario Concrete hardness type, sub-scenario 0 (C0) with 2 set points – Set Point Rotation Speed (SPRS) and Set Point Translational Speed (SPTS) – and 8 sensor variables – Rotation Current (RC), Rotation Voltage (RV), Rotation Speed

(RS), Thrust (TH), Torque (TO), Translational Voltage (TV), Translational Current (TC) and Translational Speed (TS) - is shown bellow. There is one such rule in the system for every sub-scenario considered. In the following rules *Variable_Name – Sub-scenario_code* is the fuzzy set representing the nominal situation in variable *Variable_Name* and sub-scenario *Sub-scenario_code*.

If

Set Point Rotation Speed is SPRS-C0 and Set Point Translational Speed is SPTS-C0 and Rotation Current is RC-C0 and Rotation Voltage is RV-C0 and Rotation Speed is RS-C0 and Thrust is TH-C0 and Torque is TO-C0 and Translational Voltage is TV-C0 and Translational Current is TC-C0 and Translational Speed is TS-C0

Then

Terrain is T-C0

The output of the alarm inference system is the alarm level, on a scale from 0 to 1 [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006]. An example of a rule corresponding to the variables and sub-scenario above is shown bellow.

If

Set Point Rotation Speed is SPRS-C0 and Set Point Translational Speed is SPTS-C0 and (Rotation Current is not RC-C0 and Rotation Voltage is not RV-C0 and Rotation Speed is not RS-C0 and Thrust is not TH-C0 and Torque is not TO-C0 and Translational Voltage is not TV-C0 and Translational Current is not TC-C0 and Translational Speed is not TS-C0

Then

Alarm Level

In this thesis only the terrain recognition system will be used as a test case. The same analysis could be done to the monitoring system, using appropriate measures of performance of the inference system.

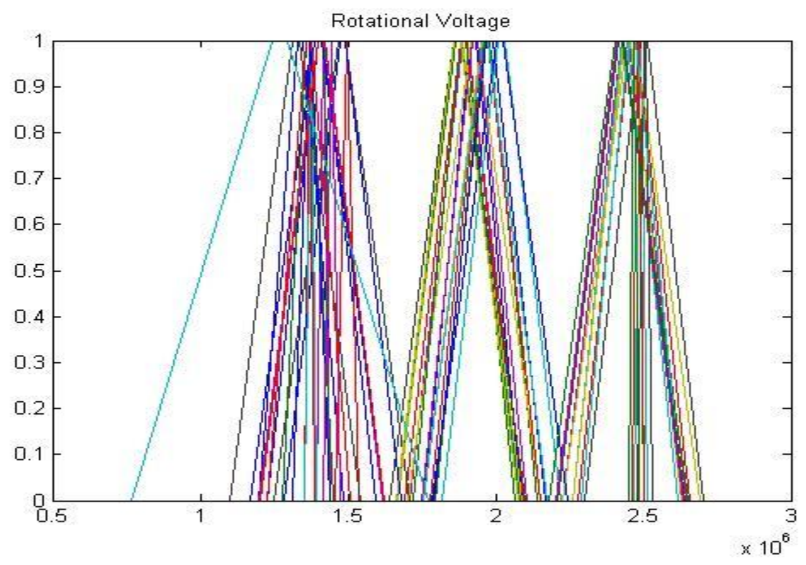


Figure 5.3: Example of linguistic variable – Rotational Voltage

Considering that an automatic process was used to create the terms for the linguistic variables, in Figure 5.3 we can see that many terms of this linguistic variable overlap heavily or are even included in others. It is also easy to observe some implicit clusters of similar membership functions. Merging the membership functions, pertaining to a cluster, into a single one seems an obvious way of reducing the number of membership functions in the system.

The contribution to this project was to define an algorithm capable of reducing the number of terms of a linguistic variable to improve the overall computational effort of the system without compromising the performance of the system.

		Classified							
		Concrete	GasConcrete	Marble	NotDrilling	Travertine	Tuff	Unknown	Total
Real	Concrete	299	0	0	0	0	0	242	541
	GasConcrete	0	267	0	0	26	0	229	522
	Marble	0	0	296	0	0	0	226	522
	NotDrilling	0	0	0	210	0	0	311	521
	Travertine	0	21	0	0	244	0	259	524
	Tuff	0	0	0	0	0	271	251	522
	Total	299	282	296	210	270	271	1529	**3152**

Table 5.1: MODI Confusion Matrix

The measures of performance used to compare the resulting terrain recognition inference systems were the Precision of the classification [Santos, Fonseca et al. (to appear 2008)], the Mean Certainty Level and a combination of these two measures. Consider the confusion matrix in Table 5.1 that shows the hardness types of terrain of a test data set and its classification with the MODI inference system.

The overall Precision (P) of the terrain recognition inference system is the ratio between the number of well classified samples (grey cells) and the total number of samples [Santos, Fonseca et al. 2008]. For the confusion matrix in Table 5.1 the precision would be 0.5035 (50.35%). The Mean Certainty Level (MCL) is the average of the certainty levels obtained for each sample, for the samples that were correctly classified. To combine these two measures of performance, it is used an average of these two (based on the F1 score [Rijsbergen 1979]) given by:

$$F = \frac{2 \cdot P \cdot MCL}{P + MCL} \tag{5.1}$$

These three measures of performance take values between 0 and 1 and are to be maximized.

5.2 Heuristics

In [Setnes, Babuska et al. 1998], Setnes presents a rule base simplification algorithm that can be summarized by the fluxogram in Figure 5.4. Given a fuzzy variable with initial membership functions M and a threshold minS for the similarity (set by the user or by some other algorithm), we select the most similar pair of fuzzy sets A and B (for a certain fuzzy similarity measure). If this similarity value is above the previously defined threshold (minS), we combine these two fuzzy sets, update the rule-base and the algorithm proceeds by choosing the most similar pair in M. Otherwise, the algorithm stops and M is returned.

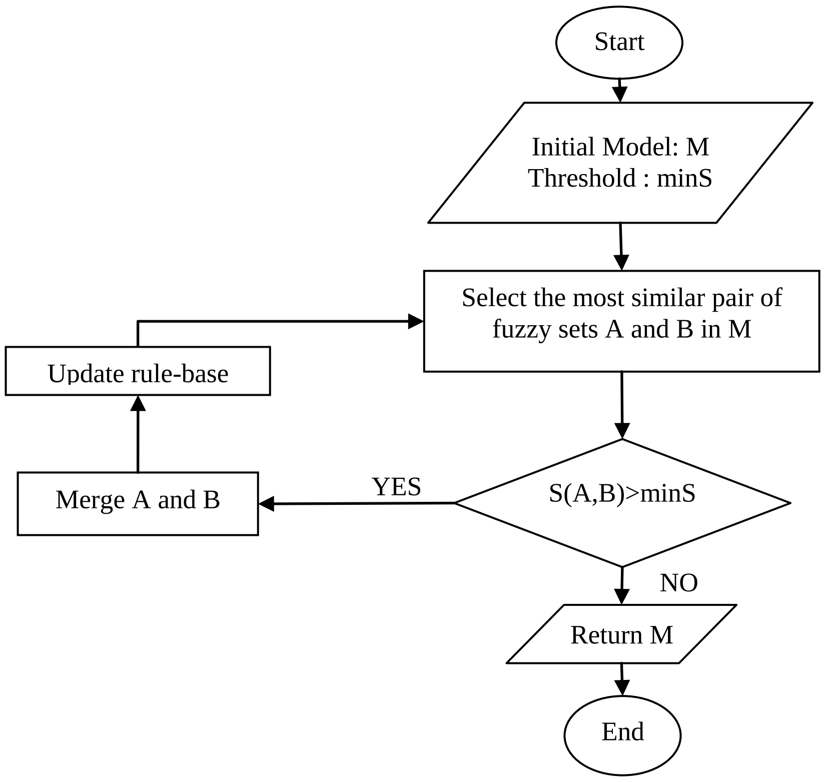


Figure 5.4: Original algorithm [Setnes, Babuska et al. 1998]

Instead of basing the decision of merging or not two membership functions A and B only on their similarity, it was decided to look at the differences in terms of "design" between the original model and the ones achieved each iteration. This way a more global view of the changes being made to the linguistic variable being pruned was done. Therefore it was defined a measure for comparing two sets of membership functions representing the same linguistic variable. This way it would be possible to compare the models obtained through the algorithms to the initial model or linguistic variable.

A distinguishability measure Δ between fuzzy sets can be defined as the complement of their similarity measure [Mencar, Castellano et al. 2007].

$$\Delta(A,B) = 1 - S(A,B) \tag{5.2}$$

Given a new set of membership functions (obtained, for instance, by merging two or more membership functions of the original set) and the correspondence between the new set and the original one, we can define what can be intuitively seen as a model error by taking the average distinguishability measures between all original sets and the one that represents them in the new set. Consider the example in Figure 5.5:

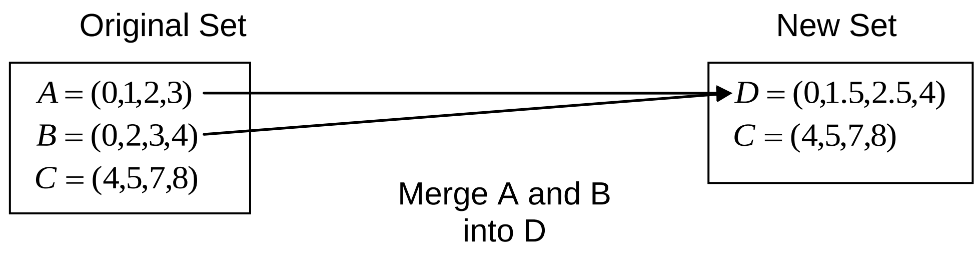


Figure 5.5: Example of model error

Using the Jaccard similarity measure [Mencar, Castellano et al. 2007] given by equation (1.19) with the pair (minimum, maximum) as intersection and union operators we obtain:

$$S(A,D) = 0.6364 \qquad S(B,D) = 0.8182 \qquad S(C,C) = 1 \tag{5.3}$$

Thus,

$$\begin{aligned} ModelError &= \frac{\Delta(A,D) + \Delta(B,D) + \Delta(C,C)}{3} = \\ &= \frac{(1-0.6364) + (1-0.8182) + (1-1)}{3} = 0.1818 \end{aligned} \tag{5.4}$$

The adapted algorithm can then be summarized by the fluxogram in Figure 5.6. Given a fuzzy variable with initial linguistic variables M and a threshold ε for the model error (set by the user or by some other algorithm), we select the most similar pair of fuzzy sets A and B (for a certain fuzzy similarity measure) and combine these two fuzzy sets, thus obtaining a new set of linguistic variables, M'. If the model error, denoted by ε, is above the threshold or if all pairs of fuzzy sets are totally dissimilar (similarity zero) the algorithm stops. Otherwise, the algorithm proceeds by choosing the most similar pair in M'.

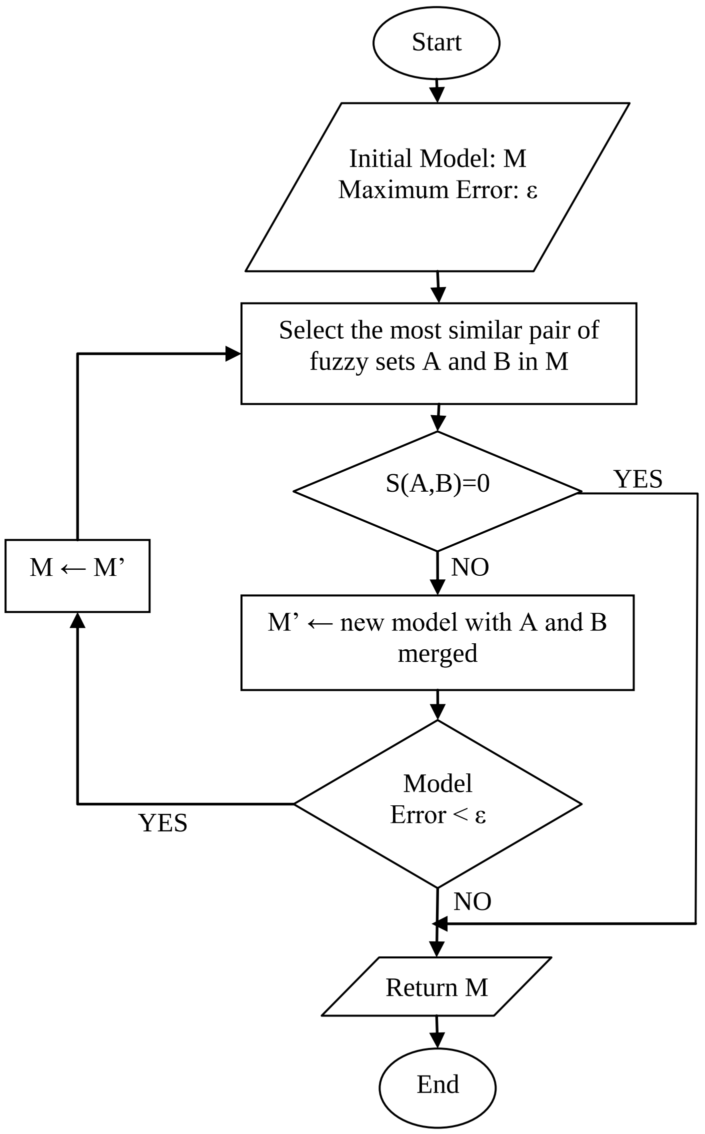
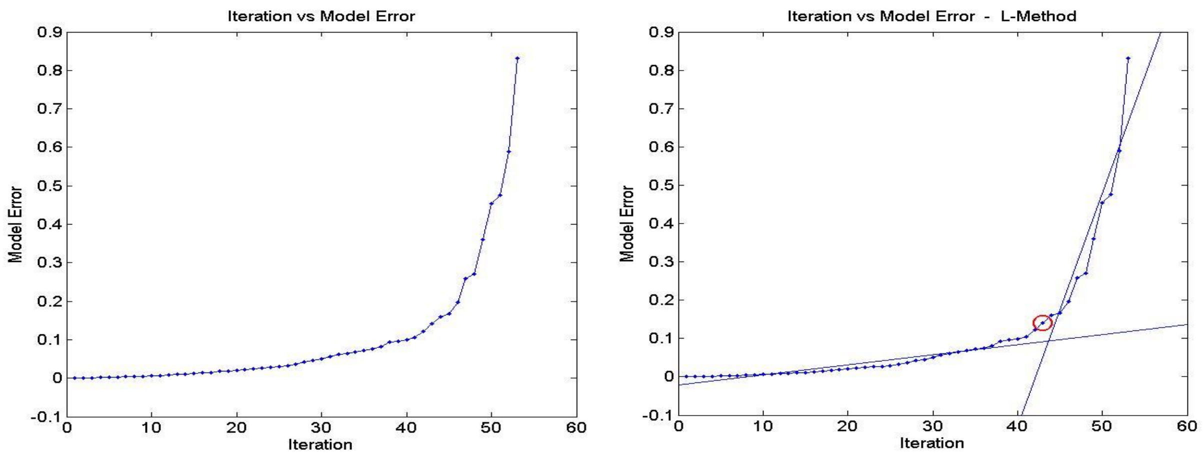

Start
Initial Model: M
Maximum Error: ε
Select the most similar pair of fuzzy sets A and B in M
S(A,B)=0
YES
NO
M ← M'
M' ← new model with A and B merged
Model Error < ε
YES
NO
Return M
End

Figure 5.6: Adapted algorithm

The above algorithm can be seen as a hierarchical clustering algorithm. We start with as many clusters as the initial number of membership functions and in each step we merge two clusters into a single one. Several methods of choosing the optimal number of clusters (and thus the optimal number of iterations) for this kind of clustering algorithms have been proposed [Milligan and Cooper 1985; Salvador and Chan 2004]. Some consist of finding the knee of a curve obtained by representing the number of iterations or number of clusters versus some metric of evaluation of the clustering algorithm. Since using the inference system performance measure (5.1) as evaluating metric during the clustering would be time consuming, it was decided to use the model error instead. Running the adapted algorithm using the

Jaccard similarity measure in (1.19) with the minimum and maximum operators until all membership functions are disjoint and plotting each iteration number versus its model error, gave a curve like the one in Figure 5.7 a). The Model Error increases slowly with each iteration until it reaches a point where it starts to grow exponentially. We can choose the number of iterations to be the x-axis coordinate of that point. This way, we will have an automatic method for choosing the number of iterations of the algorithm and we will no longer need to heuristically choose the parameter ε.



a) Iteration vs Model Error b) L-Method

Figure 5.7: a) Iteration vs Model Error; b) L-method

To find the knee of this curve the L-method, proposed by Salvador and Chan in [Salvador and Chan 2004], was used. Let n be the number of the last iteration, that is, $x = 1, \ldots, n$. Let L_c and R_c be the left and right sequences of data points partitioned at $x = c,\ c = 2, ..., n-2$. Fitting a line to L_c and another to R_c we can define the total root mean square error (RMSE) by:

$$RMSE_c = \frac{c}{n} RMSE(L_c) + \frac{n-c}{n} RMSE(R_c) \tag{5.5}$$

i.e., the total root mean squared error is a weighted average of the root mean squared error of both fittings. The optimal number of iterations is the value c that minimizes $RMSE_c$. In Figure 5.7 b), c is the number of chosen iterations corresponding to the point circled ($c = 43$). The c value is the stopping criterion threshold used in the adapted algorithm.

As will be seen through the computational results presented in section 5.3, although this approach improves the original inference system used as a test case, the use of the L-method as a stopping criterion is not the best choice. By penalizing dissimilarity, it assumes that the initial model is the one with best performance, which is not the case in this case study. Since the original inference system was obtained automatically from sensor data, it is full of redundancy. The stopping criterion should not be based on a comparison in terms of “design” to the original model but on the actual performance measures of the inference system.

The final algorithm, called bestP, is illustrated in Figure 5.8.

We ran the original algorithm (Figure 5.4) until all membership functions are disjoint (similarity zero). In each iteration, the rules are updated and the performance of the resulting inference system, P(M), is obtained and compared with the performance of the best model found so far, P(BestM). The bestP algorithm returns the inference system with best performance, from the ones generated throughout the iterations. Note that this algorithm is defined for any performance measure, P(.). In our case study, we used the performance measure F given by (5.1). This means that we will be maximizing the inference system performance. If the initial system is the best system in terms of this measure of performance, there will be no reduction in the number of membership functions. If we want to find a compromise between the number of membership functions and the inference system performance we can use a linear combination of these two objectives in the following way:

$$P(M) = \alpha F - \frac{1-\alpha}{n_0} n \tag{5.6}$$

where $\alpha \in [0,1]$ is the weight given to F, n_0 is the initial number of membership functions, used as a scaling factor and n is the number of membership functions of model M.

Finally, the adapted algorithm is applied sequentially to each input variable to be pruned (in this case, all input variables except the set points). After this pruning is completed, duplicate rules are removed from the rule system to improve the computational efficiency of the final inference system.

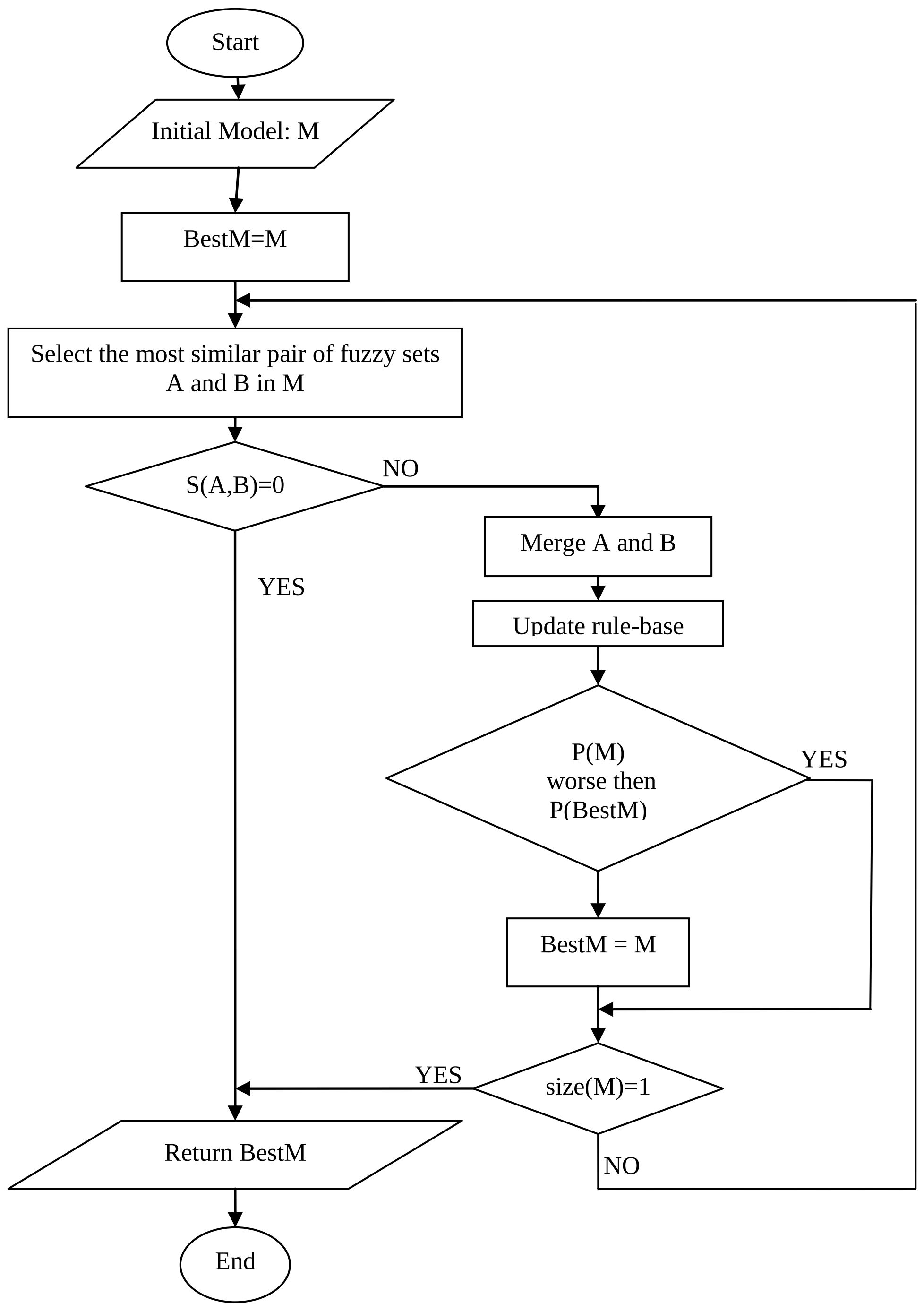


Figure 5.8: Final algorithm – bestP

5.3 Computational Results

The heuristic algorithms described in the previous section were applied to a terrain recognition inference system constructed as described in section 5.1. All experiments were done using Matlab and Java.

The set points used were:

1. Rotation Speed (rpm) - SPRS
2. Translational Speed (mm / min) - SPTS

The sensor's variables used were:

1. Rotation Current (A) - RC
2. Rotation Voltage (V) - RV
3. Rotation Speed (rpm) - RS
4. Thrust (N) - TH
5. Torque (N) - TO
6. Translational Voltage (V) - TV
7. Translational Current (A) - TC
8. Translational Speed (mm/min) - TS

Six different types of terrain hardness were tested. The 6 scenarios considered were: Not Drilling (drilling in air); Concrete; Gas-Concrete; Marble; Travertine; and Tuff. Setting 3 different values for each of the set points, these scenarios were further sub-divided into 54 ($6 \times 3 \times 3$) sub-scenarios that provided the basis for the construction of the linguistic variables representing each input variable.

The original membership functions in this inference system are represented in Figure 5.9. In all linguistic variables in the system it is clear that some of its terms should be merged. The sensor values were collected as integers. Integer programming is more efficient and was thus used to guarantee real time tasks. Therefore, the x-axis of the plots representing the linguistic variables in the system have no physical meaning, i.e., they cannot be interpreted as voltages, rotations per minute, ...

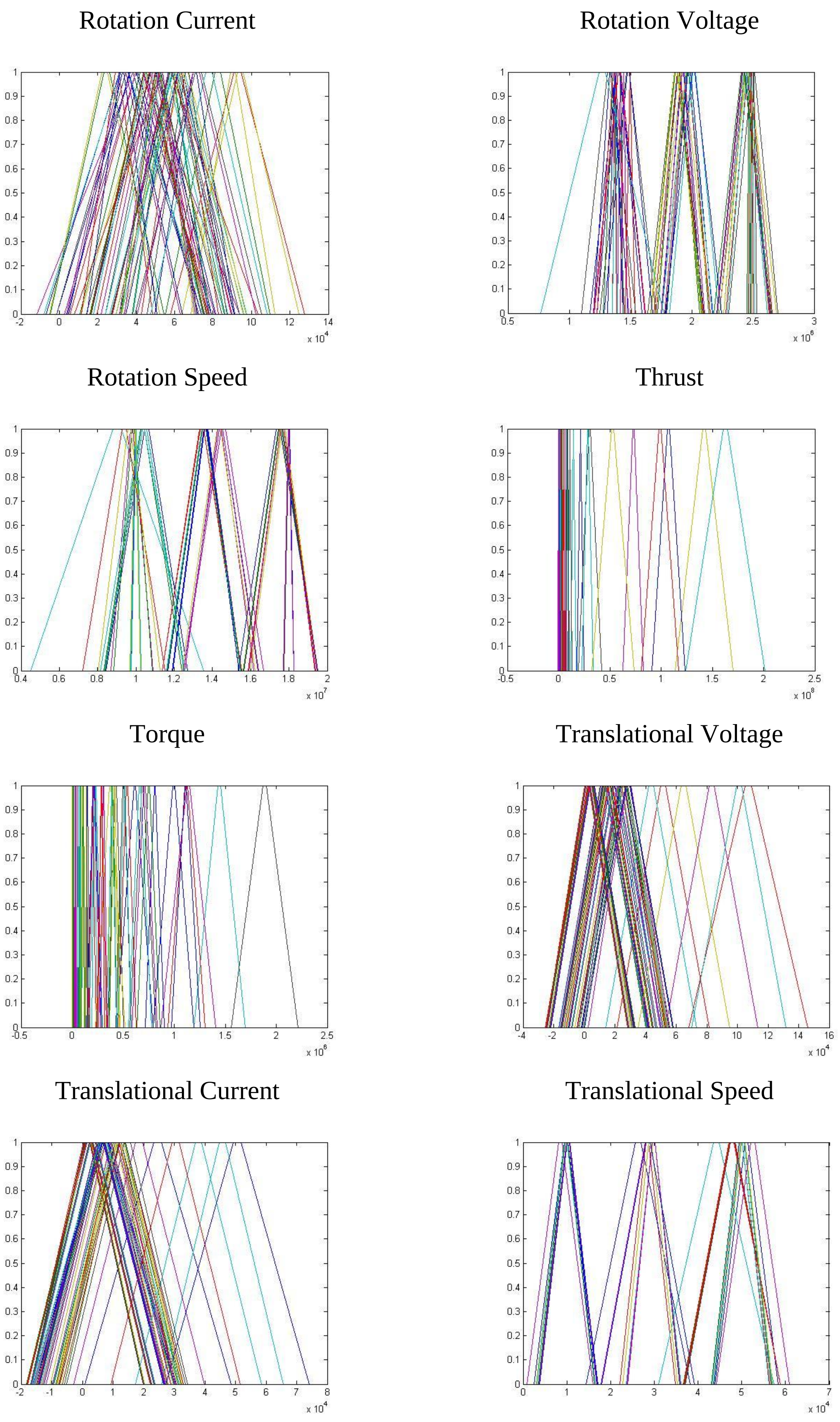


Figure 5.9: Original input linguistic variables (except set points)

First, the algorithm summarized by Figure 5.6 (adapted algorithm) with the stopping criterion defined by minimization of (5.5) – L-method [Salvador and Chan 2004] - was applied sequentially to all input variables except set points. Then, the algorithm summarized by Figure 5.8, were the inference system with best performance is chosen, was used in the same sequence. The following figures show the evolution of the performance measure F (to be maximized) given by equation (5.1) when the algorithms run until all membership functions are disjoint. The vertical lines mark the iteration chosen by the L-method and the circle marks the iteration with best performance, i.e., the iteration chosen by bestP.

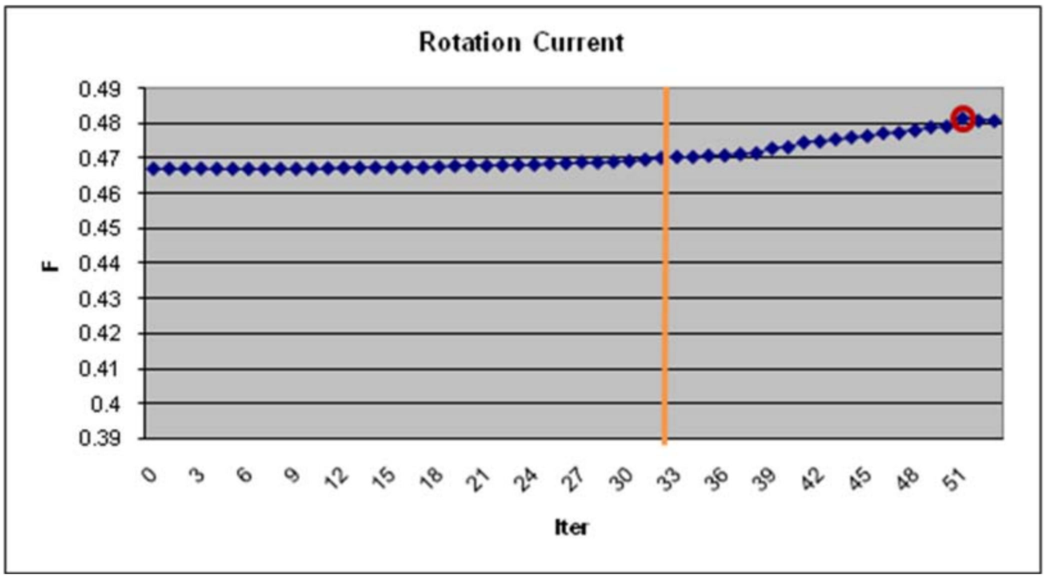


Figure 5.10: Evolution of performance measure F during the algorithm – Rotation Current

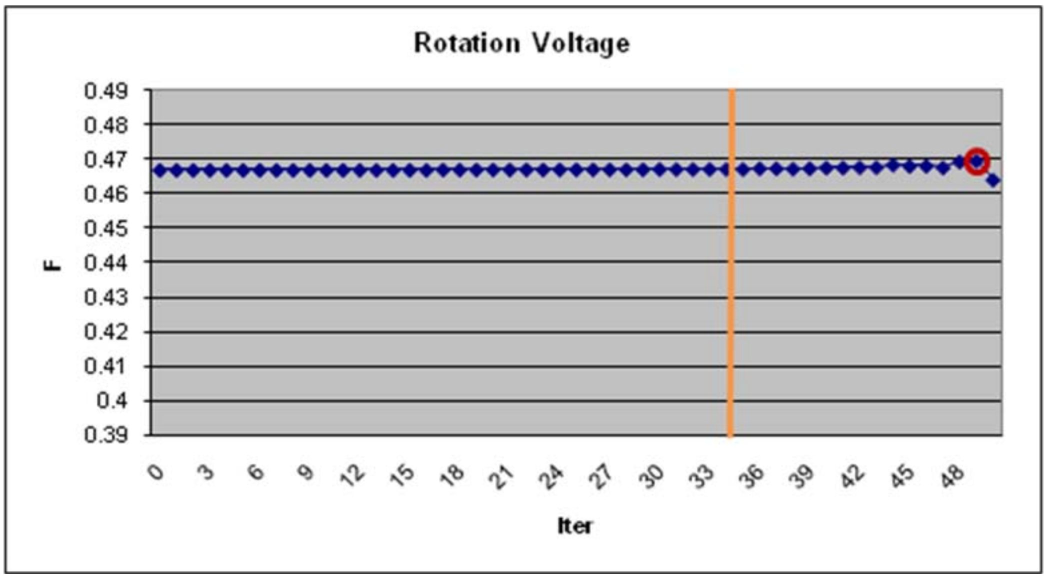


Figure 5.11: Evolution of performance measure F during the algorithm – Rotation Voltage

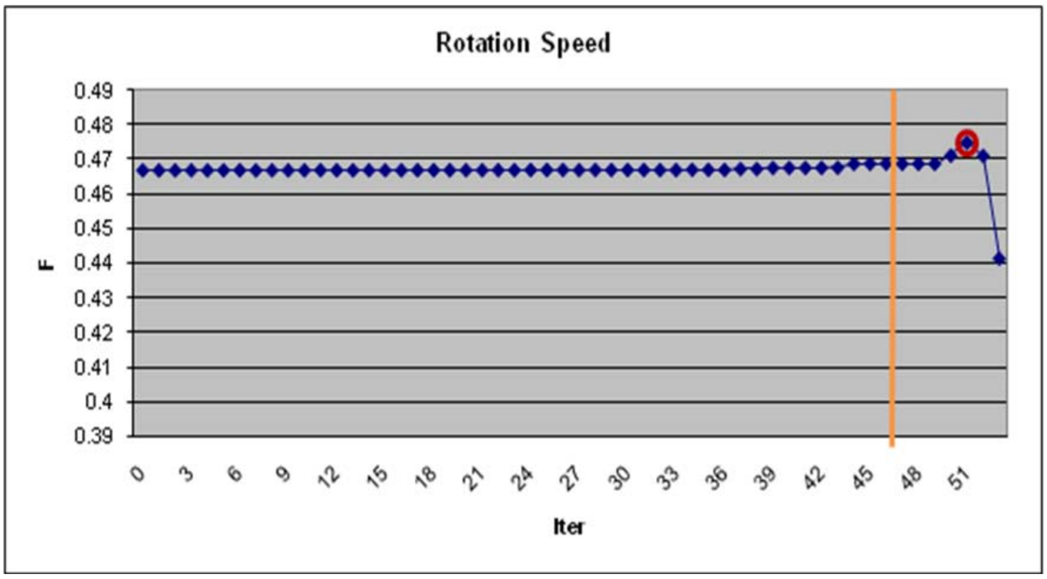


Figure 5.12: Evolution of performance measure F during the algorithm – Rotation Speed

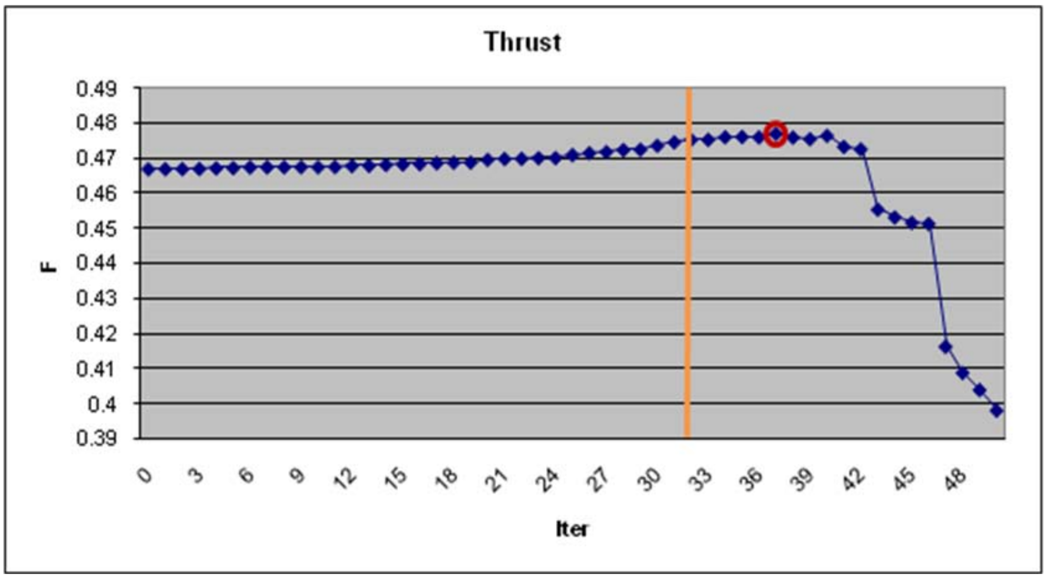


Figure 5.13: Evolution of performance measure F during the algorithm – Thrust

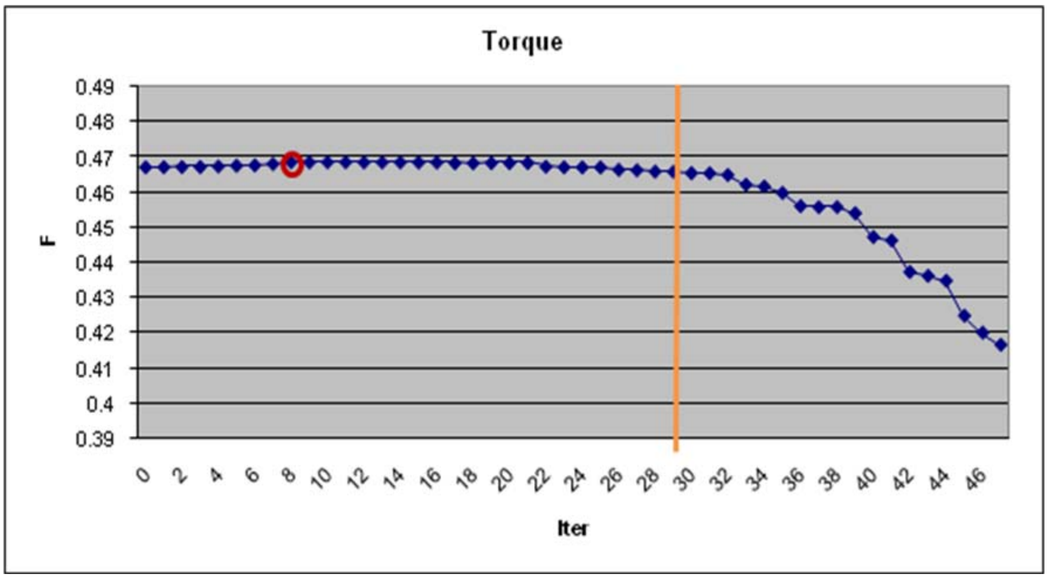


Figure 5.14: Evolution of performance measure F during the algorithm – Torque

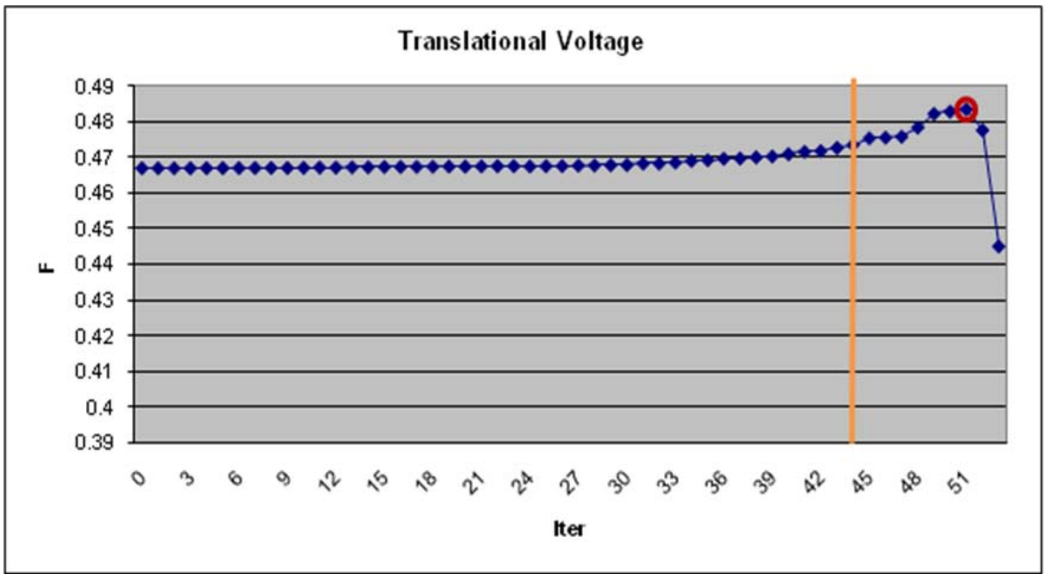


Figure 5.15: Evolution of performance measure F during the algorithm – Translational Voltage

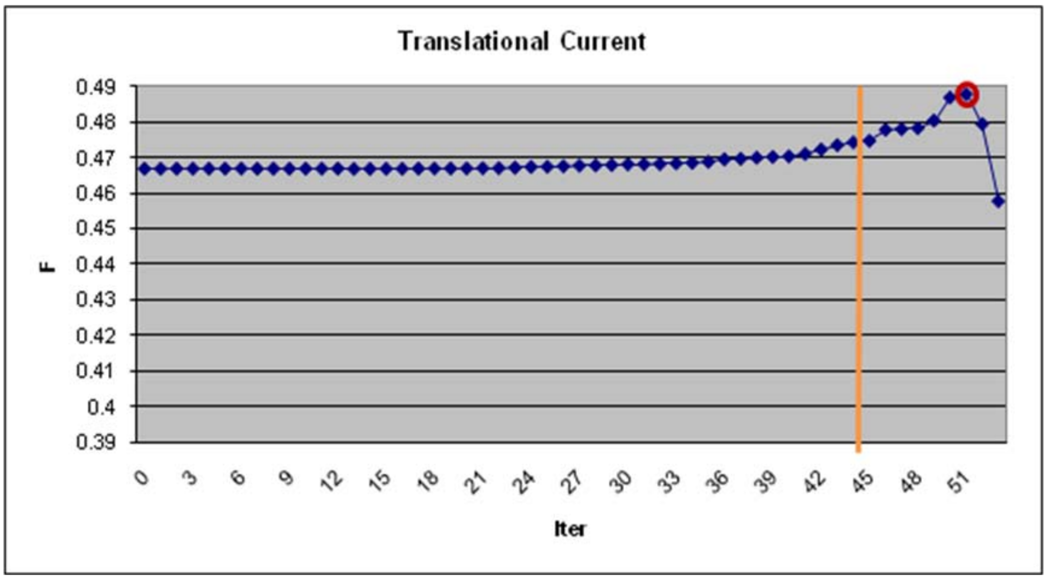


Figure 5.16: Evolution of performance measure F during the algorithm – Translational Current

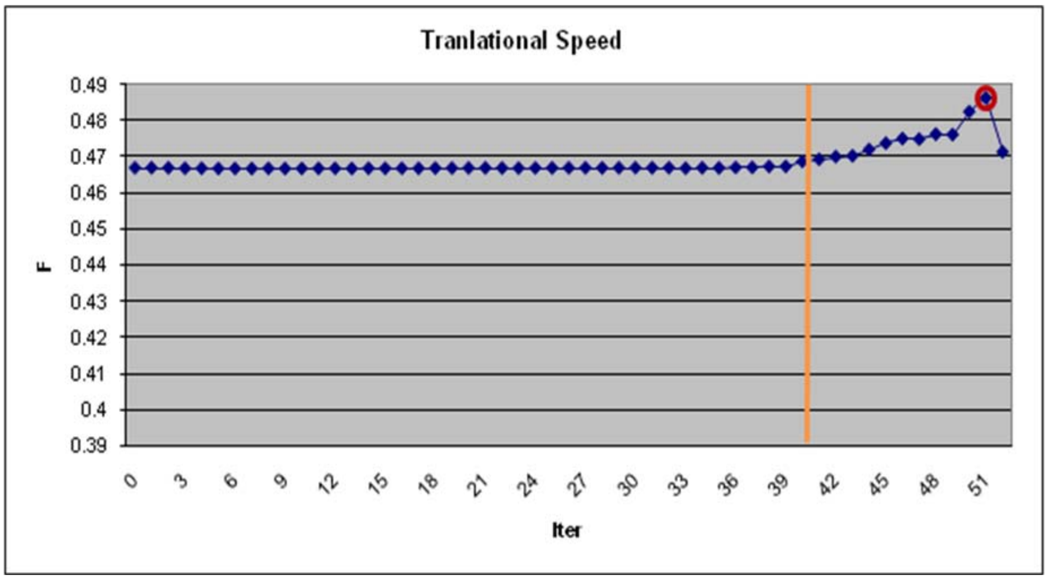


Figure 5.17: Evolution of performance measure F during the algorithm – Translational Speed

	Original	Adapted Algorithm	BestP
Rotation Current	54	21	3
Rotation Voltage	54	20	5
Rotation Speed	54	8	3
Thrust	54	22	17
Torque	54	25	46
Translational Voltage	54	10	3
Translational Current	54	10	3
Translational Speed	54	13	3
TOTAL	432	129	45

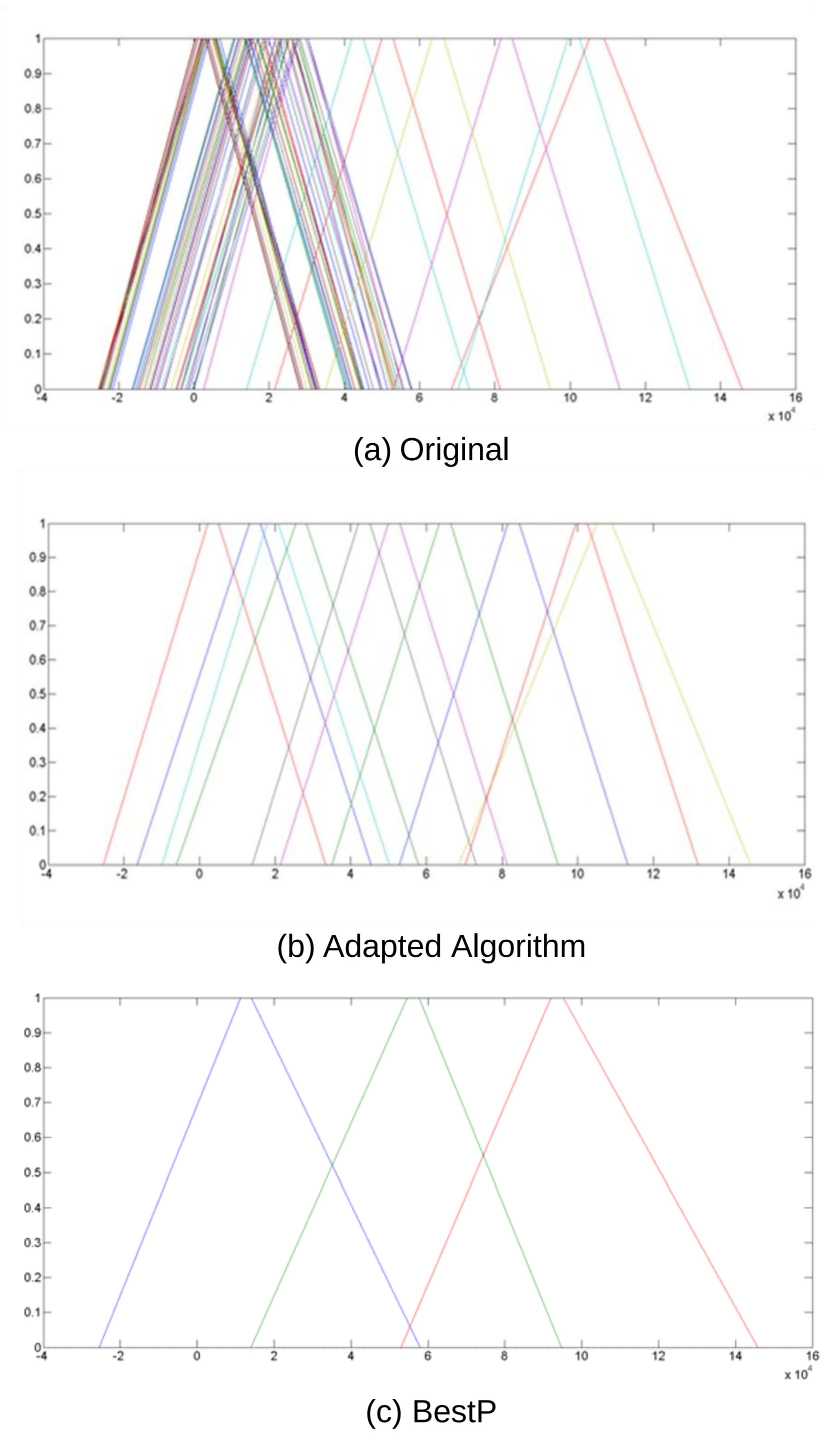
Table 5.2: Number of membership functions before and after optimization

As can be seen by the previous figures and by Table 5.2, using the L-method as a stopping criterion is not the best choice. In most cases this stopping criterion chooses to stop too early. Although in this case study this method already reduces the number of membership functions without losing performance, this method fails to choose the best iteration to stop. Analyzing the previous figures it is clear why concentrating on the model error instead of the inference system actual performance is not a good strategy. By minimizing the model error, changes to the original inference system are being penalized. If the original system was “perfect”, reducing the number of terms in linguistic variables by merging similar membership functions should negatively affect the performance of the system. The previous figures show that this is not the case, merging membership functions is actually increasing the inference system performance. This means that the initial system is full of redundancy and that some of this redundancy is being eliminated by the algorithm. For this reason, instead of using the model error, a comparison between the initial inference system and the ones obtained by the algorithm, it is wiser to focus on performance measures such as the ones described in the previous section. This is what motivated the use of the bestP algorithm.

	Original	Adapted Algorithm	BestP
P	72.33%	76.49%	85.47%
MCL	34.49%	36.88%	44.00%
F	46.71%	49.77%	58.09%
N	423	129	45

Table 5.3: Comparison of inference systems

Table 5.3 compares the three inference systems – original inference system and inference systems obtained using the Adapted Algorithm and the BestP algorithm – in terms of systems performance measures and number of membership functions. Figure 5.18 shows the linguistic variable Translational Voltage before and after using both algorithms. By using the bestP heuristic it was possible to reduce the number of membership functions in the system and improve its overall performance. It was possible to reduce the number of membership functions in input linguistic variables (except set points) from 432 to 45. All measures of performance used improved after this optimization. The easiest to interpret performance measure, the Precision of the classification (P), increased from 72.33% to 85.47%. The advantages of using this algorithm in this case study was clear. If it was desirable to further reduce the number of membership functions a measure such as the one in equation (5.6) could be used. Figure 5.19 shows the linguistic variables returned by the algorithm. Some linguistic variables seem to be too reduced, i.e., its new linguistic variables do not seem well representative of the original linguistic variables. We have to keep in mind that the original system was created automatically and is full of redundancy.



(a) Original

(b) Adapted Algorithm

(c) BestP

Figure 5.18: Linguistic Variable Translational Voltage before (a) and after using the adapted (b) and BestP (c) algorithms

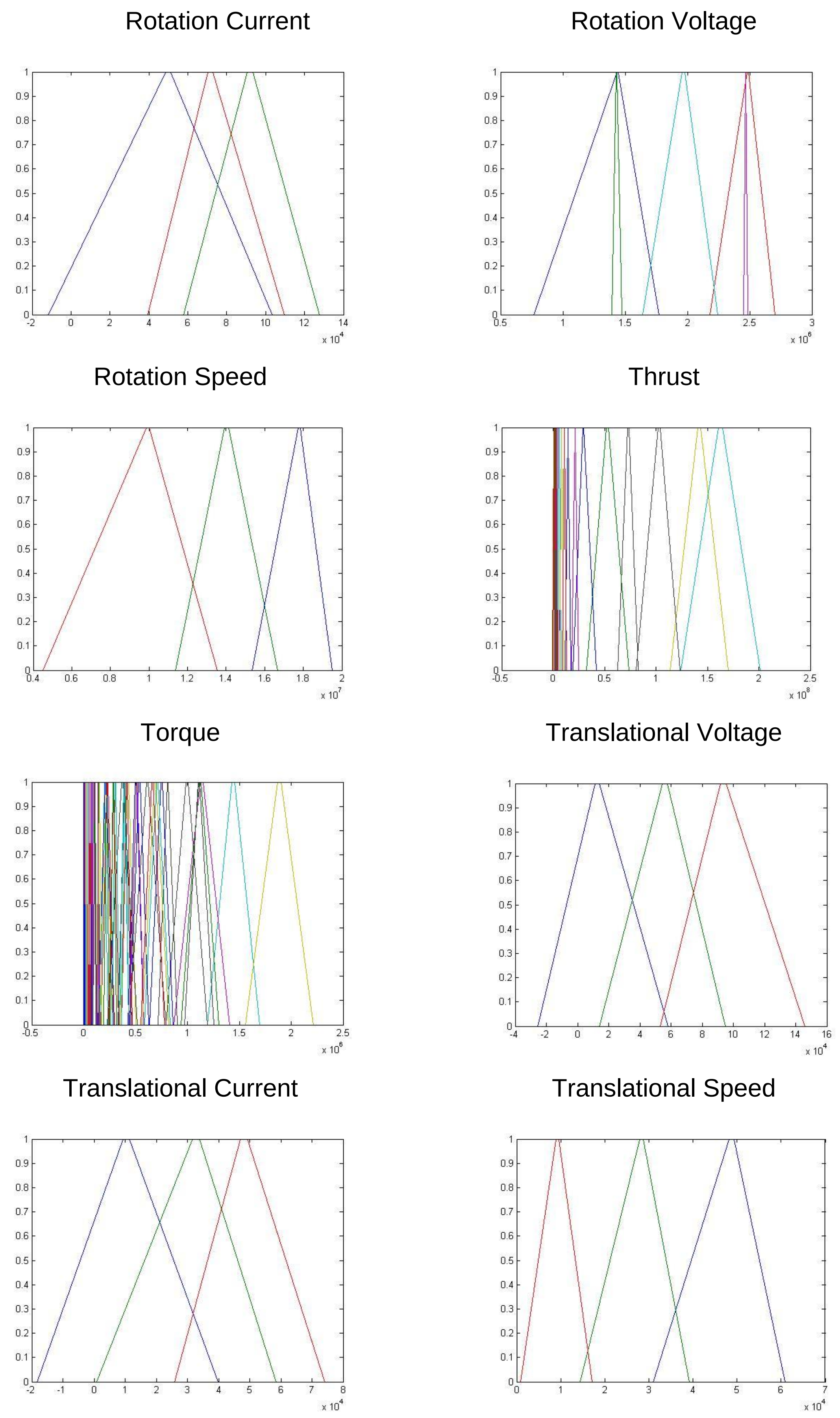


Figure 5.19: Input linguistic variables after optimization with BestP (except set points)

5.4 Summary

In this chapter the goal was to reduce the number of membership functions in linguistic variables of an already existing inference system. An inference system construted in the scope of a project developed for ESA served as case study. The inference system was constructed automatically from sensor data and was thus full of redundancy. The aim was to reduce the complexity of the inference system while maintaining its structure, without losing too much performance. Not only was this objective achieved, but also the inference system performance was increased. Moreover, a paper about this subject was published [Gomes, Santos et al. 2008].

I could have presented only the last algorithm, bestP, and ignore the intermediate attempts to find an algorithm for reducing the number of membership functions. However, I preferred to describe my first and more intuitive approach to this problem for two reasons: first, to show the importance of trial and error in science and second because the results obtained with the first adapted algorithm justify the necessity to evaluate the system performance during the algorithm instead of concentrating on “design” measures such as the model error.

There are still some questions that should be answered about this procedure. The algorithm was applied sequentially to input linguistic variables. There was no study about the importance of the order in which this pruning is made. I believe that this order can have some impact on the results. To solve this problem, instead of sequentially running the algorithm, a global algorithm where in each iteration the most similar membership functions from any linguistic variable was merged could be designed. Another question concerns the similarity measure used. Experiments with other similarity measures should be used to justify the choice of similarity measure.

Chapter 6. Conclusions and Future Work

The purpose of this thesis was to develop algorithms to reduce the number of membership functions in a linguistic variable. One of the main advantages of fuzzy models is that they are usually less complex and easy to interpret than classical models. By reducing the number of membership functions in linguistic variables the aim was to achieve simpler and more efficient fuzzy models.

The problem of reducing the number of membership functions in a linguistic variable was approached as a clustering problem. The pruned linguistic variable was the result of merging clusters of similar membership functions into a new membership function. Some possible formulations to the clustering problem were presented. Exact methods were used with one of these formulations and the combinatorial nature of clustering problems was clear. As expected, only very small data sets can be optimally solved through these methods in a reasonable amount of time. Although it was never the purpose of this thesis to use exact methods to solve this problem, this was an important step to better understand the dimension of the problem at hands.

To find good quality solutions in a more reasonable amount of time a Scatter Search procedure was developed and compared to the K-Means++ algorithm. Both procedures were implemented in Matlab and tested with two different case studies. The linguistic variables from these case studies could later be used in a fuzzy inference system or any other fuzzy model constructed taking into account the pruned membership functions instead of the original ones. The computational results were not as expected. The second phase of the Scatter Search algorithm was not able to produce good quality solutions. However, the first part of the algorithm was sufficient to obtain better results than the ones given by the K-Means algorithm. Both methods achieved a high reduction in the number of membership functions in each linguistic variable.

The last chapter presented a different case study. The objective was to reduce the number of membership functions in linguistic variables of an automatically constructed inference system without losing two much performance. It was seen that, in this case, concentrating on the characteristics of a single linguistic variable was insufficient. It is important to concentrate on the actual performance of the inference system, using appropriate measures of performance. Therefore the heuristics implemented before were not applied to this case study. Although the fitness function of the scatter search algorithm could have been changed to account for these performance measures, the time it takes to evaluate the inference system and the number of times it would be necessary to evaluate it explain why it was chosen not to use this algorithm in this case. The algorithms used in this case study can be categorized as hierarchical clustering algorithms. The results achieved in this case study were more than satisfying. It was possible to improve the initial inference system performance and simplify the system at the same time.

Although a lot of work was developed during this thesis, there is still much to be done in the future. In the first procedures a comparison of the different cluster validity indexes and of the shapes of the clusters themselves, translated by the clustering criteria used, should be made. Different strategies inside the scatter search algorithm could be tested to try to overcome the poor results obtained in terms of the way solutions are combined. More work is needed in estimating the correct number of clusters or a good maximum number of clusters to be given as input for the scatter search algorithm or other clustering algorithms. It would also be interesting to construct fuzzy models to identify the type of breast cancer (malign or benign) from the cell nuclei characteristics and to support credit approval decision processes using the linguistic variables from section 4.3 before and after being pruned, comparing results. In the MODI case study, as mentioned, the impact of the order in which the linguistic variables are pruned in the results or the possibility of designing a global algorithm that looked at all variables at the same time are possible directions for future work, along with a study of how results are affected by using different similarity measures. Furthermore, given the good results obtained in this case study, it is important to confirm the validity of the algorithm by running it on different case studies.

Chapter 7. References

Abdule-Wahab, R. S., N. Monmarché, M. Slimane, M. A. Fahdil and H. H. Saleh (2006). "A Scatter Search Algorithm for the Automatic Clustering Problem." Advances in Data Mining: 350-364.

Adlassnig, K. (1986). "Fuzzy set theory in medical diagnosis " IEEE Trans. Syst. Man Cybern. **16**(2): 260-265.

Aggarwal, C. C. and P. S. Yu (2000). Finding generalized projected clusters in high dimensional spaces. Proceedings of the 2000 ACM SIGMOD international conference on Management of data. Dallas, Texas, United States, ACM. 70-71.

Agrawal, R., J. Gehrke, D. Gunopulos and P. Raghavan (1998). Automatic subspace clustering of high dimensional data for data mining applications. Proceedings of the 1998 ACM SIGMOD international conference on Management of data. Seattle, Washington, United States, ACM. 94-105.

Anoop Kumar, D. and H. Moskowitz (1991). "Application of fuzzy theories to multiple objective decision making in system design." European Journal of Operational Research **53**(3): 348-361.

Asuncion, A. N., D.J. (2007). "UCI Machine Learning Repository " Retrieved March 2008, from http://archive.ics.uci.edu/ml/datasets/Credit+Approval.

Ball, G. H. and D. J. Hall (1965). ISODATA, a novel method of data analysis and classification. T. Rep. Stanford, CA, Stanford Univ.

Barnhart, C., E. Johnson, G. Nemhauser, M. Savelsbergh and P. Vance (1998). "Branch-and-Price: Column Generation for Solving Huge Integer Programs." Operations Research **46**(3): 316-329.

Brown, D. E. and C. L. Huntley (1990). A Practical Application of Simulated Annealing to Clustering, University of Virginia.

CA3. (2006). "MODI's homepage." Retrieved April 2007, 2007, from http://www2.uninova.pt/ca3/en/project_MODI.htm.

Chen, S.-J. and S.-M. Chen (2008). "Fuzzy risk analysis based on measures of similarity between interval-valued fuzzy numbers" Computers & Mathematics with Applications Pergamon **55**(8): 1670-1685.

Chen, S.-M., M.-S. Yeh and P.-Y. Hsiao (1995). "A comparison of similarity measures of fuzzy values". Fuzzy Sets and Systems **72**(1)**:** 79-89.

Chen, S. M. (1996). "New Methods for Subjective Mental Workload Assessment and Fuzzy Risk Analysis." Cybernetics and Systems **27**: 449-472.

Cheng, C., W. Lee and K. Wong (2002). "A genetic algorithm-based clustering approach for database partitioning." IEEE Transactions on Systems, Man and Cybernetics, Part C **32**(3): 215–230.

Colin, R. R., Ed. (1993). Modern heuristic techniques for combinatorial problems, John Wiley & Sons, Inc.

Costa, A., A. D. Gloria, F. Giudici and M. Olivieri (1997). "Fuzzy Logic Microcontroller." IEEE Micro **17**(1): 66-74.

David, A. and V. Sergei (2007). k-means++: the advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. New Orleans, Louisiana, Society for Industrial and Applied Mathematics. 1027-1035

Dempster, A., N. Laird and D. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society. Series B (Methodological) **39**(1): 1–38.

Edwards, A. W. F. and L. L. Cavalli-Sforza (1965). "A Method for Cluster Analysis." Biometrics **21**(2): 362-375.

Engelbrecht, A. (2002). Computational Intelligence: An Introduction, Halsted Press.

ESA. (2008). "Aurora Exploration Program." Retrieved January 2008, from http://www.esa.int/esaMI/Aurora/index.html.

Ester, M., H.-P. Kriegel, J. Sander and X. Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. 2nd International Conference on Knowledge Discovery and Data Mining. Portland, OR**:** 226-231.

Estivill-Castro, V. (2002). "Why so many clustering algorithms: a position paper". ACM. **4:** 65-75.

Everitt, B., S. Landau and M. Leese (2001). Cluster Analysis, Arnold Publishers.

Florek, K., J. Lukaszewicz, H. Steinhaus and S. Zubrzycki (1951). "Sur la liaison et la division des points d'un ensemble fini." Colloquium Mathematicum **2**: 282-285.

Gan, G., C. Ma and J. Wu (2007). Data Clustering: Theory, Algorithms, and Applications Philadelphia, SIAM.

Gautam, G. and B. B. Chaudhuri (2004). "A novel genetic algorithm for automatic clustering", Elsevier Science Inc. **25:** 173-187.

Glover, F. (1977). "Heuristics for Integer Programming Using Surrogate Constraints." Decision Sciences **8**(1): 156-166.

Glover, F. (1986). "Future paths for interger programming and links to artificial intelligence." Computer & Operation Research **13**(5): 533-549.

Glover, F. and M. Laguna (1997). Tabu Search, Kluwer Academic Publishers.

Gomes, M. M., B. R. Santos, T. Simas, P. Sousa and R. A. Ribeiro (2008). Reducing the Number of Membership Functions in Linguistic Variables: Application to a Fuzzy Monitoring System. Eight International Conference on Application of Fuzzy Systems and Soft Computing Helsinki, Finland, b- Quadrat Verlag.

Gower, J. C. and G. J. S. Ross (1969). "Minimum Spanning Trees and Single Linkage Cluster Analysis." Applied Statistics **18**(1): 54-64.

Hansen, P. and M. Delattre (1978). "Complete-Link Cluster Analysis by Graph Coloring." Journal of the American Statistical Association **73**(362): 397-403.

Hartigan, J. A. (1975). Clustering Algorithms. New York, Jonh Wiley and Sons.

Hillier, F. S. and G. J. Lieberman (2005). Introduction to Operation Research. Singapore, McGraw-Hill.

Holland, J. H. (1975). Adaptation in natural and artificial systems. Ann Arbor, University of Michigan Press.

Hubert, L. (1974). "Some applications of graph theory to clustering." Psychometrika **39**(3): 283-309.

Isermann, R. (1998). "On fuzzy logic application for automatic control, supervision, and fault diagnosis." IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **28**(2): 221-235.

Jain, A. K. and R. C. Dubes (1988). Algorithms for Clustering Data. New Jersey, Prentice Hall, Englewood Cliffs.

Jain, A. K., M. N. Murty and P. J. Flynn (1999). "Data Clustering: A Review." ACM Computing Surveys **31**(3): 264-323.

Jameaux, D., R. Vitulli, R. A. Ribeiro, T. Fonseca, B. R. Santos and M. Barata (2006). Monitoring & Diagnosis on-board software module for Mars driller. Proceedings of the 5th International Workshop on Planning and Scheduling for Space.

Jardine, N. and R. Sibson (1968). "The Construction of Hierarchic and Non-Hierarchic Classifications." The Computer Journal **11**(2): 177-184.

Jean-Philippe, H. and H. Jin-Kao (2002). Scatter Search for Graph Coloring. Selected Papers from the 5th European Conference on Artificial Evolution, Springer-Verlag.166-179

Jiang, T. and S. Ma (1996). Cluster analysis using genetic algorithms. Proceedings of the third international conference on signal processing **2**: 1277–1279.

Jimenez, J. F., F. J. Cuevas and J. M. Carpio (2007). Genetic algorithms applied to clustering problem and data mining. Proceedings of the 7th WSEAS International Conference on Simulation, Modelling and Optimization. Beijing, China, World Scientific and Engineering Academy and Society (WSEAS). 219-224

Joyce, C. W. and K. N. Michael (2000). A Tabu Search Based Algorithm for Clustering Categorical Data Sets. Proceedings of the Second International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents, Springer-Verlag.559-564

Kaufman, L. and P. Rousseeuw (1990). Finding Groups in Data - An Introduction to Cluster Analysis. New York, John Wiley and Sons, Inc.

King, B. (1967). "Step-wise clustering procedures." Journal of the American Statistical Association **69**: 86–101.

Kirkpatrick, S., C. D. Gelatt, Jr. and M. P. Vecchi (1983). "Optimization by Simulated Annealing." Science **220**(4598): 671-680.

Klir, G. J. and B. Yuan (1995). Fuzzy Sets and Fuzzy Logic: Theory and Applications. New Jersey, Prentice Hall.

Kruskal, J. B., Jr. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society **7**(1): 48-50.

Kuiper, F. and L. Fisher (1975). "Shorter communications 391: A Monte Carlo comparison of six clustering procedures." Biometrics **31**(3): 777-783.

Lai, Y. J. and C. L. Hwang (1994). Fuzzy multi objective decision making methods and applications. Berlin, Springer-Verlag.

Land, A. H. and A. G. Doig (1960). "An Automatic Method for Solving Discrete Programming Problems." Econometrica **28**(3): 497-520.

Lee, C. C. (1990a). "Fuzzy logic in control systems: fuzzy logic controller. I." Systems, Man and Cybernetics, IEEE Transactions on **20**(2): 404-418.

Lee, C. C. (1990b). "Fuzzy logic in control systems: fuzzy logic controller. II." Systems, Man and Cybernetics, IEEE Transactions on **20**(2): 419-435.

Lourenço, L. L. (1995). Contributos da Optimização Discreta para a Análise Classificatória. Aplicação de Heurísticas Genéticas a uma Classificação com Precedências. Dissertação de Mestrado em Matemática Aplicada à Economia e Gestão, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa.

Martí, R., M. Laguna and V. Campos (1997). "Scatter Search vs. Genetic Algorithms: An Experimental Evaluation with Permutation Problems." Adaptive Memory and Evolution: Tabu Search and Scatter Search. C. Rego, Alidaee, B.(Eds.), Kluwer Academic Publishers.

Mathworks Matlab Fuzzy Logic Toolbox **-** User's Guide.

Maulik, U. and S. Bandyopadhyay (2000). "Genetic algorithm-based clustering technique." Pattern Recognition **33**(9): 1455–1465.

McErlean, F. J., D. A. Bell and S. I. McClean (1990). "The use of simulated annealing for clustering data in databases". Information Systems. **15**(2)**:** 233-245.

McQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability: 281–297.

McQuitty, L. (1957). "Elementary linkage analysis for isolating orthogonal and oblique types
and typal relevancies." Educational and Psychological Measurement **17**: 207-222.

Mencar, C., G. Castellano and A. M. Fanelli (2007). "Distinguishability quantification of fuzzy sets." Information Sciences **177**(1): 130-149.

Milligan, G. W. and M. C. Cooper (1985). "An examination of procedures for determining the number of clusters in a data set." Psychometrika **50**(2): 159-179.

Mirkin, B. (2005). Clustering For Data Mining: A Data Recovery Approach New York, Chapman & Hall/CRC.

Miyamoto, S. (1990). Fuzzy Sets in Information Retrieval and Clustering Analysis, Kluwer Academic Publishers.

Murtagh, F. (1983). "A survey of recent advances in hierarchical clustering algorithms." The Computer Journal **26**(4): 354-359.

Nakashima, T., H. Ishibuchi and T. Murata (1998). Evolutionary algorithms for constructing linguistic rule-based systems for high-dimensional pattern classification problems. Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.

Pacheco, J. A. (2005). "A scatter search approach for the minimum sum-of-squares clustering problem". Computers and Operations Research. **32:** 1325-1335.

Petra, K. (2007). Clustering Genetic Algorithm. Proceedings of the 18th International Conference on Database and Expert Systems Applications, IEEE Computer Society.138-142

Prim, R. (1957). "Shortest connection matrix network and some generalizations." Bell Systems Technical Journal **36**: 1389–1401.

Procopiuc, C. M., M. Jones, P. K. Agarwal and T. M. Murali (2002). A Monte Carlo algorithm for fast projective clustering. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. Madison, Wisconsin, ACM.

Ribeiro, R. A. (1996). Fuzzy multiple attribute decision making: a review and new preference elicitation techniques. Fuzzy Sets and Systems. **78:** 155-181.

Ribeiro, R. A. (2006). "Fuzzy space monitoring and fault detection applications." Journal of Decision Systems. 2-3.

Rijsbergen, C. J. v. (1979). Information Retrival. London, Butterwoths.

Ross, T. J. (2004). Fuzzy Logic with Engineering Applications, John Wiley and Sons.

Russell, R. A. and W.-C. Chiang (2006). "Scatter search for the vehicle routing problem with time windows." European Journal of Operational Research **169**(2): 606-622.

Salvador, S. and P. Chan (2004). Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. Tools with Artificial Intelligence, 2004. ICTAI 2004.

Santos, B. R., T. Fonseca, M. Barata, R. A. Ribeiro and P. Sousa (2006). New Data preparation process - A case study for an ExoMars Drill. Proceedings of the World Automation Congress (WAC2006).

Santos, B. R., T. Fonseca, M. Barata, R. A. Ribeiro and P. Sousa (2008). "A method for automatic fuzzy set generation using sensor data." Autosoft-Intelligent Automation and Soft Computing International Journal.

Santos, B. R., T. Fonseca, M. Barata, R. A. Ribeiro and P. Sousa ((to appear 2008)). "A method for automatic fuzzy set generation using sensor data." Autosoft-Intelligent Automation and Soft Computing
International Journal.

Santos, B. R., G. Martins, M. Gomes and R. A. Ribeiro (2008). CCN for MODI - Simulation of Knowledge Enabled Monitoring and Diagnosis Tool for Mars Lander Payloads: Final Report, Uninova/CA3.

Setnes, M., R. Babuska, U. Kaymak and H. R. van Nauta Lemke (1998). "Similarity measures in fuzzy rule base simplification." Systems, Man and Cybernetics, Part B, IEEE Transactions on **28**(3): 376-386.

Shokri, Z. S. and K. Alsultan (1991). A simulated annealing algorithm for the clustering problem, Elsevier Science Inc. **24:** 1003-1008.

Shu-Jen, J. C. and C. L. Hwang (1992). Fuzzy Multiple Attribute Decision Making: Methods and Applications, Springer-Verlag New York, Inc.

Sneath, P. (1957). "The applications of computers to taxonomy." Journal of General Microbiology **17**: 201–226.

Sokal, R. and P. Sneath (1963). Principles of Numerical Taxonomy. San Francisco, W.H. Freeman.

Song, B. G., R. J. Marks, II, S. Oh, P. Arabshahi, T. P. Caudell and J. J. Choi (1993). "Adaptive membership function fusion and annihilation in fuzzy if-then rules". Fuzzy Systems.

Spath, H. (1980). Cluster analysis algorithms for data reduction and classification of objects. New York, Ellis Horwood.

Sung, C. and H. Jin (2000). "A tabu-search-based heuristic for clustering." Pattern Recognition **33**(5): 849–858.

Tapas, K., M. M. David, S. N. Nathan, D. P. Christine, S. Ruth and Y. W. Angela (2002). "An Efficient k-Means Clustering Algorithm: Analysis and Implementation", IEEE Computer Society. **24:** 881-892.

Ujjwal, M. and B. Sanghamitra (2002). "Performance Evaluation of Some Clustering Algorithms and Validity Indices", IEEE Computer Society. **24:** 1650-1654.

Ward Jr., J. (1963). "Hierarchical grouping to optimize an objective function." Journal of the American Statistical Association **58**(301): 236–244.

Ward Jr., J. and M. Hook (1963). "Application of a hierarchical grouping procedure to a problem of grouping profiles." Educational and Psychological Measurement **23**(1): 69–81.

Wirth, M., G. Estabrook and D. Rogers (1966). "A graph theory model for systematic biology, with an example for the oncidiinae (orchidaceae)." Systematic Zoology **15**(1): 59–69.

Wolsey, L. A. (1998). Integer Programming, Wiley-Interscience.

Yongguo, L., Y. Zhang, W. Hong, Y. Mao and C. Kefei (2008). "A tabu search approach for the minimum sum-of-squares clustering problem". Information Sciences. **178:** 2680-2704.

Zadeh, L. A. (1965). "Fuzzy Sets." Information and Control **8**: 338-353.

Zadeh, L. A. (1975). "The concept of a linguistic variable and its application to approximate reasoning--I." Information Sciences **8**(3): 199-249.

Zimmermann, H. J. (1990). Fuzzy set theory and its applications. Boston, Kluwer Academic Publishers.

APPENDIX

Resumo Alargado

1. Introdução

Uma variável linguística [Zadeh 1965] é composta por conjuntos vagos que podem ser matematicamente representados por funções de pertença. Por exemplo a variável linguística *Altura* pode ser composta pelos conjuntos vagos *Baixa*, *Média* e *Alta* (Figure 1.1). Em vez de um indivíduo pertencer apenas a um destes conjuntos como aconteceria com a lógica clássica, o grau de pertença de qualquer indivíduo a cada um destes conjuntos é dado pela respectiva função de pertença.

O objectivo desta tese era desenvolver algoritmos para reduzir o número de funções de pertença em variáveis linguísticas. Este problema é extremamente importante quando é utilizado um processo automático de criação de variáveis linguísticas, podendo-se assim obter uma variável linguística com um elevado número de funções de pertença. De uma forma mais geral, o problema que se coloca pode ser visto da seguinte maneira: como reduzir a quantidade de dados a analizar (aqui representados pelas diferentes funções de pertença) sem com isso perder informação? Esta é precisamente a mesma questão que nos é posta em problemas de agrupamento ou *clustering*. Assim, o problema da redução do número de funções pertença numa variável linguística foi aqui abordado como um problema de agrupamento. Começamos por identificar possíveis grupos de funções de pertença semelhantes. Funções de pertença pertencentes a um mesmo grupo são então agrupadas numa nova função de pertença, obtendo-se assim um novo conjunto mais pequeno de funções de pertença que representam aproximadamente o mesmo conceito que a variável linguística inicial.

Podemos considerar que nesta tese foram abordados dois grandes tipos de problemas. No primeiro o objectivo é a redução do número de funções de pertença em variáveis linguísticas que mais tarde poderiam vir a fazer parte de um qualquer modelo (não necessariamente um sistema de inferência) que seria construído já tendo em conta as características da variável linguística depois desta redução. No segundo, pelo contrário, o objectivo seria a redução do número de funções de pertença de variáveis linguísticas pertencentes a um sistema de inferência previamente construído, tendo em conta as características das variáveis linguísticas originais. Assim, neste caso, a variável linguística terá que ser encarada como parte de um sistema e o objectivo passa a ser obter um equilíbrio entre o desempenho do sistema e a sua simplificação por meio da redução do número de funções de pertença.

2. Conceitos Importantes

No Capítulo 1Chapter 1 é introduzida alguma informação sobre lógica difusa necessária para melhor compreender o contexto em que esta tese se insere, bem como alguma notação que será utilizada noutros capítulos. Apenas as ideias mais importantes são aqui referidas.

2.1. Representação de funções de pertença

Algumas famílias de funções de pertença podem ser mapeadas para R^p, em que p é o número de parâmetros dessa família de funções e cada dimensão representa um parâmetro diferente.

Por exemplo, para $p = 3$ uma função de pertença triangular pode ser representada por um triplo (a,b,c) (Figure 1.9). Se o triângulo for simétrico podemos tomar $p = 2$, representando a função de pertença por (a,ε), em que $\varepsilon = b - a = c - b$.

De modo semelhante, podemos representar uma função de pertença trapezoidal por um vector (a,b,c,d) contendo os seus vertices . No caso de esta ser simétrica, ou seja, $\frac{a+d}{2} = \frac{b+c}{2}$, podemos usar um triplo (m,ε,δ), em que $m = \frac{a+d}{2} = \frac{b+c}{2}$, $\varepsilon = c - b$ and $\delta = d - a$ (Figure 1.10).

Esta representação será usada para tratar o problema da redução do número de funções de pertença numa variável linguística como um problema de agrupamento tradicional.

2.2. Fundir Funções de Pertença

A ideia por trás dos algoritmos a utilizar é a fusão de grupos de funções de pertença semelhantes.

Dadas n funções de pertença trapezoidais, $T^i = (a_i, b_i, c_i, d_i)$, $i = 1, \ldots, n$, estas serão fundidas numa nova função de pertença $T = (a, b, c, d)$, $i = 1, \ldots, n$ usando uma generalização do método proposto em [Setnes, Babuska et al. 1998]:

$$a = \min_{i=1,\ldots,n} a_i \tag{1}$$

$$b = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{2}$$

$$c = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{3}$$

$$d = \max_{i=1,\ldots,n} d_i \tag{4}$$

As formulas para fundir um grupo de funções de pertença triangular vêm directamente das anteriores.

3. Métodos Exactos

Nesta tese são discutidas algumas formulações em programação inteira para este problema (secção 2.3). Embora nunca tenha sido o objectivo desta tese encontrar soluções óptimas para estes problemas usando métodos exactos como o Branch & Bound [Land and Doig 1960], uma destas formulações foi introduzida no CPLEX para dois conjuntos de problemas de pequena dimensão, um com apenas 12 funções de pertença em cada variável linguística e outro com 54. Enquanto que no primeiro conjunto de problemas foi possível encontrar soluções óptimas em menos de 2 minutos, no segundo conjunto de problemas já não foi possível encontrar soluções óptimas, tendo o programa parado por falta de memória. Estas experiências permitiram ter uma maior noção da dimensão e dificuldade deste tipo de problemas e justificaram a necessidade de recorrer a métodos heurísticos para encontrar boas soluções num espaço de tempo mais realista.

4. Métodos Heurísticos baseados em Pesquisa Local

Dada a ineficácia dos métodos exactos em encontrar a solução óptima para o agrupamento dos pontos num determinado conjunto X, foram explorados métodos heurísticos. Foi desenvolvida uma meta-heurística, Scatter Search, baseada em [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006] que foi posteriormente comparada com uma variação do algoritmo das K-Médias ou *K-Means* [McQueen 1967], denominada K-Means++ [David and Sergei 2007].

4.1. K-Means++

O algoritmo *K-Means* [McQueen 1967] começa por escolher aleatoriamente para centros dos clusters K pontos do conjunto de dados $X = \{x_1, \cdots, x_n\}$. Depois desta inicialização, determina-se uma partição dos dados em K grupos, afectando cada ponto ao grupo com centro mais próximo. A partir deste momento os centros dos grupos vão sendo actualizados e os pontos vão sendo afectados ao grupo mais próximo até que algum ser satisfeito algum critério de paragem.

O algoritmo K-Means++ [David and Sergei 2007] difere do algoritmo original apenas na maneira como os centros iniciais são escolhidos. Depois de o primeiro centro ser escolhido aleatoriamente e de forma uniforme, isto é, considerando iguais probabilidades de escolha para cada ponto de X, os restantes são escolhidos de acordo com probabilidades proporcionais à sua distância ao centro mais próximo, de entre os centros já escolhidos. Quanto mais longe um ponto se encontra dos centros já escolhidos, maior será a probabilidade de este ser escolhido. Desta forma pretende-se dispersar a distribuição dos centros iniciais para que o algoritmo convirja mais rapidamente.

Neste algoritmo o número de grupos a formar, K, é escolhido *a priori*. Quando não sabemos *a priori* o número de grupos a formar, corremos o algoritmo para várias escolhas de K e escolhemos a melhor configuração encontrada, tendo em conta um determinado índice para a qualidade dos agrupamentos. O índice usado nesta tese, discutido em [Ujjwal and Sanghamitra 2002], deve ser maximizado e é dado pelas expressões seguintes:

$$I(K) = \left(\frac{1}{K} \times \frac{E_1}{E_K} \times D_K \right)^p \tag{5}$$

em que
$$E_K = \sum_{k=1}^{K} \sum_{j=1}^{n} u_{kj} \left\| x_j - c_k \right\| \tag{6}$$

e
$$D_K = \max_{i,j=1,\cdots,K} \left\| c_i - c_j \right\| \tag{7}$$

sendo que $U = \lfloor u_{kj} \rfloor_{K \times n}$ é uma matrix binária representando uma partição dos dados em K grupos (i.e., $u_{kj} = 1$ se e só se x_j está no k-ésimo grupo) e o centro do grupo k é representado por c_k.

4.2. Scatter Search

O algoritmo Scatter Search (Figure 4.1), opera sobre um pequeno conjunto de referência, composto por boas soluções e por soluções com elevada diversificação (em relação às restantes). Um conjunto inicial de soluções é criado pelo Método de Geração de Diversificação (DG – Diversification Generation Method). Cada solução neste conjunto é melhorada pelo Método de Melhoria (Imp - Improvement Method) antes da criação do conjunto de referência pelo Método de Actualização do Conjunto de Referência (RSU – Reference Set Update Method), que escolhe para fazer parte deste conjunto as melhores soluções bem como soluções com elevado nível de diversificação. O Método de Geração de Subconjuntos (SG – Subset Generation Method) forma subconjuntos de soluções do conjunto de referência para serem combinados pelo Método de Combinação de Soluções (SC – Solution Combination Method) em novas soluções. A qualidade das soluções assim obtidas é mais uma vez melhorada pelo Método de Melhoria antes do conjunto de referência ser actualizado. O algoritmo continua até que algum critério de paragem seja satisfeito.

O algoritmo pode ser implementado de diversas maneiras de acordo com as estratégias adoptadas em cada um dos seus cinco métodos principais. As estratégias utilizadas nesta tese para cada um dos métodos são adaptadas de [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006] e resumidamente descritas de seguida. Como função de adaptação foi usado o índice I.

4.2.1 Método de Geração de Diversificação

Este método é responsável pela criação de um conjunto inicial de OS_{size} soluções.

Para cada solução, começamos por gerar aleatoriamente um número de grupos a formar, K, entre 1 e $K_{\max}$, sendo $K_{\max}$ o número máximo de grupos permitido (dado pelo utilizador). São escolhidos aleatoriamente K centros $S = \{c_1, \cdots, c_K\}$. No entanto, em vez de poderem ser escolhidos para centros quaisquer pontos de X, foi introduzido um parâmetro $\alpha \in [0,1]$ que controla o nível de aleatoriedade deste processo, determinando o conjunto de pontos que em cada passo podem ser escolhidos para centros, como proposto em [Pacheco 2005]. Para evitar a repetição na escolha dos centros das várias soluções criadas durante esta fase do algoritmo guardou-se a frequência com que casa ponto foi escolhido como centro, penalizando-se a escolha de pontos com elevada frequência. A penalização é controlada pelo parâmetro β.

Depois de terem sido escolhidos os centros dos grupos, os restantes pontos são atribuídos a estes grupos usando o processo heurístico *greedy* descrito em [Pacheco 2005], com o objectivo de minimizar a soma dos quadrados das distâncias de cada ponto ao centro do grupo a que pretence.

4.2.2 Método de Melhoria

Foi escolhido o método de melhoria apresentado em [Abdule-Wahab, Monmarché et al. 2006], baseado no algoritmo das K-Médias [Gan, Ma et al. 2007] e que utiliza a simplificação proposta por Spath [Spath 1980] para aproximar o incremento em termos de soma dos quadrados das distâncias de cada ponto ao centro do seu grupo resultante de mover o ponto x_i do grupo C_l para o grupo C_j. Em cada iteração deste método cada ponto de X é movido para o grupo que corresponde a um maior decréscimo nesta soma dos quadrados das distâncias. São feitas *MaxIterImp* iterações sempre que o método é utilizado.

4.2.3 Método de Actualização do Conjunto de Referência

Para construir o conjunto de referência, RS, começamos por escolher as melhores b_1 soluções, de entre as OS_{size} soluções criadas inicialmente. São depois adicionadas iterativamente b_2 soluções de acordo com a sua diversidade. As soluções escolhidas são as que maximizam

$$\delta_{\min}(\lambda) = \min\{dif(\lambda, \lambda') : \lambda' \in RS\} \tag{8}$$

em que $dif(\lambda, \lambda')$ é o número de pontos que são atribuídos a grupos diferentes nas soluções λ e λ'.

Nesta implementação o conjunto de referência é apenas actualizado quando são encontradas soluções de boa qualidade.

4.2.4 Método de Geração de Subconjuntos

Este método gera uma colecção de subconjuntos de soluções do conjunto de referência para serem posteriormente combinadas em novas soluções. Nesta implementação foram considerados todos os pares de soluções do conjunto de referência, isto é, são considerados $C_2^{b_1+b_2}$ pares de soluções.

4.2.5 Método de Combinação de Soluções

Para combinar um par de soluções numa ou mais novas soluções foi considerada uma estratégia do tipo path relinking, descrita em [Pacheco 2005]. A ideia deste tipo de estratégia é de que no “caminho” (série de movimentos simples que permitem alcançar uma solução a partir da outra) entre duas boas soluções deverão existir outras boas soluções. Neste caso um movimento corresponde a trocar um ponto de um grupo para outro. São propostas uma a três soluções escolhidas aleatoriamente neste caminho.

4.3. Resultados

Para ambos casos de estudo considerados, foi apresentada uma pequena análise dos parâmetros envolvidos no algoritmo *Scatter Search*. Apenas 5 experiências foram feitas para cada conjunto de valores dos parâmetros do algoritmo, pelo que os resultados não devem ser generalizados mas devem ser tidos em conta apenas a título indicativo. Em todas as experiências foi escolhido $K_{\max} = 100$, $b_1 = b_2$ e $OS_{size} = 10 \times (b_1 + b_2)$.

Foi possível ver a importância do parâmetro α no controlo da aleatoriedade do algoritmo, uma vez que para $\alpha = 0$ (escolha dos centros totalmente aleatória) os resultados finais apresentavam um elevado desvio padrão, não acompanhado de uma melhoria dos resultados em termos médios. O uso da memória durante a geração do conjunto de soluções iniciais mostrou-se positivo. Ao aumentar a dimensão do conjunto de referência de 4 $(b_1 = 2)$ para 10 $(b_1 = 5)$ conseguimos aumentar a qualidade das soluções com algum esforço computacional adicional. No entanto para ambos os casos de estudo este esforço adicional não foi considerado excessivo. Claro que, numa situação real, esta conclusão dependeria sempre do problema em concreto e do tempo disponível para realizar esta tarefa. O método de melhoria das soluções não melhorou significativamente a qualidade média das soluções para todas as variáveis linguísticas. Ao estudar a evolução do conjunto de referência verificou-se que a segunda parte do algoritmo não produziu boas soluções.

	K-Means++			Scatter Search		
	Tempo (seg.)	Nº Clusters	I	Tempo (seg.)	Nº Clusters	I
Radius	426,3904	4	18,9609	352,8277	3	26,9636
Texture	398,9221	3	16,8796	344,1798	5	30,18262
Perimeter	463,4574	7	830,33	371,4513	3	1286,17
Area	416,6392	6	426500,7	394,1495	3	668728,5
Smoothness	419,2019	3	0,000143	413,9573	4	0,0002044
Compactness	407,7729	3	0,004632	308,5648	3	0,004547
Concativity	418,1248	3	0,080241	305,7193	3	0,080673
Concave Points	405,9813	3	0,002273	470,995	3	0,002743
Symmetry	411,8899	3	0,000921	672,4182	3	0,000932
Fractal Dimension	413,8989	4	0,000167	679,9634	3	0,000255

Tabela 1: Caso de Estudo 1- K-Means++ vs Scatter Search (melhores resultados)

	K-Means++			Scatter Search		
	Time (sec.)	Nr. Clusters	I	Time (sec.)	Nr. Clusters	I
A2	854.5374	5	266.0796	821.3367	3	384.8384
A3	862.5715	4	90.18299	794.977	4	96.4486
A8	810.7072	6	105.4446	1277.734	6	94.76789
A11	735.1273	23	6.97E+28	1112.77	23	6.97E+28
A14	1134.509	6	346890.2	1812.912	4	402917.4
A15	1062.561	6	5.51E+09	2031.943	6	5.57E+09

Tabela 2: Caso de Estudo 2 – K-Means++ vs Scatter Search (melhores resultados)

Em termos médios, foi considerando $\alpha = 0.5$, $\alpha = 0.8$, $b_1 = 5$ e $\text{MaxIterImp} = 2$ que se obtiveram os melhores resultados. Os resultados apresentados na Tabela 1 foram obtidos com estes parâmetros. O algoritmo *Scatter Search* desenvolvido foi capaz de obter melhores resultados que o algoritmo K-Means++ para a maior parte das variáveis. Com ambos os algoritmos, foi possível reduzir significativamente o número de funções de pertença das variáveis linguísticas analisadas (Figure 4.24 - Figure 4.33 e Figure 4.52 - Figure 4.57).

5. Caso de Estudo: Um Sistema de Inferência *Fuzzy*

Como foi referido na Introdução, a natureza deste caso de estudo é diferente da dos casos de estudo do capítulo anterior. Neste caso de estudo o objectivo é a redução do número de funções de pertença em variáveis linguísticas pertencentes a um sistema de inferência previamente construído. Pretende-se reduzir a complexidade do sistema sem perder demasiado desempenho.

Este caso de estudo foi desenvolvido no CA3 – UNINOVA [CA3 2006] no âmbito do projecto "MODI- Simulation of a Knowledge Enabled Monitoring and Diagnosis Tool for ExoMars Pasteur Payloads" [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006; Santos, Martins et al. 2008] para a Agência Espacial Europeia [ESA 2008]. Foram construídos de forma automática dois sistemas de inferência: um para um sistema de alarme para a detecção de comportamentos anormais durante perfurações em Marte e outro para o reconhecimento da dureza do terreno a ser perfurado. Os resultados aqui apresentados utilizam somente o sistema de reconhecimento de terreno.

As variáveis linguísticas de entrada foram criadas automaticamente usando dados recolhidos por sensores durante a fase de aprendizagem [Santos, Fonseca et al. 2008]. Durante a fase de aprendizagem foram realizados furos para diferentes velocidades de translação e rotação em diversos tipos de terreno. Cada variável linguística representa um sensor diferente e cada função de pertença trapezoidal numa dada variável linguística corresponde a um diferente subcenário testado. O resultado da inferência é um dos tipos de terreno possível e o nível de certeza nessa classificação [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Martins et al. 2008].

5.1. Algoritmo

O algoritmo adoptado baseia-se no algoritmo proposto por [Setnes, Babuska et al. 1998], em que os conjuntos difusos mais semelhantes vão sendo fundidos de forma iterativa até que os restantes conjuntos sejam suficientemente distintos, o que é feito através da imposição de um limite mínimo para a semelhança entre dois conjuntos juntar, minS (Figure 5.4). Este algoritmo pode ser visto como um algoritmo de agrupamento hierárquico.

Viu-se que neste caso de estudo em que o sistema de inferência foi construído previamente seria importante ter em conta medidas de desempenho do sistema de inferência. Assim, em vez de se definir um valor para minS, corremos o algoritmo até todas as funções de pertença serem disjuntas, avaliando o desempenho do sistema de inferência actual, P(M), e comparando-o com o desempenho do melhor sistema encontrado até ao momento, P(BestM). O algoritmo devolve o sistema de inferência com melhor desempenho, de entre os sistemas gerados durante o algoritmo (Figure 5.8). O algoritmo foi definido para qualquer medida de desempenho para um sistema de inferência, P(.). Neste caso foi utilizada a seguinte medida de desempenho (a ser maximizada) :

$$F = \frac{2 \cdot P \cdot MCL}{P + MCL} \tag{9}$$

em que a Precisão (P) do sistema de inferência é o quociente entre o número de amostras bem classificadas sobre o total de amostras e o Nível de Certeza Média (MCL) é a média dos níveis de certeza para as amostras correctamente classificadas.

Se quisermos uma solução de compromisso entre o número de funções de pertença no sistema e o seu desempenho podemos combinar estes objectivos considerando

$$P(M) = \alpha F - \frac{1-\alpha}{n_0} n \tag{10}$$

em que $\alpha \in [0,1]$ é o peso dado a F , n_0 é o número inicial de funções de pertença e n é o número de funções de petença do sistema M a ser avaliado.

O algoritmo está ainda definido para uma medida de semelhança entre dois conjuntos difusos genérica. Neste caso foi usada a medida de semelhança de Jaccard com :

$$S_J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{11}$$

em que $|C| = \int_U \mu_C(x)\,dx$ e $\cap$ e $\cup$ representam a intersecção e a união de conjuntos difusos.

5.2. Resultados

Foram testados 6 tipos de terreno, 3 valores para a velocidade de rotação e 3 valores para a velocidade de translação da broca, obtendo-se assim 54 funções de pertença para cada uma das variáveis linguísticas que representam os diferentes sensores instalados na broca (Figure 5.9).

O algoritmo foi aplicado a cada uma das variáveis linguísticas de forma sequencial. Os resultados estão resumidos nas tabelas abaixo. Como se pode ver na Tabela 3 e pelos gráficos das variáveis linguísticas finais (Figure 5.19) foi possível reduzir de forma muito significativa o número de funções de pertença em praticamente todas as variáveis linguísticas. Foi também possível melhorar o desempenho do sistema de inferência, como se pode ver pela Tabela 4. Por exemplo, a Precisão do sistema (P), aumentou de 72.33% para 85.47%.

	Original	BestP
Rotation Current	54	3
Rotation Voltage	54	5
Rotation Speed	54	3
Thrust	54	17
Torque	54	46
Translational Voltage	54	3
Translational Current	54	3
Translational Speed	54	3
TOTAL	432	45

Tabela 3: Redução do número de funções de pertença

	Original	BestP
P	72.33%	85.47%
MCL	34.49%	44.00%
F	46.71%	58.09%
N	423	45

Tabela 4: Comparação dos sistemas de inferência

6. Conclusões

O objectivo desta tese era desenvolver algoritmos para reduzir o número de funções de pertença numa variável linguística. Este problema foi abordado como um problema de agrupamento.

Foi desenvolvida uma metaheurística Scatter Search para encontrar boas soluções para o problema. Usando dois casos de estudo, esta metaheurística foi comparada com o algoritmo K-Means++. Os resultados obtidos não foram os esperados. A segunda parte do algoritmo Scatter Search não conseguiu produzir boas soluções. No entanto, a primeira parte do algoritmo foi suficiente para obter melhores resultados que os resultados conseguidos com o K-Means++. Com ambos os métodos, foi possível reduzir significativamente o número de funções de pertença em cada variável linguística.

No último capítulo foi apresentado um caso de estudo em que as variáveis linguísticas faziam parte de um sistema de inferência construído de forma automática. Neste caso é importante ter em conta o desempenho do sistema de inferência durante o algoritmo de redução, usando medidas de desempenho adequadas. Os resultados obtidos foram bastante satisfatórios. Não só foi possível reduzir de forma bastante significativa o número de funções de pertença no sistema, mas também foi possível aumentar o seu desempenho.

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

REDUCING THE NUMBER OF MEMBERSHIP FUNCTIONS IN LINGUISTIC VARIABLES

Margarida Santos Mattos Marques Gomes

Dissertation presented at Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia in fulfilment of the requirements for the Masters degree in Mathematics and Applications, specialization in Actuarial Sciences, Statistics and Operations Research

Supervisor: Paula Alexandra da Costa Amaral Jorge
Co-supervisor: Rita Almeida Ribeiro

Lisboa
2009

Acknowledgments

I would like to thank Professor Rita Almeida Ribeiro for inviting me to work at CA3/Uninova in the context of the MODI project and for accepting to co-supervise this thesis. The work developed during this project was the basis of this thesis. Thank you for introducing me to the fuzzy world.

I want to express my gratitude to Professor Paula Amaral, for her excellent guidance and important contribution to overcome some of the difficulties encountered through this thesis.

To all my family and friends, especially my friends at CA3, thank you for the emotional support that gave me strength to finish the thesis.

Abstract

The purpose of this thesis was to develop algorithms to reduce the number of membership functions in a fuzzy linguistic variable. Groups of similar membership functions to be merged were found using clustering algorithms. By “summarizing” the information given by a similar group of membership functions into a new membership function we obtain a smaller set of membership functions representing the same concept as the initial linguistic variable.

The complexity of clustering problems makes it difficult for exact methods to solve them in practical time. Heuristic methods were therefore used to find good quality solutions. A Scatter Search clustering algorithm was implemented in Matlab and compared to a variation of the K-Means algorithm. Computational results on two data sets are discussed.

A case study with linguistic variables belonging to a fuzzy inference system automatically constructed from data collected by sensors while drilling in different scenarios is also studied. With these systems already constructed, the task was to reduce the number of membership functions in its linguistic variables without losing performance. A hierarchical clustering algorithm relying on performance measures for the inference system was implemented in Matlab. It was possible not only to simplify the inference system by reducing the number of membership functions in each linguistic variable but also to improve its performance.

Resumo

O objectivo desta tese era desenvolver algoritmos para reduzir o número de funções de pertença numa variável linguística. Foram usados algoritmos de agrupamento ou *clustering* para encontrar grupos de funções de pertença semelhantes. Concentrando a informação dada por um grupo de funções de pertença semelhantes numa nova função de pertença obtém-se um conjunto mais reduzido de funções de pertença que representam o mesmo conceito que a variável linguística original.

Dada a complexidade computacional dos problemas de agrupamento, métodos exactos para a resolução de problemas de programação inteira apenas conseguem encontrar uma solução óptima em tempo útil para pequenas instâncias. Assim, foram usados métodos heurísticos para encontrar boas soluções. Foi implementado em Matlab um algoritmo do tipo *Scatter Search* e este foi comparado com uma variante do algoritmo *K-Means*. São apresentados resultados computacionais para dois casos de estudo.

É também apresentado um caso de estudo em que as variáveis linguísticas pertencem a um sistema de inferência previamente construído a partir de dados recolhidos por sensores. O objectivo era reduzir o número de funções de pertença das suas variáveis linguísticas sem comprometer o desempenho do sistema. Foi implementado em Matlab um algoritmo de agrupamento hierárquico que tem em conta medidas de desempenho do sistema de inferência. Para além de ter sido possível simplificar o sistema, a redução do número de funções de pertença levou a um aumento do desempenho do próprio sistema, através da remoção de alguma redundância existente no sistema inicial.

Table of Contents

List of Figures

List of Tables

Introduction

In human reasoning many concepts are not crisp in the sense of being completely true or false, instead they can be interpreted in a more qualitative way. In everyday life we use concepts like tall, small, fast, slow, good, bad … that are difficult to translate numerically. Classical logic and inference have been insufficient to deal with these apparently vague concepts. Although humans reason with these concepts in a natural way on a daily basis, our search for scientific knowledge has lead us to address the problem of representing these concepts in a more systematic and precise way. As Engelbrecht [Engelbrecht 2002] states, "In a sense, fuzzy sets and logic allow the modelling of common sense".

Since 1965, when Zadeh first formalized the concept of fuzzy set [Zadeh 1965], the field of fuzzy logic and approximate reasoning has attracted the interest of the scientific community. Fuzzy set theory and fuzzy logic concepts have been applied in almost all fields, from decision making to engineering [Costa, Gloria et al. 1997; Ross 2004], from medicine [Adlassnig 1986] to pattern recognition and clustering [Nakashima, Ishibuchi et al. 1998].

In engineering, fuzzy logic has been used, for instance, in monitoring and classification applications [Isermann 1998; Ribeiro 2006]. The main goal when constructing a fuzzy monitoring system is to develop a fuzzy inference system (FIS) [Lee 1990a; Lee 1990b] to monitor certain variables and warn decisors (or an automatic system) when variables behaviour is not correct, so that they can intervene. For the development of monitoring systems, in general, a formal and precise mathematical understanding of the underlying process is usually needed. These mathematical models may become too complex to formalize or to implement, reducing the advantage of an automatic and independent system over a human expert. Once again, fuzzy knowledge can be used to overcome this problem, modelling complex systems by mimicking human thinking.

In decision making, for instance, the advantages of using fuzzy logic is even more evident. In many cases the processes behind a decision are too complex to be defined through a precise classical mathematical model and the underlying preferences and choices of decision makers have many uncertainties and are better represented through a fuzzy number. Although crisp decision models do exist, more and more papers and books propose the use of fuzzy sets and fuzzy models to deal with the underlying uncertainty [Anoop Kumar and Moskowitz 1991; Lai and Hwang 1994; Ribeiro 1996; Ross 2004].

The main idea when choosing a fuzzy model over a classical one is to obtain models that are less complex and easy to interpret. The trade off between interpretability and precision must be studied for each application. To achieve such interpretability, it is desirable that the linguistic variables in a fuzzy model [Zadeh 1975] are as intuitive as possible. This in addition to a search for computationally efficient models motivated the research of this master thesis. When linguistic variables are constructed directly from expert knowledge its interpretability is usually clearer. This is not the case when an automatic procedure is used to create the membership functions of a certain linguistic variable or when membership functions represent a single sample from a large data base. As an example consider a fuzzy set used to represent an agent preference between two alternatives and suppose the number of agents involved in the process to be modelled is considerably large.

The purpose of this thesis is to develop algorithms to reduce the number of membership functions in a linguistic variable. The problem of reducing the amount of data to be analysed, while maintaining as most information as possible from the original data, is not exclusive from fuzzy domains. Large crisp data sets often have to be clustered to become treatable [Hartigan 1975; Murtagh 1983; Everitt, Landau et al. 2001; Gan, Ma et al. 2007]. Clustering data corresponds to finding natural groups of data that represent similar objects. The same approach can be used to reduce the number of membership functions in linguistic variables. We start by identifying clusters of similar membership functions. If each cluster of membership functions can be “summarized” into a new membership function, we obtain a new and smaller set of membership functions that approximately represents the same concept as the initial linguistic variable. This will be the basic approach that will be developed during this thesis. The problem of reducing the number of membership functions in linguistic variables will be formulated as a clustering problem. Resulting clusters of membership functions will be merged in a way of “summarizing” the information contained in the original membership functions.

In Chapter 1 theoretical background that is needed to understand following development is presented. An introduction to fuzzy logic and fuzzy inference systems is described. Similarity measures and merging methods that will be used to reduce the number of membership functions in linguistic variables are also introduced.

Since, as stated before, the problem of reducing the number of membership functions in a linguistic variable can be stated as a clustering problem, Chapter 2 will present different approaches to the clustering problem in statistics and optimization and the state of the art. Also, some possible formulations to the clustering problem will be discussed.

The complexity of clustering problems makes it difficult for exact methods to solve them in practical time. Exact methods can only find an optimal solution in a reasonable amount of time for very small data sets, especially if the number of clusters is unknown. However, before deciding for heuristic methods, it is important to use exact methods to better understand the complexity of the problem at hands. Since it was never the purpose of this thesis to solve these problems through exact methods, Chapter 3 gives only a brief introduction to some of the exact methods used for combinatorial and integer programming.

When finding optimal solutions through exact procedures is too time consuming, it is still usually possible to find good quality solutions in a reasonable amount of time, using heuristic methods that take advantage of the problem structure to achieve good solutions (not necessarily optimal) in less computational time. Both a heuristic and a metaheuristic to solve the automatic clustering problem were implemented in Matlab. Chapter 4 describes these algorithms and presents computational results on two case studies. In both case studies several linguistic variables are pruned. These linguistic variables could later be used in a fuzzy inference system or any other fuzzy model. The model would be constructed taking into account the already clustered membership functions instead of the original ones.

Chapter 5 introduces another case study. This case study has different characteristics from those used in Chapter 5. In this case study linguistic variables belong to an already existing fuzzy inference system. Instead of using the algorithms from Chapter 4, a heuristic relying on measures of performance of the inference system is used. The work presented in this chapter was developed within the scope of project “MODI – Simulation of a Knowledge Enabled Monitoring and Diagnosis Tool for ExoMars Pasteur Payloads”[CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006], a CA3 – UNINOVA project for the European Space Agency [ESA] Aurora programme [ESA 2008]. In this project two inference systems were constructed: one for monitoring exploratory drilling processes and another capable of detecting the type of terrain being drilled. These systems were automatically constructed using data collected from sensors while drilling in different scenarios. With these systems already constructed, the task was to reduce the number of membership functions in its linguistic variables without losing performance. This project was on the origin of the development of the ideas presented in this thesis. The contribution to this project can also be found in [Gomes, Santos et al. 2008]. This paper summarizes the main results obtained when reducing the number of membership functions of MODI’s linguistic variables and was presented at the Eight International Conference on Application of Fuzzy Systems and Soft Computing (ICAFS-2008) in September 2008 in Helsinki, Finland.

Finally, Chapter 6 presents the conclusions of this thesis and some guidelines for future work.

Chapter 1. Preliminaries

In this Chapter we present the background on fuzzy set theory necessary to understand the results presented later.

Sections 1.1 and 1.2 introduce the main concepts of fuzzy logic and fuzzy inference systems. Formal definitions of the concepts of linguistic variable, fuzzy set, membership function and the most used operations on fuzzy sets are given. A description of the structure of a fuzzy inference system and of its underlying modules is also presented.

In section 1.3, analytical and R^p representations of some of the most common types of membership functions - triangular, trapezoidal and Gaussian membership functions – are introduced.

The notion of similarity or proximity between membership functions will be the main idea underneath the algorithms for reducing the number of membership functions in linguistic variables. Section 1.4 describes these concepts and presents the measures of proximity of fuzzy sets that will be used. After identifying the most similar membership functions, these will be merged to give rise to a new set of membership functions simultaneously as small and as representative of the original linguistic variable as possible. Section 1.5 presents some membership functions merging methods.

1.1 Fuzzy Logic

In crisp logic, if we want to categorize a group of individuals as tall, medium or small, we have to distribute those individuals into two disjoint sets, as in Figure 1.1, by a crisp rule. For instance, if the height of an individual is above 1.75m, the individual is tall, if the height is bellow 1.60m, the individual is short and otherwise the individual is medium.

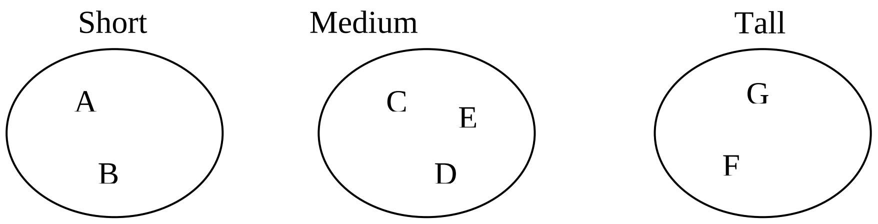


Figure 1.1: Concepts Short, Medium and Tall represented by Crisp Sets

This does not accurately represent human reasoning. In our mind, the frontier between these sets is not as well defined as in Figure 1.1. These concepts are better represented by fuzzy sets [Zadeh 1965], as in Figure 1.2. This representation allows for an individual to be considered simultaneously short and medium or medium and tall, with different degrees of membership. The definition of fuzzy set is given bellow.

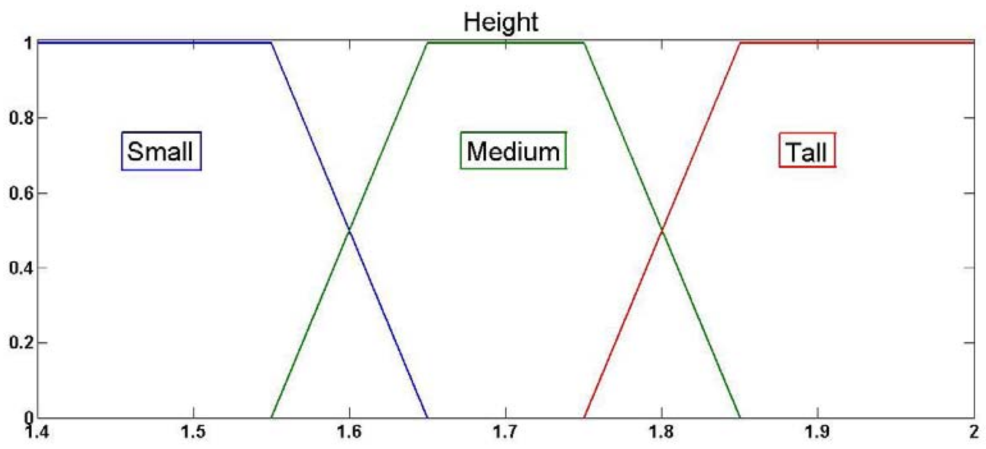


Figure 1.2: Concepts Short, Medium and Tall represented by Fuzzy Sets

Definition 1.1 [Zimmermann 1990] - If X is a collection of objects denoted generically by x then a fuzzy set $\tilde{A}$ in X is a set of ordered pairs:

$$\tilde{A} = \{(x, \mu_A(x)) : x \in X\} \tag{1.1}$$

where $\mu_{\tilde{A}}(x)$ is called the membership function or grade of membership of x in $\tilde{A}$ which maps X to the membership space M. The range of the membership function is a subset of nonnegative real numbers whose supremum is finite. Usually M is the real interval $[0,1]$.

♦

The representation of some common types of membership functions will be further presented in the next section.

Zadeh [Zadeh 1975] defines a linguistic variable as a quintuple $(x, T(x), U, G, M)$ in which x is the name of the variable; $T(x)$ is the term set of x, that is, the collection of its linguistic values; U is a universe of discourse; G is a syntactic rule which generates the terms in $T(x)$; and M is a semantic rule which associates with each linguistic value $T(x)$ its meaning, $M(X)$, where $M(X)$ denotes a subset of U.

The fuzzy sets in Figure 1.2 represent a linguistic variable Height.

T-norms and t-conorms generalize the idea of intersection and union of sets to fuzzy set theory.

Definition 1.2 [Klir and Yuan 1995] – A t-norm is a function $t:[0,1]\times[0,1]\to[0,1]$ satisfying the following properties:

Boundary Condition: $t(a,1)=a$ (1.2)

Monotonicity: $t(a,b)\le t(a,c) \quad \text{if} \quad b\le c$ (1.3)

Commutativity: $t(a,b)=t(b,a)$ (1.4)

Associativity: $t(a,t(b,c))=t(t(a,b),c)$ (1.5)

♦

Definition 1.3 [Klir and Yuan 1995] – A t-conorm or s-norm is a function $u:[0,1]\times[0,1]\to[0,1]$ satisfying the following conditions:

Boundary Condition: $u(a,0)=a$ (1.6)

Monotonicity: $u(a,b) \leq u(a,c) \quad \text{if} \quad b \leq c$ (1.7)

Commutativity: $u(a,b) = u(b,a)$ (1.8)

Associativity: $u(a,u(b,c)) = u(u(a,b),c)$ (1.9)

♦

The fuzzy minimum and the fuzzy maximum, defined bellow, are the most used t-norms and t-conorms. Examples of these operators can be found in Figure 1.3 and Figure 1.4, respectively.

Definition 1.4 [Klir and Yuan 1995] – Given two fuzzy sets A and B, their standard intersection, $A \cap B$, and standard union, $A \cup B$, also known as fuzzy minimum and fuzzy maximum, are defined for all $x \in X$ by the equations:

$$(A \cap B)(x) = \min[A(x), B(x)] \tag{1.10}$$

$$(A \cup B)(x) = \max[A(x), B(x)] \tag{1.11}$$

♦

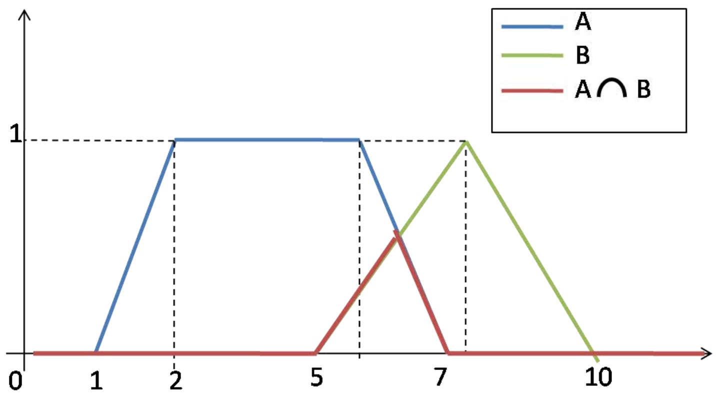


Figure 1.3: Fuzzy min

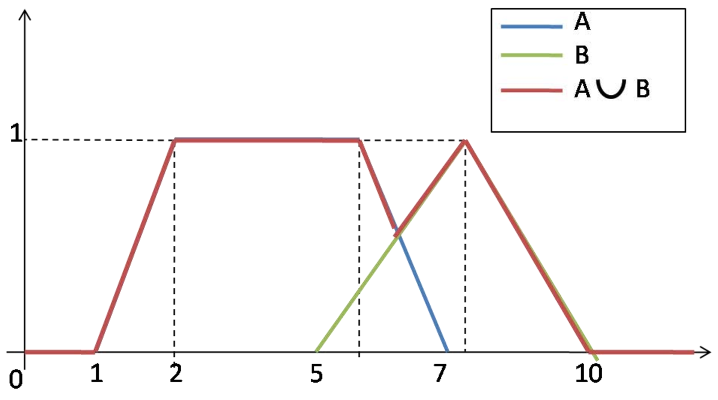


Figure 1.4: Fuzzy max

To generalize the concept of negation, complement operators are used. The membership function of a fuzzy set A represents, for each x in its universe of discourse, the degree to which x belongs to A. The membership functions of the complement of A represents the degree to which x does not belong to A.

Definition 1.5 [Klir and Yuan 1995] – A complement of a fuzzy set A is specified by a function $c:[0,1]\to[0,1]$ satisfying the following properties [Klir and Yuan 1995]:

Boundary Conditions: $c(0)=1; \quad c(1)=0$ (1.12)

Monotonicity: $c(a)\geq c(b) \quad \text{if} \quad a\leq b$ (1.13)

♦

The standard complement is defined bellow and exemplified in Figure 1.5.

Definition 1.6 [Klir and Yuan 1995] - The standard complement, $\overline{A}$, of a fuzzy set A with respect to the universal set X is defined for all $x\in X$ by the equation:

$$\overline{A}(x)=1-A(x) \tag{1.14}$$

♦

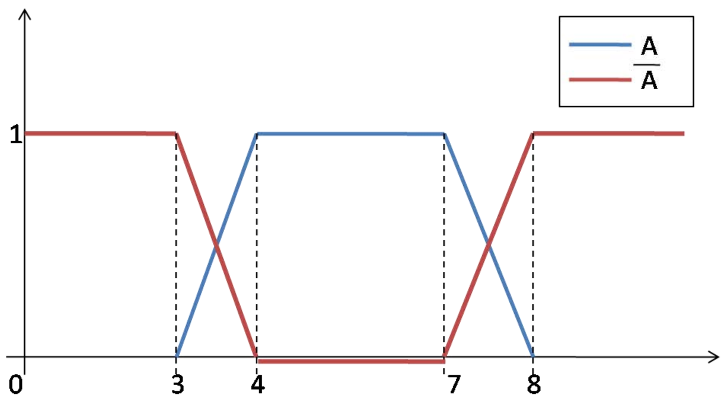


Figure 1.5: Standard fuzzy complement

1.2 Fuzzy Inference Systems

A fuzzy inference system is composed of fuzzy *if-then* rules relating different fuzzy sets, which are stored in a knowledge-base, and an inference engine that performs approximate reasoning [Ross 2004]. As mentioned before, one of the main advantages of inference systems [Ross 2004] is the ability to build models that mimic human reasoning and are relatively simple and easy to interpret. These models might be less accurate than classical and more formal ones but when dealing with real world applications interpretability, significance and computational efficiency can overcome some lack of accuracy, as depicted in Figure 1.6, taken from [Mathworks].

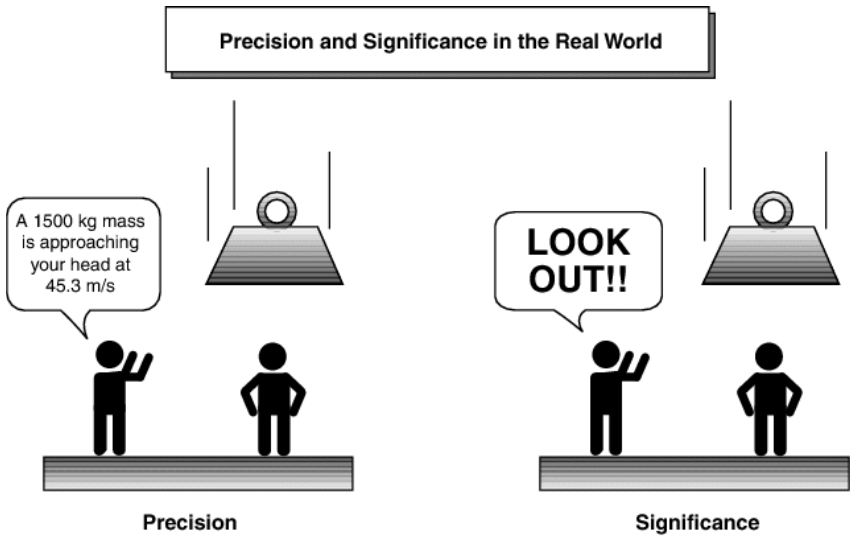

Precision and Significance in the Real World
A 1500 kg mass is approaching your head at 45.3 m/s
LOOK OUT!!
Precision
Significance

Figure 1.6: Precision vs. Significance in the Real World [Mathworks]

There are two main kinds of fuzzy inference systems, Mamdani and Sugeno [Lee 1990a; Lee 1990b]. The knowledge base of a Mamdani inference system contains rules where both the antecedents and the consequents are fuzzy sets. Sugeno inference systems, on the other hand, use rules with fuzzy antecedents and crisp consequents. In this thesis only Mamdani inference systems will be used but the ideas and algorithms developed can also be used in Sugeno inference systems.

Fuzzy if-then rules used in Mamdani inference systems are expressions of the type [Ross 2004]:

“if x is A then y is B”

where A and B are fuzzy sets, *“x is A”* is called the antecedent and *“y is B”* is called the consequent of the rule.

The antecedent part of the rule can have multiple parts connected by fuzzy operators, typically t-norms and t-conorms giving meaning to the linguistic expressions “and” and “or” respectively. The consequent can have multiple parts representing distinct conclusions that can be inferred from the given antecedent. The firing level or firing strength of the rule is the degree to which the antecedent part of the rule is satisfied.

To determine the outcome of fuzzy if-then rules given the crisp inputs, we need to fuzzify the inputs, apply the fuzzy operators that connect the multiple parts of the antecedent (if needed) to find the firing level of the rule and use an implication operator to apply the firing level to the consequent part (or parts) [Lee 1990a; Lee 1990b]. The output of the rule is a fuzzy set (or fuzzy sets). These concepts are better explained through an example. The following example in Figure 1.7 is taken from Matlab Fuzzy Logic Toolbox documentation [Mathworks].

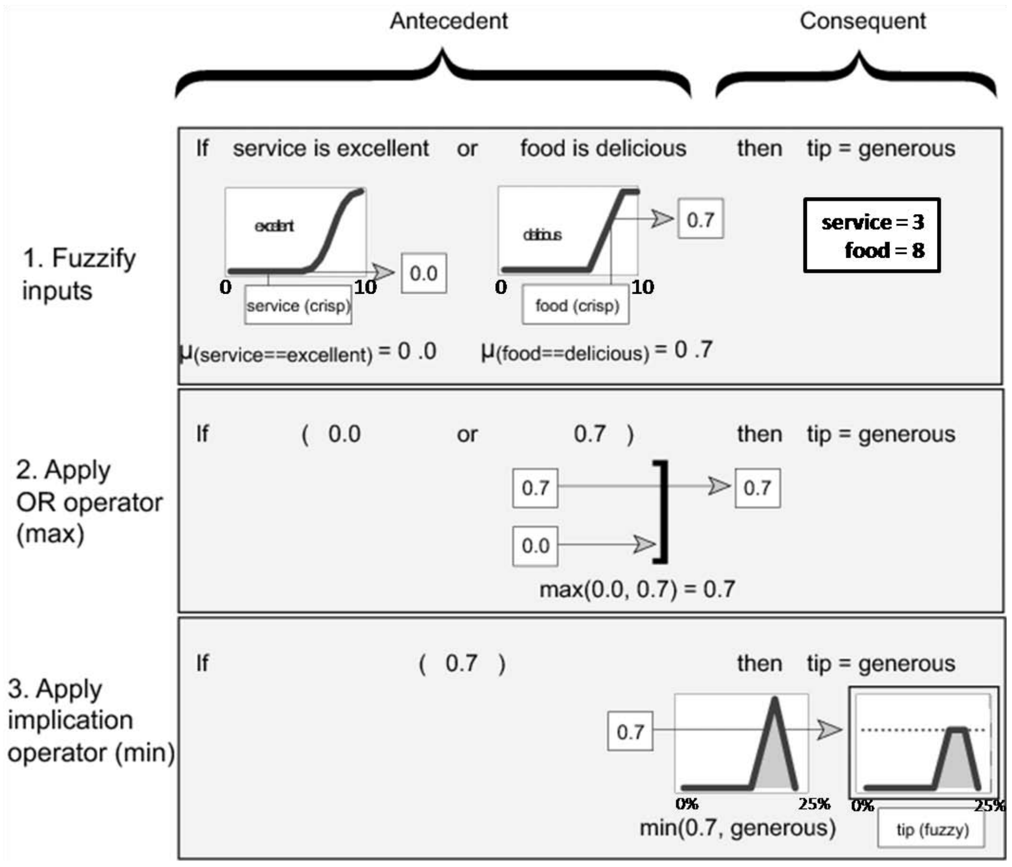

Antecedent
Consequent
If service is excellent or food is delicious then tip = generous
1. Fuzzify inputs
excellent
0
10
0.0
service (crisp)
delicious
0.7
food (crisp)
service = 3
food = 8
μ(service==excellent) = 0 .0
μ(food==delicious) = 0 .7
2. Apply OR operator (max)
If (0.0 or 0.7) then tip = generous
0.7
0.0
max(0.0, 0.7) = 0.7
3. Apply implication operator (min)
If (0.7) then tip = generous
0%
25%
min(0.7, generous)
tip (fuzzy)

Figure 1.7: Example of fuzzy if-then rule [Mathworks]

Given crisps values for the service and food quality the correspondent degrees of membership in the antecedent are computed and combined through the OR operator to give the rule firing level. For instance, if we consider service=3 and food=8, the degrees of membership in excellent (for service) and delicious (for food) are 0 and 0.7, respectively, and the firing level of the rule is given by $\max(0,0.7) = 0.7$. The implication operator is then applied taking into account this firing level to obtain the fuzzy set representing the output of the rule.

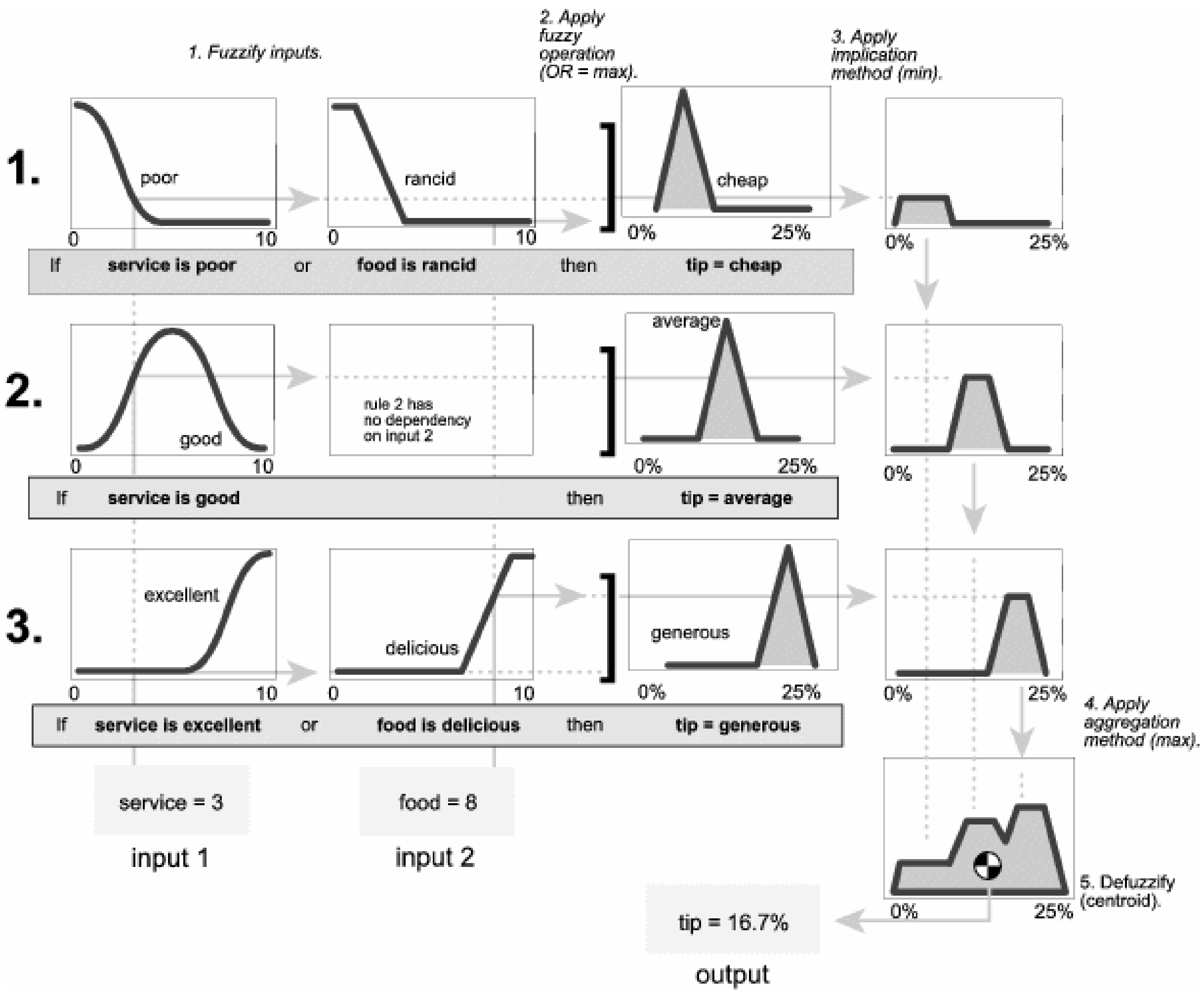

1. Fuzzify inputs.
2. Apply fuzzy operation (OR = max).
3. Apply implication method (min).
1.
poor
rancid
cheap
If service is poor or food is rancid then tip = cheap
2.
good
rule 2 has no dependency on input 2
average
If service is good then tip = average
3.
excellent
delicious
generous
If service is excellent or food is delicious then tip = generous
4. Apply aggregation method (max).
5. Defuzzify (centroid).
service = 3
input 1
food = 8
input 2
tip = 16.7%
output

Figure 1.8: Example of fuzzy inference system [Mathworks]

For each rule in the knowledge base the previously described steps are performed and the resulting fuzzy sets are aggregated through an appropriate operator (usually standard fuzzy maximum) to obtain a new fuzzy set representing the output of the system. This fuzzy set is then defuzzified to obtain a crisp value for the inference. Several defuzzification methods can be used, e.g. the centroid [Lee 1990a; Lee 1990b]. Continuing with the tipping example from Fuzzy Logic Toolbox documentation [Mathworks], Figure 1.8 shows a possible inference system with three rules and the necessary steps to determine the tip to be given crisp values for the service and food quality. In Figure 1.8, the three first fuzzy sets on the right represent the output of each rule after implication, using the same input values as before, i.e., service=3 and food=8. Aggregating these three fuzzy sets, the fuzzy set in the bottom right of Figure 1.8 is obtained. In this example the centre of area or centroid defuzzification method, defined by (1.15), is used and a tip of 16.7% is recommended.

Definition 1.7 [Klir and Yuan 1995] - Consider a fuzzy set A with membership function $\mu_A : X \to [0,1]$. The centre of area or centroid defuzzification method returns the value $d_{CA}(A)$ within X for which the area underneath the graph of membership function μ_A is divided into two equal subareas. This value is given by the following expression:

$$d_{CA}(A) = \frac{\int_X \mu_A(x) \cdot x \, dx}{\int_X \mu_A(x) \, dx} \tag{1.15}$$

♦

1.3 Representation of Membership Functions

Some types of membership functions can be mapped to R^p, where p is the number of parameters of that family of membership functions and each dimension represents a different parameter. In this section both analytical and R^p representations of some of the most common types of membership functions are presented.

1.3.1 Triangular Membership Functions

Definition 1.8 - A triangular membership function is given by the analytical expression:

$$\mu(a,b,c,x) = \begin{cases} \dfrac{x-a}{b-a} & , \quad a \le x \le b \\ \dfrac{c-x}{c-b} & , \quad b \le x \le c \\ 0 & , \quad \text{otherwise} \end{cases} \tag{1.16}$$

where a, b and c correspond to the x-axis coordinates of the vertices of the triangle, as in Figure 1.9.

♦

There are several possibilities for mapping these membership functions into R^p, $p = 2, 3$. For instance, for $p = 3$ we can consider a vector with the x-axis coordinates of the vertices of the triangle, (a,b,c), or a vector $(b, \varepsilon_L, \varepsilon_R)$ where $\varepsilon_L = b - a$ and $\varepsilon_R = c - b$ represent its left and right spreads, respectively. This way we define a mapping between the family of triangular membership functions and R^3. If we only consider symmetrical membership functions, i.e., if $\varepsilon_L = \varepsilon_R = \varepsilon$, we can use a pair (b, ε) to represent a membership function of this family. In this way the mapping can be done in R^2.

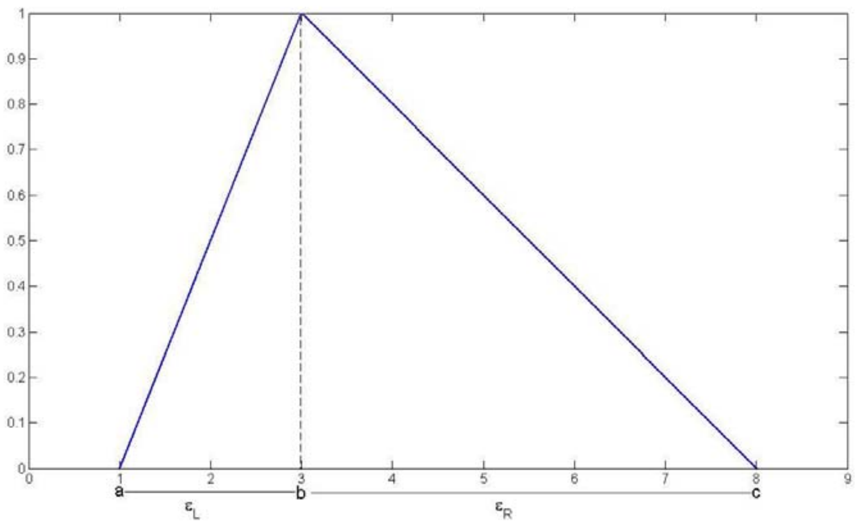


Figure 1.9: Triangular membership function $(a,b,c) = (1,3,8)$

1.3.2 Trapezoidal Membership Functions

Definition 1.9 - A trapezoidal membership function is given by the analytical expression:

$$\mu(a,b,c,d,x)=\begin{cases}1 & , \quad b\le x\le c\\ \dfrac{x-a}{b-a} & , \quad a\le x\le b\\ \dfrac{d-x}{d-c} & , \quad c\le x\le d\\ 0 & , \quad \text{otherwise}\end{cases} \tag{1.17}$$

where a,b,c and d correspond to the x-axis coordinates of the vertices of the trapezoid, as in Figure 1.10.

♦

Similarly to the case of triangular membership functions, we can now map the family of trapezoidal membership functions to R^4 and R^3 (symmetric trapezoidal). We can consider a vector with the x-axis coordinates of the vertices of the trapezoidal, (a,b,c,d), to map this family of membership functions to R^4 and if we only consider symmetrical membership functions, i.e., if $\frac{a+d}{2}=\frac{b+c}{2}$, we can use a vector (m,ε,δ), where $m=\frac{a+d}{2}=\frac{b+c}{2}$, $\varepsilon=c-b$ and $\delta=d-a$, to represent a membership function of this family.

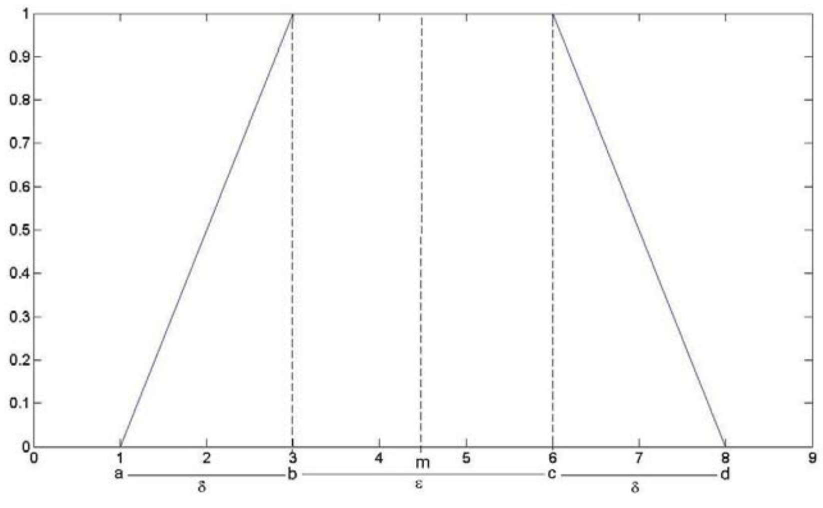


Figure 1.10: Symmetrical trapezoidal membership function $(a,b,c,d) = (1,3,6,8)$

1.3.3 Gaussian Membership Functions

Definition 1.10 – A Gaussian membership function is given by the analytical expression:

$$e^{-\sigma(x-\mu)^2} \tag{1.18}$$

where μ and σ are the mean and spread of the Gaussian function.

♦

The mapping of this family of membership functions to R^2 is straightforward and is given by the pair (μ, σ).

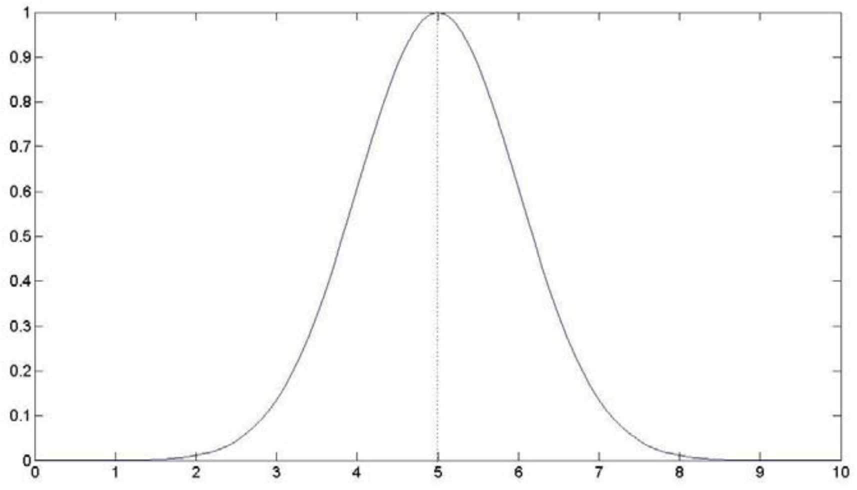


Figure 1.11: Gaussian membership function $(\mu,\sigma)=(5,1)$

1.4 Proximity Measures between Membership Functions

As stated in the introduction of this Chapter, the notion of similarity or proximity between membership functions will be the main idea underneath the algorithms for reducing the number of membership functions in linguistic variables. When faced with the problem of reducing the number of terms in linguistic variables, we intuitively think of joining or merging membership functions that are somehow similar. For crisp data sets, a similar idea is the foundation of cluster analysis. Clusters are groups of objects that are similar according to some proximity measure [Hartigan 1975]. The problem presented in this thesis can then be approached as a clustering problem where the objects are membership functions and suitable proximity measures are used.

In general, similarity measures between membership functions or fuzzy sets can be classified as geometric or set-theoretical [Miyamoto 1990]. Geometric measures are based on distance-measures and represent proximity between fuzzy sets. Set-theoretical similarity measures, based on operations such as union and intersection, translate the degree to which two fuzzy sets are equal and are not influenced by scaling and ordering of the domain.

One of the most used set-theoretical similarity measures, the fuzzy Jaccard index or Jaccard similarity measure [Miyamoto 1990], is defined by:

$$S_J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1.19}$$

where $|C| = \int_U \mu_C(x)\,dx$ and $(\cap, \cup)$ is a pair of fuzzy t-norms $(\cap)$ and t-conorms $(\cup)$.

An overview of some similarity measures for comparing fuzzy sets can be found in [Chen, Yeh et al. 1995]. The Jaccard similarity measure will be used in the algorithms presented in Chapter 5, but other similarity measures could also be used. For instance, since in Chapter 5 only trapezoidal membership functions are used, the following two similarity measures used for comparing trapezoidal fuzzy sets could be considered.

The first one can be calculated by the following expression [Chen 1996]:

$$S_C(A,B) = 1 - \frac{\sum_{i=1}^{4} |a_i - b_i|}{4} \tag{1.20}$$

where $A = (a_1, a_2, a_3, a_4)$ and $B = (b_1, b_2, b_3, b_4)$.

The second one was proposed by Shi-Jay Chen and Shyi-Ming Chen [Chen and Chen 2008] and is given by:

$$S_{SCGM}(A,B) = \left[1 - \frac{\sum_{i=1}^{4} |a_i - b_i|}{4} \right] \times (1 - |x_A^* - x_B^*|)^{B(S_A, S_B)} \times \frac{\min(y_A^*, y_B^*)}{\max(y_A^*, y_B^*)} \tag{1.21}$$

where

$$B(S_A, S_B) = \begin{cases} 1, & \text{if } S_A + S_B > 0 \\ 0, & \text{if } S_A + S_B = 0 \end{cases} \tag{1.22}$$

$$S_A = a_4 - a_1 \tag{1.23}$$

$$S_B = b_4 - b_1 \tag{1.24}$$

and (x_A^*, y_A^*) and (x_B^*, y_B^*) are the centre of gravity points of A and B, respectively. These points can be easily determined by the simple centre of gravity method (SCGM) [Chen and Chen 2008], using the following expressions:

$$y_A^* = \begin{cases} \dfrac{\dfrac{a_3 - a_2}{a_4 - a_1} + 2}{6}, if\ a_1 \neq a_4 \\ \dfrac{1}{2}, if\ a_1 = a_4 \end{cases} \tag{1.25}$$

$$x_A^* = \frac{y_A^*(a_3 + a_2) + (a_4 + a_1)(1 - y_A^*)}{2} \tag{1.26}$$

In the previous section it was shown how the most used families of membership functions can be mapped to R^p. By mapping a membership function to R^p the problem to be addressed becomes equivalent to finding clusters given a data set in R^p, provided that we are considering linguistic variables where all membership functions belong to the same family, which is usually the case. Therefore, the proximity measures used for comparing objects in R^p can also be used to compare membership functions of the same family. For instance, the Euclidean Distance given by (1.27) can be used to compare two membership functions of the same family, $A = (a_1, \ldots, a_p)$ and $B = (b_1, \ldots, b_p)$, represented in R^p. This will be done in the algorithms presented in Chapter 4 where the problem of reducing the number of membership functions in linguistic variables will be approached by clustering the vectors of parameters representing the membership functions.

$$D(A, B) = \sqrt{\sum_{i=1}^{p} (a_i - b_i)^2} \tag{1.27}$$

1.5 Merging Membership Functions

In this section we discuss some methods on how to merge membership functions to reduce the number of membership functions in a linguistic variable, by using the concept of similarity. This section is not intended as an overview of the possible methods for merging membership functions since these methods could vary according to several factors: the type of membership functions being merged, the algorithms in use, the context of the problem, among others.

Membership functions of the types referred in section 1.2 will be considered, since these are the most used ones. Also, throughout this thesis, it will be assumed that all membership functions of a certain linguistic variable to be pruned share the same type (either triangular or trapezoidal) and that the merging of two membership functions should yield a new membership function of the same type as the original ones. This simplification does not change the nature of the problem and the algorithms that will be use to solve it are as general as possible. If one of these conditions fails we only have to redefine the way two membership functions are merged but the algorithms still apply.

1.5.1 Merging Trapezoidal Membership Functions

Given two trapezoidal membership functions $A=(a_1,a_2,a_3,a_4)$ and $B=(b_1,b_2,b_3,b_4)$, merging them using the method proposed in [Setnes, Babuska et al. 1998] gives a new trapezoidal membership function $C=(c_1,c_2,c_3,c_4)$ where:

$$c_1 = \min(a_1, b_1) \tag{1.28}$$

$$c_2 = \lambda_2 a_2 + (1-\lambda_2) b_2 \tag{1.29}$$

$$c_3 = \lambda_3 a_3 + (1-\lambda_3) b_3 \tag{1.30}$$

$$c_4 = \max(a_4, b_4) \tag{1.31}$$

The parameters λ_2 and λ_3 belong to the interval $[0,1]$. These parameters allow weighting the importance of A and B in the final membership C. In subsequent chapters this operator will be used with $\lambda_2 = \lambda_3 = 0.5$. See for instance Figure 1.12, which shows the trapezoidal membership functions $A = (1,2,4,6)$ and $B = (2,3,5,7)$ combined into $C = (1,2.5,4.5,7)$. Notice that (1.28) and (1.31) guarantee that the same "coverage" as A and B, i.e., points with positive membership in either A or B will still have positive membership in C. This might be crucial for some applications.

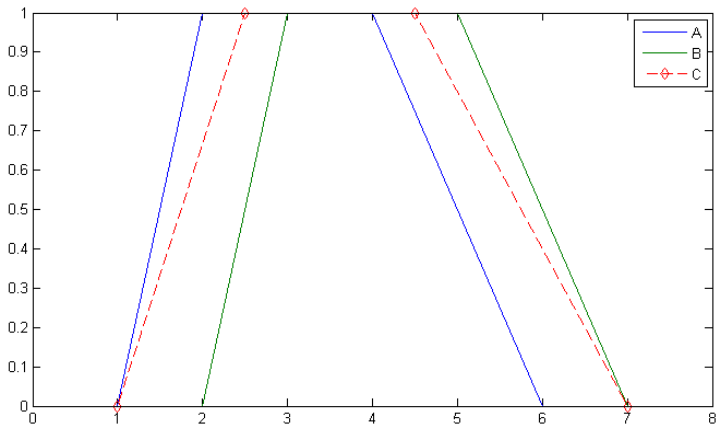


Figure 1.12: Merging trapezoidal membership functions $A = (1,2,4,6)$ and $B = (2,3,5,7)$ into $C = (1,2.5,4.5,7)$.

In the previous merging method only two membership functions are merged at a time. When merging more than two membership functions at a time, a generalization of this method was used. Given n trapezoidal membership functions $T^i = (a_i, b_i, c_i, d_i)$, $i = 1,\ldots,n$, these will be simultaneously merged into a membership function $T = (a,b,c,d)$, $i = 1,\ldots,n$ where:

$$a = \min_{i=1,\ldots,n} a_i \tag{1.32}$$

$$b = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{1.33}$$

$$c = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{1.34}$$

$$d = \max_{i=1,\ldots,n} d_i \tag{1.35}$$

1.5.2 Merging Triangular Membership Functions

It is straightforward to adapt the previous methodology to the case of triangular membership functions. Considering that a triangular membership function is a trapezoidal membership function with $b_i = c_i$ Given n triangular membership functions $S^i = (a_i, b_i, d_i)$, $i = 1,\ldots,n$, these will be simultaneously merged into a membership function $S = (a, b, d)$, $i = 1,\ldots,n$ where:

$$a = \min_{i=1,\ldots,n} a_i \tag{1.36}$$

$$b = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{1.37}$$

$$d = \max_{i=1,\ldots,n} d_i \tag{1.38}$$

1.5.3 Merging Gaussian Membership Functions

In [Song, Marks et al. 1993] the fusion of two Gaussian membership functions with parameters (μ_1, σ_1) and (μ_2, σ_2) is a Gaussian membership function with parameters (μ, σ) defined by the following equations. See for instance Figure 1.13.

$$\mu = \frac{\mu_1\sigma_1 + \mu_2\sigma_2}{\sigma_1 + \sigma_2} \tag{1.39}$$

$$\sigma^2 = \frac{\sigma_1^3 + \sigma_2^3}{\sigma_1 + \sigma_2} \tag{1.40}$$

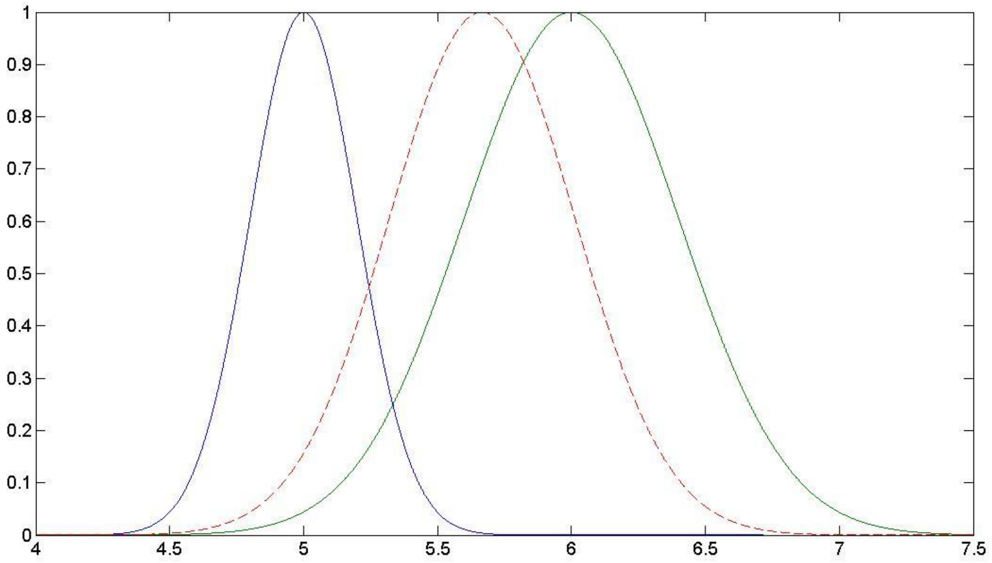


Figure 1.13: Merging Gaussian membership functions $(\mu_1,\sigma_1)=(5,0.2)$ and $(\mu_2,\sigma_2)=(6,0.4)$ into $(\mu,\sigma)\approx(5.667,0.3464)$

We can extend this method by defining the merge of n Gaussian membership functions with parameters $(\mu_i,\sigma_i),\quad i=1,\ldots,n$ as by the pair (μ,σ), where:

$$\mu = \frac{\sum_{i=1}^{n}\mu_i\sigma_i}{\sum_{i=1}^{n}\sigma_i} \tag{1.41}$$

$$\sigma^2 = \frac{\sum_{i=1}^{n}\sigma_i^3}{\sum_{i=1}^{n}\sigma_i} \tag{1.42}$$

1.6 Summary

In this Chapter the concepts of fuzzy set theory necessary to understand the work presented in this thesis were introduced. The concepts in sections 1.1 and 1.2

are the basic concepts of fuzzy logic and inference systems. Analytical and R^p representations of three of the most used families of membership functions are given in section 1.3. In this thesis it will be seen how to reduce the number of terms in a linguistic variable by merging similar membership functions. Sections 1.4 and 1.5 present the proximity measures between membership functions that will be used in later chapters and the methods for merging membership functions.

Chapter 2. A Clustering Problem Approach

As stated in the introduction, the problem of reducing the number of membership functions in linguistic variables can be formulated as a clustering problem. We need to identify groups of similar membership functions and merge them. As a result, we should obtain a smaller set of membership functions capable of approximately represent the initial linguistic variable.

In section 2.1 the clustering problem will be introduced. Section 2.2 will present the *state of the art* and finally in section 2.3 some integer programming formulations for the clustering problem will be given.

2.1 The Clustering Problem

There is no uniform formal definition for data clustering. The task of defining the meaning of clustering have been pointed out as a difficult one by several authors [Everitt, Landau et al. 2001; Estivill-Castro 2002]. In [Gan, Ma et al. 2007] the following informal definition can be found:

"Data clustering (or just clustering), also called cluster analysis, segmentation analysis, taxonomy analysis, or unsupervised classification, is a method of creating groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct."

As can be seen in Figure 2.1, two main types of clustering problems exist: hard clustering and fuzzy clustering [Gan, Ma et al. 2007]. In hard clustering problems an object or record has to belong to one and only one cluster, that is, a partition of the data into mutually exclusive groups is obtained. Fuzzy clustering problems on the other hand, allow an object to belong to several clusters, with different degrees of membership. In this thesis the problem of reducing the number of membership functions in a linguistic variable will be formulated as a hard clustering problem. Therefore the term *clustering* will be used instead of *hard clustering*. Approaching this problem as a fuzzy clustering problem by allowing membership functions to belong to more than one cluster and defining appropriate membership function merging techniques is a possibility to be studied in the future.

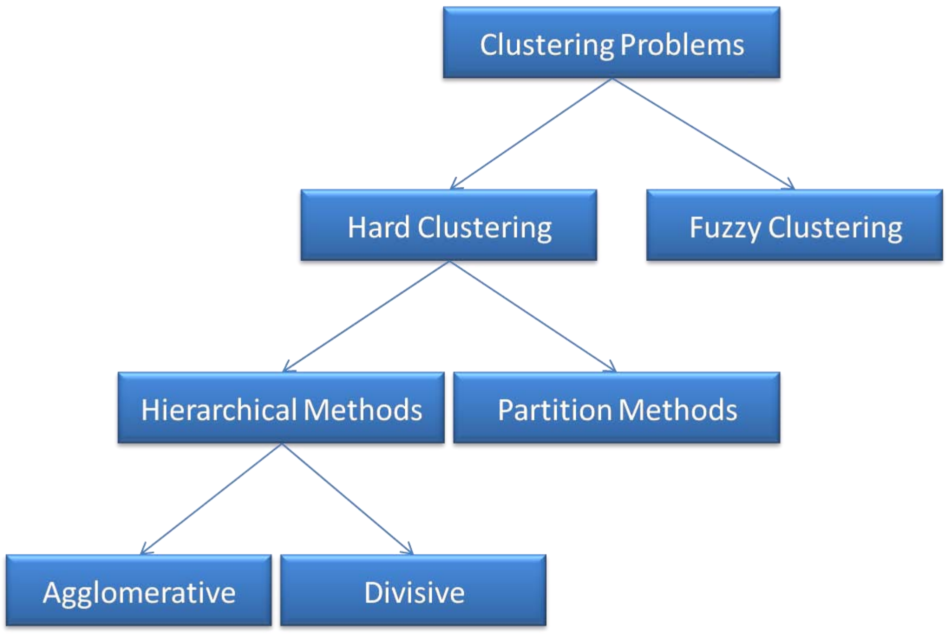


Figure 2.1: Diagram of Clustering Algorithms [Gan, Ma et al. 2007]

2.2 State of the Art

As stated in the previous section, there is no unique definition of clustering. Appropriate criteria for clustering have to be chosen for each application, according to the type of groups to be found in data. This partially explains the existing diversity of clustering algorithms [Estivill-Castro 2002]. Given this diversity, this *state of the art* will not be exhaustive in describing all the existing methods. Some more extensive reviews can be found in [Sokal and Sneath 1963; Hartigan 1975; Rijsbergen 1979; Jain and Dubes 1988; Kaufman and Rousseeuw 1990; Jain, Murty et al. 1999; Everitt, Landau et al. 2001; Engelbrecht 2002; Mirkin 2005; Gan, Ma et al. 2007].

As depicted in Figure 2.1, conventional (hard) clustering algorithms can be divided into two categories, according to the type of structures they return.

Hierarchical methods return a hierarchy or set of nested partitions while partition methods return a single partition of the data.

The next subsections will present the main ideas of hierarchical methods (section 2.2.1), classical partition methods (section 2.2.2), graph based methods (section 2.2.3), metaheuristics (section 2.2.4) and other clustering methods (section 2.2.5).

2.2.1 Hierarchical Methods

As stated before, hierarchical methods return a hierarchy or set of nested partitions, as depicted in Figure 2.2. Agglomerative hierarchical algorithms start with each data point in a different cluster and proceed by merging clusters, according to some criterion, until there is only one cluster containing all data points in the data set. Divisive hierarchical algorithms start with one cluster containing all data points and proceed by splitting clusters until each data point is in a different cluster.

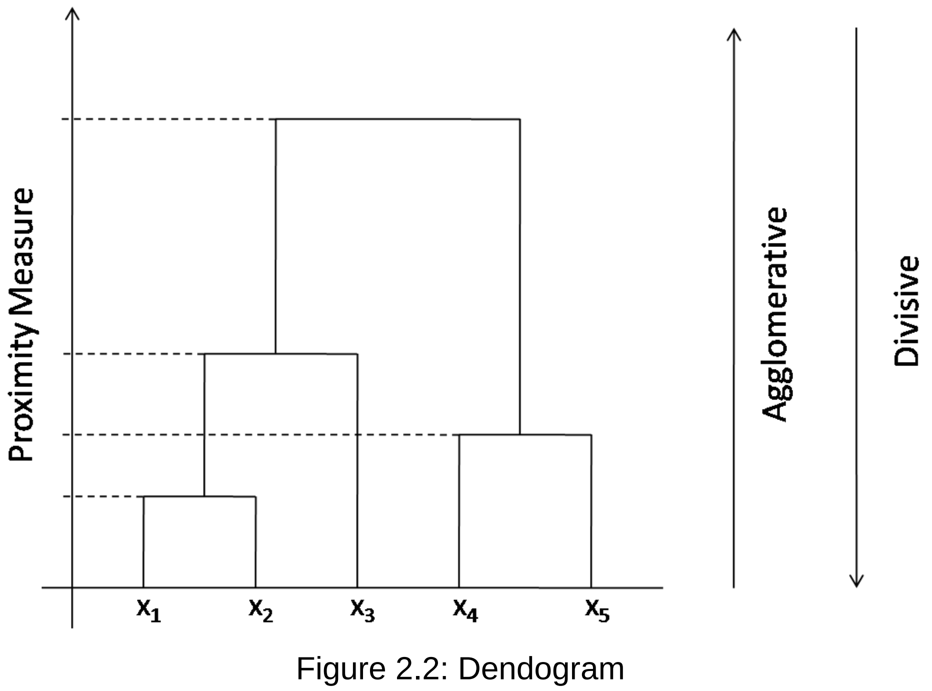


Figure 2.2: Dendogram

An hierarchical agglomerative clustering algorithm consists of the following steps [Jain, Murty et al. 1999]:

1. Compute the proximity matrix containing the distance between each pair of data points. Treat each data point as a cluster;
2. Find the most similar pair of clusters using the proximity matrix and merge them into one cluster;
3. Update the proximity matrix to reflect the merging operation in 2;
4. If all data points are in one cluster, stop. Otherwise, go to step 2.

Different algorithms can be developed according to the way the proximity measure is updated in step 3. The most used are the single-link, complete link and Ward's methods [Jain, Murty et al. 1999].

The single-link method, also known as nearest neighbour method and minimum method, was first introduced by [Florek, Lukaszewicz et al. 1951] and then independently by [McQuitty 1957] and [Sneath 1957]. Let C_1 and C_2 be two clusters and $d(\cdot,\cdot)$ a distance measure between two points. In the single-link method, the distance between C_1 and C_2, also referred to as linkage function, is given by:

$$D(C_1, C_2) = \min_{x \in C_1,\, y \in C_2} d(x, y) \tag{2.1}$$

The complete-link [King 1967], also known as farthest neighbour method, updates the proximity measure using the following expression, using the same notation as in (2.1).

$$D(C_1, C_2) = \max_{x \in C_1,\, y \in C_2} d(x, y) \tag{2.2}$$

The Ward's method [Ward Jr. 1963; Ward Jr. and Hook 1963], also known as minimum-variance method, aims at forming partitions $P_n, \cdots, P_1$ of the original data minimizing the loss of information, quantified in terms of the error sum of squares (ESS) criterion, associated with each merge. Consider a partition of the data into K clusters $C_1, \cdots, C_K$. The information loss is represented by:

$$ESS = \sum_{i=1}^{K} ESS(C_i) \tag{2.3}$$

where

$$ESS(C) = \sum_{\underline{x} \in C} \left(\underline{x} - \mu(C)\right)\left(\underline{x} - \mu(C)\right)^T \tag{2.4}$$

and

$$\mu(X) = \frac{1}{|C|} \sum_{\underline{x} \in X} \underline{x} \tag{2.5}$$

At each step of the Ward's method the two clusters whose fusion results in the minimum increase in loss of information are merged. The linkage function is computed as the increase in ESS after merging two clusters, i.e.:

$$D(C_1, C_2) = ESS(C_1 C_2) - ESS(C_1) - ESS(C_2) \tag{2.6}$$

where $C_1 C_2$ denotes the cluster resulting from merging C_1 and C_2.

Other linkage functions are described in [Hartigan 1975; Everitt, Landau et al. 2001; Gan, Ma et al. 2007]. In [Kuiper and Fisher 1975] a comparison of several hierarchical clustering algorithms is done using the Monte Carlo method.

As stated before, divisive hierarchical algorithms proceed the opposite way of the agglomerative algorithms. We start with one cluster containing all data points and proceed by splitting clusters until each data point is in a different cluster. Since given a cluster C there are $2^{|C|-1} - 1$ nontrivial ways of splitting it into two subclusters, it is not feasible to enumerate all the possible divisions of a cluster to find the optimal division, except for small clusters [Edwards and Cavalli-Sforza 1965]. Several divisive hierarchical clustering algorithms can therefore be designed considering different criteria for choosing the cluster to be split and different methods for splitting clusters. Examples of divisive algorithms can be found in [Edwards and Cavalli-Sforza 1965; Spath 1980; Kaufman and Rousseeuw 1990].

To illustrate divisive hierarchical clustering algorithms we will consider the DIANA (DIvisive ANAlysis) algorithm proposed by [Kaufman and Rousseeuw 1990].

For a given distance measure $d(\cdot,\cdot)$, the diameter of a cluster C is given by:

$$Diam(C) = \max_{x,y \in C} d(x,y) \tag{2.7}$$

Denote the average dissimilarity from a point x to the points in a set S by $D(x,S)$, i.e.,

$$D(x,S) = \frac{1}{|S|} \sum_{y \in S} d(x,y) \tag{2.8}$$

In each step of the DIANA algorithm, the cluster with largest diameter, C ($|C| \geq 2$), is split into two subclusters, A and B. These subclusters are determined by the following procedure:

1. Do $A = C$ and $B = \{\ \}$;
2. Do $z = \arg\max\{D(x, A \setminus \{x\}),\ x \in A\}$;
3. Move point z from A to B, i.e., $A \leftarrow A \setminus \{z\}$ and $B \leftarrow B \cup \{z\}$;
4. Do $z = \arg\max\{D(x, A \setminus \{x\}) - D(x,B),\ x \in A\}$;
5. If $D(z, A \setminus \{z\}) - D(z,B) > 0$ then move point z from A to B, i.e., $A \leftarrow A \setminus \{z\}$ and $B \leftarrow B \cup \{z\}$, and return to 4. Otherwise stop the procedure, returning A and B.

The procedure starts by considering $A = C$ and $B = \{\ \}$, i.e., all points belong to subcluster A. Then the point with highest dissimilarity is moved from subcluster A to B. The procedure continues by moving points from A to B whenever their average dissimilarity to B is smaller than the average dissimilarity to the rest of the points in A.

Generally, hierarchical methods have a complexity of $O(n^2)$ for memory space and $O(n^3)$ for CPU time [Hartigan 1975; Murtagh 1983], n being the number of points to be clustered. Therefore, they become impractical for large data sets.

2.2.2 Classical Partition Clustering Methods

Unlike hierarchical methods, partition methods create a single partition of the data points.

The most known partition method is the K-Means algorithm [McQueen 1967]. This algorithm is a centre-based method. Each cluster is represented by a centre and the corresponding clusters have convex shapes. The algorithm starts by choosing initial K cluster centres from the original data. After the initialization, a partition of the data is determined by assigning each point to the cluster with closest centre. After this assignment the centroids of each cluster are calculated according to the following expression:

$$c_i = \frac{1}{|C_i|} \sum_{\underline{x} \in C_i} \underline{x}, \quad i = 1, \cdots, K \tag{2.9}$$

where c_i is the centre of cluster C_i.

Then the points are reassigned to the clusters regarding the closeness to the centroids. Again, the centroids are recalculated and the algorithm proceeds in the same way until some stopping criterion is met. Usually the algorithm will proceed until the cluster centroid and partition no longer change or until a predefined number of iterations is reached. This way the K-Means algorithm is a heuristic method that tries to minimize the sum of squared distances from each point to its cluster centre. The number of clusters K is determined by the user *a priori.* In practice, if the user can not identify the correct number of clusters, the algorithm is run for a certain range for the number of clusters, i.e. $K \in \{K_{\min}, \cdots, K_{\max}\}$, and the best configuration found, according to some criterion, is chosen.

Many variations of the original K-Means algorithm have been developed. Some try to improve the efficiency of the algorithm by reducing the computational effort demanded by the algorithm [Tapas, David et al. 2002]. Others differ from the original algorithm in the way the initial cluster centres are chosen, as is the case of the algorithm presented in [David and Sergei 2007] called K-Means++ that will be further discussed in section 4.1. Some allow merging or splitting clusters according to centres distances or cluster within variance [Ball and Hall 1965].

Another widely used partition method is the Expectation Maximization Algorithm (EM) [Dempster, Laird et al. 1977], a model based clustering algorithm. In model based clustering it is assumed that the data comes from a certain mixture of distributions $f(x)=\sum_{k=1}^{K} p_k\, f(x,a_k)$ $\left(p_k \geq 0,\quad \sum_{k=1}^{K} p_k =1\right)$, with each component $f(x,a_k)$ representing a different cluster, where $f(x,a_k)$ is a family of density functions over x and a_k is the parameter vector that identifies a particular density from that family. Model based clustering algorithms try to optimize the fit between the data and the proposed model.

To estimate the individual cluster parameter the EM algorithm uses the maximum likelihood approach. The logarithm of the likelihood of the observed data given by (2.10) is maximized under the assumption that the data comes from a mixture of distributions.

$$L=\log\left\{\prod_{i=1}^{N}\sum_{k=1}^{K} p_k\, f(y_i,a_k)\right\} \tag{2.10}$$

Maximization of (2.10) can be reformulated as the maximization of (2.11).

$$L=\sum_{i=1}^{N}\sum_{k=1}^{K} g_{ik}\log p_k+\sum_{i=1}^{N}\sum_{k=1}^{K} g_{ik}\log f(y_i\, fa_k)-\sum_{i=1}^{N}\sum_{k=1}^{K} g_{ik}\log g_{ik} \tag{2.11}$$

where

$$g_{ik}=\frac{p_k\, f(y_i,a_k)}{\sum_{j=1}^{K} p_k\, f(y_j,a_k)} \tag{2.12}$$

and K and N are the number of clusters and data points, respectively.

The EM algorithm can then be summarized in the following way [Mirkin 2005]:

1. Start with any initial values of the parameters p_k, a_k and g_{ik}, $i = 1, \cdots, N$, $k = 1, \cdots, K$;
2. (E-step) Given p_k and a_k estimate g_{ik};
3. (M-step) Given g_{ik} find p_k and a_k maximizing (2.11);
4. Repeat steps 2 and 3 until there is no change in the parameter values (or the absolute difference is below some previously defined threshold).

2.2.3 Graph Based Methods

The relationship between graph theory and the clustering problem has been discussed by [Wirth, Estabrook et al. 1966; Jardine and Sibson 1968; Gower and Ross 1969; Hubert 1974; Hansen and Delattre 1978], among other authors. Algorithms that take advantage of the graph theoretical properties of data are called graph based methods.

The single-link and complete-link hierarchical methods discussed in section 2.2.1 can be approached from a graph theoretical view. More computationally efficient algorithms for single and complete link hierarchical methods than the ones already presented are described in [Gower and Ross 1969; Hansen and Delattre 1978; Jain and Dubes 1988].

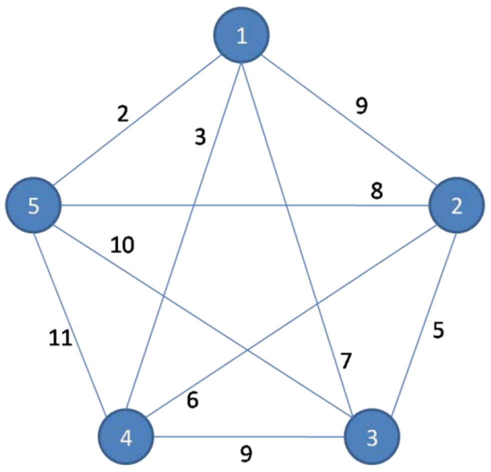
A minimum spanning tree (MST) of a connected, undirected, weighted graph is a subgraph that connects all its edges without cycles (tree) with minimum weight. Several methods for finding a minimum spanning tree of a graph have been developed [Kruskal 1956; Prim 1957]. In [Jain and Dubes 1988] the following algorithm for the single-link method using a minimum spanning tree is given, where the data is represented by a complete weighted graph $G = (V, E, W)$, V being the vertices of the graph representing the objects or data points to be clustered, E being the set of edges connecting all pairs of vertices and W being the weights of the edges representing the distance between two points:

1. Begin with each object in its own cluster and find the MST of G;
2. Merge the two clusters connected by the MST edge with smallest weight to define the next clustering;
3. Replace the weight of the edge selected in 2 by a weight larger than the largest proximity;
4. Repeat steps 2 and 3 until all objects are in one cluster.

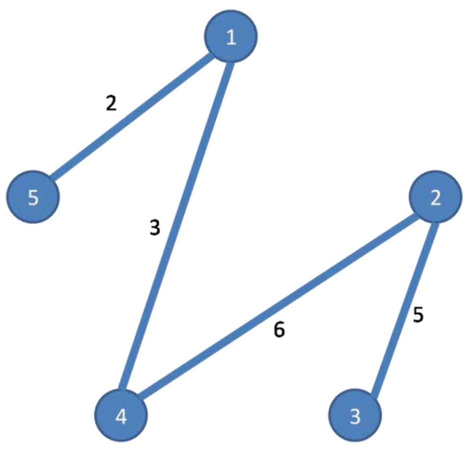
Figure 2.3 presents an example of this procedure. The information in the distance matrix D serves as a basis for the construction of the graph in Figure 2.3 (b). Figure 2.3 (c) depicts a possible minimum spanning tree for this graph. Merging the clusters corresponding to connected vertices in the MST from the smallest to the largest edge weight gives the dendogram in Figure 2.3 (d).

$$D = \begin{array}{ccccc|c} 1 & 2 & 3 & 4 & 5 & \\ \hline 0 & & & & & 1 \\ 9 & 0 & & & & 2 \\ 7 & 5 & 0 & & & 3 \\ 3 & 6 & 9 & 0 & & 4 \\ 2 & 8 & 10 & 11 & 0 & 5 \end{array}$$

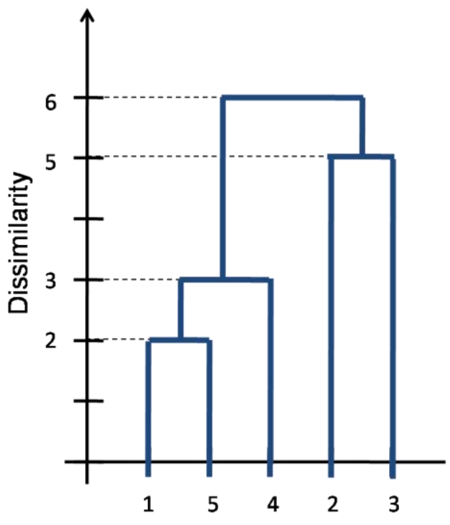
(a)



(b)



(c)



(d)

Figure 2.3: Example of single-link method using a MST: (a) distance matrix; (b) weighted graph; (c) MST; (d) Dendogram

Just as the single-link method can be approached using a minimum spanning tree, the complete-link method can be approached using node colouring theory [Hansen and Delattre 1978]. Other graph based methods for clustering data are reviewed in [Gan, Ma et al. 2007].

2.2.4 Metaheuristics

Heuristic approaches consist on a search strategy starting from a given feasible or unfeasible solution or set, an iterative process designed to favour the improvement of the solutions regarding feasibility and value and a stopping criterion. In [Colin 1993], the following definition of heuristic is given:

Definition 2.1 – A heuristic is a technique which seeks good (i.e. near-optimal) solution at a reasonable computational cost without being able to guarantee either feasibility or optimality, or even in many cases to state how close to optimality a particular feasible solution is.

♦

The most classical clustering methods in statistics and data mining, namely hierarchical clustering methods and partitioning methods, like K-Means [Gan, Ma et al. 2007], are heuristic. They take advantage of the problem structure to find good solutions but they cannot guarantee optimality.

The most basic heuristic methods may be trapped at local optima. Although it is possible that this local optimum is also the global optimum in general this will not be the case. To overcome this deficiency more sophisticated and elaborated heuristics incorporate techniques to increase the search space and escape local optima. With this purpose, in recent decades more algorithms that use information regarding the search process itself have been developed. These methods are designated as metaheuristics. In [Hillier and Lieberman 2005], the following definition of metaheuristics in given.

Definition 2.2 – A metaheuristic is a general kind of solution method that orchestrates the interaction between local improvement procedures and higher level strategies to create a process that is capable of escaping from local optima and performing a robust search of a feasible solution.

♦

Among the most well-known metaheuristics we have Simulated Annealing, Genetic Algorithms and Tabu Search.

Simulated Annealing, proposed by [Kirkpatrick, Gelatt et al. 1983], mimics the process of healing and cooling of material. At each iteration of the algorithm we move from the current solution to a neighbour solution, similarly to what happens in a descent heuristic for minimization. However, instead of moving always in the direction of improvement, worse solutions are accepted with a probability that depends on the magnitude of increase of the cost function (in a minimization problem) and on a parameter representing the temperature of the system. This parameter is decreased during the algorithm, simulating the cooling of material, until the temperature is close enough to zero. Following thermodynamics rules, at high temperatures the probability of accepting a randomly generated neighbor solution is higher. As the temperature decreases, this probability of acceptance also decreases. Application of the Simulated Annealing algorithm to the clustering problem can be found in [Brown and Huntley 1990; McErlean, Bell et al. 1990; Shokri and Alsultan 1991].

Genetic Algorithms [Holland 1975] are population based methods and are inspired in Charles Darwin theory of evolution. During the algorithm, a population consisting of a usually large set of solutions (chromosomes) is evolved through crossover and mutation operators. Pairs of solutions (parents) are chosen randomly to serve as input for the crossover operator that will generate one or more children. Fittest members are more likely to become parents, thus the next generation tends to be more fitted than the current one, following the natural selection and the principle of survival of the fittest. Additionally, with a typically small probability, mutation of one or more genes (variables) of a chromosome occurs. Through the natural selection process, at the end of the algorithm we expect a population of good quality solutions. Genetic Algorithms have been widely used on the clustering problem. A variety of papers on this subject have been published, for instance [Jiang and Ma 1996; Maulik and Bandyopadhyay 2000; Cheng, Lee et al. 2002; Gautam and Chaudhuri 2004; Jimenez, Cuevas et al. 2007; Petra 2007].

Unlike the two previous metaheuristics, Tabu Search [Glover 1986; Glover and Laguna 1997] is a deterministic process. The keyword in Tabu Search is “memory”. Tabu Search uses different structures of memory – long term and short term memory - to control the search process. In this way it is possible to avoid search cycles, conduct the search to domains of the solution space that would otherwise be skipped, concentrate the search around good quality solutions and avoid getting stuck at local optima. By concentrating the search around good solutions, usually called elite solutions, we are intensifying the search process. On the other hand, by moving to solutions somehow distant to the ones already visited, to avoid local optima, we are diversifying the search process. Efficiency of the Tabu Search Algorithm widely depends on a good balance between these two opposite strategies – intensification and diversification. Just as the previous metaheuristics, Tabu Search has also been applied to the clustering problem [Joyce and Michael 2000; Sung and Jin 2000; Yongguo, Zhang et al. 2008].

In this thesis a Scatter Search algorithm [Glover 1977] will be implemented. In a Scatter Search algorithm a reference set of both good quality and diverse solutions chosen from a larger original set of solutions is sequentially updated to produce better solutions. The algorithm implements both diversification and intensification search strategies to achieve a more intelligent search. Scatter Search algorithms were already applied to the clustering problem in [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006]. The scatter search algorithm that was implemented is based on the algorithms presented in these two papers. The algorithm is presented in detail in section 4.2.

2.2.5 Other Methods

Density-based or grid-based clustering methods are useful for finding arbitrarily shaped clusters consisting of denser regions than their surroundings in large multidimensional spaces. As pointed out in [Gan, Ma et al. 2007], “the grid-based clustering approach differs from the conventional clustering algorithms in that it is concerned not with the data points but with the value space that surrounds the data points”. The main idea of a density-based cluster is that for each point of a cluster the density of points in its ε-neighbourhood, for some $\varepsilon > 0$, has to exceed some threshold [Ester, Kriegel et al. 1996]. The most well-known density-based algorithm, proposed by [Ester, Kriegel et al. 1996], is called DBSCAN.

For high dimensional data it is hard to find good clusters using conventional clustering algorithms. Dimension reduction or feature selection techniques can be used before performing clustering, thus reducing the dimensionality of the data to be clustered. However, these approaches imply a loss of information and consequently the clusters obtained may not fully reflect the original structure of a given data set [Gan, Ma et al. 2007]. The goal of subspace clustering or projected clustering is to find clusters embedded in subspaces of the original data space with their own associated dimensions. The first subspace clustering algorithm, CLIQUE, was proposed by [Agrawal, Gehrke et al. 1998]. Other subspace clustering algorithms were proposed by [Agrawal, Gehrke et al. 1998; Aggarwal and Yu 2000; Procopiuc, Jones et al. 2002], among others. In this thesis we are clustering data points representing the parameters of membership functions belonging to a certain family of membership functions, typically Triangular, Trapezoidal or Gaussian membership functions. Since these families of membership functions can be described using a small number of parameters, the dimensionality of the data involved is low. Therefore, the methodology for subspace clustering will not be further described. Details on some of these algorithms can be found in [Gan, Ma et al. 2007].

2.3 Formulations in Integer Programming

In this section some formulations of the clustering problem to be solved are given. In these formulations only binary and integer variables will be used. The problem consists of clustering n fuzzy sets into k clusters, $1 \le k \le n$. The number of clusters is not known *a priori*. In all formulations d_{ij} denotes the distance between fuzzy sets i and j. If the fuzzy sets are represented in R^p, the Euclidean Distance defined by (1.27) or other distance for comparing objects in R^p can be used. It is also possible to use distance measures based on similarity measures for comparing fuzzy sets. The formulations presented are as general as possible and do not assume any particular distance measure.

2.3.1 A Binary Linear Programming Formulation - I

This first formulation is a linear programming formulation using only binary variables.

$$Min \quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^2 \sum_{k=1}^{n} y_{ijk} + \alpha \sum_{k=1}^{n} z_k \tag{2.13}$$

$s.t.$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad , \quad i \in \{1, \cdots, n\} \tag{2.14}$$

$$x_{ik} + x_{jk} - 1 \leq y_{ijk} \quad , \quad i, j, k \in \{1, \cdots, n\} \tag{2.15}$$

$$x_{ik} + x_{jk} \geq 2y_{ijk} \quad , \quad i, j, k \in \{1, \cdots, n\} \tag{2.16}$$

$$x_{ik} \geq y_{ijk} \quad , \quad i, j, k \in \{1, \cdots, n\} \tag{2.17}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} y_{ijk} \leq M\, z_k \quad , \quad k \in \{1, \cdots, n\} \tag{2.18}$$

$$x_{ik} y_{ijk}, z_k \in \{0, 1\} \quad , \quad i \in \{1, \cdots, n\}, \quad j \in \{1, \cdots, n\}, \quad k \in \{1, \cdots, n\} \tag{2.19}$$

As can be seen by the integrality conditions (2.19), x_{ik} , y_{ijk} and z_k are binary variables. Variable x_{ik} takes value 1 if and only if point i is in cluster k , y_{ijk} equals 1 if and only if points i and j belong to cluster k and z_k takes value 1 if and only if cluster k is not empty. Notice that $y_{ijk} = x_{ik} \cdot x_{jk}$ and $y_{iik} = x_{ik}$.

One of the most used criteria for clustering is to minimize the sum of squared distances (or equivalently the mean of squared distances) of data points belonging to the same cluster. However, if the number of clusters is not defined *a priori*, this yields an optimal solution where each data point forms a different cluster, with an optimal value of zero. Therefore the objective function has to account for the number of clusters formed. Since $\sum_{k=1}^{n} y_{ijk} = 1$ if points i and j belong to the same cluster and $\sum_{k=1}^{n} y_{ijk} = 0$ otherwise, $\frac{1}{n} \sum_{i=1}^{n} \sum_{j=i+1}^{n} d_{ij}^2 \sum_{k=1}^{n} y_{ijk}$ is the mean of squared distances of all pair of points belonging to the same cluster. The number of non-empty clusters is given by $\sum_{k=1}^{n} z_k$ and the parameter $\alpha > 0$ is not only used to control the importance given to both objectives – minimization of mean of squared distances and minimization of the number of clusters – but also to deal with the difference in scales present in the objective function.

Equations (2.14) ensure that each point belongs to exactly one cluster. Equations (2.15) translate that if point i belongs to cluster k ($x_{ik} = 1$) and point j belongs to cluster j ($x_{jk} = 1$), then both clusters belong to cluster k ($y_{ijk} = 1$). The reciprocal is ensured by minimization of the objective function but can also be expressed by equations (2.16) or by equations (2.17). Equations (2.18), where M is a large constant, allow identifying if the clusters are empty or not. If $y_{ijk} = 1$ for some i and j then $z_k = 1$, i.e., the cluster is not empty. Minimization of the objective function guarantees that $z_k = 0$ whenever cluster k is empty.

2.3.2 A Binary Linear Programming Formulation - II

This formulation is another linear programming formulation using only binary variables. In the previous formulation the x_{ik} variables are redundant, since $x_{ik} = y_{iik}$. Also, since $y_{ijk} = y_{jik} \quad \forall\, i, j \in \{1,\cdots,n\}$, it is possible to further reduce the number of variables in the formulation by considering only variables y_{ijk} for $i \in \{1,\cdots,n\}$ and $j \in \{i,\cdots,n\}$.

$$Min \quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^{2} \sum_{k=1}^{n} y_{ijk} + \alpha \sum_{k=1}^{n} z_k \tag{2.20}$$

s.t.

$$\sum_{k=1}^{n} y_{iik} = 1 \quad , \quad i \in \{1,\cdots,n\} \tag{2.21}$$

$$y_{iik} + y_{jjk} - 1 \le y_{ijk} \quad , \quad i \in \{1,\cdots,n\}, \quad j \in \{i+1,\cdots,n\}, \quad k \in \{1,\cdots,n\} \tag{2.22}$$

$$y_{ijk} \le z_k \quad , \quad i \in \{1,\cdots,n\}, \quad j \in \{i,\cdots,n\}, \quad k \in \{1,\cdots,n\} \tag{2.23}$$

$$y_{ijk}, z_k \in \{0,1\} \quad , \quad i \in \{1,\cdots,n\}, \quad j \in \{i,\cdots,n\}, \quad k \in \{1,\cdots,n\} \tag{2.24}$$

In this formulation y_{ijk} are binary variables, as can be seen by the integrality conditions (2.24), taking value 1 if and only if points i and j belong to cluster k and z_k is a binary variable that takes value 1 if and only if cluster k is not empty. Notice that $y_{iik} = 1$ if and only if point i belongs to cluster k.

The objective function in (2.20) was already explained in the previous formulation. Equations (2.21) ensure that each point belongs to exactly one cluster, as was the case for equations (2.14). Equations (2.22), similarly to equations (2.15), translate that if point i belongs to cluster k ($y_{iik} = 1$) and point j belongs to cluster j ($y_{jjk} = 1$), then both clusters belong to cluster k ($y_{ijk} = 1$). The reciprocal is ensured by minimization of the objective function. Equations (2.23) allow identifying if the clusters are empty or not. If $y_{ijk} = 1$ then $z_k = 1$, i.e., the cluster is not empty. Minimization of the objective function guarantees that $z_k = 0$ whenever cluster k is empty.

Additional valid inequalities, i.e., constraints that are satisfied by all admissible solutions, can be considered. The following inequalities are just some of the possible valid inequalities that can be used.

$$y_{iik} \geq y_{ijk} \quad , \quad i \in \{1, \cdots, n\}, \quad j \in \{i+1, \cdots, n\}, \quad k \in \{1, \cdots, n\} \tag{2.25}$$

$$y_{iik} + y_{jjk} \geq 2y_{ijk} \quad , \quad i \in \{1, \cdots, n\}, \quad j \in \{i+1, \cdots, n\}, \quad k \in \{1, \cdots, n\} \tag{2.26}$$

$$\sum_{i=1}^{n} \sum_{j=i}^{n} y_{ijk} \leq M\, z_k \quad , \quad k \in \{1, \cdots, n\} \tag{2.27}$$

Equations (2.25), similarly to equations (2.17) express that if both points i and j are in cluster k, then point i is in cluster k. Equations (2.26) can be immediately obtained from equations (2.25). Just like equations (2.23), Equations (2.27) allow to identify if the clusters are empty or not. These equations could replace equations (2.23), as in the case of the previous formulation.

2.3.3 A Formulation using precedence

In the previous formulation, we do not take advantage of the fact that membership functions have their domain in R. Consider the linguistic variable in Figure 2.4. In the previous formulation, membership functions A_1 and A_3 can belong to the same cluster even if A_2 does not belong to this cluster. Intuitively this should not happen. The space of admissible solutions can be reduced if we consider an ordering of the membership functions.

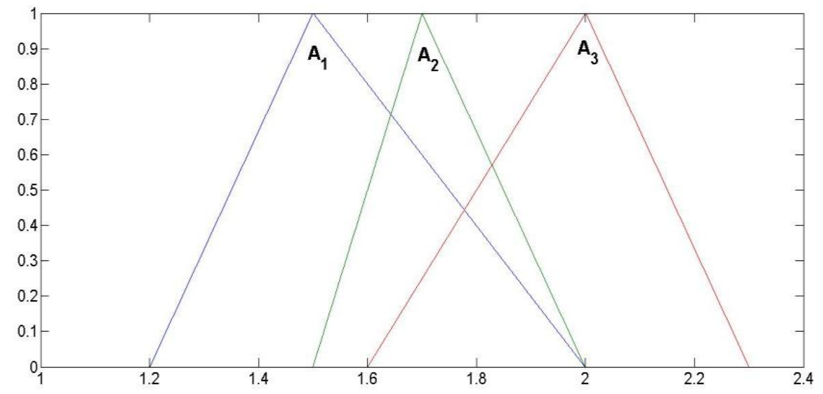


Figure 2.4: Example of a Linguistic Variable with three fuzzy sets

Consider that an ordering of the membership functions to be clustered $A_1 \preceq A_2 \preceq \cdots \preceq A_n$ exists. Then we can formulate the problem if the following way.

$$Min \quad \frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^2 x_{ij} + \alpha z_n \tag{2.28}$$

$s.t.$

$$z_1 = 1 \tag{2.29}$$

$$z_{i+1} - z_i \le 1 \quad , \quad i \in \{1, \cdots, n-1\} \tag{2.30}$$

$$z_{i+1} - z_i \ge 0 \quad , \quad i \in \{1, \cdots, n-1\} \tag{2.31}$$

$$z_j - z_i \geq 1 - x_{ij} \quad , \quad i \in \{1, \cdots, n-1\} \quad , \quad j \in \{i+1, \cdots, n\} \tag{2.32}$$

$$z_j - z_i \leq M(1 - x_{ij}) \quad , \quad i \in \{1, \cdots, n-1\} \quad , \quad j \in \{i+1, \cdots, n\} \tag{2.33}$$

$$z_i \in \{1, \cdots, n\} \quad , \quad i \in \{1, \cdots, n\} \tag{2.34}$$

$$x_{ij} \in \{0,1\} \quad , \quad i \in \{1, \cdots, n-1\} \quad , \quad j \in \{i+1, \cdots, n\} \tag{2.35}$$

where z_i is the number of the cluster that contains membership function A_i, $i \in \{1, \cdots, n\}$, x_{ij} is a binary variable that takes value 1 if and only if membership functions A_i and A_j belong to the same cluster, $i \in \{1, \cdots, n-1\}$, $j \in \{i+1, \cdots, n\}$, and M is an arbitrarily large constant.

The equality in (2.29) guarantees that the first membership function is always in the first cluster. Since $z_i, i \in \{1, \cdots, n\}$ are integers, inequalities (2.30) and (2.31) state that two consecutive membership functions A_i and A_{i+1}, $i \in \{1, \cdots, n\}$ are in the same cluster ($z_{i+1} = z_i$) or A_{i+1} is in the cluster immediately after the cluster that contains A_i ($z_{i+1} = z_i + 1$). Equations (2.32) and (2.33) make the correspondence between the two groups of variables. Membership functions A_i and A_j, $i, j \in \{1, \cdots, n\}$ belong to the same cluster ($x_{ij} = 1$) if and only if they have the same cluster number ($z_i = z_j$).

The objective function has the same meaning as the one in (2.20).

This formulation assumes that an ordering of the fuzzy sets exists. There are several methods for ordering fuzzy sets [Shu-Jen and Hwang 1992]. However, this ordering is not unique. It varies according to the method used. Therefore, an optimal solution to the previous formulation is only optimal for that particular ordering and not for the problem itself.

2.3.4 Quadratic Formulation

The previous formulations were all linear formulations. It is also possible to formulate this problem as a quadratic integer programming problem. Although the problem is easy to formulate with a quadratic objective function, quadratic problems are usually more difficult to solve then linear ones.

$$Min \quad \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n} d_{ij}^2 x_{ik} x_{jk} + \alpha \sum_{k=1}^{n} c_k \tag{2.36}$$

$s.t.$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad , \quad i \in \{1,\cdots,n\} \tag{2.37}$$

$$\sum_{i=1}^{n} x_{ik} \le Mc_k \quad , \quad k \in \{1,\cdots,n\} \tag{2.38}$$

$$\sum_{i=1}^{n} x_{ik} \ge c_k \quad , \quad k \in \{1,\cdots,n\} \tag{2.39}$$

$$x_{ik} \in \{0,1\} \quad , \quad i \in \{1,\cdots,n\} \quad , \quad j \in \{1,\cdots,n\} \tag{2.40}$$

$$c_k \in \{0,1\} \quad , \quad k \in \{1,\cdots,n\} \tag{2.41}$$

where x_{ij} is a binary variable that takes value 1 if and only if membership function i is in cluster k, $i,k \in \{1,\cdots,n\}$, c_k is a binary value that takes value 1 if and only if cluster k is not empty, $k \in \{1,\cdots,n\}$, and M is an arbitrarily large constant.

Equations (2.37) state that each membership function is in exactly one cluster. Equations (2.38) and (2.39) are equivalent to $\sum_{i=1}^{n} x_{ik} = 0 \Leftrightarrow c_k = 0$, $k \in \{1,\cdots,n\}$. By identifying if the clusters are empty or not it is possible to get the number of non-empty clusters to be used in the objective function.

2.4 Summary

The problem of reducing the number of membership functions in linguistic variables can be formulated as a clustering problem, as explained before. Therefore, this chapter started by introducing the clustering problem and the *state of the art* in this area (sections 2.1 and 2.2) and proceeded by discussing some integer programming formulations to the clustering problem (section 2.3).

Chapter 3. Exact Methods

The initial purpose of the work in this thesis was not to solve the reduction of membership functions through exact methods. The complexity of clustering problems is one of the main reasons why exact methods are in general not efficient and so finding an optimal solution in a reasonable amount of time will most likely only be possible for small data sets, particularly if the number of clusters is unknown. Nevertheless it seemed important to explain, even briefly, how to approach the problem if a global optimal solution is intended. Therefore, this chapter presents only a brief introduction to some of the exact methods used for combinatorial and integer programming.

Finding an optimal solution of a discrete optimization problem is in general difficult and known methods are not efficient for large instances. The complexity that characterizes these NP-Hard problems has the consequence that the computational implementation of exact algorithms is in general too heavy in terms of memory and too time-consuming for large problems. Partial enumeration methods, like Branch-and-Bound [Land and Doig 1960] or Branch-and-Cut [Wolsey 1998], are examples of such algorithms. The dimension of the instances above which is no more practical to apply an exact method varies according to the problems under study. This is one reason why exact methods should always be, at least, tested before switching to a heuristic approach. Cluster problems are among those problems for which a dimensionality above 40 variables makes the application of exact methods almost impractical [Lourenço 1995].

3.1 Branch-and-Bound

The Branch-and-Bound algorithm [Land and Doig 1960] is a *divide and conquer* technique that implicitly enumerates all feasible solutions of an integer (or mixed integer) linear programming problem. The three main aspects of this algorithm are the branching, fathoming or pruning and bounding strategies used. The original problem is divided into smaller problems by the branching strategy, usually represented by a solution tree. The bounding strategy tries to update the lower and upper bound on the optimal value of the objective function, L and U, by solving the linear relaxations of the integer problems considered, providing information that allows pruning some of the branches of the solution tree.

Consider the integer linear programming maximization problem defined by (3.1).

$$\begin{aligned} & Max \quad Z = cx \\ & s.t. \quad Ax \leq b \\ & \qquad x = (x_1, \cdots, x_n) \geq 0 \quad \text{and integers} \end{aligned} \tag{3.1}$$

and its linear relaxation

$$\begin{aligned} & Max \quad Z = cx \\ & s.t. \quad Ax \leq b \\ & \qquad x = (x_1, \cdots, x_n) \geq 0 \end{aligned} \tag{3.2}$$

To initialize the upper bound U the linear relaxation (3.2) at the root node is solved through a linear programming method. The lower bound is set to $L = -\infty$.

If the optimal solution x^* of the linear relaxation problem is integer, i.e., if $x_1, \cdots, x_n$ are integers, then this is also the optimal solution of the integer problem. Otherwise, a branching variable x_j is chosen among the basic variables that have non-integer values in this solution and two sub-problems are considered by adding the constraints $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lfloor x_j^* \rfloor + 1$ to (3.2), where $\lfloor a \rfloor$ stands for the largest integer smaller or equal to *a*.

The bounding strategy is applied for each new sub-problem. If an integer optimal solution for one of the sub-problems is found, we may try to update L because this solution is a feasible solution of the original problem. If z_{sub} denotes the objective function value of such solution we have $L = \max\{L, z_{sub}\}$.

The pruning strategy allows reducing the number of nodes in the solution tree that need to be explicitly visited. If a sub-problem satisfies one of the following conditions – pruning by optimality, pruning by bound or pruning by infeasibility

[Wolsey 1998] - the corresponding node will node be branched. These conditions are presented below:

1. Pruning by optimality – an integer optimal solution to the sub-problem was found;
2. Pruning by bound – $z_{sub} < L$, i.e., solutions found by branching this node will always be worse than a feasible known solution whose value is equal to the lower bound;
3. Pruning by infeasibility – the sub-problem (and thus all possible branches of this node) is infeasible.

The branching, bounding and pruning steps are iteratively applied to each sub-problem until there are no remaining non-pruned sub-problems or until $L = U = Z^*$. In this case either an optimal solution was found or the problem is infeasible. It is also common to stop the algorithm when the amplitude of the interval $[L,U]$ is small, where the concept small is given by considering an error measure and threshold for this error, but in this case optimality is not guaranteed.

3.2 Branch-and-Cut

The Branch-and-Cut algorithm [Wolsey 1998] is a hybrid of Branch-and-Bound and cutting plane algorithms. A cutting plane for an integer programming problem is a valid inequality, i.e., a constraint that is satisfied by all admissible solutions, that reduces the admissible region of the linear programming relaxation.

Several implementations of this algorithm exist. Basically, cutting planes are generated during the Branch-and-Bound algorithm. The goal is to find better bounds in each node in order to reduce the number of nodes to be visited. As stated in [Wolsey 1998], “though this may seem to be a minor difference, in practice there is a change of philosophy”. Instead of quickly solving the node problems, emphasis is given to improving the formulation at each node.

Other than generating cutting planes, additional strategies can be used to improve the formulation at each node. Some of these strategies consist of fixing variables to the only possible value that can take part in an optimal solution or eliminating redundant constraints. The efficiency of a Branch-and-Cut algorithm depends on a good implementation of such strategies. Knowing when to include or eliminate constraints is a major aspect of this algorithm. Although general implementations exist, to solve a specific (and more complex) problem an implementation that takes advantage of the underlying problem structure should be developed.

3.3 Branch-and-Price

The Branch-and-Price algorithm [Barnhart, Johnson et al. 1998] is another variation of Branch-and-Bound. Just like in Branch-and-Cut, emphasis is given to the strategies employed in each node to obtain better solutions or better bounds. However, instead of using cutting planes (row generation) to improve the formulations at each node, column generation methods are used.

This algorithm is especially suited for solving problems with a large number of variables. The basic idea is that in many problems most of the variables will have a zero value in the optimal solution. By using column generation, a master problem corresponding to the original problem but where only a subset of variables is considered can be more efficiently solved. To identify which columns should enter the master problem, subproblems based on the dual linear programming problem, called pricing problems, are solved. This allows choosing variables with positive (negative) reduced cost in the minimization (maximization) problem that should therefore enter the master problem. When no such variables exist and the integrality conditions are not satisfied, branching is performed as in the original Branch-and-Bound algorithm.

3.4 Computational Results

To better understand the dimension of the problem and the difficulty of using exact methods for clustering some computational experiments were done. These experiments were done using data from the case study that will be presented in

Chapter 5. The formulation presented in section 2.3.1 was implemented in GAMS and latter run on CPLEX. In these experiments the distance between two membership functions i and j, d_{ij}, was chosen to be $d_{ij} = 1 - s_{ij}$, where s_{ij} is the Jaccard Similarity given by (1.19) using the fuzzy minimum and fuzzy maximum, given by (1.10) and (1.11), as intersection and union operators.

All experiments were done in Pentium(R) 4 CPU 2.6 GHz, 504 MB of RAM.

First we considered linguistic variables with 12 membership functions each. The results are summarized in Table 3.1. Instead of running CPLEX until an optimal solution was found (and proved to be optimal) a threshold of 10% for the relative gap between the lower and upper bounds on the objective function was used as a stopping criterion. As can be seen in Table 3.1, CPLEX took less than 2 minutes – 40.41 seconds in average – to stop. Given these results we ran CPLEX for linguistic variables with 54 membership functions to see if exact methods could still be used to solve these problems in a reasonable amount of time. However, for these problems CPLEX stopped because lack of memory, without returning an optimal solution. These results show what was already expected by the combinatorial nature of clustering problems: exact methods can only deal with very small data sets.

	Solution	Best Possible	Absolute Gap	Relative Gap	Elapsed Time (sec.)	Number of Clusters
Rotation Current	0.767505	0.690776	0.076729	0.099972	95.063	8
Rotation Voltage	0.898916	0.809514	0.089402	0.099455	33.672	8
Rotation Speed	0.768401	0.694777	0.073624	0.095815	19.313	7
Thrust	0.939391	0.845645	0.093746	0.099794	41.172	10
Torque	0.75501	0.680006	0.075003	0.099341	45.922	7
Translational Voltage	0.58426	0.527426	0.056835	0.097276	30.219	5
Translational Current	0.555146	0.501321	0.053825	0.096956	26.531	4
Translational Speed	0.684707	0.616784	0.067923	0.099199	31.422	7

Table 3.1: Computational Results

3.5 Summary

In this Chapter some exact methods for solving integer problems were briefly described. These methods consist of a set of strategies to methodically examine the search space of an integer or mixed integer problem without having to implicitly enumerate all possible solutions. Even though these methods allow to optimally solve many problems that by explicit enumeration could not be solved in a reasonable amount of time, for a wide class of combinatorial problems the search for an optimal solution is still too time-consuming. When this is the case, heuristic methods such as the ones described in Chapter 4 can provide good quality solutions with less computational effort without guaranteeing optimality.

Chapter 4. Heuristic Methods Based on Local Search

In real applications, the dimension and complexity of combinatorial and integer problems and the need to find good solutions in useful time have lead to the development of algorithms that take advantage of the problem structure to achieve good solutions (not necessarily optimal). Computational implementations of these algorithms, contrary to exact methods, are quite efficient regarding time and memory. Whenever the application of global optimization methods is not advisable, it is still usually possible to find good quality solutions by using heuristic methods.

As was pointed out in section 2.2.4, the most classical clustering methods in statistics and data mining are heuristic and can therefore be trapped at local optima. For this reason, a variety of metaheuristics have been applied to the clustering problem.

In this thesis a Scatter Search algorithm was implemented. Although this metaheuristic is not as well-known as the metaheuristics described in section 2.2.4, it already proved to be efficient at finding good quality solutions for many problems. Scatter Search has been applied to find solutions to the nodes graph coloring problem [Jean-Philippe and Jin-Kao 2002], to vehicle routing problems [Russell and Chiang 2006], to clustering problems [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006], among many other applications.

Both a heuristic and a metaheuristic to solve the automatic clustering problem were implemented in Matlab. Section 4.1 describes a heuristic approach called K-Means++ [David and Sergei 2007]. Section 4.2 describes the general Scatter Search algorithm and the details of this particular implementation. In Section 4.3 computational results on two case studies are presented in order to compare these two implementations.

4.1 A heuristic approach: K-means ++

One of the most used algorithms for clustering data is the K-Means algorithm [McQueen 1967], already described in section 2.2.2. The algorithm starts by choosing initial K cluster centres from the original data X. After the initialization, a partition of the data is determined by assigning each point to the cluster with closest centre. After this assignment the centroids of each cluster are calculated and the points are reassigned to the clusters regarding the closeness to the centroids. Again the centroids are recalculated and the algorithm proceeds in the same way until some stopping criterion is met.

A variation of this method, called K-Means++ [David and Sergei 2007], was implemented in Matlab for finding a feasible solution of the clustering problem. This method differs from the original K-Means algorithm in the way the initial clusters are chosen. Sections 4.1.1 and 4.1.2 describe the K-Means++ algorithm used for partitioning n points into K clusters. Section 4.1.3 discusses how to choose the correct number of clusters by evaluation of cluster validity indexes.

4.1.1 Initialization

The K-Means algorithm starts by choosing K cluster centres from the original data to be clustered. Usually the cluster centres are chosen uniformly at random from the original data, i.e., they are chosen with equal probabilities. The K-Means++ [David and Sergei 2007] differs from the original K-Means algorithm in the way the initial cluster centres are chosen. Cluster centres are still chosen randomly, but they are not chosen uniformly. After the first cluster centre is chosen uniformly at random from the original data, the remaining $K-1$ centres are chosen proportionally to their distance to the centres already chosen, a method referred in [David and Sergei 2007] by "D^2 weighting". The empirical reasoning of this rule is to diversify the location centres within the set of points to allow for a better assignment of points to clusters.

Let $D(x)$ denote the shortest distance from a point $x \in X$ to the closest centre already chosen. In this work we used the Euclidean distance and so $D(x)$ is given by:

$$D(x) = \min_{i \in S} \{ d(x, c_i) \} \tag{4.1}$$

where S is the set of all already chosen cluster centres and $d(\cdot,\cdot)$ is the Euclidean distance.

Then the necessary steps to perform the cluster centres initialization are the following:

1. Choose an initial centre uniformly at random from X, i.e., $p_i = \frac{1}{n}$, $i = 1, \cdots, n$, where p_i denotes the probability of choosing x_i.
2. Choose the next centre randomly according to the probability distribution $p_i = \frac{D(x_i)^2}{\sum_{j=1}^{n} D(x_j)^2}$, $i = 1, \cdots, n$.
3. Repeat step 2 until K centres have been chosen.

We should notice that the probability of choosing a point is proportional to the distance to the closest already chosen centre. So the further away the point is the likely it is that it will be chosen as a new centre. After the K initial cluster centres are chosen, the algorithm proceeds as the original K-Means. The iterative procedure is described in the next section.

4.1.2 Iterative Procedure

Now that the initial cluster centres are chosen, the remaining points are assigned to its closest cluster and the cluster centroids are updated. This procedure is repeated until a stopping criterion is met. The steps of this procedure, after the initialization phase, are summarized below.

1. Assign each point to the closest centre.
2. Update cluster centres by recalculating the cluster centres according to (2.9).
3. Repeat steps 1 and 2 until the centres no longer change.

In addition, a maximum number of iterations could be used as a stopping criterion. However, due to the rapid convergence of the algorithm, in the computational experiments that will be presented in section 4.3, it was not necessary to prematurely stop the algorithm.

4.1.3 Choosing the number of clusters

The previous algorithm partitions n data points into K clusters (or less, because empty cluster might be formed). However, when the number of clusters is not known *a priori*, the correct number of clusters has to be estimated. Usually this is done by running the algorithm for a range of values for K and choosing the best partition according to some cluster validity index.

In [Ujjwal and Sanghamitra 2002] several validity indexes for clustering algorithms are compared using different clustering algorithms. The experiments conducted by the authors lead them to the following conclusion: “Compared to the other considered validity indexes, I is found to be more consistent and reliable in indicating the correct number of clusters”. This index is defined by equation (4.2).

$$I(K) = \left(\frac{1}{K} \times \frac{E_1}{E_K} \times D_K \right)^p \tag{4.2}$$

where
$$E_K = \sum_{k=1}^{K} \sum_{j=1}^{n} u_{kj} \left\| x_j - c_k \right\| \tag{4.3}$$

and
$$D_K = \max_{i,j=1,\cdots,K} \left\| c_i - c_j \right\| \tag{4.4}$$

In these equations $X = \{x_1, \cdots, x_n\}$ is the data to be clustered and $U = \lfloor u_{kj} \rfloor_{K \times n}$ is a partition binary matrix representing a possible clustering of the data into K disjoint clusters, i.e., $u_{kj} = 1$ if and only if x_j is in the k^th cluster. The centroid of cluster k is denoted by c_k. To find the correct number of clusters we chose the value of K which maximizes $I(K)$.

Analyzing $I(K)$ we see that as the error E_K decreases, the factor $\frac{E_1}{E_K}$ increases. It is always possible to obtain a partition with zero error by considering n clusters, each consisting of a single data point. To balance the error with the number of cluster the factor $\frac{1}{K}$ is introduced. As the number of clusters decreases, this factor increases. To achieve well separated clusters, the factor D_K should be large, that is, the maximum distance between two cluster centres should be large. The previous considerations intuitively justify that $I(K)$ should be maximized.

4.2 Scatter Search

Scatter Search [Glover 1977] has some similarities to Tabu Search and Genetic Algorithms. The use of memory is one of the main features of Tabu Search and is usually present in Scatter Search. Such as Genetic Algorithms, Scatter Search is an evolutionary algorithm. While in Genetic Algorithms an usually large population is evolved through crossover and mutation operations, in Scatter Search instead of a population it is used a smaller reference set (composed of good quality solutions and diverse solutions) and it plays the most important role in the algorithm.

Essentially, Scatter Search operates on a small set of solutions, the reference set, and consists on the application of the following methods, which can be implemented in different ways, according to the problem at hand:

1. A Diversification Generation (DG) method to produce a collection of diverse trial solutions from which the initial reference set is built;
2. An Improvement (Imp) method to enhance the quality of trial solutions;
3. A Reference Set Update (RSU) method responsible for constructing a reference set of both high quality solutions and diverse solutions from the collection of solutions obtained by the diversification generation method and of updating this reference set when new solutions are created during the algorithm;

4. A Subset Generation (SG) method that, in each iteration of the algorithm, creates a collection of subsets of solutions belonging to the reference set, such that the solutions in each subset are to be combined through the solution combination method;
5. A Solution Combination (SC) method that takes a subset of solutions given by the subset generation method and generates one or more new trial solutions.

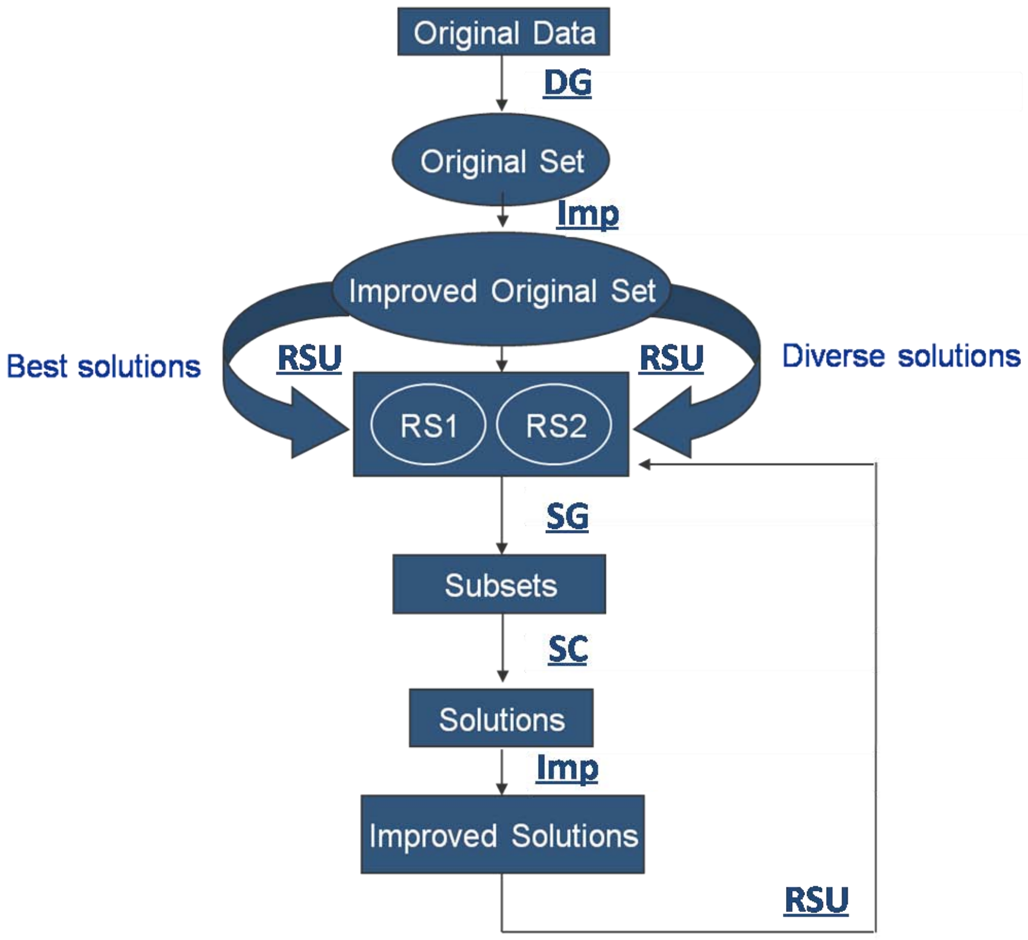


Figure 4.1: Scatter Search Algorithm

The way the previous methods operate is summarized in Figure 4.1. The original set is created by the Diversification Generation Method (DG). Each solution in this set is then improved by the Improvement Method (Imp) before the Reference Set Update Method (RSU) constructs the reference set, selecting the best quality solutions in the original set, along with diverse solutions. The subsets of solutions to be combined through the Solution Combination Method (SC) are chosen by the Subset Generation Method (SG). The solutions resulting from this combination are improved before the Reference Set Update Method (RSU) updates the reference set. The process continues until some stopping criterion is met.

The scatter search algorithm that was implemented is based on the algorithms in [Pacheco 2005] and [Abdule-Wahab, Monmarché et al. 2006]. The next sections describe the implementations of each of the five methods mentioned above.

4.2.1 Fitness Function

In [Pacheco 2005], quality of solutions was measured by the sum of squared distances from each point to the centroid of its cluster. This measure cannot be used in the automatic clustering problem, where the number of clusters is not defined *a priori.* Using this measure when the number of clusters can vary, yields the construction of as many clusters as the number of points to be clustered, giving a sum of the squared distances from each point to the centroid of its cluster (the point itself) of zero. Therefore, a different quality measure was used. The validity index I described in section 4.1.3 and defined by equations (4.2) - (4.4) was used as fitness function, to be maximized.

4.2.2 Diversification Generation Method

The diversification generation method used was proposed by Pacheco [Pacheco 2005], based on GRASP – Greedy Randomized Adaptive Search Procedure. However in this work the number of clusters is predefined by the user. To achieve an automatic clustering procedure, as it is aimed in our work, the correct number of clusters should be determined by the algorithm. Therefore, before creating a new solution with the diversification generation method, it is necessary to generate a random number of clusters, i.e., an integer K between 1 and $K_{\max}$, where $K_{\max}$ is the maximum number of clusters allowed.

Given the number of clusters K to be considered for the generation of a solution, the cluster centres $S = \{c_1, \cdots, c_K\}$ are randomly chosen from the data set X in the following way [Pacheco 2005]:

1. Find x_{j^*}, the farthest point from the centroid of X and do $c_1 = x_{j^*}$ and $S = \{c_1\}$. Set $h = 2$.
2. Fix $0 \leq \alpha \leq 1$ and while $|S| < K$ do:
    a. Determine $\Delta_j = \min\{\|x_j - c_l\| : c_l \in S\}, \forall x_j \in X \setminus S$
    b. Determine $\Delta_{\max} = \max\{\Delta_j : x_j \in X \setminus S\}$
    c. Do $L = \{x_j : \Delta_j \geq \alpha \Delta_{\max}\}$
    d. Choose $x_{j^*} \in L$ uniformly at random and do $c_h = x_{j^*}$, $S = S \cup \{c_h\}$ and $h = h + 1$.

If $\alpha = 0$, the cluster centres are chosen completely at random from the original data X. If $\alpha = 1$ the process is deterministic if there is only one point in L, and so the only farthest point from the centres already chosen will enter S. Therefore, generally speaking, the parameter α controls the level of randomization of the process.

A memory structure is used to avoid repetition of centres and consequently of solutions. The number of times that a point x_j is selected as a centre is stored in $freq(j)$ and the values of Δ_j in subsequent iterations are modified according to equation (4.6), where:

$$freq_{\max} = \max\{freq(j) : \forall j\} \tag{4.5}$$

and β controls the importance of memory in the diversification generation method.

$$\Delta'_j = \Delta_j - \beta \Delta_{\max} \frac{freq(j)}{freq_{\max}} \tag{4.6}$$

Equation (4.6) decreases the value of Δ_j proportionally to $freq(j)$ and so the possibility of inclusion of x_j in L also diminishes.

After the clusters centres are defined, the remaining points are assigned to these clusters. This is done with the goal of minimizing the sum of squared distances from each point to its cluster centre. When the number of clusters is not predetermined, minimizing the sum of squared distances from each point to its cluster centre yields a solution where each point is a centre itself and we have as many clusters as points. However, since for a particular solution to be generated the number of clusters is previously determined, the remaining points can be assigned in order to minimize this measure, as in [Pacheco 2005], in the following way:

1. Let A be the set of unsigned points, i.e., $A = X \setminus S$.
2. For each point $x_j \in A$ and each cluster C_i , $i = 1,\cdots,K$ determine Γ_{ij} given by

$$\Gamma_{ij} = \frac{|C_i|}{|C_i|+1}\left\|c_i - x_j\right\|^2 \tag{4.7}$$

   where c_i is the centroid of C_i.
3. Calculate $\Gamma_{i^* j^*} = \min\left\{\Gamma_{ij} : x_j \notin A, i = 1,\cdots,K\right\}$.
4. Assign x_{j^*} to C_{i^*} and set $A = A \setminus \left\{x_{j^*}\right\}$.
5. If $|A| \neq 0$ return to 2, else stop.

The formula in (4.7) gives the increase in terms of sum of squared distances from each point to its cluster centre when point x_j is assigned to cluster C_i. Steps 1 through 5 define a greedy heuristic for assigning the remaining points to the clusters whose centres were previously chosen.

This algorithm is used to generate OS_{size} initial solutions, called the original set that will serve as basis for constructing the reference set.

4.2.3 Improvement Method

The improvement method is used to enhance the quality of the solutions generated both during the diversification phase and after the combination of two solutions. In this thesis it was chosen to implement the improvement method presented in [Abdule-Wahab, Monmarché et al. 2006], based on the K-Means algorithm [Gan, Ma et al. 2007]. The following steps are taken a number of times equal to *MaxIterImp,* where *MaxIterImp* is a parameter to be chosen by the user.

1. For each point $x_j \in X$ do:
    a. For each cluster C_i $, i = 1, \cdots, K$ determine v_{ij} given by

    $$v_{ij} = \frac{|C_i|}{|C_i|+1}\left\|c_i - x_j\right\|^2 - \frac{|C_l|}{|C_l|+1}\left\|c_l - x_j\right\|^2 \tag{4.8}$$

    where x_j currently belongs to cluster C_l and c_i and c_l are the centroids of C_i and C_l, respectively.

    b. Determine $a = \arg\min_{j=1,\cdots,K,\, j\neq l}\{v_{ij}\}$.
    c. If $a < 0$ reassign x_j to C_a.
2. Compute the fitness of the new solution obtained.
3. If the fitness of the new solution, given by equation (4.2) is better than the original solution, replace the original solution by the new solution.

The formula in (4.8) is given by Spath [Spath 1980] to simplify the K-Means algorithm and approximates the increase in terms of sum of squared distances from each point to its cluster centre when point x_i is moved from cluster C_l to cluster C_j.

4.2.4 Reference Set Update Method

The reference set, RS, is composed of b_1 high quality solutions and b_2 diverse solutions. To construct the initial reference set, the first b_1 best solutions are inserted in the reference set, where the quality of solutions is given by the fitness function presented in section 4.2.1. Next, b_2 solutions are added one by one to reference set according to its diversity. In this work it was used the diversity measure proposed by Pacheco[Pacheco 2005]. Let $dif(\lambda, \lambda')$ be the number of assignments in solution λ that are different from the assignments in solution λ'. For instance, suppose that we have an instance with 5 objects and 4 clusters. Consider two solutions, $y_1 = (1,1,2,3,2)$ and $y_2 = (1,2,2,4,3)$, where the i-element of the vector corresponds to the cluster to which object i is assigned. In this case $dif(\lambda, \lambda')$ is equal to 3.

Iteratively, it is chosen to enter the reference set, the solution that maximizes:

$$\delta_{\min}(\lambda) = \min\{dif(\lambda, \lambda') : \lambda' \in RS\} \tag{4.9}$$

In this implementation the reference set is only updated when better quality solutions are found. Other implementations [Abdule-Wahab, Monmarché et al. 2006] also update the reference set according to the measure of diversity, reinforcing the diversification strategy.

4.2.5 Subset Generation Method

In each iteration of the algorithm, the subsets of solutions from the reference set that will be latter combined by the solution combination method consist of pairs of solutions. The collection of subsets created through this method is composed of all pairs of solutions from the reference set (i, j), $i = 1, \cdots, b_1 + b_2 - 1$, $j = 2, \cdots, b_1 + b_2$, $i < j$. Supposing that we have 3 solutions, we should consider the following subsets (1,2), (1,3), (2,3).

Each element of the collection of subsets generated by this method serves as an input to the solution combination method that generates one or more trial solutions that, after being enhanced by the improvement method in section 4.2.3, can enter the reference set as described in section 4.2.4.

4.2.6 Solution Combination Method

To combine two solutions it was implemented the path relinking strategy described in [Pacheco 2005]. The idea of path relinking is that, in the "path" between two good quality solutions other good quality solutions should exist. A "path" is a series of simple movements that lead from one solution to another. In this case we can start on one solution and move points from one cluster to another until the second solution is reached. For instance, consider two solutions, $y_1 = (1,1,2,3,2)$ and $y_2 = (1,2,2,4,3)$ to be combined. A path between these solutions could be given by: $(1,1,2,3,2) - (1,2,2,3,2) - (1,2,2,4,2) - (1,2,2,4,3)$, where in each movement the first point assigned differently than in y_2 is assigned as in y_2. Solutions in this path can be chosen as trial solutions.

As in [Pacheco 2005], given two solutions, the number of trial solutions that will be generated through the solution combination method varies. If the two solutions to be combined were chosen from the b_1 high quality solutions in the reference set, three trial solutions will be generated. If the two solutions were chosen from the b_2 diverse solutions in the reference set, only one solution will be generated. Otherwise, two solutions will be created. These solutions are randomly chosen from the solutions in the path.

4.2.7 The Final Algorithm

After the basic methods of the scatter search algorithm have been described, we may now describe the final algorithm.

The algorithm starts by generating the original set through the diversification method. The best quality solutions and most diverse solutions are chosen to form the reference set before the iterative part of the algorithm starts. In each iteration, the subset generation method forms all subsets consisting of pairs of solutions from the reference set. These pairs of solutions are then combined through the solution combination method and the generated trial solutions are improved through the improvement method. The reference set update method is then responsible for deciding if any of the generated solutions should replace one of the solutions in the reference set. The iterative procedure continues until there are no new elements in the reference set. The best solution is then returned.

4.3 Computational Results

In this section computational results on two case studies will be presented. In both case studies real world data sets are used. Fuzzyfication of the data is needed before using the previously described algorithms. The reason for fuzzyfying data comes either from a way to deal with different source of uncertainty or to categorize numerical data. Depending on the application that it will be given to the fuzzified data, it may be needed to reduce the number of membership functions to simplify the system that will use this functions. The idea is to reduce the number of terms in each linguistic variable by merging membership functions in the same cluster. These linguistic variables could then be used, for instance in a fuzzy inference system or other fuzzy model. All experiments were conducted in a Intel Core2 Duo, CPU 2.2 GHz, 2 GB of RAM.

4.3.1 Wisconsin Diagnostic Breast Cancer Data Set

The data used in this section is taken from [Asuncion 2007]. Wisconsin Diagnostic Breast Cancer (WDBC) data set contains 569 samples of data describing characteristics of the cell nuclei present in digitalized images of a fine needle aspirate (FNA) of a breast mass. Ten real-valued features were computed for each cell nucleus [Asuncion 2007]:

a) Radius (mean of distances from centre to points on the perimeter)
b) Texture (standard deviation of gray-scale values)
c) Perimeter
d) Area
e) Smoothness (local variation in radius lengths)

f) Compactness
g) Concavity (severity of concave portions of the contour)
h) Concave Points (number of concave portions of the contour)
i) Symmetry
j) Fractal dimension (“coastline approximation” -1)

For each of these features, the mean, standard error and mean of the three largest values (“worst”) of these features were computed for each image. Therefore, each sample has the following 32 attributes:

1. ID number
2. Diagnosis (M = malignant, B = benign)
3. Mean Radius
4. Mean Texture
5. Mean Perimeter
6. Mean Area
7. Mean Smoothness
8. Mean Compactness
9. Mean Concavity
10. Mean Concave Points
11. Mean Symmetry
12. Mean Fractal dimension
13. Radius Standard Deviation
14. Texture Standard Deviation
15. Perimeter Standard Deviation
16. Area Standard Deviation
17. Smoothness Standard Deviation
18. Compactness Standard Deviation
19. Concavity Standard Deviation
20. Concave Points Standard Deviation
21. Symmetry Standard Deviation

22. Fractal dimension Standard Deviation
23. Worst Radius
24. Worst Texture
25. Worst Perimeter
26. Worst Area
27. Worst Smoothness
28. Worst Compactness
29. Worst Concavity
30. Worst Concave Points
31. Worst Symmetry
32. Worst Fractal dimension

In this thesis only features 3 through 22 where used. The mean radius can be used as a measure of the cell nuclei radius. The radius standard deviation gives a measure of the error associated with this measure. For this reason it is advisable to fuzzify the data as a way to deal with uncertainty. To do so we represent each sample's radius by a symmetric triangular membership function $(\mu, 2\sigma)$, as described in section 1.3.1 with μ equal to the mean radius. The width of the triangular membership functions was chosen to be 4σ because in a random variable following a normal distribution, approximately 95% of the samples are expected to belong to the interval $[\mu - 2\sigma, \mu + 2\sigma]$. The same fuzzification scheme, and with the same reasoning, was used for the rest of the features, resulting in 10 linguistic variables with 569 membership functions each, depicted in Figure 4.2 through Figure 4.11. The objective is to reduce the number of membership functions in each linguistic variable using the algorithms described in Chapter 4. The resulting linguistic variables could then be used in a fuzzy inference system or other fuzzy model to diagnose the type of cancer. The construction of such model is outside the scope of this thesis.

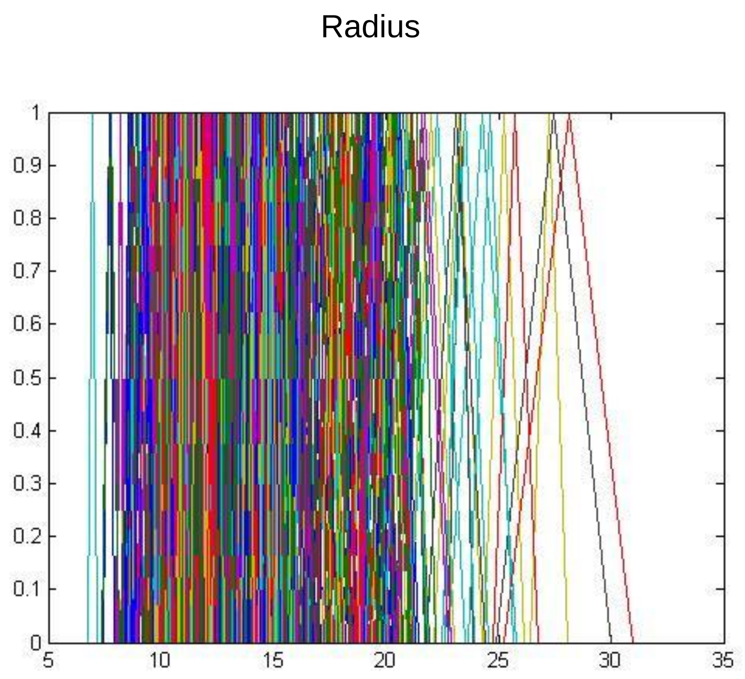


Figure 4.2: Linguistic Variable Radius

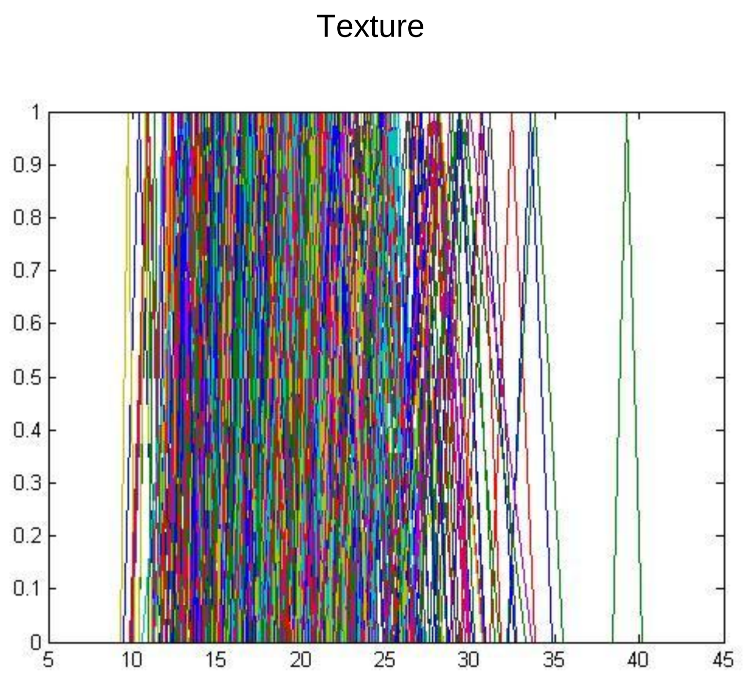


Figure 4.3: Linguistic Variable Texture

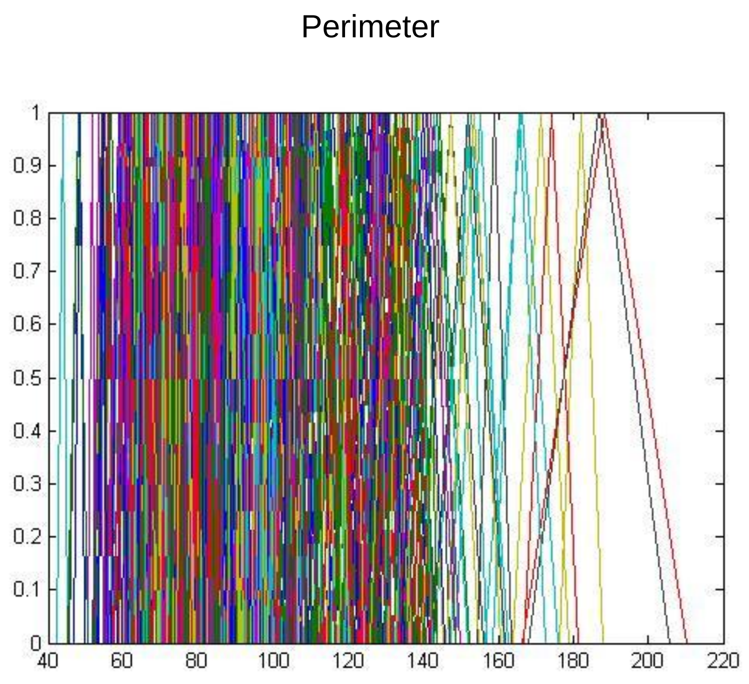


Figure 4.4: Linguistic Variable Perimeter

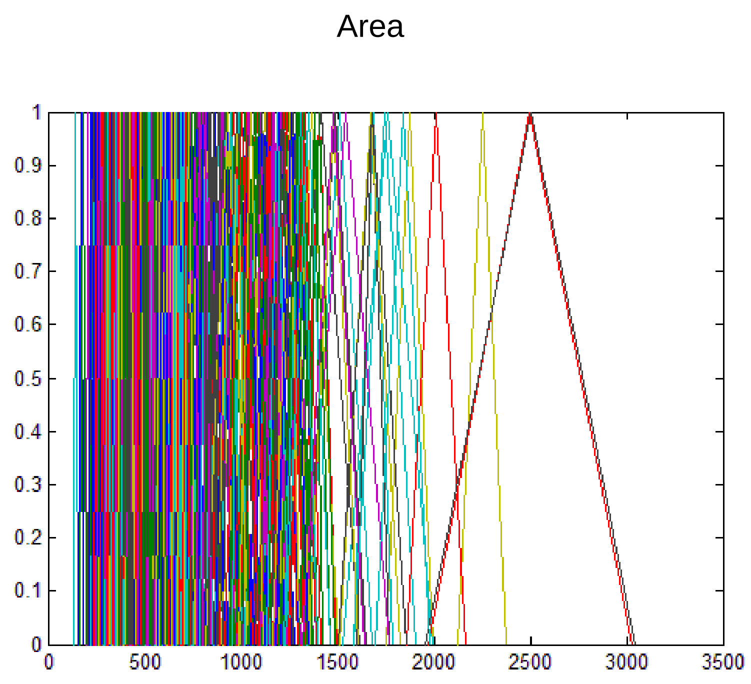


Figure 4.5: Linguistic Variable Area

Smoothness

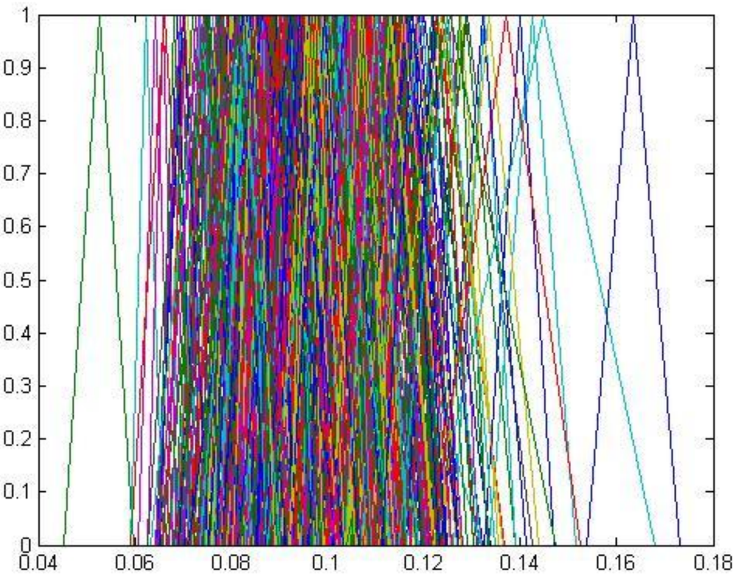


Figure 4.6: Linguistic Variable Smoothness

Compactness

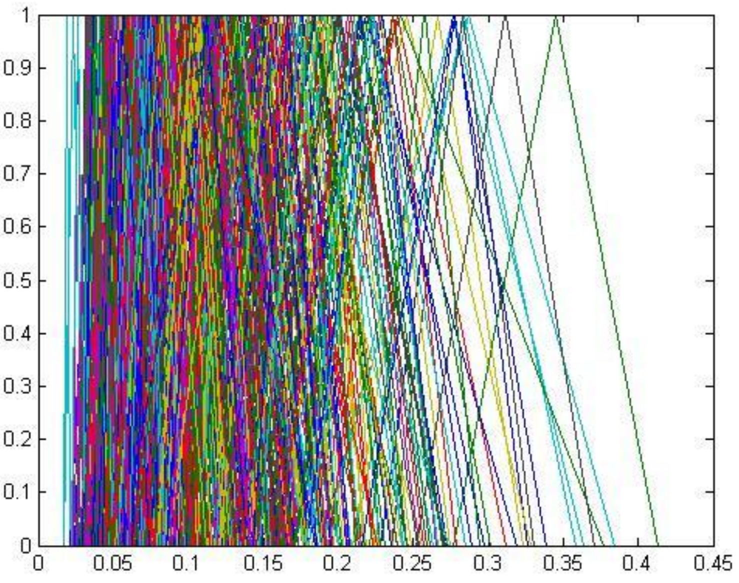


Figure 4.7: Linguistic Variable Compactness

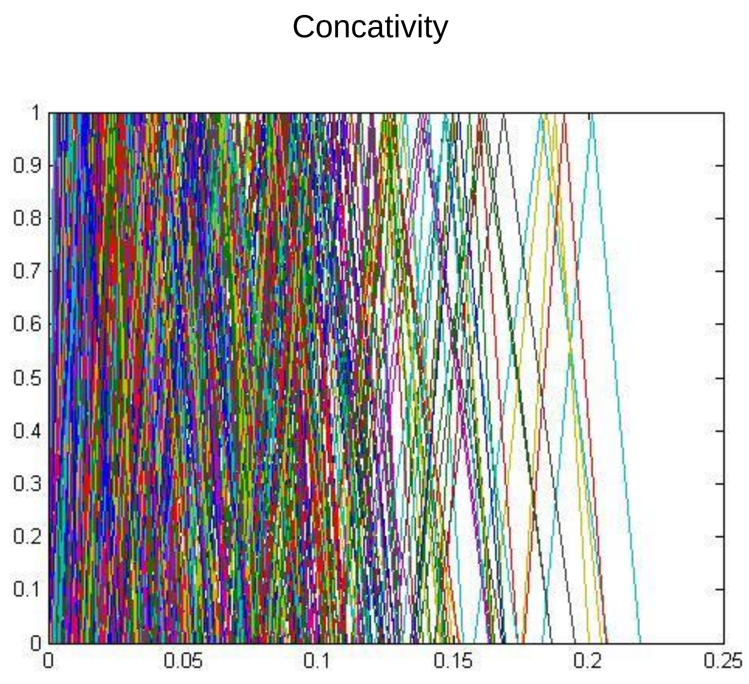


Figure 4.8: Linguistic Variable Concativity

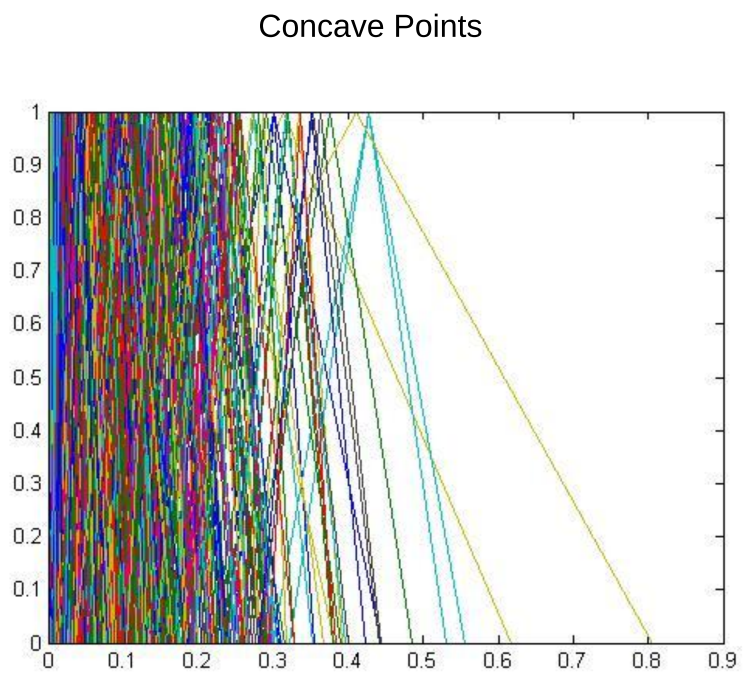


Figure 4.9: Linguistic Variable Concave Points

Symmetry

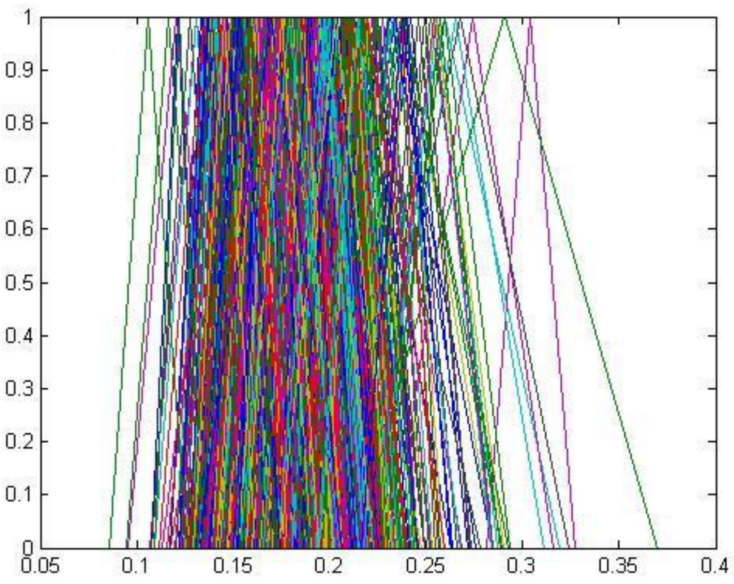


Figure 4.10: Linguistic Variable Symmetry

Fractal Dimension

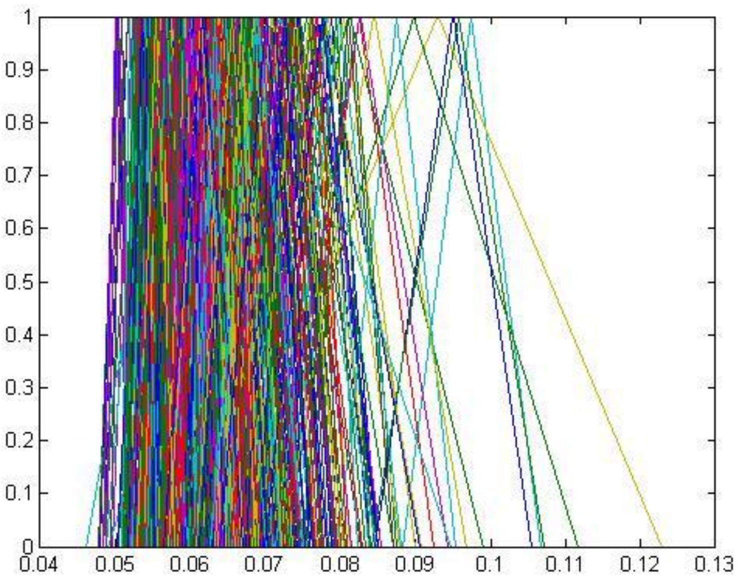


Figure 4.11: Linguistic Variable Fractal Dimension

4.3.1.1 Computational results

The K-Means++ algorithm, described in section 4.1, was applied to each linguistic variable previously presented. The number of clusters was estimated by running the algorithm for $1 \le K \le 568 = n - 1$ and choosing the iteration that maximizes the evaluation measure I given in (4.2). For $K = n$ the evaluation measure I is not defined since $E_n = 0$, as can be seen in (4.3). Figure 4.12 and Figure 4.14 show the evolution of I using K-Means++ for $1 \le K \le 568$, for linguistic variables Radius and Texture. For all other linguistic variables in this case study the overall behaviour is the same. It seems that I increases with K. However, looking at Figure 4.13 and Figure 4.15 it is possible to see that this is not always that case.

Radius - Evaluation Measure

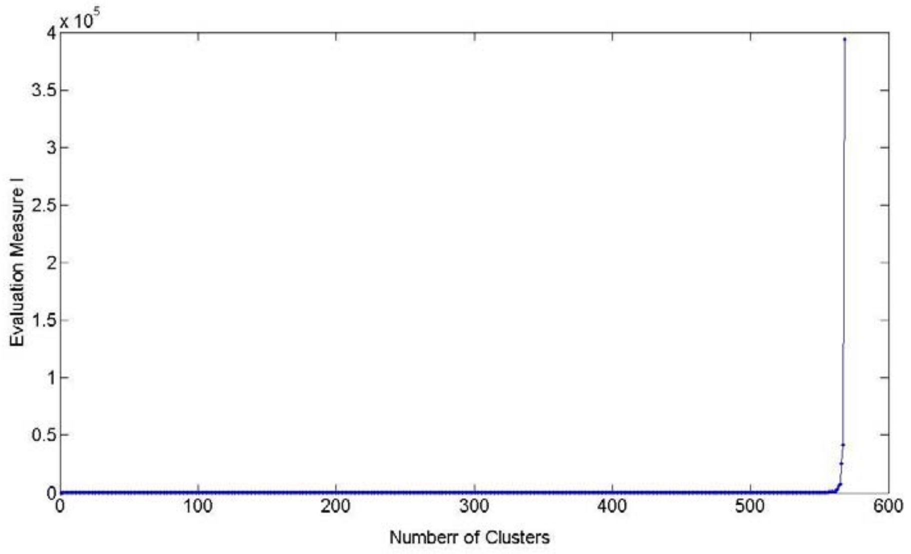


Figure 4.12: Evaluation measure I using K-means++ for $1 \le K \le 568$, linguistic variable Radius

Radius - Evaluation Measure – zoom

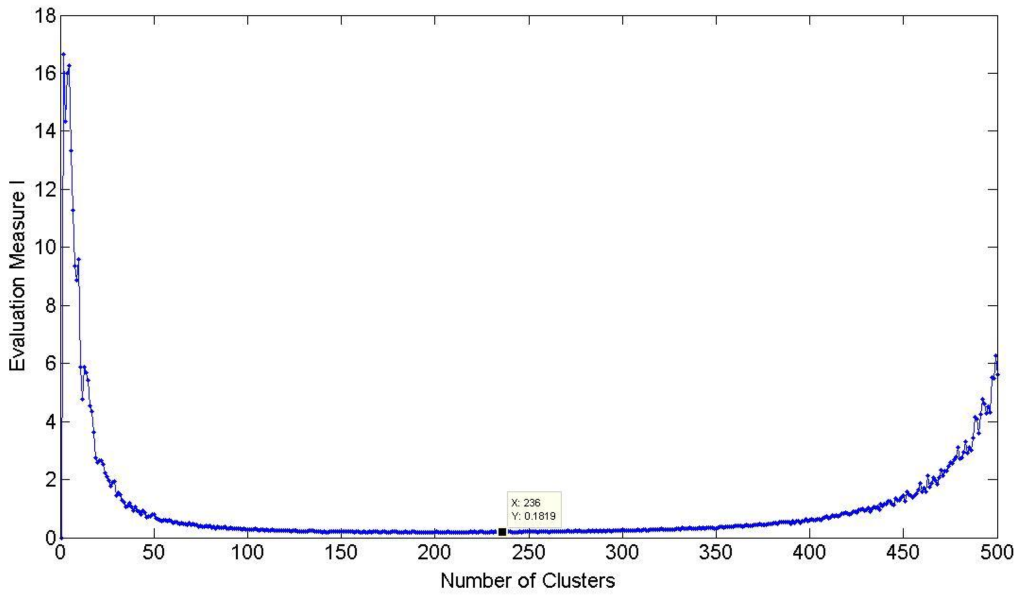


Figure 4.13: Evaluation measure I using K-means++ for $1 \le K \le 500$ (zoom in of previous plot) $1 \le K \le 568$, linguistic variable Radius

Texture - Evaluation Measure

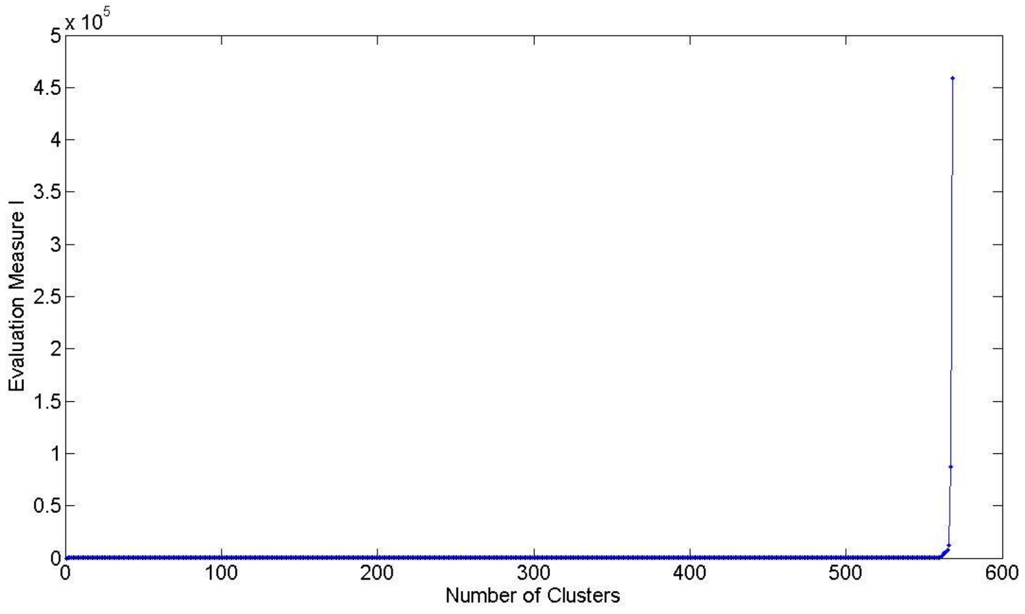


Figure 4.14: Evaluation measure I using K-means++ for $1 \le K \le 568$, linguistic variable Texture

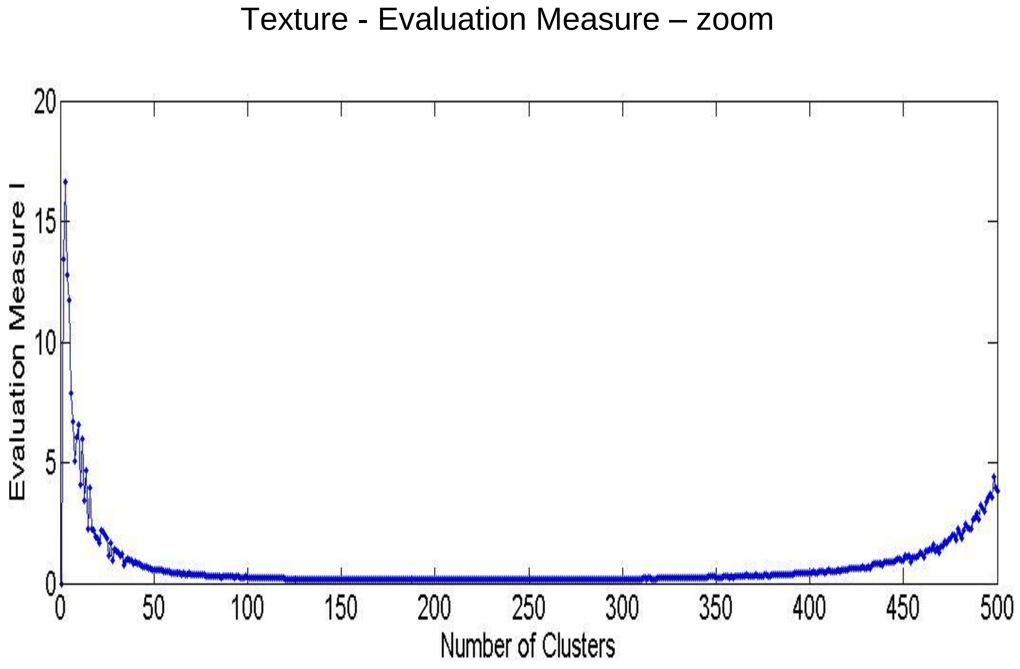


Figure 4.15: Evaluation measure I using K-means++ for $1 \leq K \leq 500$ (zoom in of previous plot), linguistic variable Texture

Running the Scatter Search algorithm from section 4.2 with $K_{\max} = n-1$ yield solutions with $n-1$ clusters. This is easily explained by the behavior of I depicted in Figure 4.12 and Figure 4.14. Given this results it was necessary to redefine the maximum number of clusters allowed, $K_{\max}$. Taking into account Figure 4.13 and Figure 4.15, it was chosen $K_{\max} = 100$. Therefore we are interested in finding a partition of the data into less that 101 clusters that maximizes the cluster validity index I.

The Scatter Search algorithm was run with several values for the parameters α, β, *MaxIterImp* and b_1. To reduce the number of parameters to be analyzed, the number of good quality solutions and of diverse solutions to be included in the reference set was chosen to be equal ($b_1 = b_2$) and the original set was chosen to be 10 times the size of the reference set ($OS_{size} = 10*(b_1 + b_2)$). This last choice is recommended in [Martí, Laguna et al. 1997; Abdule-Wahab, Monmarché et al. 2006]. The algorithm was run until no new elements entered the reference set. Since random numbers are used during the algorithm, for each combination of values of the parameters 5 experiments were run. Only 5 runs of each experiment is clearly not enough to take any statistically valid conclusions, the purpose of these experiments was only to see how results were influenced by different choices of the several parameters involved.

Only the results relative to the linguistic variable Radius will be discussed in more detail. Similar conclusions were found for the rest of the linguistic variables in this case study. Final results will be presented for all linguistic variables.

The first experiments were conducted with no improvement method (i.e. *MaxIterImp* $= 0$) and all possible combinations of $\alpha \in \{0; 0.5; 0.8; 1\}$, $\beta \in \{0; 0.5\}$ and $b_1 \in \{2; 5\}$, with the purpose of analysing the influence of the parameter α in the algorithm. In section 4.2.2 it is stated that the parameter α controls the level of randomization used when choosing cluster centres. When $\alpha = 0$, the cluster centres are chosen completely at random from the original data X. In Figure 4.16 it is clear that this randomness affects the standard deviation of the fitness of solutions returned by the algorithm. When $\alpha = 0$ the standard deviation of results is much higher than for larger values of α. This standard deviation means that it is likely to achieve very good results but also very bad results, a characteristic that is undesirable in an algorithm. In fact, although in terms of the best result found for each 5 experiments a choice of $\alpha = 0$ seems to produce good results (Figure 4.17), the same does not happen in terms of average results (Figure 4.18). Given this results, no further experiments were done with $\alpha = 0$.

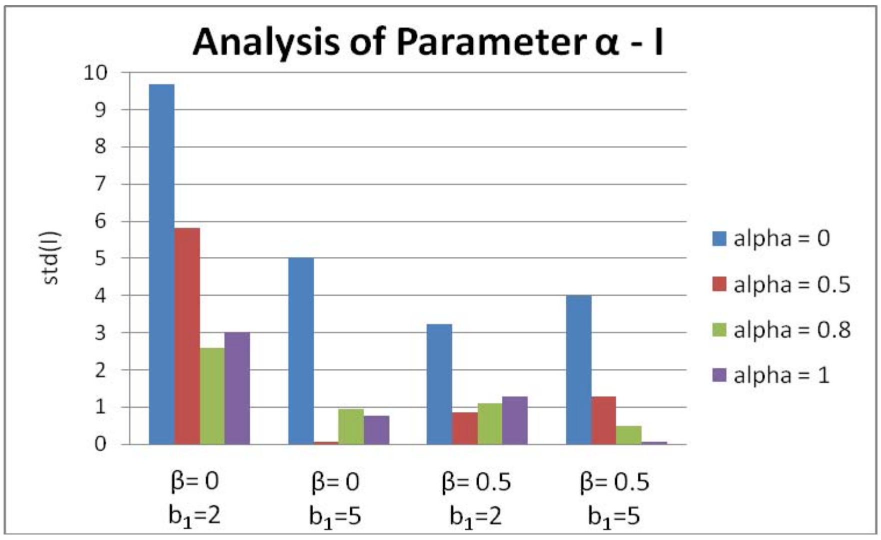


Figure 4.16: Influence of α in fitness function I standard deviation

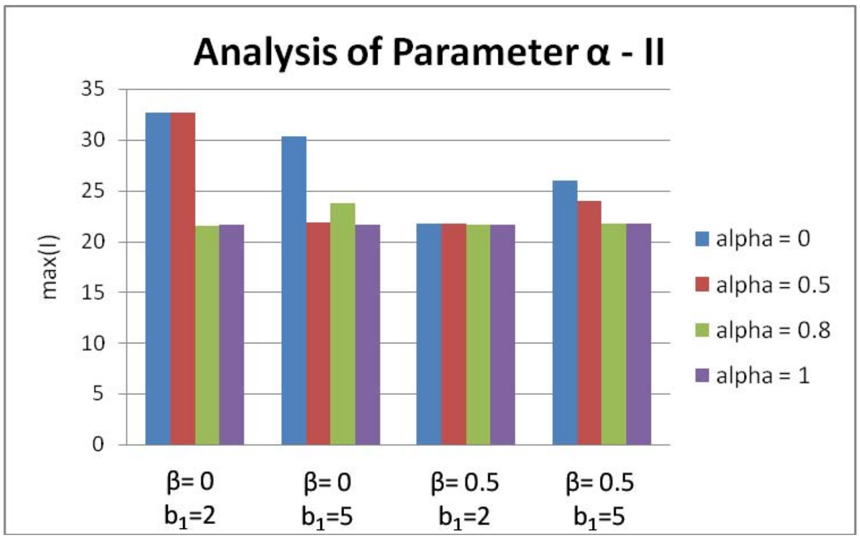


Figure 4.17: Influence of α in best results obtained for fitness function I

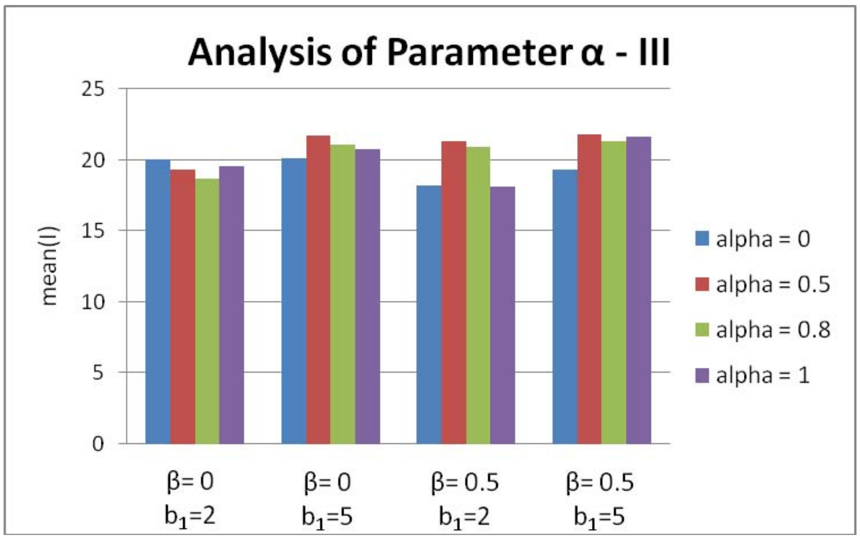


Figure 4.18: Influence of α in mean results for fitness function I

The parameter β controls the weight given to the memory in the process of choosing the cluster centres when creating initial solutions, as given by (4.6). Notice that it does not make sense to study the importance of this parameter for values of α very close to zero since the choice of the cluster centres is close to random. In this case, although the quantities Δ_j are replaced by Δ'_j, this does not significantly change the set L from which the cluster centres are chosen. As can be seen in Figure 4.19, there seems to be an improvement in terms of average results in using this memory during the creation of the original set. Therefore, no further experiments for $\beta = 0$ will be discussed.

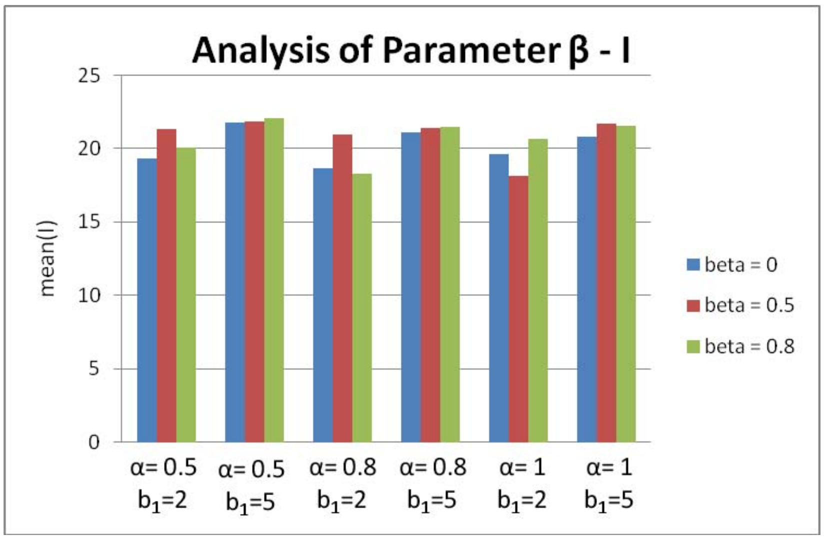


Figure 4.19: Influence of β in mean results for fitness function I

By considering a larger reference set we expect to obtain better results but worse computational efficiency. Not only does it take more time to generate the original set (since the size of the original set was chosen to be 10 times the size of the reference set), but also the number of subsets of solutions to be combined increases exponentially. In Figure 4.20 we can see that increasing the size of the reference set does indeed produce better quality results, but this improvement is achieved at a computational cost, as can be seen in Figure 4.21. However, in this case study, we are considering an increase from an execution time of around 1 minute to around 3.5 minutes. Due to these low execution times, we can afford to consider a larger reference set to obtain better solutions. In other applications where the number of membership functions to merge is higher, this increase in execution time could be unaffordable. Reference sets are typically small. Only two small values for b_1 were considered, $b_1 = 2$ and $b_1 = 5$. This is due to the way the size of original set is related to this parameter. Since it was chosen that $b_1 = b_2$ and $OS_{size} = 10 * (b_1 + b_2)$, we are considering original sets with 40 and 100 solutions. Larger original sets would mean that the initial diversification achieved would be such that a very good quality solution was probably already found in this first step of the algorithm.

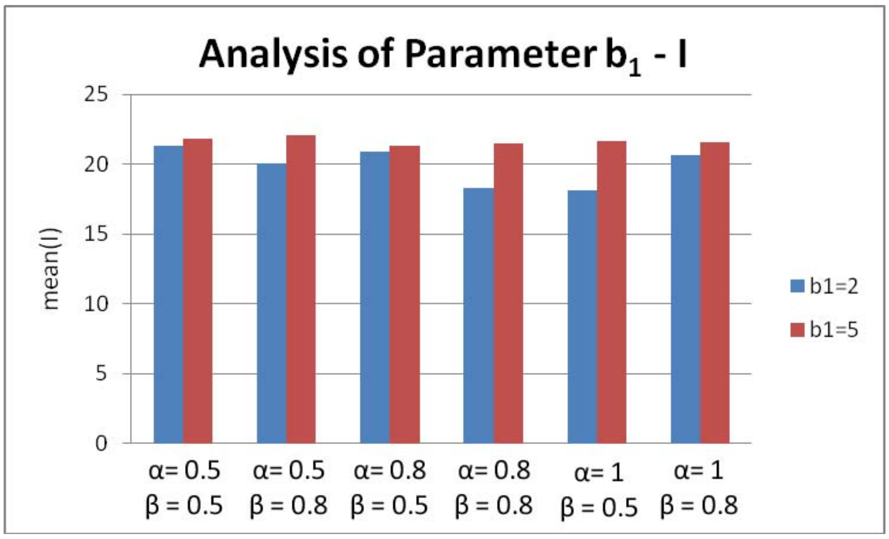


Figure 4.20: Influence of b_1 in mean results for fitness function I

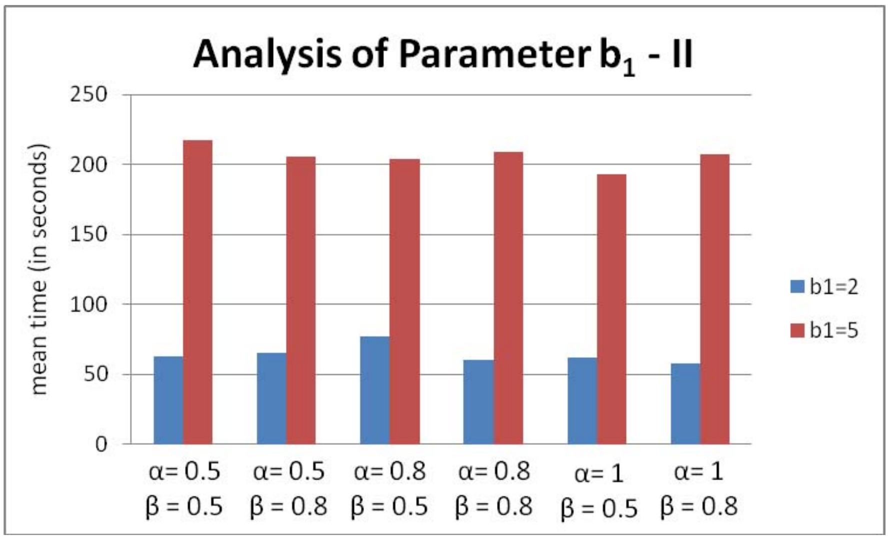


Figure 4.21: Influence of b_1 in execution time

Until this point experiments were done without the Improvement Method. This way the influence of the parameters being analyzed in the solutions obtained was clearer. The computational cost, in terms of execution time (Figure 4.23), of using the Improvement Method should lead to an improvement in the quality of the solutions obtained by the algorithm. However, this was not always the case, as can be seen in

Figure 4.22. The increase in execution time when using $MaxIterImp = 5$ is considerably high and does not result in a significant improvement in the quality of solutions. Considering $MaxIterImp = 2$ the increase in terms of execution time is not high (from approximately 1 minute to approximately 3 minutes) but only improves the average quality of solutions in some of the sets of experiences made. This result was unexpected. Since only 5 experiments per each choice of parameters were made, these results could be explained by the weak estimate of the actual mean values. Since the increase in execution time for using $MaxIterImp = 2$ is not too high, this value will be considered for this parameter. Also, the two first combinations of parameters seem to give consistently better results than the rest. From these two sets of experiences, the second presents slightly better average results. Therefore, in the results presented in Table 4.1 and Figure 4.24 through Figure 4.33 were obtained with $\alpha = 0.5$, $\beta = 0.8$, $b_1 = 5$ and $MaxIterImp = 2$. Figure 4.24 through Figure 4.33 show, for both algorithms, a scatter plot of the centre values versus the width of the triangular membership and the final configuration of the linguistic variable.

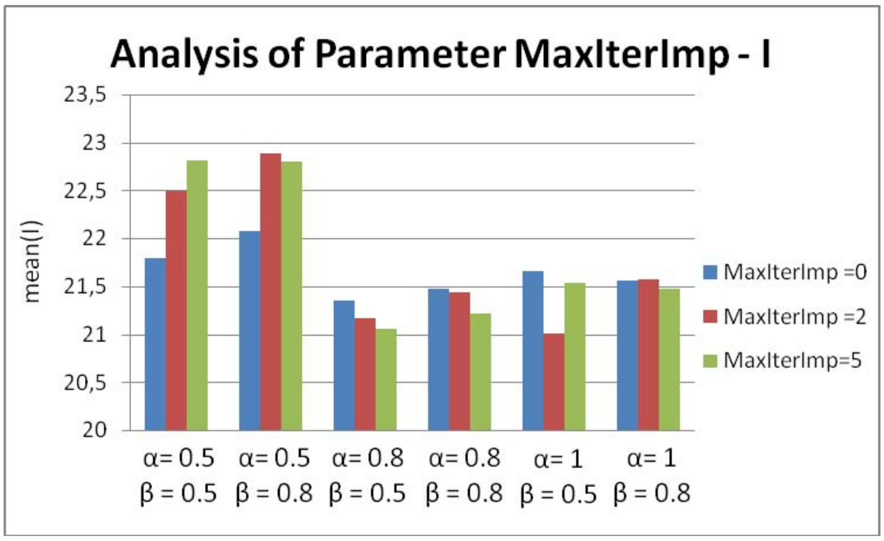


Figure 4.22: Influence of Improvement Method in mean results for fitness function I

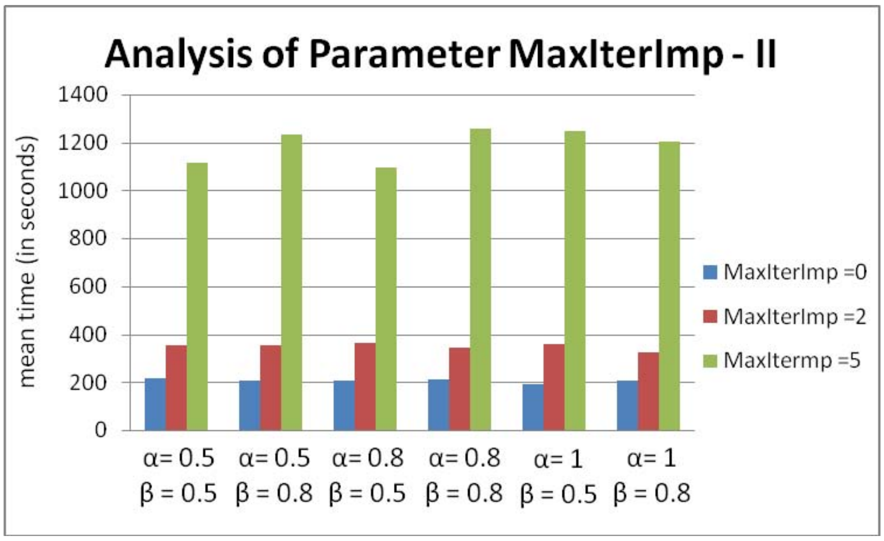


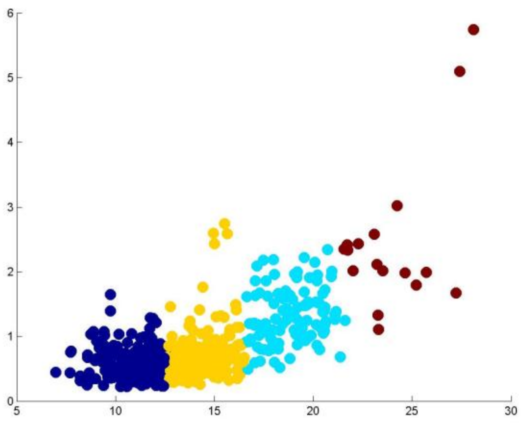
Figure 4.23: Influence of Improvement Method in execution time

To better understand what happens during the algorithm a study of how the reference set evolves was conducted. Surprisingly, for all experiments conducted the reference set was only updated a few times after its initial construction and the best solution found was always generated during the first part of the algorithm. This result was not expected. Unfortunately, the second part of the Scatter Search algorithm is not producing good solutions. Still, Table 4.1 shows that the Scatter Search returned better results than K-Means++ for almost all variables. The first part of the algorithm is sufficient to obtain better quality solutions than the K-Means++. The computational time of the Scatter Search was expected to be much higher than the computational time of the K-Means++. This did not happen only because the Scatter Search algorithm stopped after the first iteration of the second part of the algorithm.

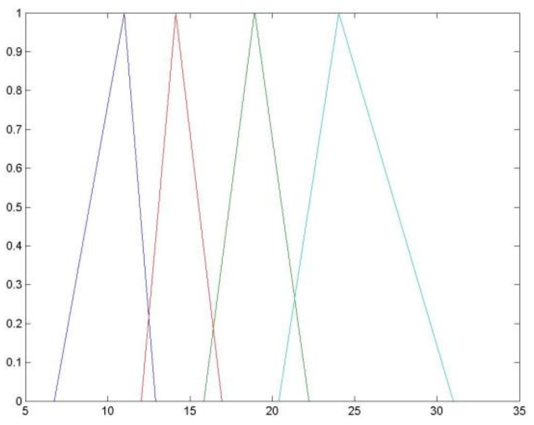
	K-Means++			Scatter Search		
	Time (sec.)	Nr. Clusters	I	Time (sec.)	Nr. Clusters	I
Radius	426,3904	4	18,9609	352,8277	3	26,9636
Texture	398,9221	3	16,8796	344,1798	5	30,18262
Perimeter	463,4574	7	830,33	371,4513	3	1286,17
Area	416,6392	6	426500,7	394,1495	3	668728,5
Smoothness	419,2019	3	0,000143	413,9573	4	0,0002044
Compactness	407,7729	3	0,004632	308,5648	3	0,004547
Concativity	418,1248	3	0,080241	305,7193	3	0,080673
Concave Points	405,9813	3	0,002273	470,995	3	0,002743
Symmetry	411,8899	3	0,000921	672,4182	3	0,000932
Fractal Dimension	413,8989	4	0,000167	679,9634	3	0,000255

Table 4.1: K-Means++ vs Scatter Search (best results)

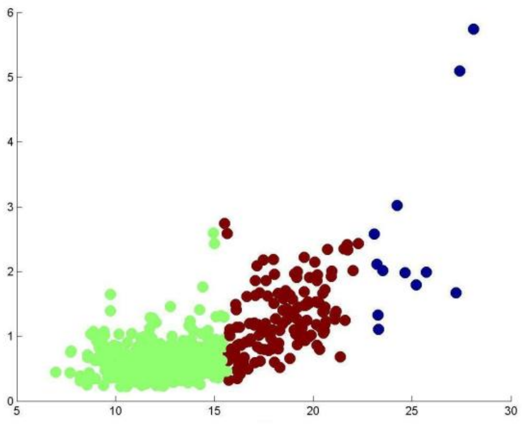
Radius



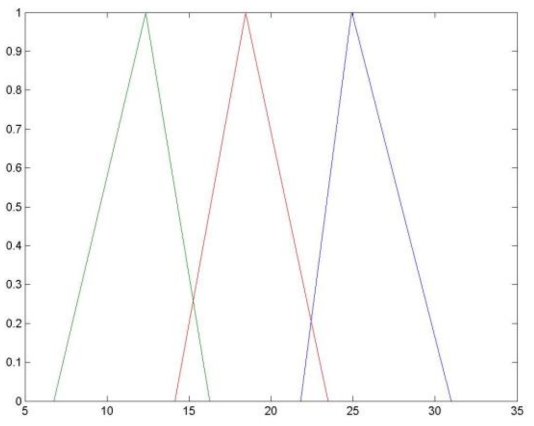
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



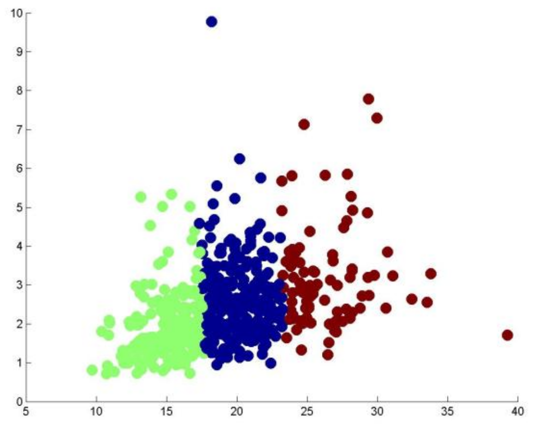
(c) Clusters with Scatter Search



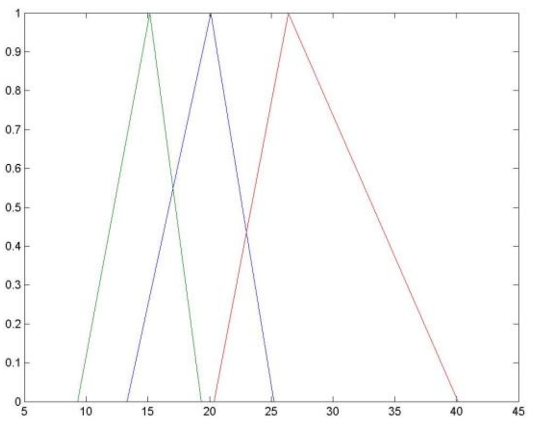
(d) Linguistic Variable after Scatter Search

Figure 4.24: Best Results for Linguistic Variable Radius

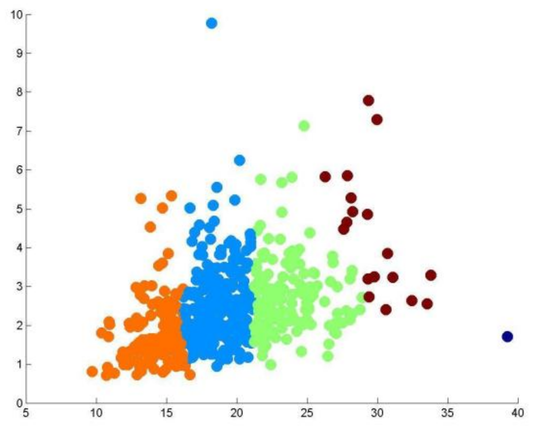
Texture



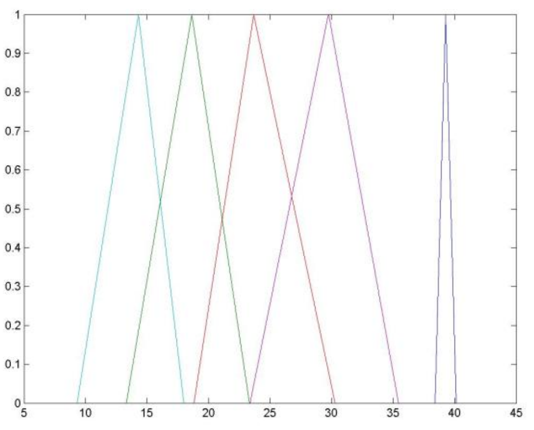
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



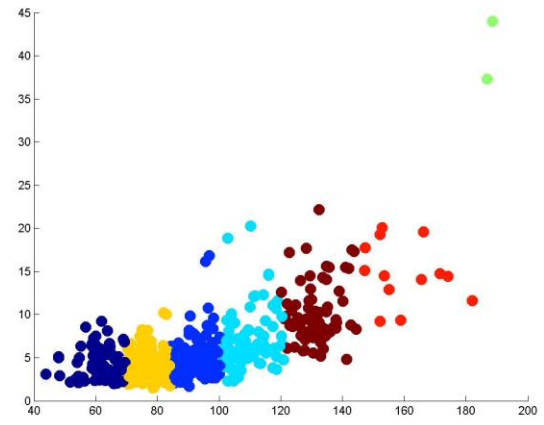
(c) Clusters with Scatter Search



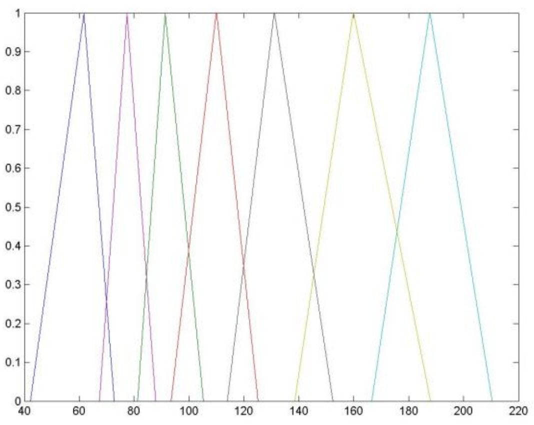
(d) Linguistic Variable after Scatter Search

Figure 4.25: Best Results for Linguistic Variable Texture

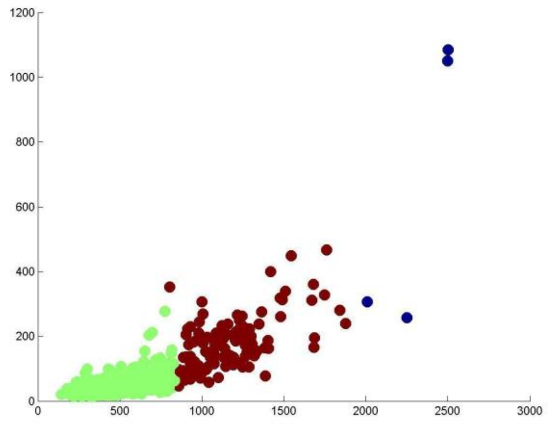
Perimeter



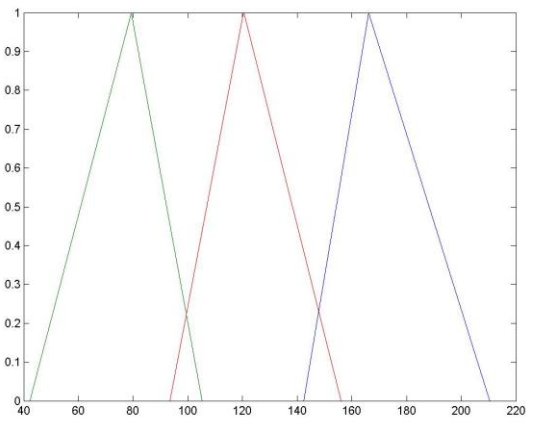
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



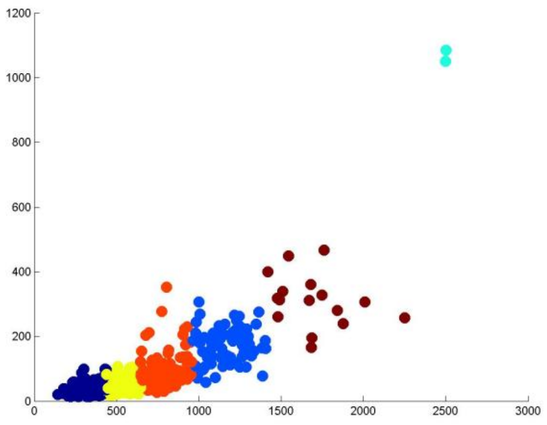
(c) Clusters with Scatter Search



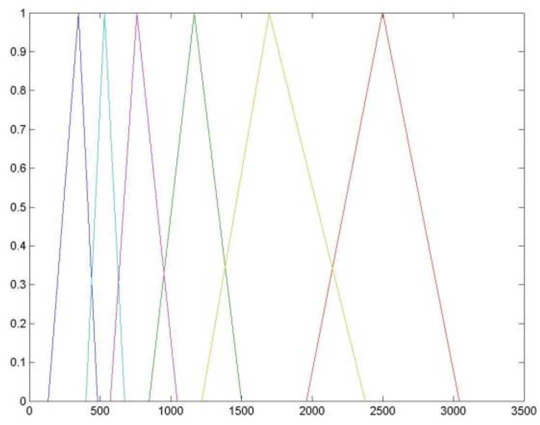
(d) Linguistic Variable after Scatter Search

Figure 4.26: Best Results for Linguistic Variable Perimeter

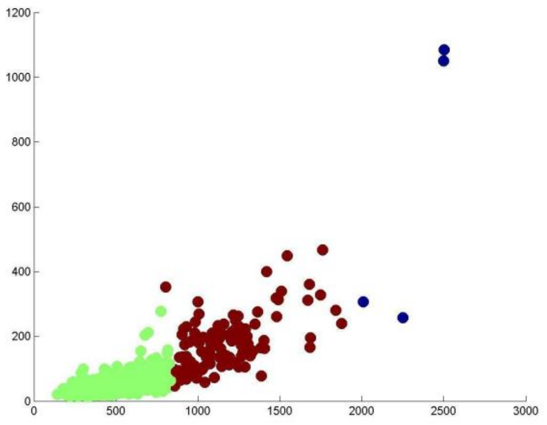
Area



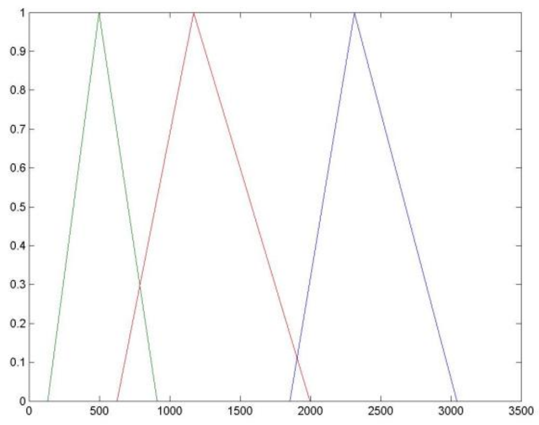
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



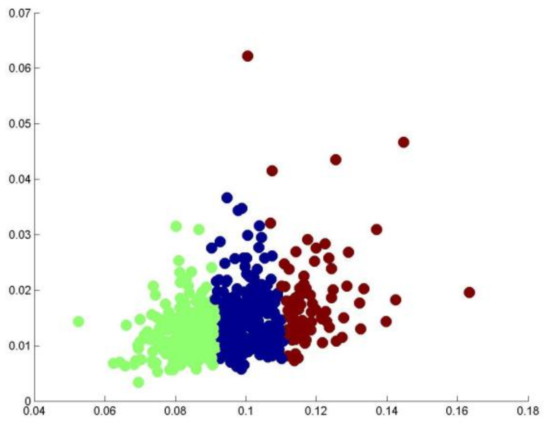
(c) Clusters with Scatter Search

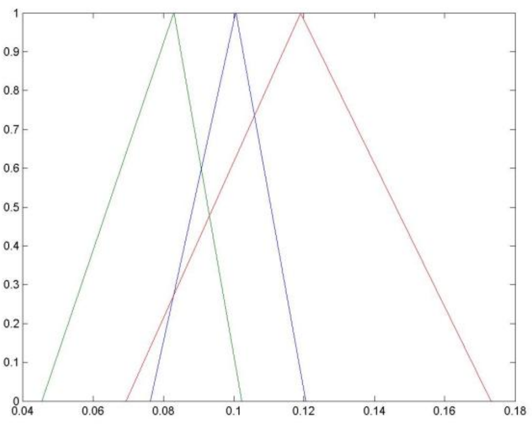


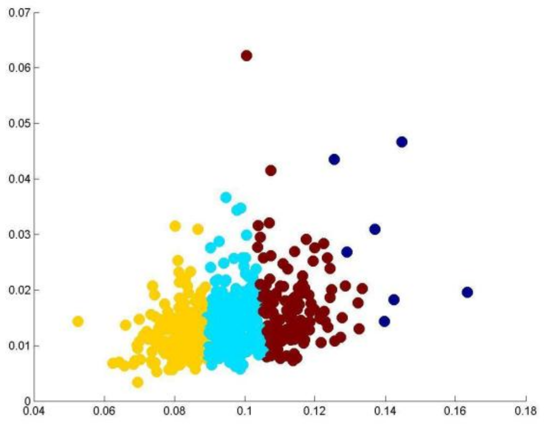
(d) Linguistic Variable after Scatter Search

Figure 4.27: Best Results for Linguistic Variable Area

Smoothness


(a) Clusters with K-Means++


(b) Linguistic Variable after K-Means++


(c) Clusters with Scatter Search

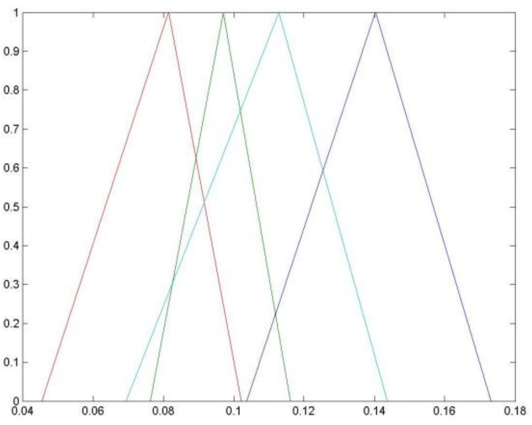
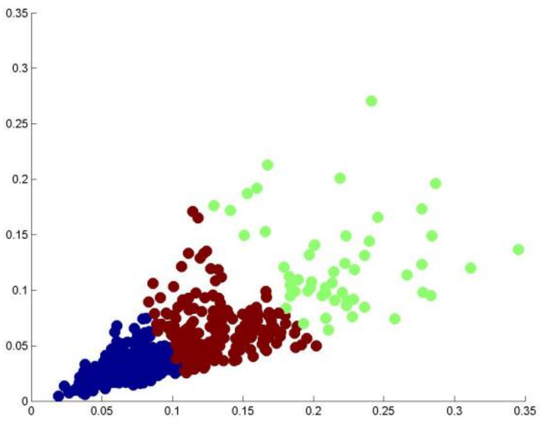
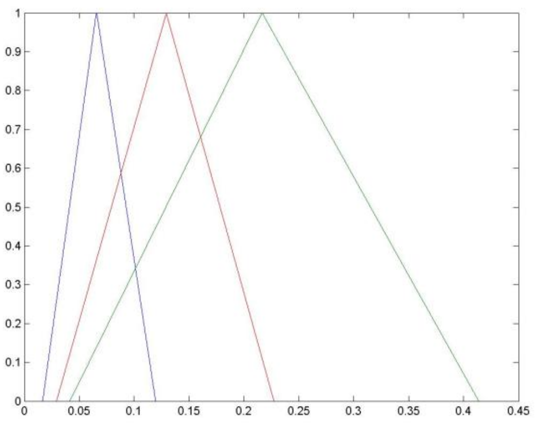

(d) Linguistic Variable after Scatter Search

Figure 4.28: Best Results for Linguistic Variable Smoothness

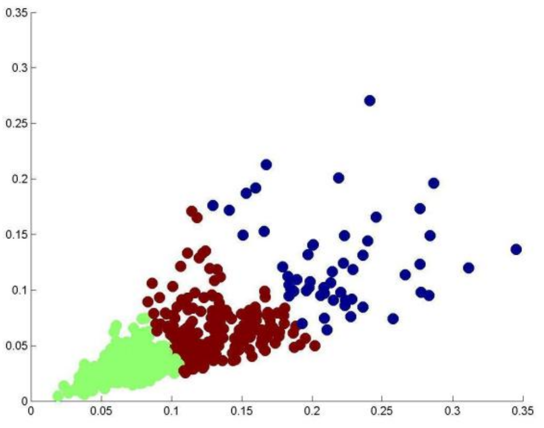
Compactness



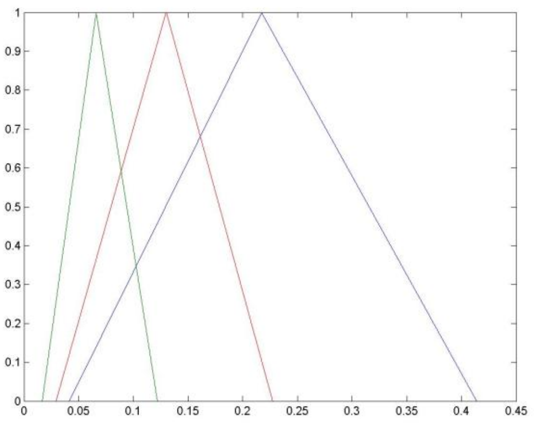
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



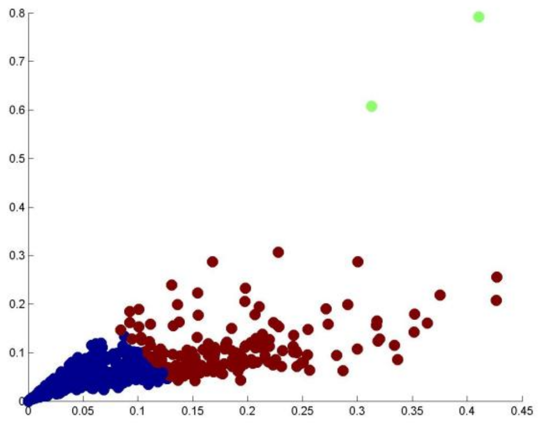
(c) Clusters with Scatter Search



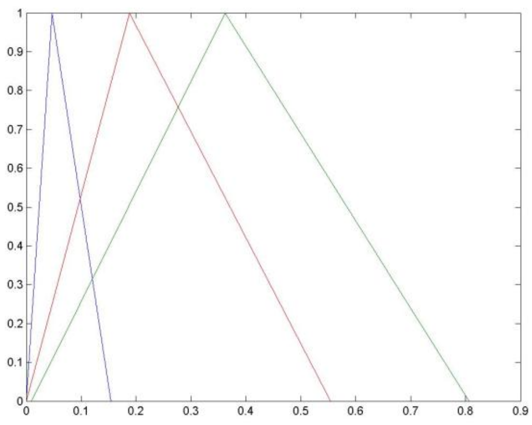
(d) Linguistic Variable after Scatter Search

Figure 4.29: Best Results for Linguistic Variable Compactness

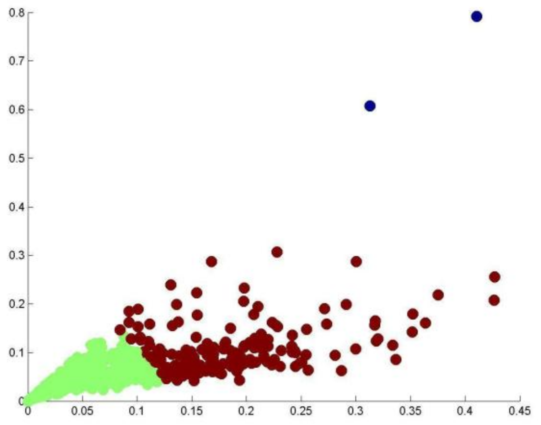
Concativity



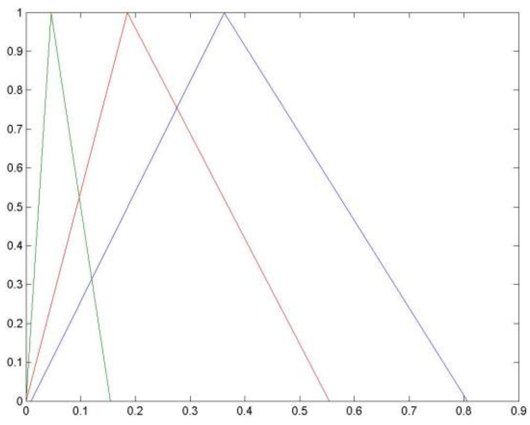
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



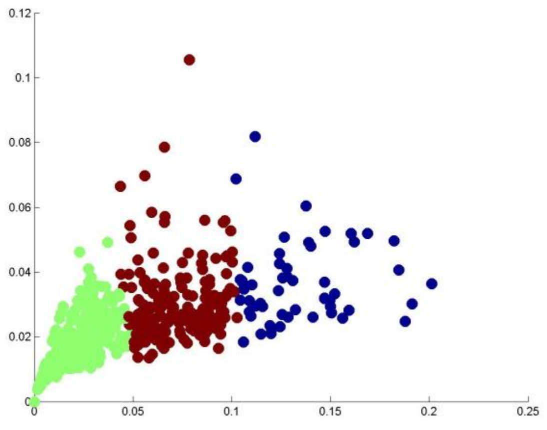
(c) Clusters with Scatter Search



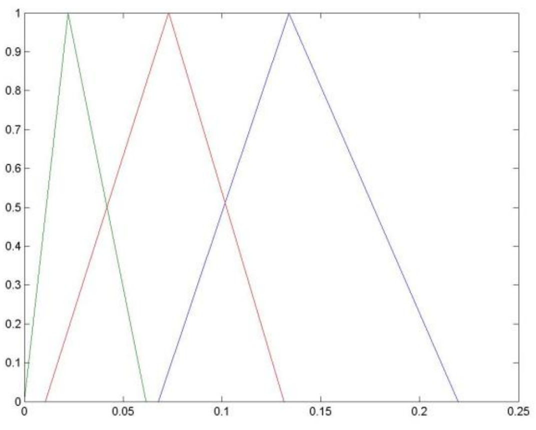
(d) Linguistic Variable after Scatter Search

Figure 4.30: Best Results for Linguistic Variable Concativity

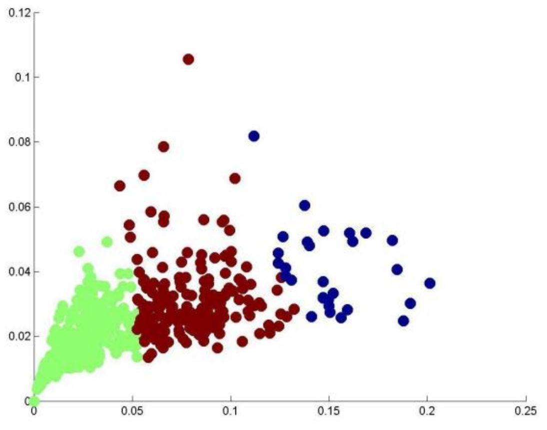
Concave Points



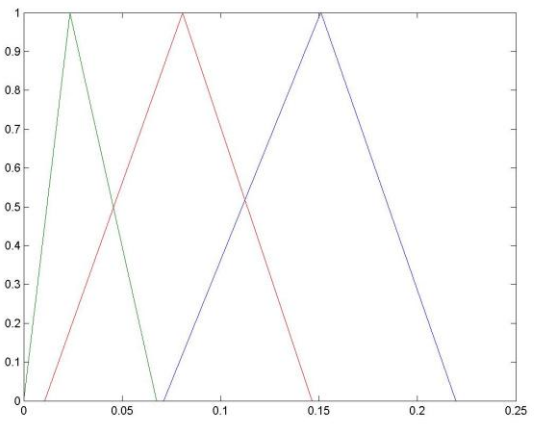
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



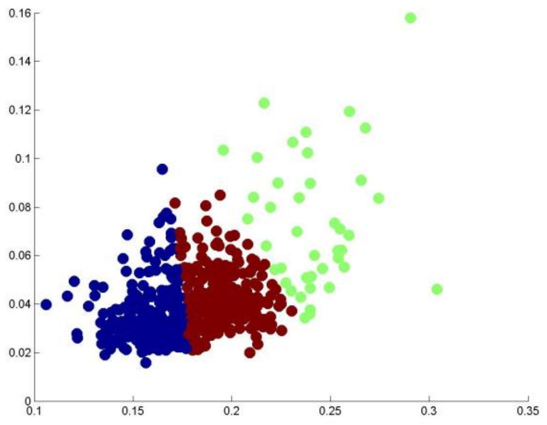
(c) Clusters with Scatter Search



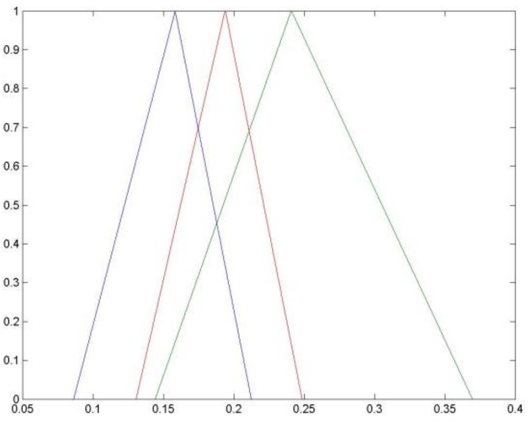
(d) Linguistic Variable after Scatter Search

Figure 4.31: Best Results for Linguistic Variable Concave Points

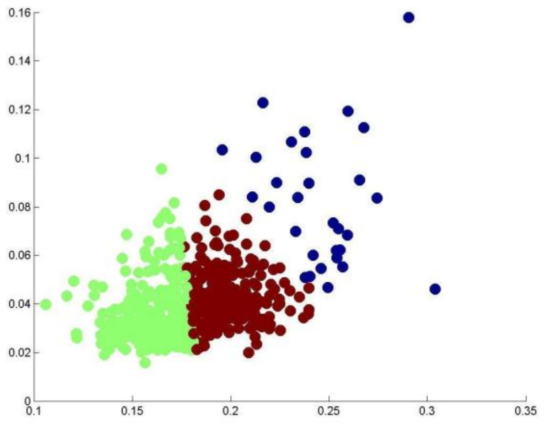
Symmetry



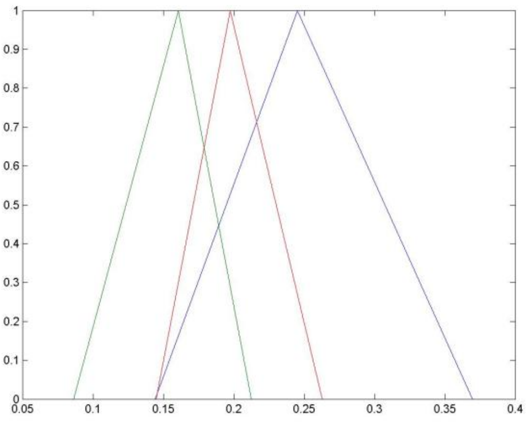
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



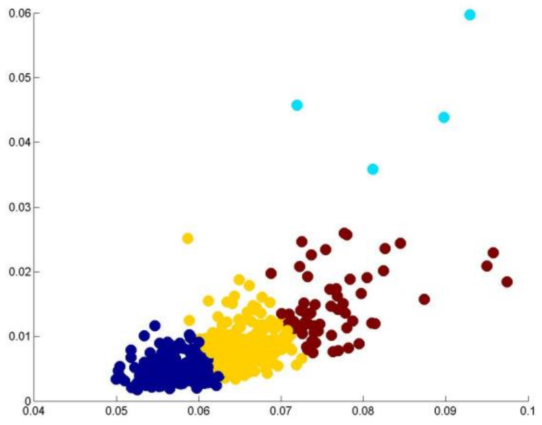
(c) Clusters with Scatter Search



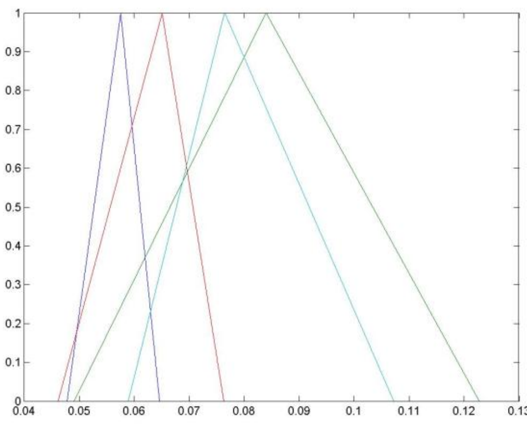
(d) Linguistic Variable after Scatter Search

Figure 4.32: Best Results for Linguistic Variable Symmetry

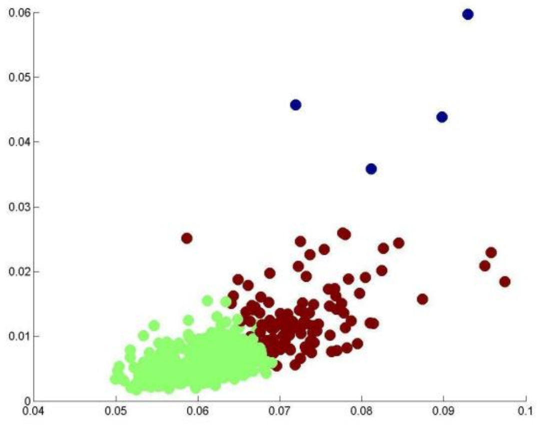
Fractal Dimension



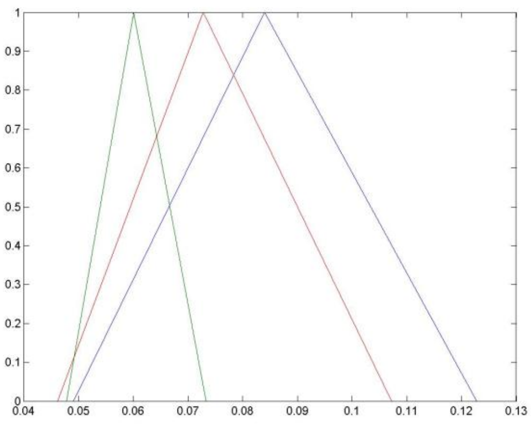
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



(c) Clusters with Scatter Search



(d) Linguistic Variable after Scatter Search

Figure 4.33: Best Results for Linguistic Variable Fractal Dimension

4.3.2 Credit Approval Data Set

The data used in this section is taken from [Asuncion 2007]. This data concerns credit approval information. Credit approval information is usually prone to uncertainty. On one hand, attributes like annual income are usually average information rather than absolute information. On the other hand, misinformation from the credit candidates, for instance undeclared income, provides additional uncertainty to the values presented. Therefore, it is natural to use fuzzy models when constructing automatic credit approval applications. Credit Approval (CA) data set contains 690 samples (665 after removing missing data) of data concerning credit approval information. Unfortunately, for confidentiality purposes, all attribute names and values have been changed to meaningless symbols. Each sample is composed of features A1 through A16. Ahead each variable its possible values are presented [Asuncion 2007]:

1. A1 – b, a
2. **A2 – continuous**
3. **A3 – continuous**
4. A4 - u, y, l, t
5. A5 – g, p, gg
6. A6 – c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff
7. A7 – v, h, bb, j, n, z, dd, ff, o
8. **A8 – continuous**
9. A9 - t, f
10. A10 – t, f
11. **A11 – continuous**
12. A12 – t, f
13. A13 – g, p, s
14. **A14 – continuous**
15. **A15 – continuous**
16. A16 - +,- (class attribute)

Only continuous attributes will be considered because other attributes are already categorized and the number of categories is already small. Once again, symmetrical triangular membership functions will be used. Since there is no information on the attributes and no additional information on accuracy of the data, triangular membership functions $(v_{ij}, 2\sigma_j)$ where used, where v_{ij} is the value of attribute j for sample i and σ_j is the standard deviation of attribute j. Of course all membership functions from the same linguistic variable, representing an attribute, will have the same width. In reality, the width of the membership functions would vary according to additional information collected from experts or from credit candidates. The six linguistic variables in this case study are represented in Figure 4.34 through Figure 4.39.

A2

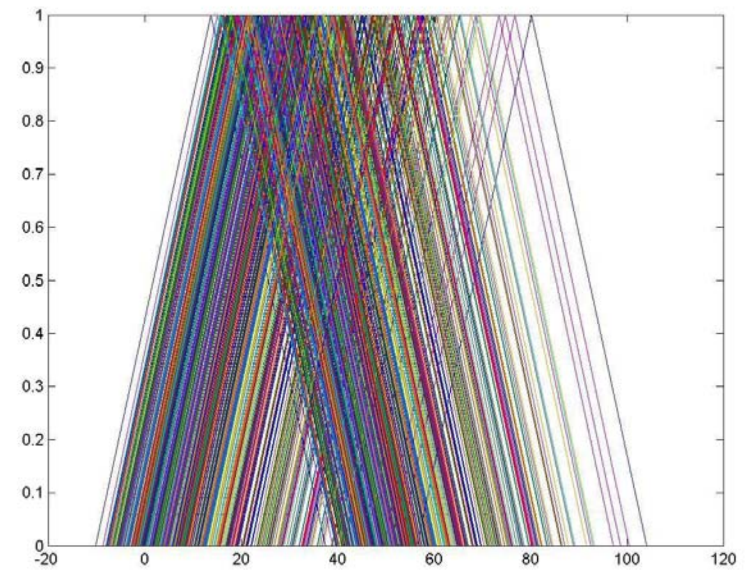


Figure 4.34: Linguistic Variable A2

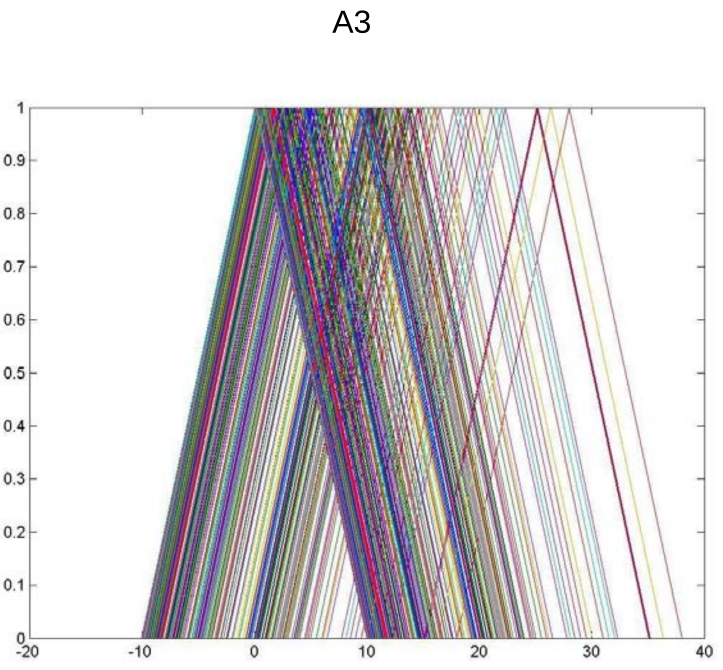


Figure 4.35: Linguistic Variable A3

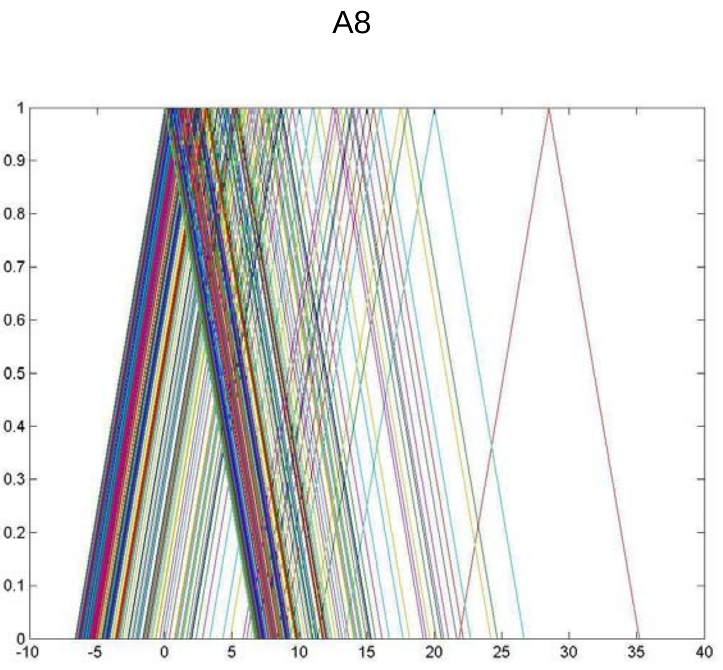


Figure 4.36: Linguistic Variable A8

A11

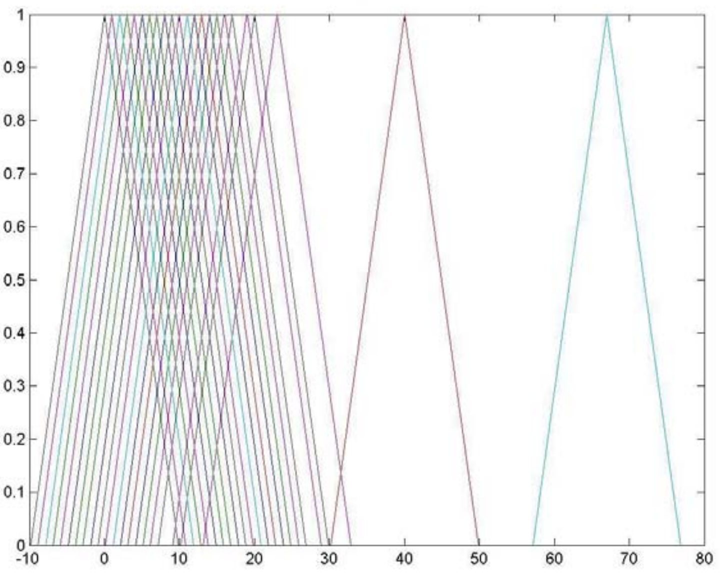


Figure 4.37: Linguistic Variable A11

A14

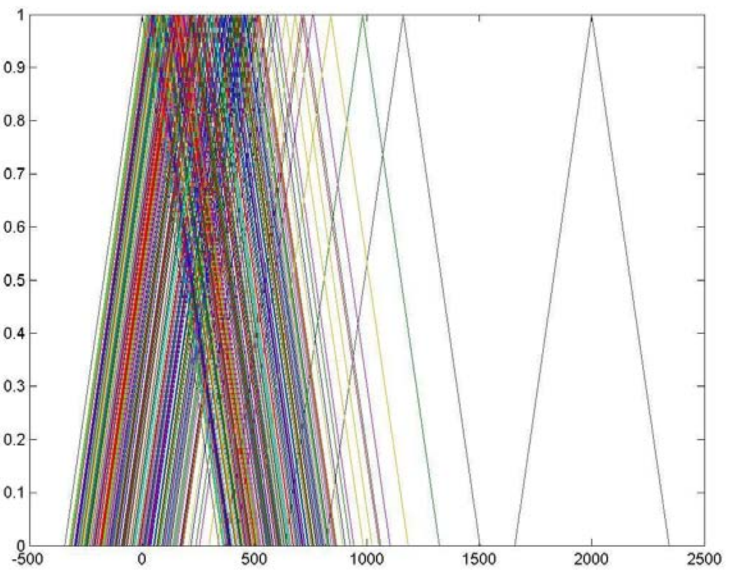


Figure 4.38: Linguistic Variable A14

A15

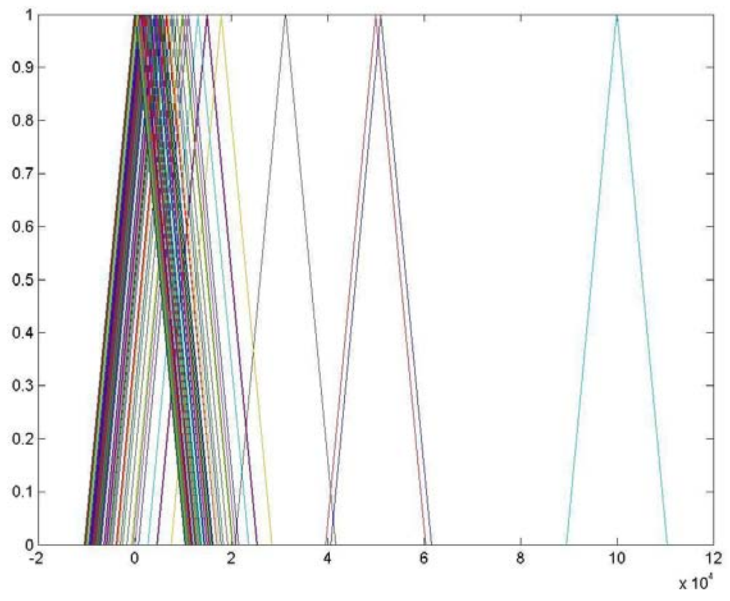


Figure 4.39: Linguistic Variable A15

4.3.2.1Computational Results

The same study conducted for the linguistic variables in the previous case study was conducted for the six linguistic variables in this case study. The K-Means++ algorithm, described in section 4.1, was applied to each of the linguistic variables.

Figure 4.40 and Figure 4.42 show the evolution of I using K-Means++ for $1 \leq K \leq 665$, for linguistic variables A2 and A3. For all other linguistic variables in this case study the overall behaviour is the same. This is not the same behaviour as in the previous case study. Now there is a very sudden improvement in the quality of solutions for a certain number of clusters. However, this sudden improvement is achieved only for high number of clusters, between 200 and 350 clusters for the linguistic variables in this case study. This is a high number of membership functions in a linguistic variable. It is desirable to achieve a greater reduction in the number of membership functions to improve its interpretability. Therefore, as in the previous case study, it was chosen $K_{\max} = 100$. It is left for future work to investigate on procedures to estimate this parameter. Figure 4.41 and Figure 4.43 show a zoom of the previous plots, for $1 \leq K \leq 150$.

A2 - Evaluation Measure

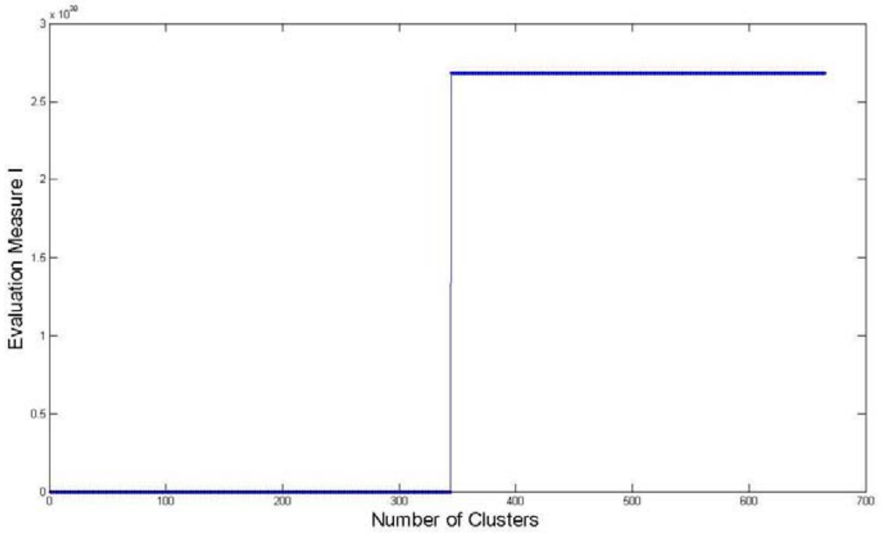


Figure 4.40: Evaluation measure I using K-means++ for $1 \leq K \leq 665$, linguistic variable A2

A2 - Evaluation Measure – zoom

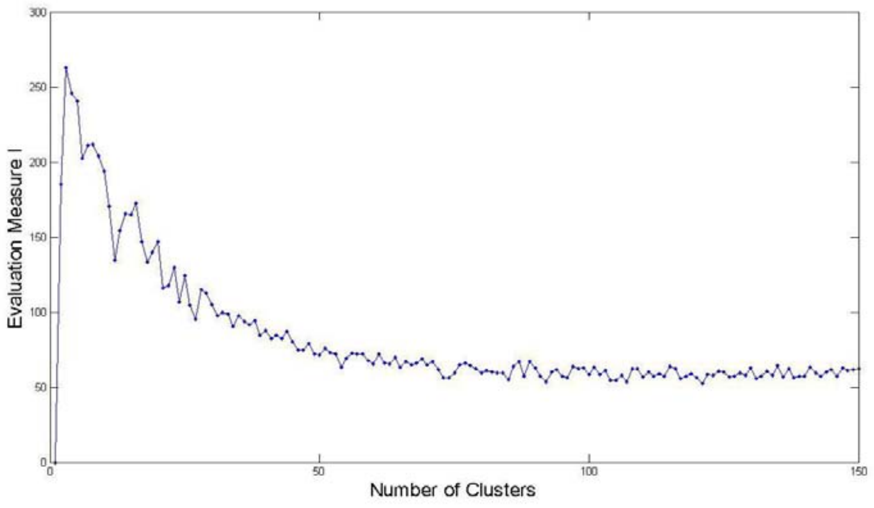


Figure 4.41: Evaluation measure I using K-means++ for $1 \leq K \leq 150$ (zoom in of previous plot), linguistic variable A2

A3 - Evaluation Measure

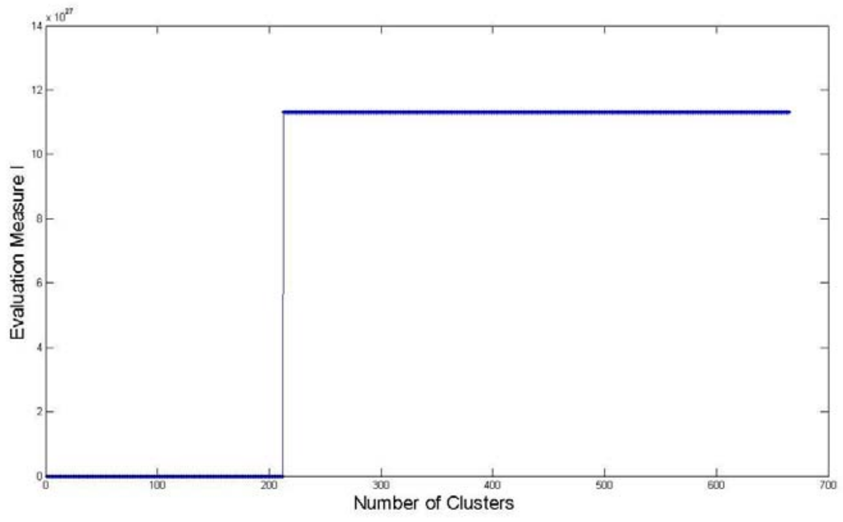


Figure 4.42: Evaluation measure I using K-means++ for $1 \leq K \leq 665$, linguistic variable A3

A3 - Evaluation Measure – zoom

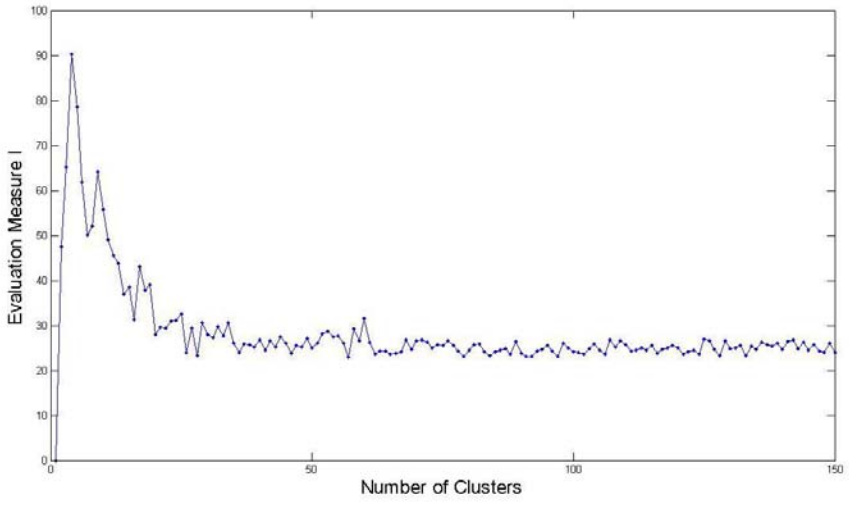


Figure 4.43: Evaluation measure I using K-means++ for $1 \leq K \leq 150$ (zoom in of previous plot), linguistic variable A3

The Scatter Search algorithm was run with the same values for the parameters α, β, *MaxIterImp* and b_1 as in the previous case study, considering 5 experiments for each choice of parameters and with the same stopping criterion as before. Once again, the number of good quality solutions and of diverse solutions to be included in the reference set was chosen to be equal ($b_1 = b_2$) and the original set was chosen to be 10 times the size of the reference set ($OS_{size} = 10*(b_1 + b_2)$), as in [Martí, Laguna et al. 1997; Abdule-Wahab, Monmarché et al. 2006].

Only the results relative to the linguistic variable A2 will be discussed in more detail.

To study the influence of α in the results, experiments were conducted with no improvement method (i.e. *MaxIterImp* $= 0$) and all possible combinations of $\alpha \in \{0; 0.5; 0.8; 1\}$, $\beta \in \{0; 0.5\}$ and $b_1 \in \{2; 5\}$. When $\alpha = 0$ the standard deviation of results is much higher than for other values of α (Figure 4.44), explaining why $\alpha = 0$ was responsible for the best results (Figure 4.45) but does not give competitive results in average (Figure 4.46). Due to the referred high standard deviation in the results, no further experiments were done with $\alpha = 0$.

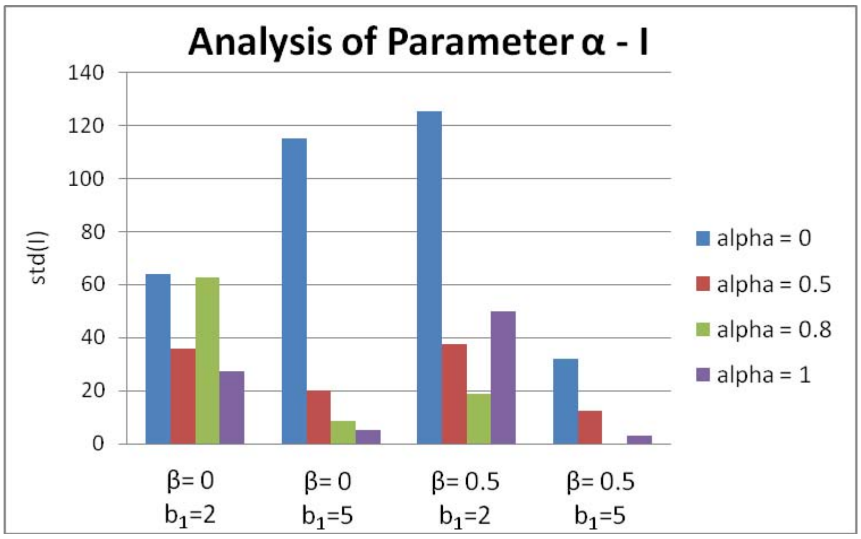


Figure 4.44: Influence of α in fitness function I standard deviation

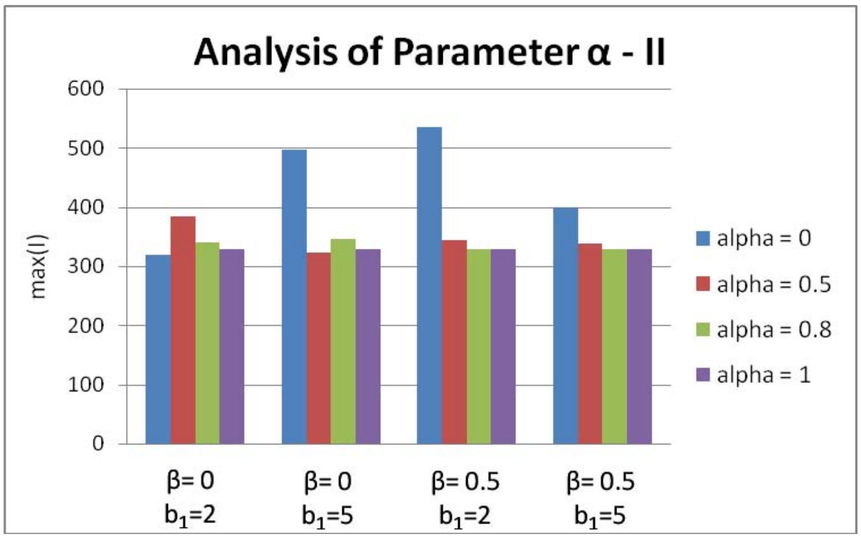


Figure 4.45: Influence of α in best results obtained for fitness function I

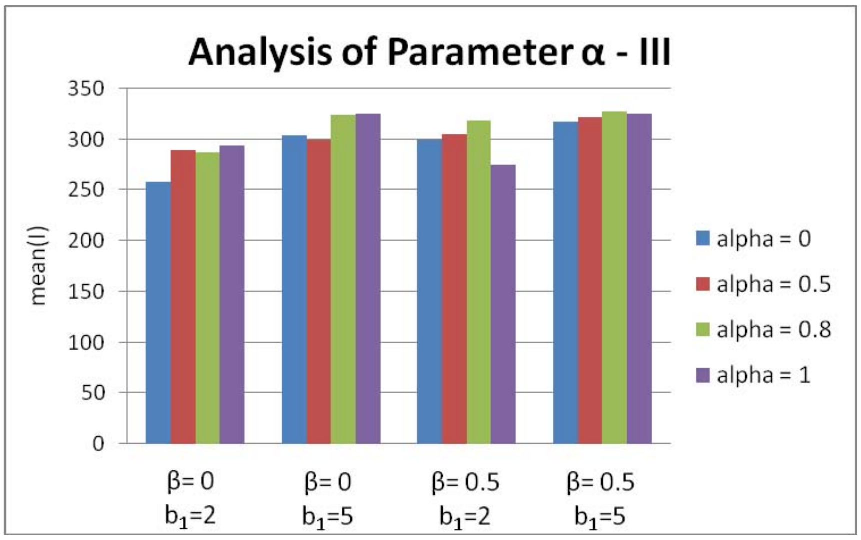


Figure 4.46: Influence of α in mean results for fitness function I

In Figure 4.47 the use of memory in the Diversification Generation method is clear since using $\beta = 0$ gives worse results in average. Therefore, no further experiments for $\beta = 0$ will be discussed.

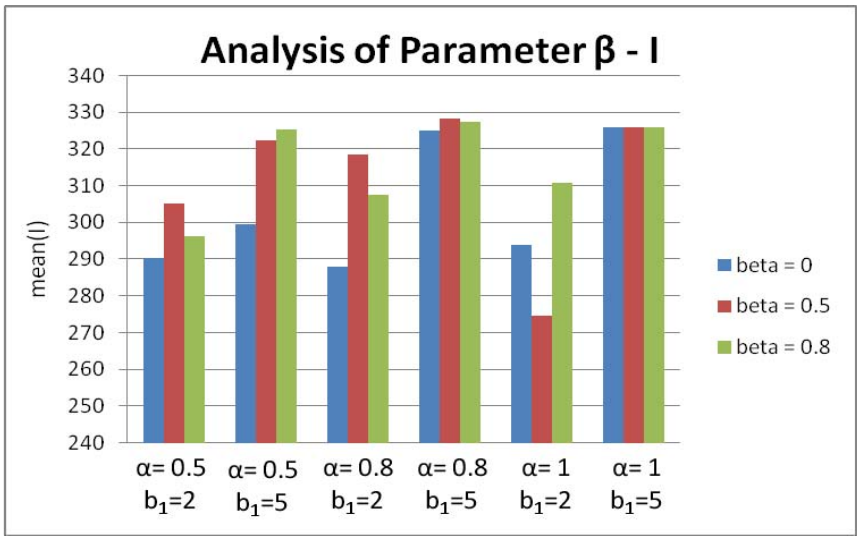


Figure 4.47: Influence of β in mean results for fitness function I

Increasing the size of the reference set produces better quality results, as can be seen in Figure 4.48. Increasing the number of solutions in the reference set from 4 to 10 ($b_1 = 2$ to $b_1 = 5$.) resulted in an increase in the execution time (Figure 4.49) by a factor between 1.6 and 2.9. Still, all experiments were run in less than 10 minutes. It is still affordable to consider a larger reference set to improve the overall quality of solutions.

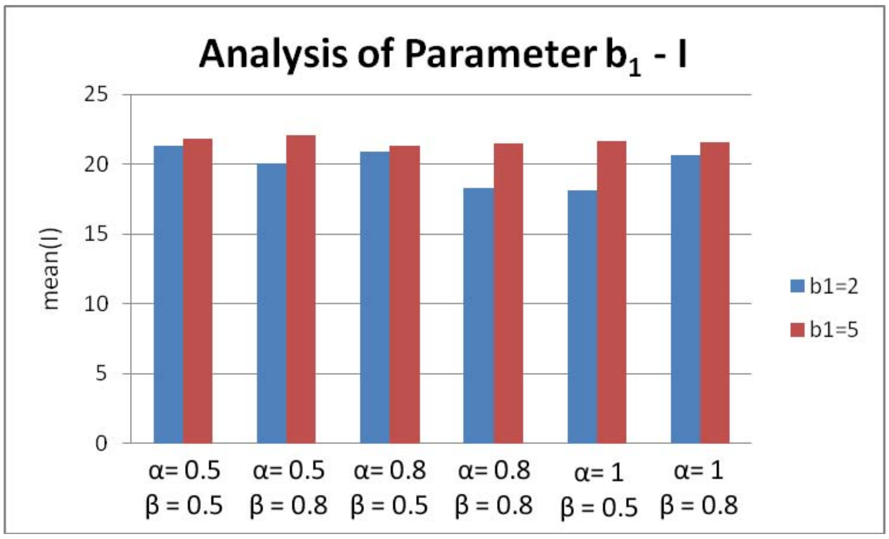


Figure 4.48: Influence of b_1 in mean results for fitness function I

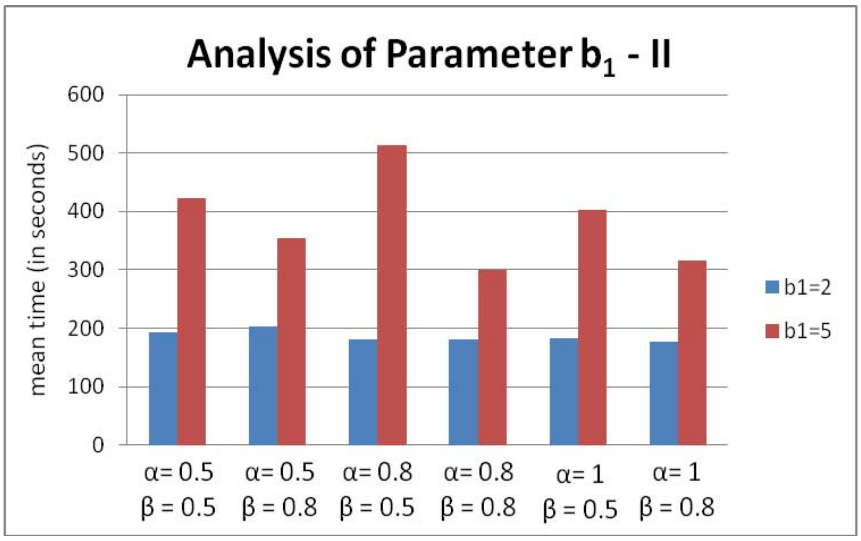


Figure 4.49: Influence of b_1 in execution time

Just as before, the increase in execution time when using *MaxIterImp* $= 5$ is considerably high (Figure 4.51) and does not result in a significant improvement in the quality of solutions in most of the cases (Figure 4.50). Since the increase in execution time for using *MaxIterImp* $= 2$ is not too high (from approximately 6 minute to approximately 10 minutes), this value will be considered for this parameter. Just as in the previous case study, the choice of parameters $\alpha = 0.5$, $\beta = 0.8$, $b_1 = 5$ and *MaxIterImp* $= 2$ gives better results, in average, and were used to obtain the results summarized in Table 4.2 and Figure 4.52 through Figure 4.57.

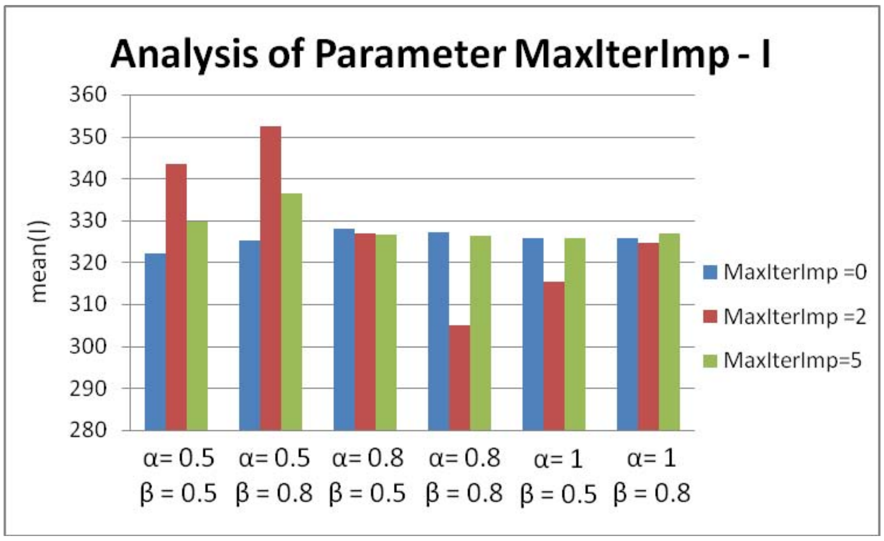


Figure 4.50: Influence of Improvement Method in mean results for fitness function I

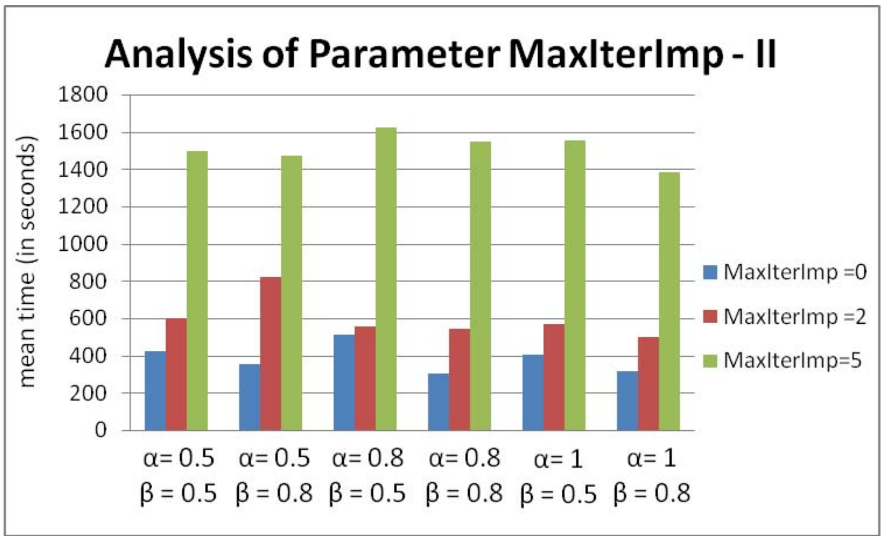


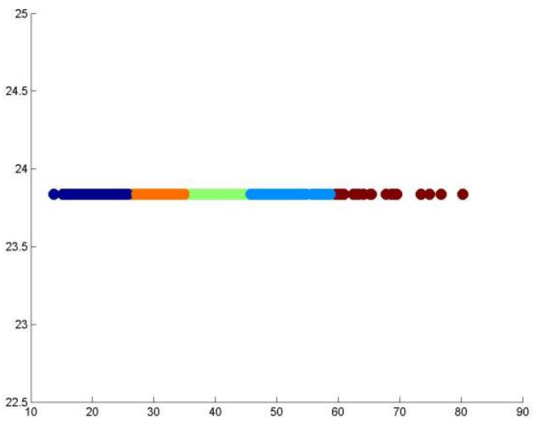
Figure 4.51: Influence of Improvement Method in execution time

Once again, the best solution found by the algorithm was always generated by the diversification generation method. The results reported in Table 4.2 were achieved only by the first stage of the algorithm, which still was sufficient to produce better results than the K-Means++ algorithm in most variables. Notice that both algorithms returned the same solution (except for cluster numbering) with 23 clusters for linguistic variable A11. This can be explained with information about this linguistic variable. Although A11 is continuous, there are only 23 different values for this variable. The solution returned by the algorithm corresponds to the 23 different membership functions corresponding to the 23 crisp values. This result show that the two algorithms are not giving “fake” low numbers of clusters (until now the number of clusters varied from 3 to 6) but are indeed capable of estimating a good number of clusters.

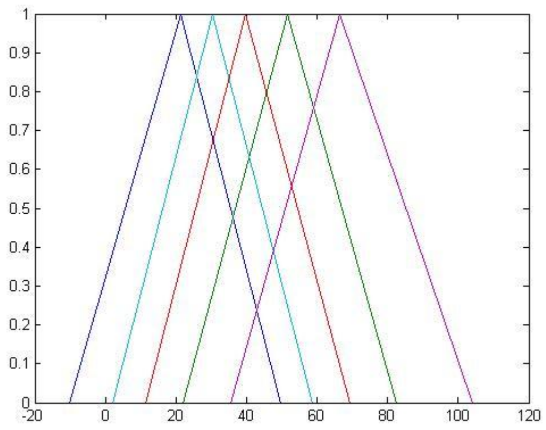
	K-Means++			Scatter Search		
	Time (sec.)	Nr. Clusters	I	Time (sec.)	Nr. Clusters	I
A2	854.5374	5	266.0796	821.3367	3	384.8384
A3	862.5715	4	90.18299	794.977	4	96.4486
A8	810.7072	6	105.4446	1277.734	6	94.76789
A11	735.1273	23	6.97E+28	1112.77	23	6.97E+28
A14	1134.509	6	346890.2	1812.912	4	402917.4
A15	1062.561	6	5.51E+09	2031.943	6	5.57E+09

Table 4.2: K-Means++ vs Scatter Search (best results)

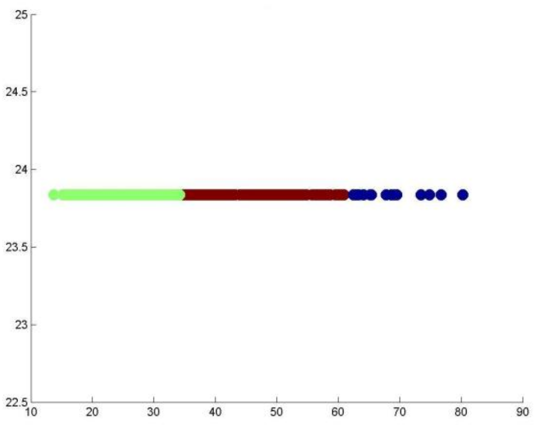
A2



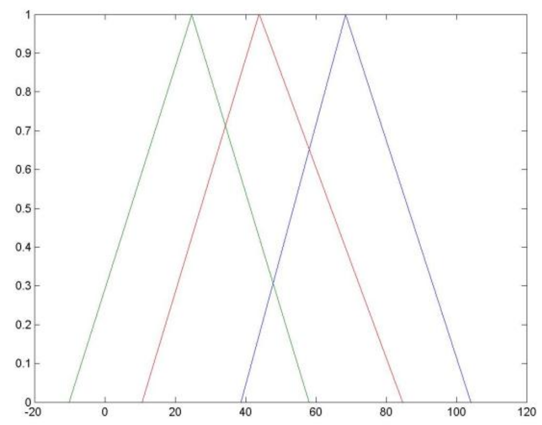
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



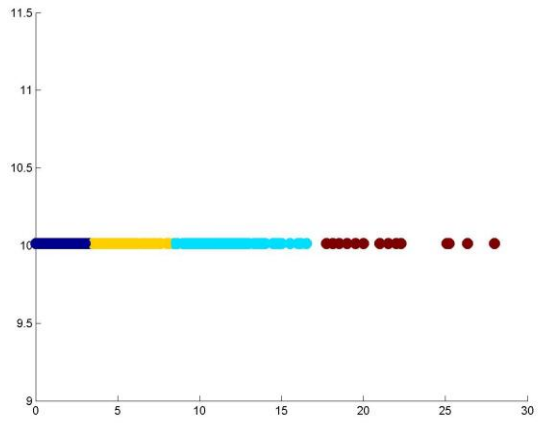
(c) Clusters with Scatter Search

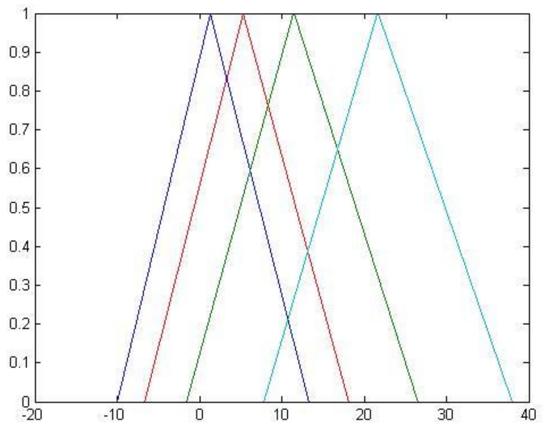


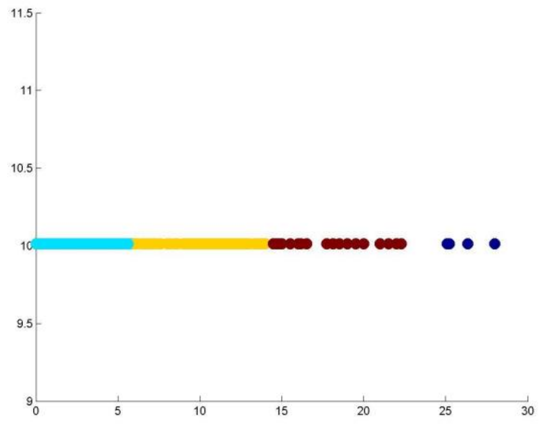
(d) Linguistic Variable after Scatter Search

Figure 4.52: Best Results for Linguistic Variable A2

A3


(a) Clusters with K-Means++


(b) Linguistic Variable after K-Means++


(c) Clusters with Scatter Search

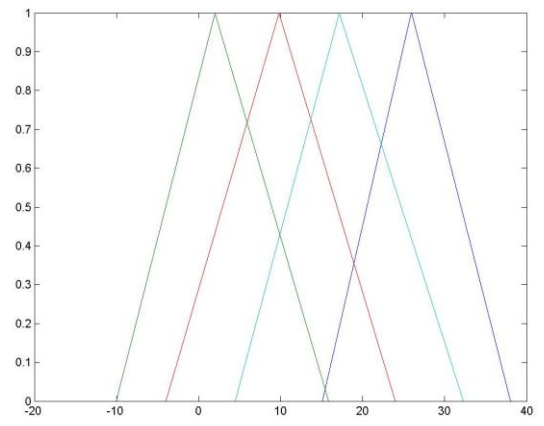
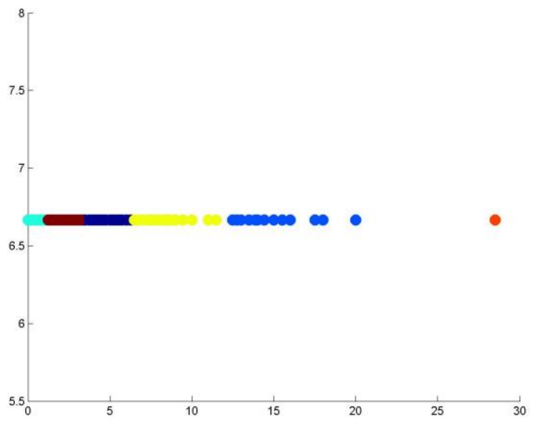
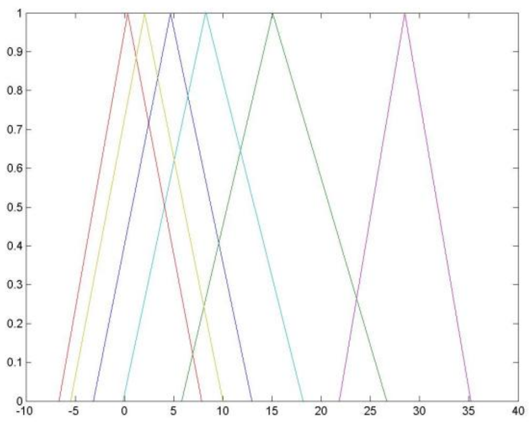

(d) Linguistic Variable after Scatter Search

Figure 4.53: Best Results for Linguistic Variable A3

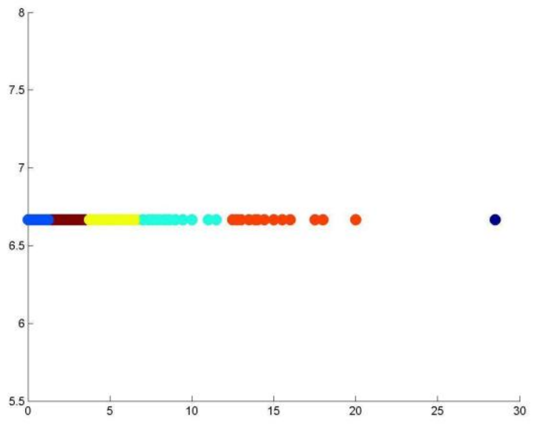
A8



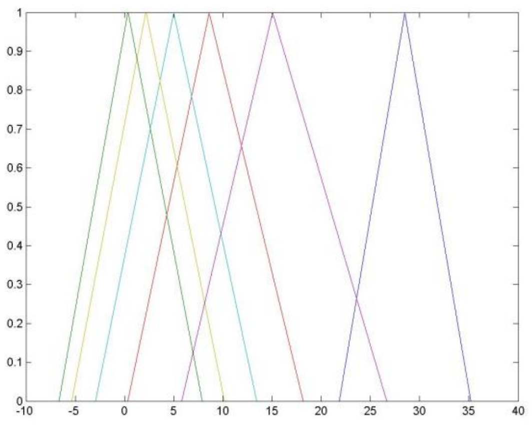
(a) Cluters with K-Means++



(b) Linguistic Variable after K-Means++



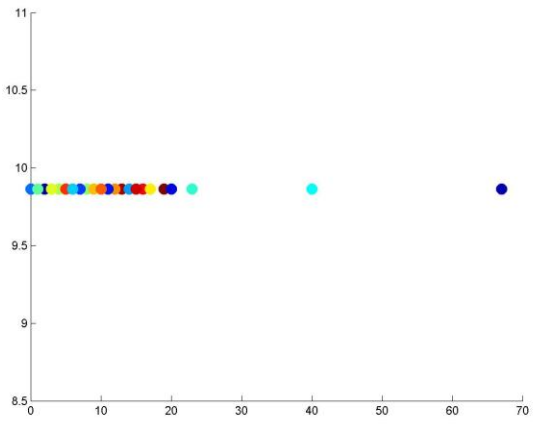
(c) Clusters with Scatter Search



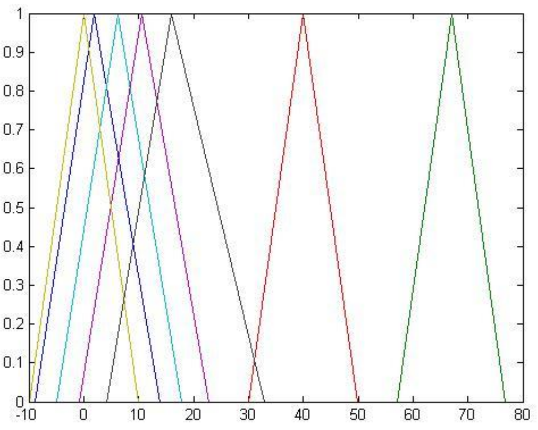
(d) Linguistic Variable after Scatter Search

Figure 4.54: Best Results for Linguistic Variable A8

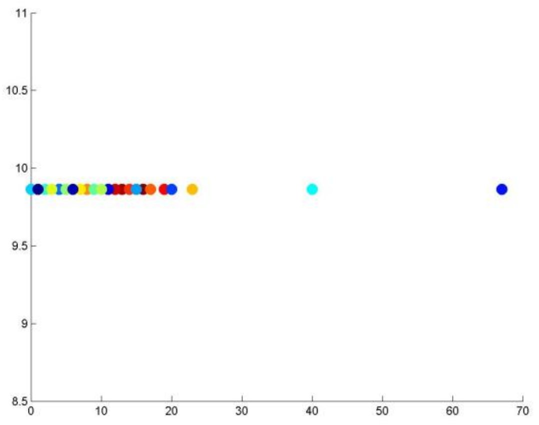
A11



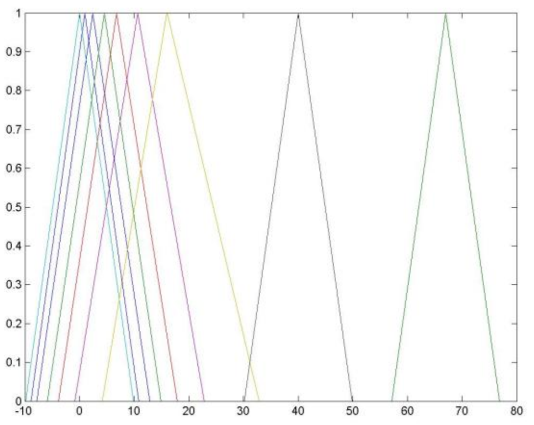
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



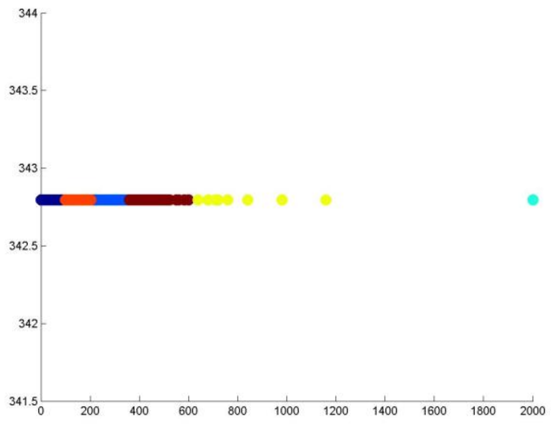
(c) Clusters with Scatter Search



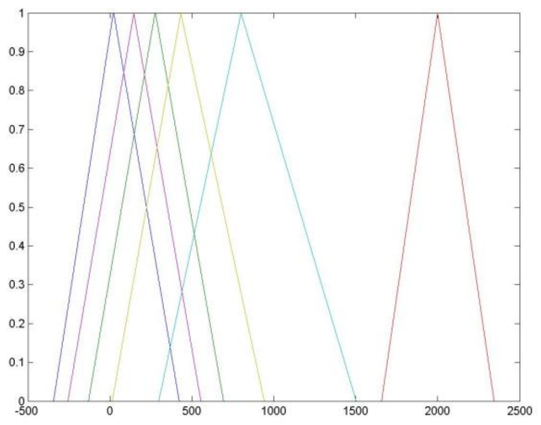
(d) Linguistic Variable after Scatter Search

Figure 4.55: Best Results for Linguistic Variable A11

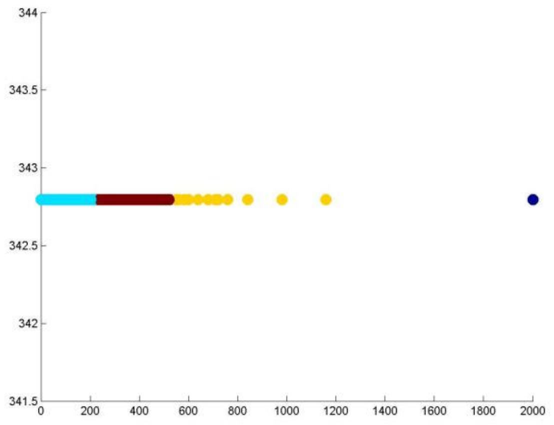
A14



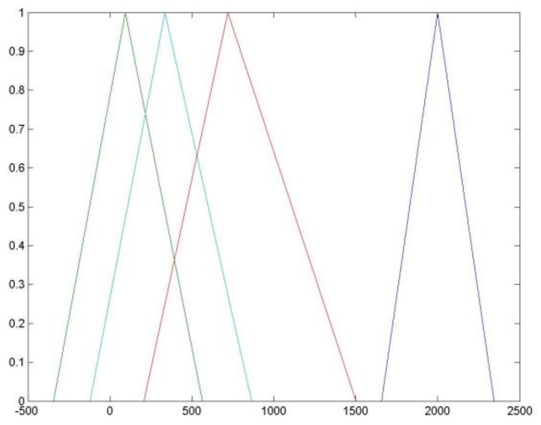
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



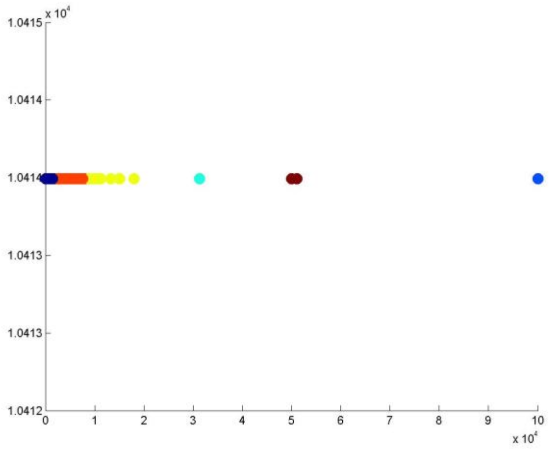
(c) Clusters with Scatter Search



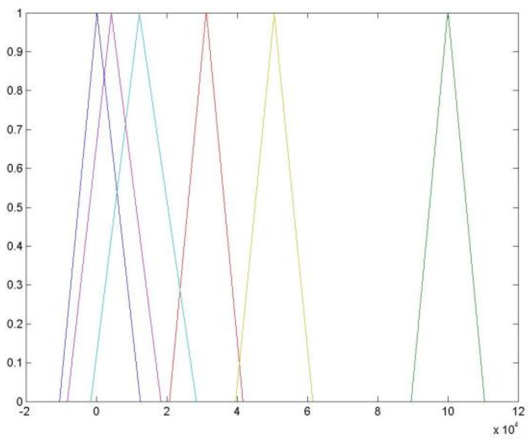
(d) Linguistic Variable after Scatter Search

Figure 4.56: Best Results for Linguistic Variable A14

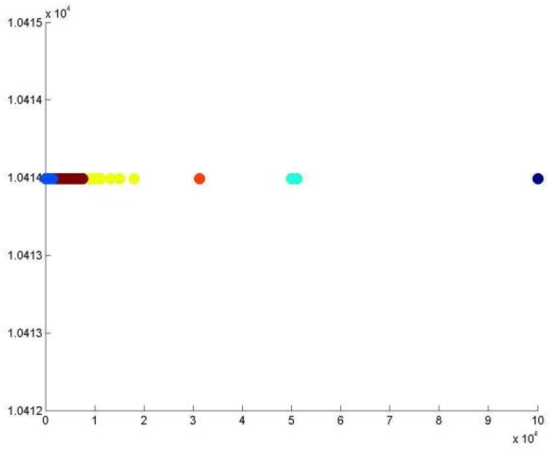
A15



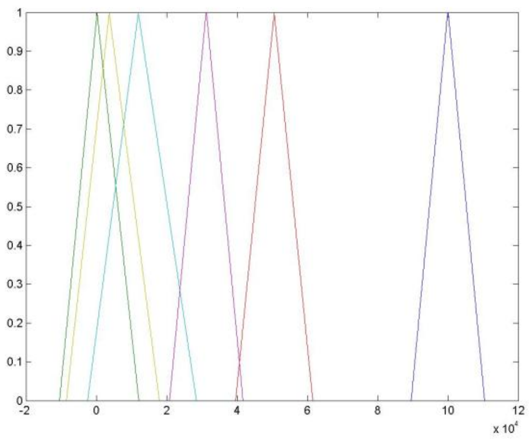
(a) Clusters with K-Means++



(b) Linguistic Variable after K-Means++



(c) Clusters with Scatter Search



(d) Linguistic Variable after Scatter Search

Figure 4.57: Best Results for Linguistic Variable A15

4.4 Summary

In this Chapter an heuristic and a metaheuristic for clustering were described. The first is the K-Means++ algorithm [David and Sergei 2007], a variation of the K-Means algorithm [Gan, Ma et al. 2007]. The second is a Scatter Search algorithm based on the work of [Pacheco 2005] and [Abdule-Wahab, Monmarché et al. 2006]. These two algorithms where implemented in Matlab.

The computational results were not as expected. The second phase of the Scatter Search algorithm was not able to produce good quality solutions. However, the first part of the algorithm was sufficient to obtain better results than the ones given by the K-Means++ algorithm. Both methods achieved a high reduction in the number of membership functions in each linguistic variable.

Chapter 5. Case Study: a Fuzzy Inference System

In the previous chapter, computational results for two case studies were presented. In both cases the problem consisted in reducing the number of membership functions in linguistic variables that could later be used in the construction of an inference system. Each membership function represented a sample or an individual and merging similar membership functions could be seen as finding groups of individuals with similar characteristics. The rule base system that could be constructed afterwards would take into account the final morphology of the linguistic variables. The case study presented in this section is of a different nature. In this case, the goal is to reduce the number of membership functions in linguistic variables of an already existing inference system [Gomes, Santos et al. 2008]. The aim is to reduce the complexity of the inference system while maintaining its structure, without losing too much performance.

In 2001, the European Space Agency [ESA] launched the Aurora Programme whose main goal is the robotic and human exploration of the solar system [ESA 2008]. ExoMars, one of the missions under this programme, will require the drilling and sampling of Martian rocks [ESA 2008]. The case study here presented has been developed at CA3 - UNINOVA [CA3 2006] in the scope of this programme.

In section 5.1 the case study will be described. Section 5.2 discusses the heuristics used for reducing the number of membership functions and in section 5.3 the computational results for the heuristics applied to this case study will be presented and discussed.

5.1 Overview of the case study: MODI

During project “MODI- Simulation of a Knowledge Enabled Monitoring and Diagnosis Tool for ExoMars Pasteur Payloads” [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006; Santos, Martins et al. 2008] two fuzzy inference systems were developed: one for an alarm system for detecting faulty behaviours during drilling in Mars and other for recognition of terrain hardness types. These inference systems were created automatically from signals first generated by a simulator and later in the project acquired from a drilling station developed during the project. Pictures of this drilling station prototype constructed as proof of concept during this project and a simulated image of the rover that might use this technology can be found in Figure 5.1 and Figure 5.2, respectively.

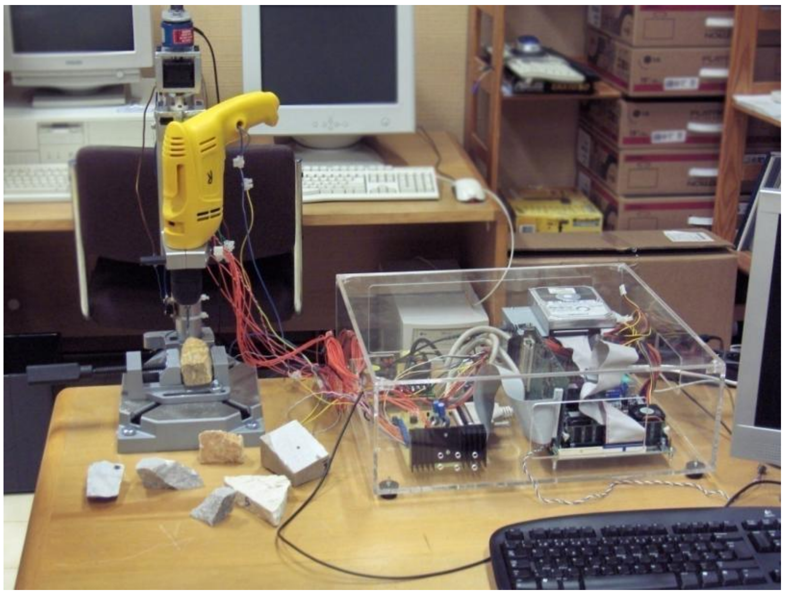


Figure 5.1: MODI drill station

Figure 5.2: ExoMars Rover (courtesy of ESA [ESA 2008])

The inference systems developed included two types of input variables: set points and sensor variables. The set points are variables whose values are pre-defined by the user to study the behaviour of the drill while drilling in different types of materials (rocks). Given the values for these two set points, the drilling process would start and sensors installed in the drill would measure the rest of the variables in our model.

As the project evolved, the sensors available increased. These sensors were able to measure rotational and translational currents and voltages, thrust, among other measures.

All linguistic variables that were created in the MODI project are trapezoidal membership functions, except the ones that describe the set points that are either triangular membership functions or singletons [Santos, Fonseca et al. 2008]. In the MODI project the linguistic variables (except the set points that are pre-defined by the user) were constructed automatically, using sensor data collected during a learning phase [Santos, Fonseca et al. 2008]. During the learning phase, drills in different types of terrain hardness, using different values for the set points, were performed. Each combination of values for the set points and terrain type defined a sub-scenario in our model. Each linguistic variable represents a different sensor and each term in a linguistic variable refers to a different sub-scenario. Trapezoidal membership functions for each sub-scenario and sensor were constructed taking into account the mean and standard deviation of the corresponding signal.

If we consider a drill with d sensors and a set of tests consisting in drilling in t different types of terrain with all possible combinations of sp_1 values for set point 1 and sp_2 values for set point 2, the model will have $d \times t \times sp_1 \times sp_2$ membership functions, excluding set points and output variables. As the number of sub-scenarios or number of sensors increases, so does the complexity of the inference system.

The output of the terrain recognition inference system is the terrain hardness for the t scenarios defined (T) and the certainty level of that classification [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006]. An example of a rule used for scenario Concrete hardness type, sub-scenario 0 (C0) with 2 set points – Set Point Rotation Speed (SPRS) and Set Point Translational Speed (SPTS) – and 8 sensor variables – Rotation Current (RC), Rotation Voltage (RV), Rotation Speed

(RS), Thrust (TH), Torque (TO), Translational Voltage (TV), Translational Current (TC) and Translational Speed (TS) - is shown bellow. There is one such rule in the system for every sub-scenario considered. In the following rules *Variable_Name – Sub-scenario_code* is the fuzzy set representing the nominal situation in variable *Variable_Name* and sub-scenario *Sub-scenario_code*.

If

Set Point Rotation Speed is SPRS-C0 and Set Point Translational Speed is SPTS-C0 and Rotation Current is RC-C0 and Rotation Voltage is RV-C0 and Rotation Speed is RS-C0 and Thrust is TH-C0 and Torque is TO-C0 and Translational Voltage is TV-C0 and Translational Current is TC-C0 and Translational Speed is TS-C0

Then

Terrain is T-C0

The output of the alarm inference system is the alarm level, on a scale from 0 to 1 [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006]. An example of a rule corresponding to the variables and sub-scenario above is shown bellow.

If

Set Point Rotation Speed is SPRS-C0 and Set Point Translational Speed is SPTS-C0 and (Rotation Current is not RC-C0 and Rotation Voltage is not RV-C0 and Rotation Speed is not RS-C0 and Thrust is not TH-C0 and Torque is not TO-C0 and Translational Voltage is not TV-C0 and Translational Current is not TC-C0 and Translational Speed is not TS-C0

Then

Alarm Level

In this thesis only the terrain recognition system will be used as a test case. The same analysis could be done to the monitoring system, using appropriate measures of performance of the inference system.

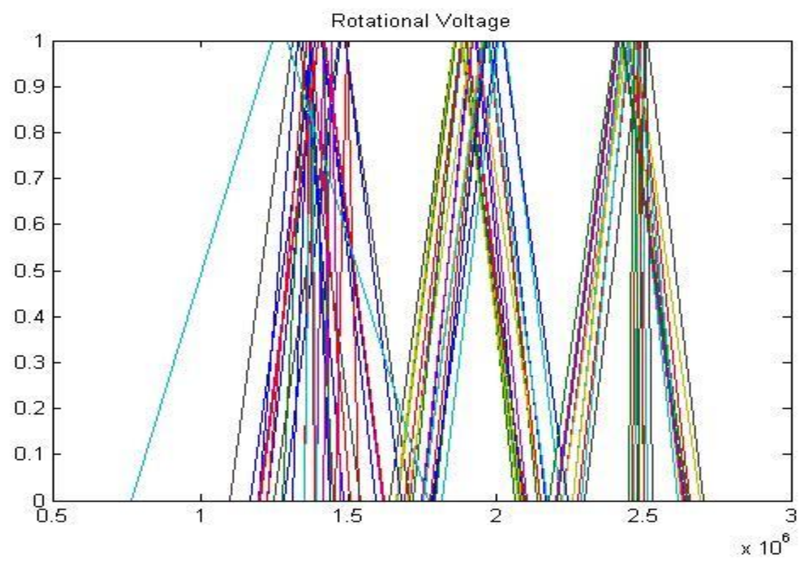


Figure 5.3: Example of linguistic variable – Rotational Voltage

Considering that an automatic process was used to create the terms for the linguistic variables, in Figure 5.3 we can see that many terms of this linguistic variable overlap heavily or are even included in others. It is also easy to observe some implicit clusters of similar membership functions. Merging the membership functions, pertaining to a cluster, into a single one seems an obvious way of reducing the number of membership functions in the system.

The contribution to this project was to define an algorithm capable of reducing the number of terms of a linguistic variable to improve the overall computational effort of the system without compromising the performance of the system.

		Classified							
		Concrete	GasConcrete	Marble	NotDrilling	Travertine	Tuff	Unknown	Total
Real	Concrete	299	0	0	0	0	0	242	541
	GasConcrete	0	267	0	0	26	0	229	522
	Marble	0	0	296	0	0	0	226	522
	NotDrilling	0	0	0	210	0	0	311	521
	Travertine	0	21	0	0	244	0	259	524
	Tuff	0	0	0	0	0	271	251	522
	Total	299	282	296	210	270	271	1529	**3152**

Table 5.1: MODI Confusion Matrix

The measures of performance used to compare the resulting terrain recognition inference systems were the Precision of the classification [Santos, Fonseca et al. (to appear 2008)], the Mean Certainty Level and a combination of these two measures. Consider the confusion matrix in Table 5.1 that shows the hardness types of terrain of a test data set and its classification with the MODI inference system.

The overall Precision (P) of the terrain recognition inference system is the ratio between the number of well classified samples (grey cells) and the total number of samples [Santos, Fonseca et al. 2008]. For the confusion matrix in Table 5.1 the precision would be 0.5035 (50.35%). The Mean Certainty Level (MCL) is the average of the certainty levels obtained for each sample, for the samples that were correctly classified. To combine these two measures of performance, it is used an average of these two (based on the F1 score [Rijsbergen 1979]) given by:

$$F = \frac{2 \cdot P \cdot MCL}{P + MCL} \tag{5.1}$$

These three measures of performance take values between 0 and 1 and are to be maximized.

5.2 Heuristics

In [Setnes, Babuska et al. 1998], Setnes presents a rule base simplification algorithm that can be summarized by the fluxogram in Figure 5.4. Given a fuzzy variable with initial membership functions M and a threshold minS for the similarity (set by the user or by some other algorithm), we select the most similar pair of fuzzy sets A and B (for a certain fuzzy similarity measure). If this similarity value is above the previously defined threshold (minS), we combine these two fuzzy sets, update the rule-base and the algorithm proceeds by choosing the most similar pair in M. Otherwise, the algorithm stops and M is returned.

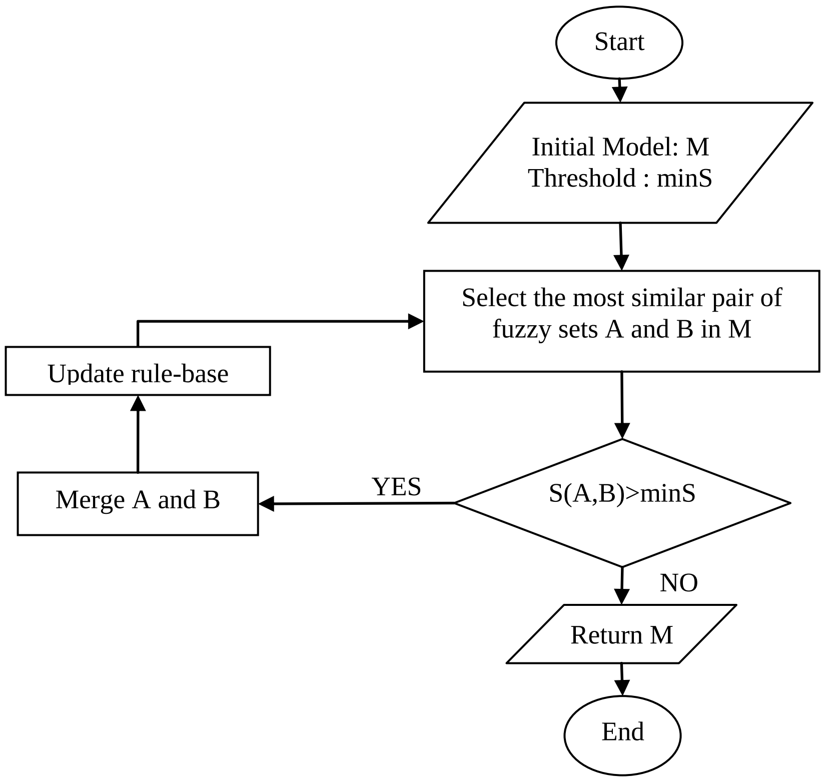


Figure 5.4: Original algorithm [Setnes, Babuska et al. 1998]

Instead of basing the decision of merging or not two membership functions A and B only on their similarity, it was decided to look at the differences in terms of "design" between the original model and the ones achieved each iteration. This way a more global view of the changes being made to the linguistic variable being pruned was done. Therefore it was defined a measure for comparing two sets of membership functions representing the same linguistic variable. This way it would be possible to compare the models obtained through the algorithms to the initial model or linguistic variable.

A distinguishability measure Δ between fuzzy sets can be defined as the complement of their similarity measure [Mencar, Castellano et al. 2007].

$$\Delta(A,B) = 1 - S(A,B) \tag{5.2}$$

Given a new set of membership functions (obtained, for instance, by merging two or more membership functions of the original set) and the correspondence between the new set and the original one, we can define what can be intuitively seen as a model error by taking the average distinguishability measures between all original sets and the one that represents them in the new set. Consider the example in Figure 5.5:

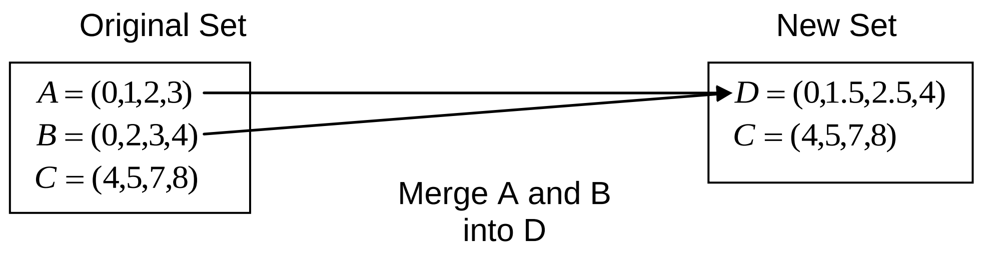


Figure 5.5: Example of model error

Using the Jaccard similarity measure [Mencar, Castellano et al. 2007] given by equation (1.19) with the pair (minimum, maximum) as intersection and union operators we obtain:

$$S(A,D) = 0.6364 \qquad S(B,D) = 0.8182 \qquad S(C,C) = 1 \tag{5.3}$$

Thus,

$$\begin{aligned} ModelError &= \frac{\Delta(A,D) + \Delta(B,D) + \Delta(C,C)}{3} = \\ &= \frac{(1-0.6364) + (1-0.8182) + (1-1)}{3} = 0.1818 \end{aligned} \tag{5.4}$$

The adapted algorithm can then be summarized by the fluxogram in Figure 5.6. Given a fuzzy variable with initial linguistic variables M and a threshold ε for the model error (set by the user or by some other algorithm), we select the most similar pair of fuzzy sets A and B (for a certain fuzzy similarity measure) and combine these two fuzzy sets, thus obtaining a new set of linguistic variables, M'. If the model error, denoted by ε, is above the threshold or if all pairs of fuzzy sets are totally dissimilar (similarity zero) the algorithm stops. Otherwise, the algorithm proceeds by choosing the most similar pair in M'.

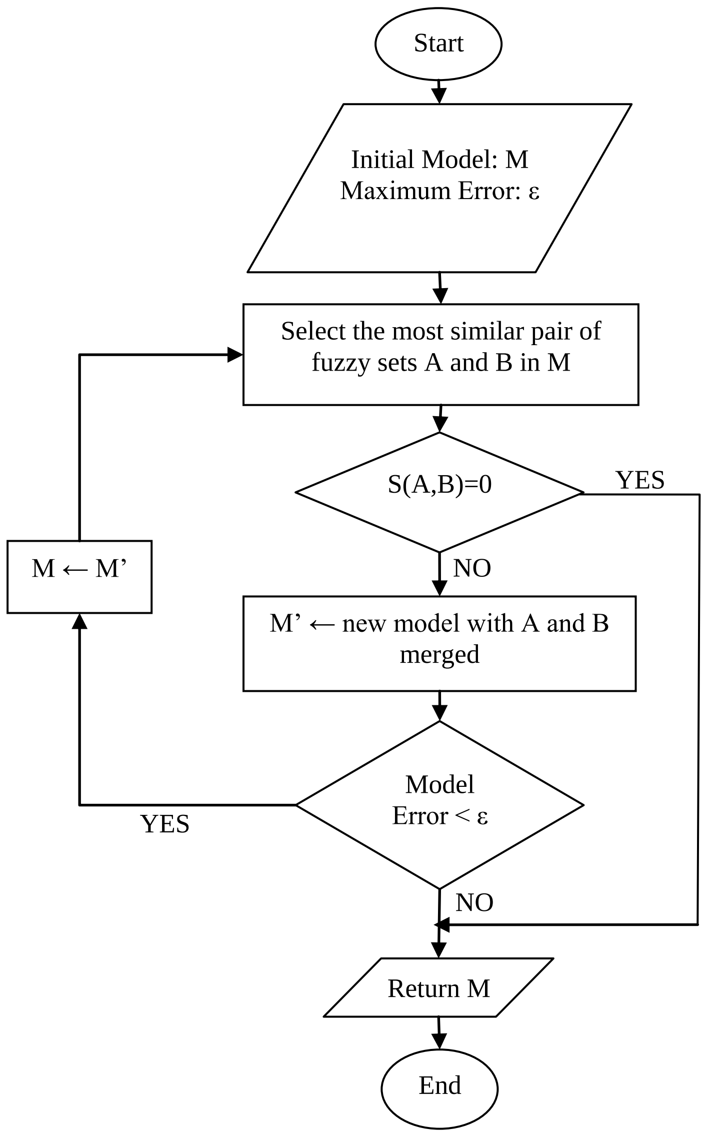
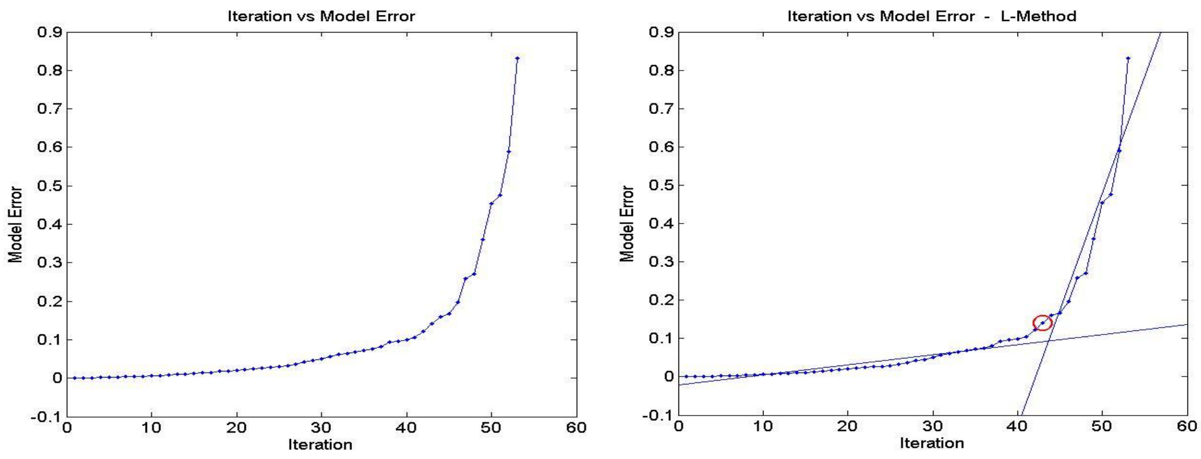

Start
Initial Model: M
Maximum Error: ε
Select the most similar pair of fuzzy sets A and B in M
S(A,B)=0
YES
NO
M ← M'
M' ← new model with A and B merged
Model Error < ε
YES
NO
Return M
End

Figure 5.6: Adapted algorithm

The above algorithm can be seen as a hierarchical clustering algorithm. We start with as many clusters as the initial number of membership functions and in each step we merge two clusters into a single one. Several methods of choosing the optimal number of clusters (and thus the optimal number of iterations) for this kind of clustering algorithms have been proposed [Milligan and Cooper 1985; Salvador and Chan 2004]. Some consist of finding the knee of a curve obtained by representing the number of iterations or number of clusters versus some metric of evaluation of the clustering algorithm. Since using the inference system performance measure (5.1) as evaluating metric during the clustering would be time consuming, it was decided to use the model error instead. Running the adapted algorithm using the

Jaccard similarity measure in (1.19) with the minimum and maximum operators until all membership functions are disjoint and plotting each iteration number versus its model error, gave a curve like the one in Figure 5.7 a). The Model Error increases slowly with each iteration until it reaches a point where it starts to grow exponentially. We can choose the number of iterations to be the x-axis coordinate of that point. This way, we will have an automatic method for choosing the number of iterations of the algorithm and we will no longer need to heuristically choose the parameter ε.



a) Iteration vs Model Error b) L-Method

Figure 5.7: a) Iteration vs Model Error; b) L-method

To find the knee of this curve the L-method, proposed by Salvador and Chan in [Salvador and Chan 2004], was used. Let n be the number of the last iteration, that is, $x = 1, \ldots, n$. Let L_c and R_c be the left and right sequences of data points partitioned at $x = c,\ c = 2, \ldots, n-2$. Fitting a line to L_c and another to R_c we can define the total root mean square error (RMSE) by:

$$RMSE_c = \frac{c}{n} RMSE(L_c) + \frac{n-c}{n} RMSE(R_c) \qquad (5.5)$$

i.e., the total root mean squared error is a weighted average of the root mean squared error of both fittings. The optimal number of iterations is the value c that minimizes $RMSE_c$. In Figure 5.7 b), c is the number of chosen iterations corresponding to the point circled ($c = 43$). The c value is the stopping criterion threshold used in the adapted algorithm.

As will be seen through the computational results presented in section 5.3, although this approach improves the original inference system used as a test case, the use of the L-method as a stopping criterion is not the best choice. By penalizing dissimilarity, it assumes that the initial model is the one with best performance, which is not the case in this case study. Since the original inference system was obtained automatically from sensor data, it is full of redundancy. The stopping criterion should not be based on a comparison in terms of “design” to the original model but on the actual performance measures of the inference system.

The final algorithm, called bestP, is illustrated in Figure 5.8.

We ran the original algorithm (Figure 5.4) until all membership functions are disjoint (similarity zero). In each iteration, the rules are updated and the performance of the resulting inference system, P(M), is obtained and compared with the performance of the best model found so far, P(BestM). The bestP algorithm returns the inference system with best performance, from the ones generated throughout the iterations. Note that this algorithm is defined for any performance measure, P(.). In our case study, we used the performance measure F given by (5.1). This means that we will be maximizing the inference system performance. If the initial system is the best system in terms of this measure of performance, there will be no reduction in the number of membership functions. If we want to find a compromise between the number of membership functions and the inference system performance we can use a linear combination of these two objectives in the following way:

$$P(M) = \alpha F - \frac{1-\alpha}{n_0} n \tag{5.6}$$

where $\alpha \in [0,1]$ is the weight given to F, n_0 is the initial number of membership functions, used as a scaling factor and n is the number of membership functions of model M.

Finally, the adapted algorithm is applied sequentially to each input variable to be pruned (in this case, all input variables except the set points). After this pruning is completed, duplicate rules are removed from the rule system to improve the computational efficiency of the final inference system.

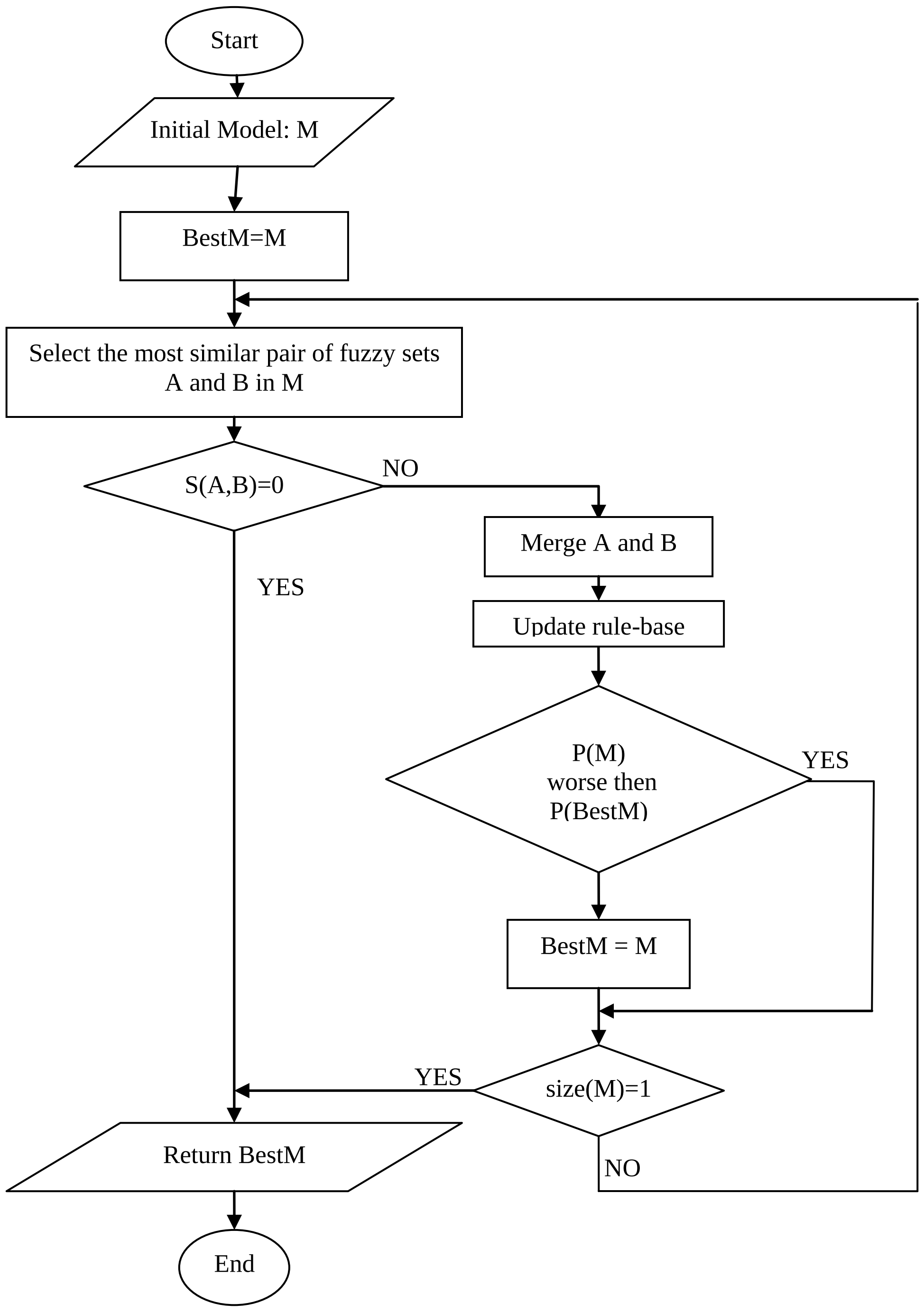


Figure 5.8: Final algorithm – bestP

5.3 Computational Results

The heuristic algorithms described in the previous section were applied to a terrain recognition inference system constructed as described in section 5.1. All experiments were done using Matlab and Java.

The set points used were:

1. Rotation Speed (rpm) - SPRS
2. Translational Speed (mm / min) - SPTS

The sensor's variables used were:

1. Rotation Current (A) - RC
2. Rotation Voltage (V) - RV
3. Rotation Speed (rpm) - RS
4. Thrust (N) - TH
5. Torque (N) - TO
6. Translational Voltage (V) - TV
7. Translational Current (A) - TC
8. Translational Speed (mm/min) - TS

Six different types of terrain hardness were tested. The 6 scenarios considered were: Not Drilling (drilling in air); Concrete; Gas-Concrete; Marble; Travertine; and Tuff. Setting 3 different values for each of the set points, these scenarios were further sub-divided into 54 ($6\times3\times3$) sub-scenarios that provided the basis for the construction of the linguistic variables representing each input variable.

The original membership functions in this inference system are represented in Figure 5.9. In all linguistic variables in the system it is clear that some of its terms should be merged. The sensor values were collected as integers. Integer programming is more efficient and was thus used to guarantee real time tasks. Therefore, the x-axis of the plots representing the linguistic variables in the system have no physical meaning, i.e., they cannot be interpreted as voltages, rotations per minute, ...

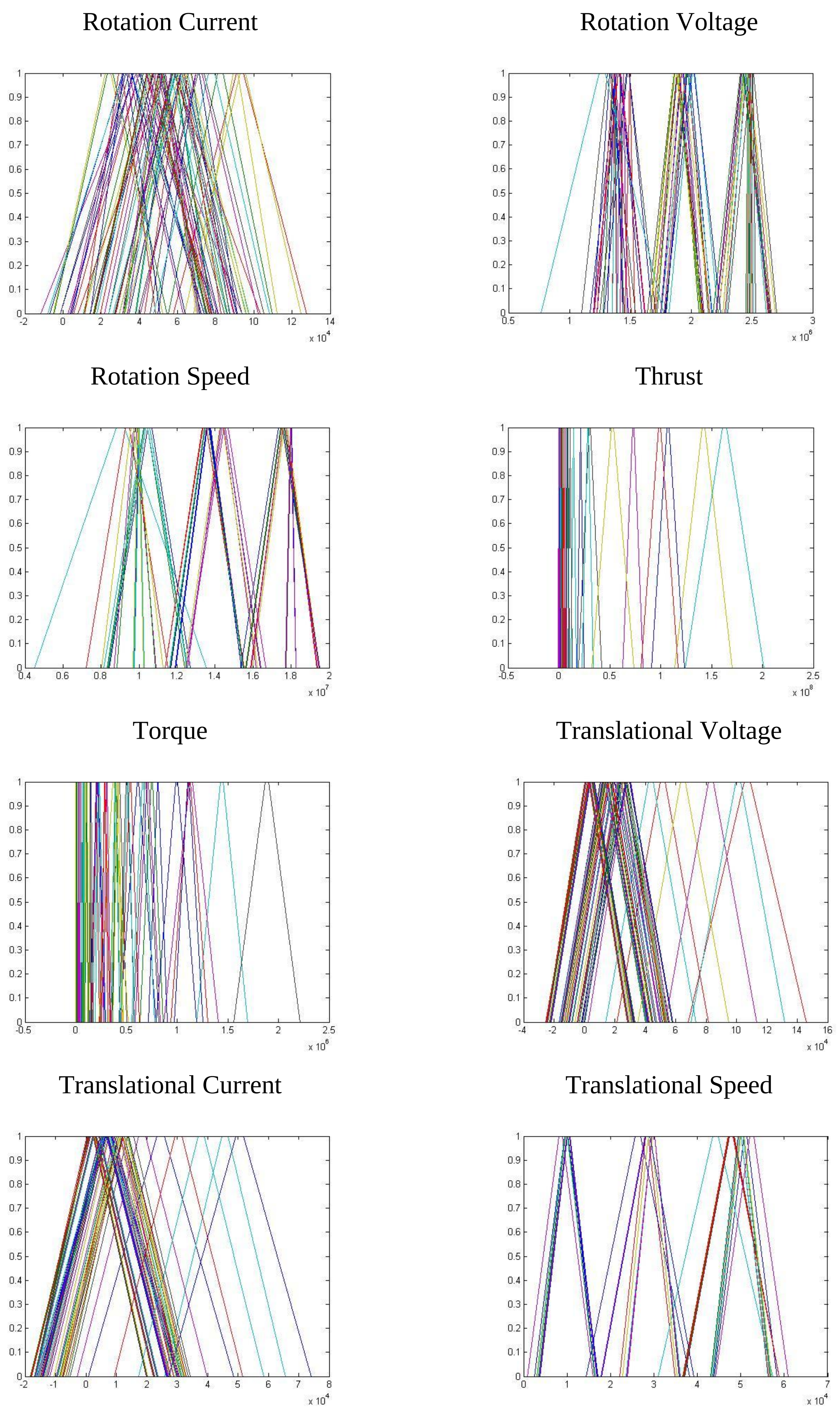


Figure 5.9: Original input linguistic variables (except set points)

First, the algorithm summarized by Figure 5.6 (adapted algorithm) with the stopping criterion defined by minimization of (5.5) – L-method [Salvador and Chan 2004] - was applied sequentially to all input variables except set points. Then, the algorithm summarized by Figure 5.8, were the inference system with best performance is chosen, was used in the same sequence. The following figures show the evolution of the performance measure F (to be maximized) given by equation (5.1) when the algorithms run until all membership functions are disjoint. The vertical lines mark the iteration chosen by the L-method and the circle marks the iteration with best performance, i.e., the iteration chosen by bestP.

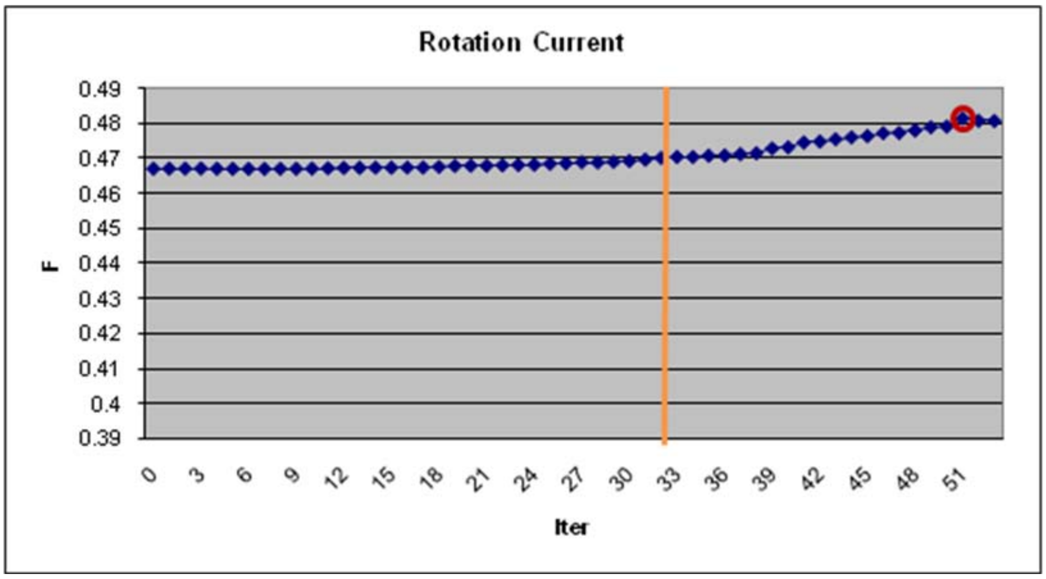


Figure 5.10: Evolution of performance measure F during the algorithm – Rotation Current

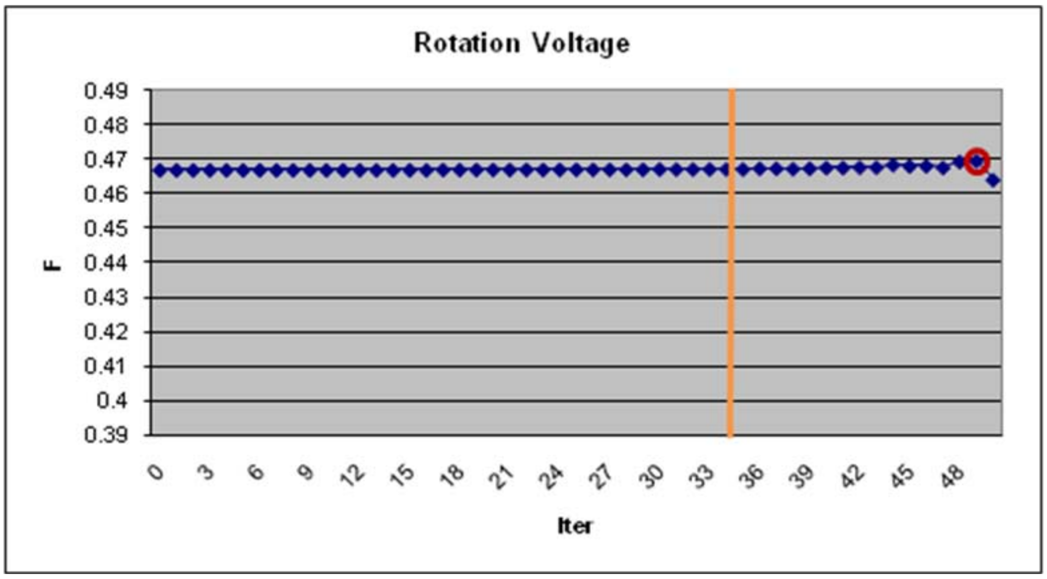


Figure 5.11: Evolution of performance measure F during the algorithm – Rotation Voltage

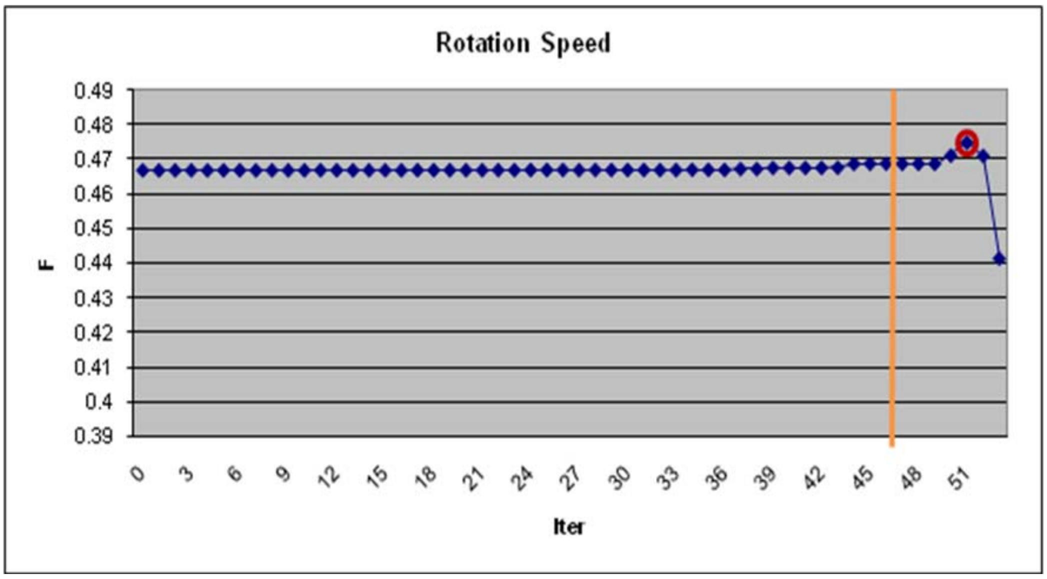


Figure 5.12: Evolution of performance measure F during the algorithm – Rotation Speed

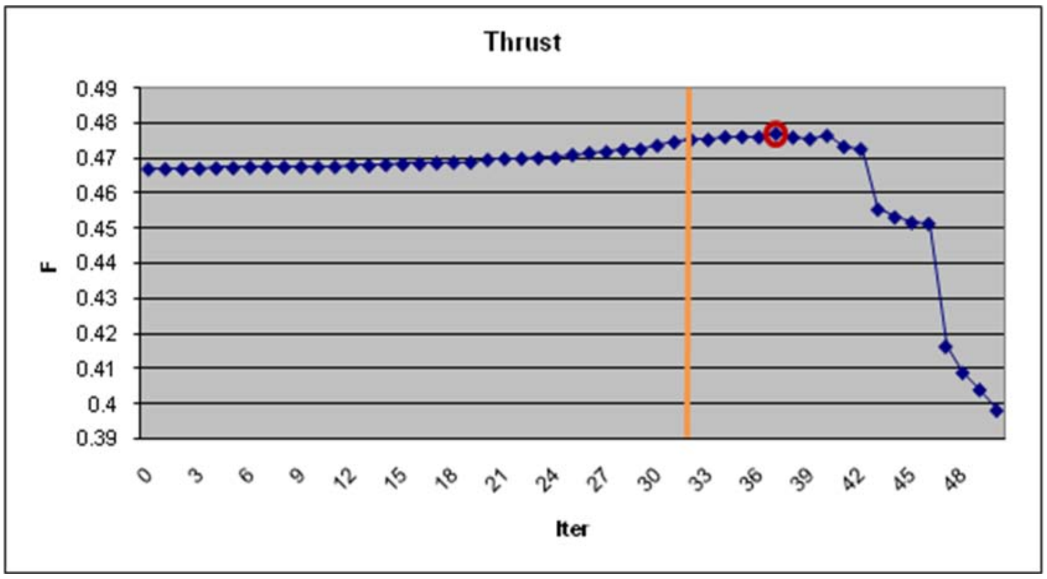


Figure 5.13: Evolution of performance measure F during the algorithm – Thrust

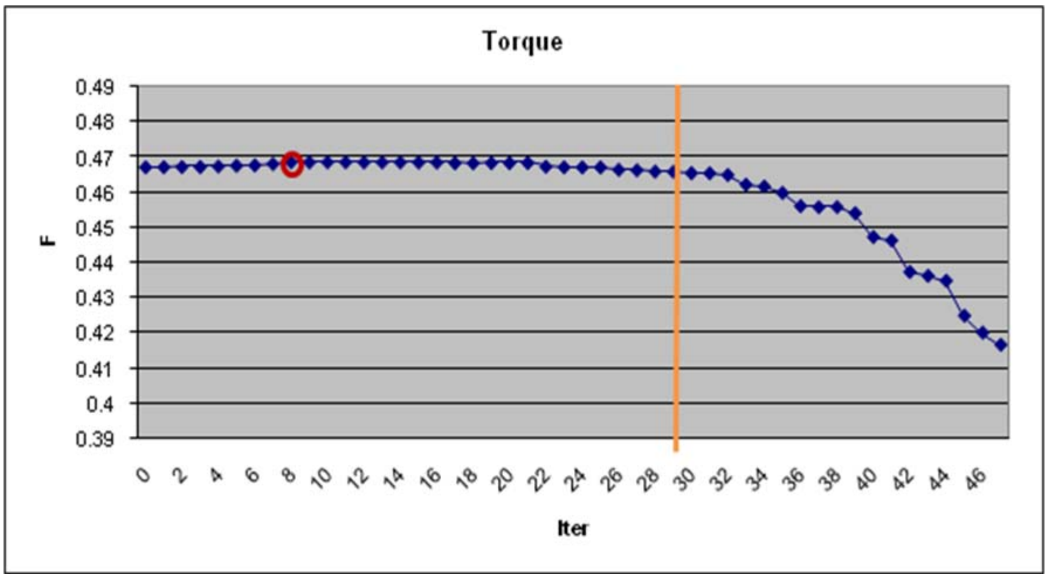


Figure 5.14: Evolution of performance measure F during the algorithm – Torque

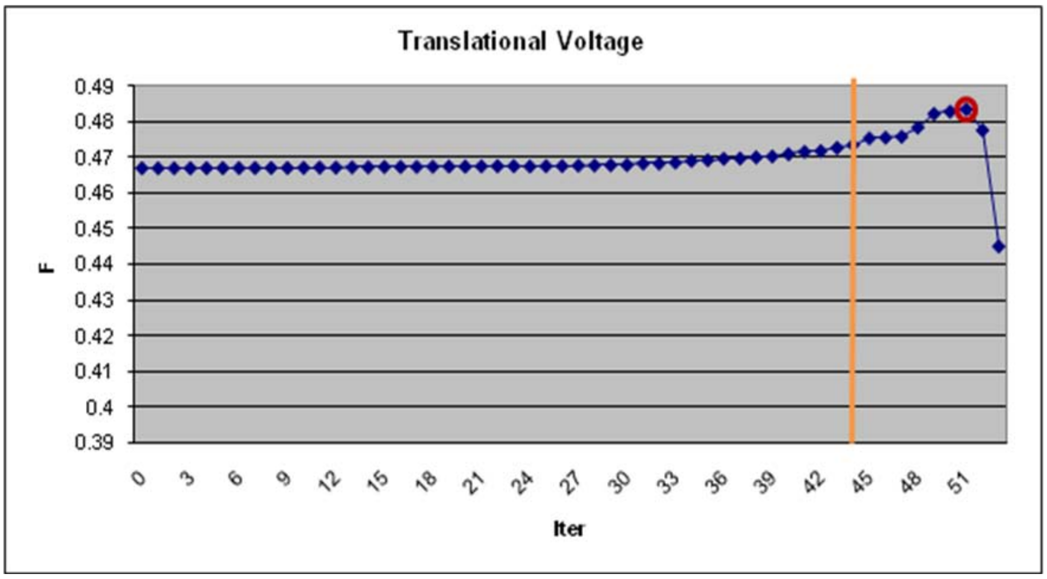


Figure 5.15: Evolution of performance measure F during the algorithm – Translational Voltage

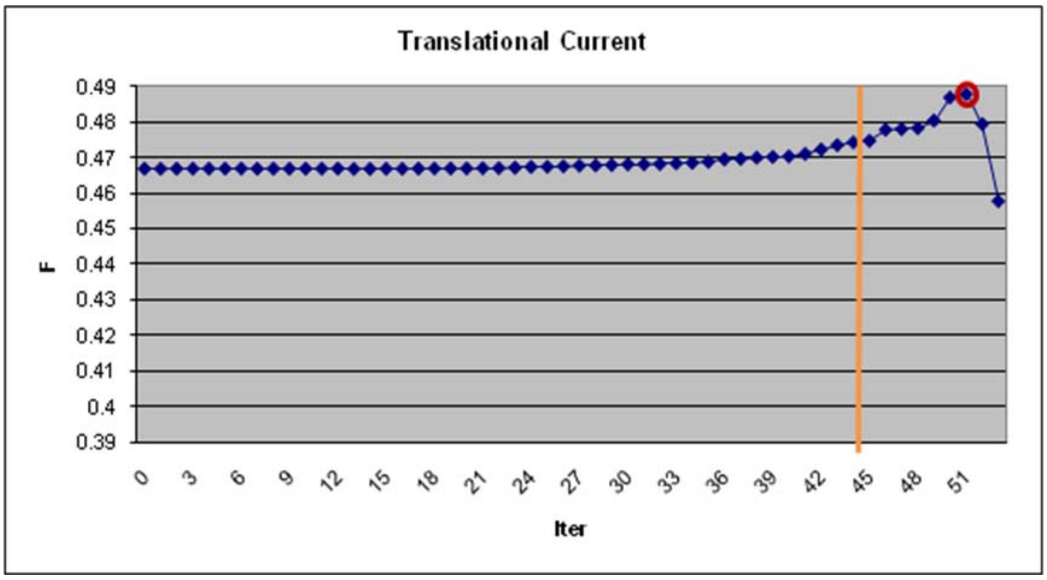


Figure 5.16: Evolution of performance measure F during the algorithm – Translational Current

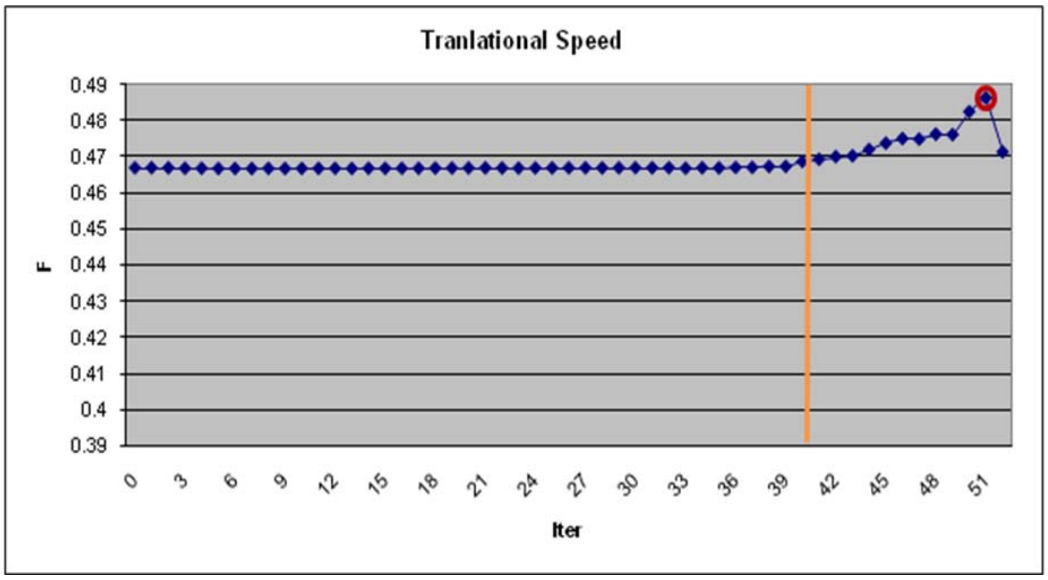


Figure 5.17: Evolution of performance measure F during the algorithm – Translational Speed

	Original	Adapted Algorithm	BestP
Rotation Current	54	21	3
Rotation Voltage	54	20	5
Rotation Speed	54	8	3
Thrust	54	22	17
Torque	54	25	46
Translational Voltage	54	10	3
Translational Current	54	10	3
Translational Speed	54	13	3
TOTAL	432	129	45

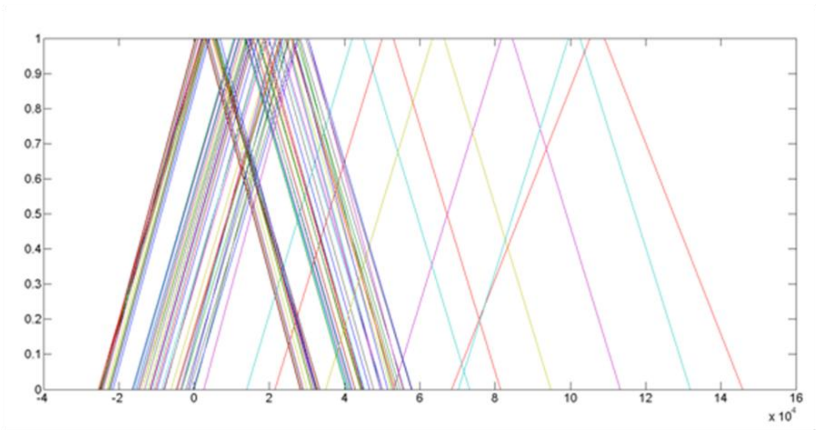
Table 5.2: Number of membership functions before and after optimization

As can be seen by the previous figures and by Table 5.2, using the L-method as a stopping criterion is not the best choice. In most cases this stopping criterion chooses to stop too early. Although in this case study this method already reduces the number of membership functions without losing performance, this method fails to choose the best iteration to stop. Analyzing the previous figures it is clear why concentrating on the model error instead of the inference system actual performance is not a good strategy. By minimizing the model error, changes to the original inference system are being penalized. If the original system was “perfect”, reducing the number of terms in linguistic variables by merging similar membership functions should negatively affect the performance of the system. The previous figures show that this is not the case, merging membership functions is actually increasing the inference system performance. This means that the initial system is full of redundancy and that some of this redundancy is being eliminated by the algorithm. For this reason, instead of using the model error, a comparison between the initial inference system and the ones obtained by the algorithm, it is wiser to focus on performance measures such as the ones described in the previous section. This is what motivated the use of the bestP algorithm.

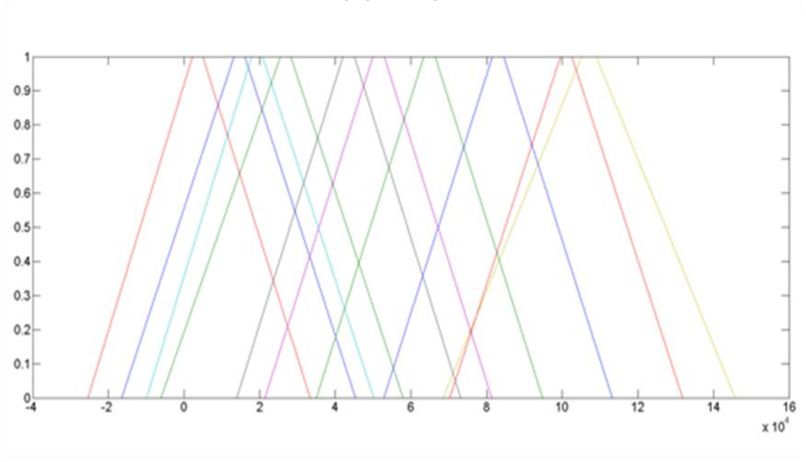
	Original	Adapted Algorithm	BestP
P	72.33%	76.49%	85.47%
MCL	34.49%	36.88%	44.00%
F	46.71%	49.77%	58.09%
N	423	129	45

Table 5.3: Comparison of inference systems

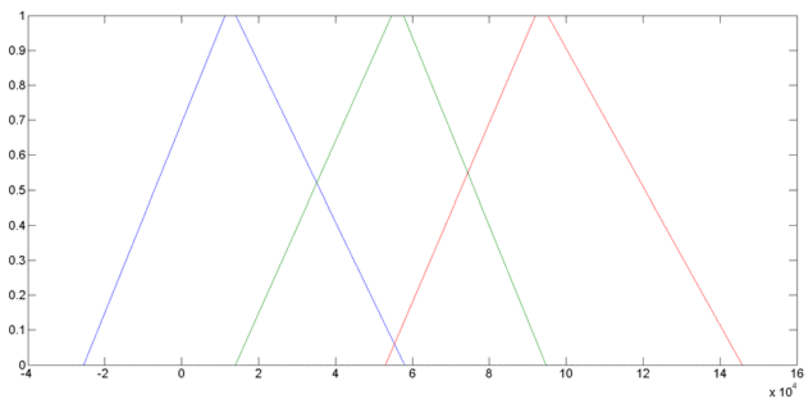
Table 5.3 compares the three inference systems – original inference system and inference systems obtained using the Adapted Algorithm and the BestP algorithm – in terms of systems performance measures and number of membership functions. Figure 5.18 shows the linguistic variable Translational Voltage before and after using both algorithms. By using the bestP heuristic it was possible to reduce the number of membership functions in the system and improve its overall performance. It was possible to reduce the number of membership functions in input linguistic variables (except set points) from 432 to 45. All measures of performance used improved after this optimization. The easiest to interpret performance measure, the Precision of the classification (P), increased from 72.33% to 85.47%. The advantages of using this algorithm in this case study was clear. If it was desirable to further reduce the number of membership functions a measure such as the one in equation (5.6) could be used. Figure 5.19 shows the linguistic variables returned by the algorithm. Some linguistic variables seem to be too reduced, i.e., its new linguistic variables do not seem well representative of the original linguistic variables. We have to keep in mind that the original system was created automatically and is full of redundancy.



(a) Original



(b) Adapted Algorithm



(c) BestP

Figure 5.18: Linguistic Variable Translational Voltage before (a) and after using the adapted (b) and BestP (c) algorithms

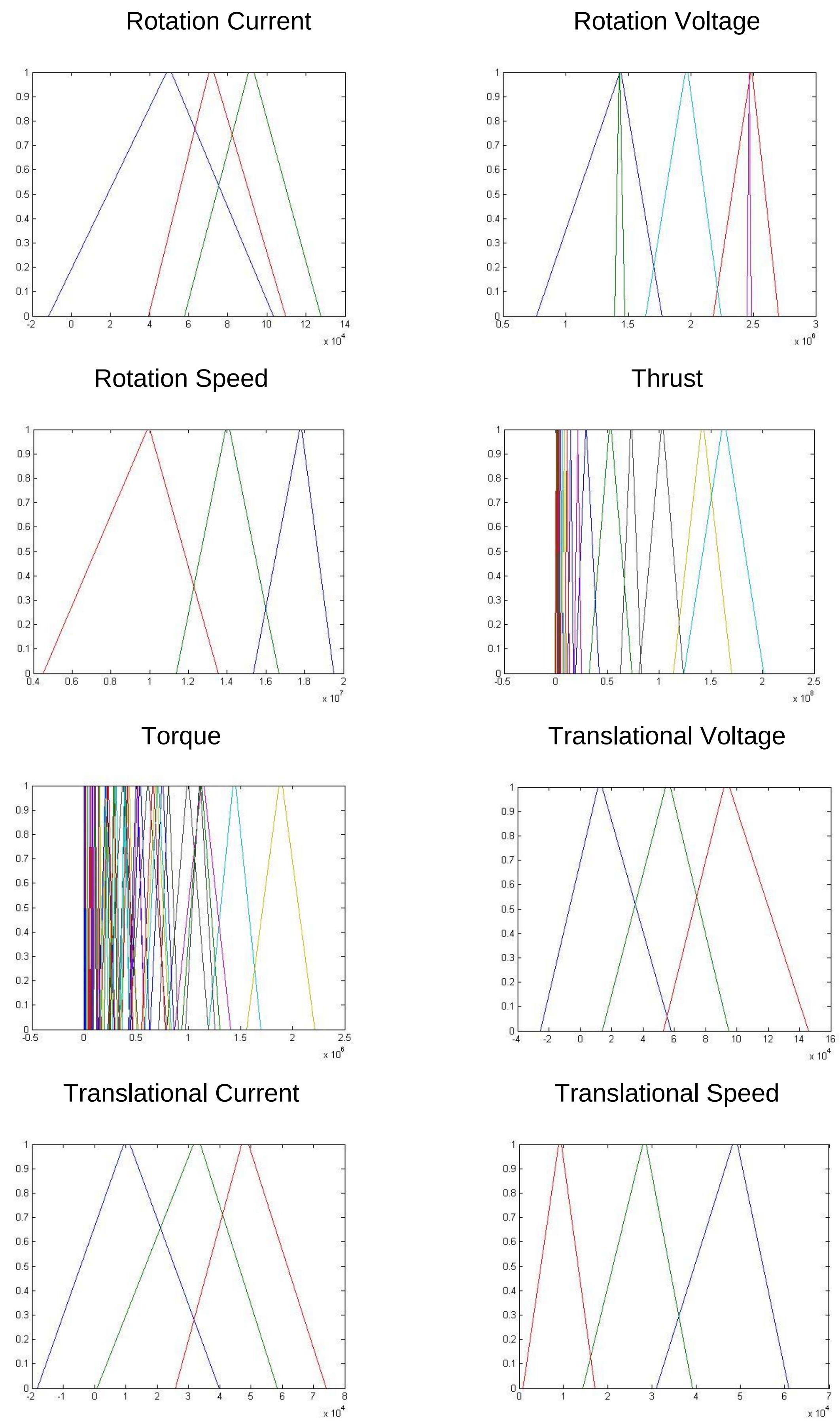


Figure 5.19: Input linguistic variables after optimization with BestP (except set points)

5.4 Summary

In this chapter the goal was to reduce the number of membership functions in linguistic variables of an already existing inference system. An inference system construted in the scope of a project developed for ESA served as case study. The inference system was constructed automatically from sensor data and was thus full of redundancy. The aim was to reduce the complexity of the inference system while maintaining its structure, without losing too much performance. Not only was this objective achieved, but also the inference system performance was increased. Moreover, a paper about this subject was published [Gomes, Santos et al. 2008].

I could have presented only the last algorithm, bestP, and ignore the intermediate attempts to find an algorithm for reducing the number of membership functions. However, I preferred to describe my first and more intuitive approach to this problem for two reasons: first, to show the importance of trial and error in science and second because the results obtained with the first adapted algorithm justify the necessity to evaluate the system performance during the algorithm instead of concentrating on “design” measures such as the model error.

There are still some questions that should be answered about this procedure. The algorithm was applied sequentially to input linguistic variables. There was no study about the importance of the order in which this pruning is made. I believe that this order can have some impact on the results. To solve this problem, instead of sequentially running the algorithm, a global algorithm where in each iteration the most similar membership functions from any linguistic variable was merged could be designed. Another question concerns the similarity measure used. Experiments with other similarity measures should be used to justify the choice of similarity measure.

Chapter 6. Conclusions and Future Work

The purpose of this thesis was to develop algorithms to reduce the number of membership functions in a linguistic variable. One of the main advantages of fuzzy models is that they are usually less complex and easy to interpret than classical models. By reducing the number of membership functions in linguistic variables the aim was to achieve simpler and more efficient fuzzy models.

The problem of reducing the number of membership functions in a linguistic variable was approached as a clustering problem. The pruned linguistic variable was the result of merging clusters of similar membership functions into a new membership function. Some possible formulations to the clustering problem were presented. Exact methods were used with one of these formulations and the combinatorial nature of clustering problems was clear. As expected, only very small data sets can be optimally solved through these methods in a reasonable amount of time. Although it was never the purpose of this thesis to use exact methods to solve this problem, this was an important step to better understand the dimension of the problem at hands.

To find good quality solutions in a more reasonable amount of time a Scatter Search procedure was developed and compared to the K-Means++ algorithm. Both procedures were implemented in Matlab and tested with two different case studies. The linguistic variables from these case studies could later be used in a fuzzy inference system or any other fuzzy model constructed taking into account the pruned membership functions instead of the original ones. The computational results were not as expected. The second phase of the Scatter Search algorithm was not able to produce good quality solutions. However, the first part of the algorithm was sufficient to obtain better results than the ones given by the K-Means algorithm. Both methods achieved a high reduction in the number of membership functions in each linguistic variable.

The last chapter presented a different case study. The objective was to reduce the number of membership functions in linguistic variables of an automatically constructed inference system without losing two much performance. It was seen that, in this case, concentrating on the characteristics of a single linguistic variable was insufficient. It is important to concentrate on the actual performance of the inference system, using appropriate measures of performance. Therefore the heuristics implemented before were not applied to this case study. Although the fitness function of the scatter search algorithm could have been changed to account for these performance measures, the time it takes to evaluate the inference system and the number of times it would be necessary to evaluate it explain why it was chosen not to use this algorithm in this case. The algorithms used in this case study can be categorized as hierarchical clustering algorithms. The results achieved in this case study were more than satisfying. It was possible to improve the initial inference system performance and simplify the system at the same time.

Although a lot of work was developed during this thesis, there is still much to be done in the future. In the first procedures a comparison of the different cluster validity indexes and of the shapes of the clusters themselves, translated by the clustering criteria used, should be made. Different strategies inside the scatter search algorithm could be tested to try to overcome the poor results obtained in terms of the way solutions are combined. More work is needed in estimating the correct number of clusters or a good maximum number of clusters to be given as input for the scatter search algorithm or other clustering algorithms. It would also be interesting to construct fuzzy models to identify the type of breast cancer (malign or benign) from the cell nuclei characteristics and to support credit approval decision processes using the linguistic variables from section 4.3 before and after being pruned, comparing results. In the MODI case study, as mentioned, the impact of the order in which the linguistic variables are pruned in the results or the possibility of designing a global algorithm that looked at all variables at the same time are possible directions for future work, along with a study of how results are affected by using different similarity measures. Furthermore, given the good results obtained in this case study, it is important to confirm the validity of the algorithm by running it on different case studies.

Chapter 7. References

Abdule-Wahab, R. S., N. Monmarché, M. Slimane, M. A. Fahdil and H. H. Saleh (2006). "A Scatter Search Algorithm for the Automatic Clustering Problem." Advances in Data Mining: 350-364.

Adlassnig, K. (1986). "Fuzzy set theory in medical diagnosis " IEEE Trans. Syst. Man Cybern. **16**(2): 260-265.

Aggarwal, C. C. and P. S. Yu (2000). Finding generalized projected clusters in high dimensional spaces. Proceedings of the 2000 ACM SIGMOD international conference on Management of data. Dallas, Texas, United States, ACM. 70-71.

Agrawal, R., J. Gehrke, D. Gunopulos and P. Raghavan (1998). Automatic subspace clustering of high dimensional data for data mining applications. Proceedings of the 1998 ACM SIGMOD international conference on Management of data. Seattle, Washington, United States, ACM. 94-105.

Anoop Kumar, D. and H. Moskowitz (1991). "Application of fuzzy theories to multiple objective decision making in system design." European Journal of Operational Research **53**(3): 348-361.

Asuncion, A. N., D.J. (2007). "UCI Machine Learning Repository " Retrieved March 2008, from http://archive.ics.uci.edu/ml/datasets/Credit+Approval.

Ball, G. H. and D. J. Hall (1965). ISODATA, a novel method of data analysis and classification. T. Rep. Stanford, CA, Stanford Univ.

Barnhart, C., E. Johnson, G. Nemhauser, M. Savelsbergh and P. Vance (1998). "Branch-and-Price: Column Generation for Solving Huge Integer Programs." Operations Research **46**(3): 316-329.

Brown, D. E. and C. L. Huntley (1990). A Practical Application of Simulated Annealing to Clustering, University of Virginia.

CA3. (2006). "MODI's homepage." Retrieved April 2007, 2007, from http://www2.uninova.pt/ca3/en/project_MODI.htm.

Chen, S.-J. and S.-M. Chen (2008). "Fuzzy risk analysis based on measures of similarity between interval-valued fuzzy numbers" Computers & Mathematics with Applications Pergamon **55**(8): 1670-1685.

Chen, S.-M., M.-S. Yeh and P.-Y. Hsiao (1995). "A comparison of similarity measures of fuzzy values". Fuzzy Sets and Systems **72**(1)**:** 79-89.

Chen, S. M. (1996). "New Methods for Subjective Mental Workload Assessment and Fuzzy Risk Analysis." Cybernetics and Systems **27**: 449-472.

Cheng, C., W. Lee and K. Wong (2002). "A genetic algorithm-based clustering approach for database partitioning." IEEE Transactions on Systems, Man and Cybernetics, Part C **32**(3): 215–230.

Colin, R. R., Ed. (1993). Modern heuristic techniques for combinatorial problems, John Wiley & Sons, Inc.

Costa, A., A. D. Gloria, F. Giudici and M. Olivieri (1997). "Fuzzy Logic Microcontroller." IEEE Micro **17**(1): 66-74.

David, A. and V. Sergei (2007). k-means++: the advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. New Orleans, Louisiana, Society for Industrial and Applied Mathematics. 1027-1035

Dempster, A., N. Laird and D. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society. Series B (Methodological) **39**(1): 1–38.

Edwards, A. W. F. and L. L. Cavalli-Sforza (1965). "A Method for Cluster Analysis." Biometrics **21**(2): 362-375.

Engelbrecht, A. (2002). Computational Intelligence: An Introduction, Halsted Press.

ESA. (2008). "Aurora Exploration Program." Retrieved January 2008, from http://www.esa.int/esaMI/Aurora/index.html.

Ester, M., H.-P. Kriegel, J. Sander and X. Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. 2nd International Conference on Knowledge Discovery and Data Mining. Portland, OR**:** 226-231.

Estivill-Castro, V. (2002). "Why so many clustering algorithms: a position paper". ACM. **4:** 65-75.

Everitt, B., S. Landau and M. Leese (2001). Cluster Analysis, Arnold Publishers.

Florek, K., J. Lukaszewicz, H. Steinhaus and S. Zubrzycki (1951). "Sur la liaison et la division des points d'un ensemble fini." Colloquium Mathematicum **2**: 282-285.

Gan, G., C. Ma and J. Wu (2007). Data Clustering: Theory, Algorithms, and Applications Philadelphia, SIAM.

Gautam, G. and B. B. Chaudhuri (2004). "A novel genetic algorithm for automatic clustering", Elsevier Science Inc. **25:** 173-187.

Glover, F. (1977). "Heuristics for Integer Programming Using Surrogate Constraints." Decision Sciences **8**(1): 156-166.

Glover, F. (1986). "Future paths for interger programming and links to artificial intelligence." Computer & Operation Research **13**(5): 533-549.

Glover, F. and M. Laguna (1997). Tabu Search, Kluwer Academic Publishers.

Gomes, M. M., B. R. Santos, T. Simas, P. Sousa and R. A. Ribeiro (2008). Reducing the Number of Membership Functions in Linguistic Variables: Application to a Fuzzy Monitoring System. Eight International Conference on Application of Fuzzy Systems and Soft Computing Helsinki, Finland, b- Quadrat Verlag.

Gower, J. C. and G. J. S. Ross (1969). "Minimum Spanning Trees and Single Linkage Cluster Analysis." Applied Statistics **18**(1): 54-64.

Hansen, P. and M. Delattre (1978). "Complete-Link Cluster Analysis by Graph Coloring." Journal of the American Statistical Association **73**(362): 397-403.

Hartigan, J. A. (1975). Clustering Algorithms. New York, Jonh Wiley and Sons.

Hillier, F. S. and G. J. Lieberman (2005). Introduction to Operation Research. Singapore, McGraw-Hill.

Holland, J. H. (1975). Adaptation in natural and artificial systems. Ann Arbor, University of Michigan Press.

Hubert, L. (1974). "Some applications of graph theory to clustering." Psychometrika **39**(3): 283-309.

Isermann, R. (1998). "On fuzzy logic application for automatic control, supervision, and fault diagnosis." IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **28**(2): 221-235.

Jain, A. K. and R. C. Dubes (1988). Algorithms for Clustering Data. New Jersey, Prentice Hall, Englewood Cliffs.

Jain, A. K., M. N. Murty and P. J. Flynn (1999). "Data Clustering: A Review." ACM Computing Surveys **31**(3): 264-323.

Jameaux, D., R. Vitulli, R. A. Ribeiro, T. Fonseca, B. R. Santos and M. Barata (2006). Monitoring & Diagnosis on-board software module for Mars driller. Proceedings of the 5th International Workshop on Planning and Scheduling for Space.

Jardine, N. and R. Sibson (1968). "The Construction of Hierarchic and Non-Hierarchic Classifications." The Computer Journal **11**(2): 177-184.

Jean-Philippe, H. and H. Jin-Kao (2002). Scatter Search for Graph Coloring. Selected Papers from the 5th European Conference on Artificial Evolution, Springer-Verlag.166-179

Jiang, T. and S. Ma (1996). Cluster analysis using genetic algorithms. Proceedings of the third international conference on signal processing **2**: 1277–1279.

Jimenez, J. F., F. J. Cuevas and J. M. Carpio (2007). Genetic algorithms applied to clustering problem and data mining. Proceedings of the 7th WSEAS International Conference on Simulation, Modelling and Optimization. Beijing, China, World Scientific and Engineering Academy and Society (WSEAS). 219-224

Joyce, C. W. and K. N. Michael (2000). A Tabu Search Based Algorithm for Clustering Categorical Data Sets. Proceedings of the Second International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents, Springer-Verlag.559-564

Kaufman, L. and P. Rousseeuw (1990). Finding Groups in Data - An Introduction to Cluster Analysis. New York, John Wiley and Sons, Inc.

King, B. (1967). "Step-wise clustering procedures." Journal of the American Statistical Association **69**: 86–101.

Kirkpatrick, S., C. D. Gelatt, Jr. and M. P. Vecchi (1983). "Optimization by Simulated Annealing." Science **220**(4598): 671-680.

Klir, G. J. and B. Yuan (1995). Fuzzy Sets and Fuzzy Logic: Theory and Applications. New Jersey, Prentice Hall.

Kruskal, J. B., Jr. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society **7**(1): 48-50.

Kuiper, F. and L. Fisher (1975). "Shorter communications 391: A Monte Carlo comparison of six clustering procedures." Biometrics **31**(3): 777-783.

Lai, Y. J. and C. L. Hwang (1994). Fuzzy multi objective decision making methods and applications. Berlin, Springer-Verlag.

Land, A. H. and A. G. Doig (1960). "An Automatic Method for Solving Discrete Programming Problems." Econometrica **28**(3): 497-520.

Lee, C. C. (1990a). "Fuzzy logic in control systems: fuzzy logic controller. I." Systems, Man and Cybernetics, IEEE Transactions on **20**(2): 404-418.

Lee, C. C. (1990b). "Fuzzy logic in control systems: fuzzy logic controller. II." Systems, Man and Cybernetics, IEEE Transactions on **20**(2): 419-435.

Lourenço, L. L. (1995). Contributos da Optimização Discreta para a Análise Classificatória. Aplicação de Heurísticas Genéticas a uma Classificação com Precedências. Dissertação de Mestrado em Matemática Aplicada à Economia e Gestão, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa.

Martí, R., M. Laguna and V. Campos (1997). "Scatter Search vs. Genetic Algorithms: An Experimental Evaluation with Permutation Problems." Adaptive Memory and Evolution: Tabu Search and Scatter Search. C. Rego, Alidaee, B.(Eds.), Kluwer Academic Publishers.

Mathworks Matlab Fuzzy Logic Toolbox **-** User's Guide.

Maulik, U. and S. Bandyopadhyay (2000). "Genetic algorithm-based clustering technique." Pattern Recognition **33**(9): 1455–1465.

McErlean, F. J., D. A. Bell and S. I. McClean (1990). "The use of simulated annealing for clustering data in databases". Information Systems. **15**(2)**:** 233-245.

McQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability: 281–297.

McQuitty, L. (1957). "Elementary linkage analysis for isolating orthogonal and oblique types
and typal relevancies." Educational and Psychological Measurement **17**: 207-222.

Mencar, C., G. Castellano and A. M. Fanelli (2007). "Distinguishability quantification of fuzzy sets." Information Sciences **177**(1): 130-149.

Milligan, G. W. and M. C. Cooper (1985). "An examination of procedures for determining the number of clusters in a data set." Psychometrika **50**(2): 159-179.

Mirkin, B. (2005). Clustering For Data Mining: A Data Recovery Approach New York, Chapman & Hall/CRC.

Miyamoto, S. (1990). Fuzzy Sets in Information Retrieval and Clustering Analysis, Kluwer Academic Publishers.

Murtagh, F. (1983). "A survey of recent advances in hierarchical clustering algorithms." The Computer Journal **26**(4): 354-359.

Nakashima, T., H. Ishibuchi and T. Murata (1998). Evolutionary algorithms for constructing linguistic rule-based systems for high-dimensional pattern classification problems. Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence.

Pacheco, J. A. (2005). "A scatter search approach for the minimum sum-of-squares clustering problem". Computers and Operations Research. **32:** 1325-1335.

Petra, K. (2007). Clustering Genetic Algorithm. Proceedings of the 18th International Conference on Database and Expert Systems Applications, IEEE Computer Society.138-142

Prim, R. (1957). "Shortest connection matrix network and some generalizations." Bell Systems Technical Journal **36**: 1389–1401.

Procopiuc, C. M., M. Jones, P. K. Agarwal and T. M. Murali (2002). A Monte Carlo algorithm for fast projective clustering. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. Madison, Wisconsin, ACM.

Ribeiro, R. A. (1996). Fuzzy multiple attribute decision making: a review and new preference elicitation techniques. Fuzzy Sets and Systems. **78:** 155-181.

Ribeiro, R. A. (2006). "Fuzzy space monitoring and fault detection applications." Journal of Decision Systems. 2-3.

Rijsbergen, C. J. v. (1979). Information Retrival. London, Butterwoths.

Ross, T. J. (2004). Fuzzy Logic with Engineering Applications, John Wiley and Sons.

Russell, R. A. and W.-C. Chiang (2006). "Scatter search for the vehicle routing problem with time windows." European Journal of Operational Research **169**(2): 606-622.

Salvador, S. and P. Chan (2004). Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. Tools with Artificial Intelligence, 2004. ICTAI 2004.

Santos, B. R., T. Fonseca, M. Barata, R. A. Ribeiro and P. Sousa (2006). New Data preparation process - A case study for an ExoMars Drill. Proceedings of the World Automation Congress (WAC2006).

Santos, B. R., T. Fonseca, M. Barata, R. A. Ribeiro and P. Sousa (2008). "A method for automatic fuzzy set generation using sensor data." Autosoft-Intelligent Automation and Soft Computing International Journal.

Santos, B. R., T. Fonseca, M. Barata, R. A. Ribeiro and P. Sousa ((to appear 2008)). "A method for automatic fuzzy set generation using sensor data." Autosoft-Intelligent Automation and Soft Computing
International Journal.

Santos, B. R., G. Martins, M. Gomes and R. A. Ribeiro (2008). CCN for MODI - Simulation of Knowledge Enabled Monitoring and Diagnosis Tool for Mars Lander Payloads: Final Report, Uninova/CA3.

Setnes, M., R. Babuska, U. Kaymak and H. R. van Nauta Lemke (1998). "Similarity measures in fuzzy rule base simplification." Systems, Man and Cybernetics, Part B, IEEE Transactions on **28**(3): 376-386.

Shokri, Z. S. and K. Alsultan (1991). A simulated annealing algorithm for the clustering problem, Elsevier Science Inc. **24:** 1003-1008.

Shu-Jen, J. C. and C. L. Hwang (1992). Fuzzy Multiple Attribute Decision Making: Methods and Applications, Springer-Verlag New York, Inc.

Sneath, P. (1957). "The applications of computers to taxonomy." Journal of General Microbiology **17**: 201–226.

Sokal, R. and P. Sneath (1963). Principles of Numerical Taxonomy. San Francisco, W.H. Freeman.

Song, B. G., R. J. Marks, II, S. Oh, P. Arabshahi, T. P. Caudell and J. J. Choi (1993). "Adaptive membership function fusion and annihilation in fuzzy if-then rules". Fuzzy Systems.

Spath, H. (1980). Cluster analysis algorithms for data reduction and classification of objects. New York, Ellis Horwood.

Sung, C. and H. Jin (2000). "A tabu-search-based heuristic for clustering." Pattern Recognition **33**(5): 849–858.

Tapas, K., M. M. David, S. N. Nathan, D. P. Christine, S. Ruth and Y. W. Angela (2002). "An Efficient k-Means Clustering Algorithm: Analysis and Implementation", IEEE Computer Society. **24:** 881-892.

Ujjwal, M. and B. Sanghamitra (2002). "Performance Evaluation of Some Clustering Algorithms and Validity Indices", IEEE Computer Society. **24:** 1650-1654.

Ward Jr., J. (1963). "Hierarchical grouping to optimize an objective function." Journal of the American Statistical Association **58**(301): 236–244.

Ward Jr., J. and M. Hook (1963). "Application of a hierarchical grouping procedure to a problem of grouping profiles." Educational and Psychological Measurement **23**(1): 69–81.

Wirth, M., G. Estabrook and D. Rogers (1966). "A graph theory model for systematic biology, with an example for the oncidiinae (orchidaceae)." Systematic Zoology **15**(1): 59–69.

Wolsey, L. A. (1998). Integer Programming, Wiley-Interscience.

Yongguo, L., Y. Zhang, W. Hong, Y. Mao and C. Kefei (2008). "A tabu search approach for the minimum sum-of-squares clustering problem". Information Sciences. **178:** 2680-2704.

Zadeh, L. A. (1965). "Fuzzy Sets." Information and Control **8**: 338-353.

Zadeh, L. A. (1975). "The concept of a linguistic variable and its application to approximate reasoning--I." Information Sciences **8**(3): 199-249.

Zimmermann, H. J. (1990). Fuzzy set theory and its applications. Boston, Kluwer Academic Publishers.

APPENDIX

Resumo Alargado

1. Introdução

Uma variável linguística [Zadeh 1965] é composta por conjuntos vagos que podem ser matematicamente representados por funções de pertença. Por exemplo a variável linguística *Altura* pode ser composta pelos conjuntos vagos *Baixa*, *Média* e *Alta* (Figure 1.1). Em vez de um indivíduo pertencer apenas a um destes conjuntos como aconteceria com a lógica clássica, o grau de pertença de qualquer indivíduo a cada um destes conjuntos é dado pela respectiva função de pertença.

O objectivo desta tese era desenvolver algoritmos para reduzir o número de funções de pertença em variáveis linguísticas. Este problema é extremamente importante quando é utilizado um processo automático de criação de variáveis linguísticas, podendo-se assim obter uma variável linguística com um elevado número de funções de pertença. De uma forma mais geral, o problema que se coloca pode ser visto da seguinte maneira: como reduzir a quantidade de dados a analizar (aqui representados pelas diferentes funções de pertença) sem com isso perder informação? Esta é precisamente a mesma questão que nos é posta em problemas de agrupamento ou *clustering*. Assim, o problema da redução do número de funções pertença numa variável linguística foi aqui abordado como um problema de agrupamento. Começamos por identificar possíveis grupos de funções de pertença semelhantes. Funções de pertença pertencentes a um mesmo grupo são então agrupadas numa nova função de pertença, obtendo-se assim um novo conjunto mais pequeno de funções de pertença que representam aproximadamente o mesmo conceito que a variável linguística inicial.

Podemos considerar que nesta tese foram abordados dois grandes tipos de problemas. No primeiro o objectivo é a redução do número de funções de pertença em variáveis linguísticas que mais tarde poderiam vir a fazer parte de um qualquer modelo (não necessariamente um sistema de inferência) que seria construído já tendo em conta as características da variável linguística depois desta redução. No segundo, pelo contrário, o objectivo seria a redução do número de funções de pertença de variáveis linguísticas pertencentes a um sistema de inferência previamente construído, tendo em conta as características das variáveis linguísticas originais. Assim, neste caso, a variável linguística terá que ser encarada como parte de um sistema e o objectivo passa a ser obter um equilíbrio entre o desempenho do sistema e a sua simplificação por meio da redução do número de funções de pertença.

2. Conceitos Importantes

No Capítulo 1Chapter 1 é introduzida alguma informação sobre lógica difusa necessária para melhor compreender o contexto em que esta tese se insere, bem como alguma notação que será utilizada noutros capítulos. Apenas as ideias mais importantes são aqui referidas.

2.1. Representação de funções de pertença

Algumas famílias de funções de pertença podem ser mapeadas para R^p, em que p é o número de parâmetros dessa família de funções e cada dimensão representa um parâmetro diferente.

Por exemplo, para $p=3$ uma função de pertença triangular pode ser representada por um triplo (a,b,c) (Figure 1.9). Se o triângulo for simétrico podemos tomar $p=2$, representando a função de pertença por (a,ε), em que $\varepsilon=b-a=c-b$.

De modo semelhante, podemos representar uma função de pertença trapezoidal por um vector (a,b,c,d) contendo os seus vertices . No caso de esta ser simétrica, ou seja, $\frac{a+d}{2}=\frac{b+c}{2}$, podemos usar um triplo (m,ε,δ), em que $m=\frac{a+d}{2}=\frac{b+c}{2}$, $\varepsilon=c-b$ and $\delta=d-a$ (Figure 1.10).

Esta representação será usada para tratar o problema da redução do número de funções de pertença numa variável linguística como um problema de agrupamento tradicional.

2.2. Fundir Funções de Pertença

A ideia por trás dos algoritmos a utilizar é a fusão de grupos de funções de pertença semelhantes.

Dadas n funções de pertença trapezoidais, $T^i = (a_i, b_i, c_i, d_i)$, $i = 1, \ldots, n$, estas serão fundidas numa nova função de pertença $T = (a, b, c, d)$, $i = 1, \ldots, n$ usando uma generalização do método proposto em [Setnes, Babuska et al. 1998]:

$$a = \min_{i=1,\ldots,n} a_i \tag{1}$$

$$b = \frac{1}{n}\sum_{i=1}^{n} b_i \tag{2}$$

$$c = \frac{1}{n}\sum_{i=1}^{n} c_i \tag{3}$$

$$d = \max_{i=1,\ldots,n} d_i \tag{4}$$

As formulas para fundir um grupo de funções de pertença triangular vêm directamente das anteriores.

3. Métodos Exactos

Nesta tese são discutidas algumas formulações em programação inteira para este problema (secção 2.3). Embora nunca tenha sido o objectivo desta tese encontrar soluções óptimas para estes problemas usando métodos exactos como o Branch & Bound [Land and Doig 1960], uma destas formulações foi introduzida no CPLEX para dois conjuntos de problemas de pequena dimensão, um com apenas 12 funções de pertença em cada variável linguística e outro com 54. Enquanto que no primeiro conjunto de problemas foi possível encontrar soluções óptimas em menos de 2 minutos, no segundo conjunto de problemas já não foi possível encontrar soluções óptimas, tendo o programa parado por falta de memória. Estas experiências permitiram ter uma maior noção da dimensão e dificuldade deste tipo de problemas e justificaram a necessidade de recorrer a métodos heurísticos para encontrar boas soluções num espaço de tempo mais realista.

4. Métodos Heurísticos baseados em Pesquisa Local

Dada a ineficácia dos métodos exactos em encontrar a solução óptima para o agrupamento dos pontos num determinado conjunto X, foram explorados métodos heurísticos. Foi desenvolvida uma meta-heurística, Scatter Search, baseada em [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006] que foi posteriormente comparada com uma variação do algoritmo das K-Médias ou *K-Means* [McQueen 1967], denominada K-Means++ [David and Sergei 2007].

4.1. K-Means++

O algoritmo *K-Means* [McQueen 1967] começa por escolher aleatoriamente para centros dos clusters K pontos do conjunto de dados $X = \{x_1, \cdots, x_n\}$. Depois desta inicialização, determina-se uma partição dos dados em K grupos, afectando cada ponto ao grupo com centro mais próximo. A partir deste momento os centros dos grupos vão sendo actualizados e os pontos vão sendo afectados ao grupo mais próximo até que algum ser satisfeito algum critério de paragem.

O algoritmo K-Means++ [David and Sergei 2007] difere do algoritmo original apenas na maneira como os centros iniciais são escolhidos. Depois de o primeiro centro ser escolhido aleatoriamente e de forma uniforme, isto é, considerando iguais probabilidades de escolha para cada ponto de X, os restantes são escolhidos de acordo com probabilidades proporcionais à sua distância ao centro mais próximo, de entre os centros já escolhidos. Quanto mais longe um ponto se encontra dos centros já escolhidos, maior será a probabilidade de este ser escolhido. Desta forma pretende-se dispersar a distribuição dos centros iniciais para que o algoritmo convirja mais rapidamente.

Neste algoritmo o número de grupos a formar, K, é escolhido *a priori*. Quando não sabemos *a priori* o número de grupos a formar, corremos o algoritmo para várias escolhas de K e escolhemos a melhor configuração encontrada, tendo em conta um determinado índice para a qualidade dos agrupamentos. O índice usado nesta tese, discutido em [Ujjwal and Sanghamitra 2002], deve ser maximizado e é dado pelas expressões seguintes:

$$I(K) = \left(\frac{1}{K} \times \frac{E_1}{E_K} \times D_K \right)^p \tag{5}$$

em que
$$E_K = \sum_{k=1}^{K} \sum_{j=1}^{n} u_{kj} \left\| x_j - c_k \right\| \tag{6}$$

e
$$D_K = \max_{i,j=1,\cdots,K} \left\| c_i - c_j \right\| \tag{7}$$

sendo que $U = \lfloor u_{kj} \rfloor_{K \times n}$ é uma matrix binária representando uma partição dos dados em K grupos (i.e., $u_{kj} = 1$ se e só se x_j está no k-ésimo grupo) e o centro do grupo k é representado por c_k.

4.2. Scatter Search

O algoritmo Scatter Search (Figure 4.1), opera sobre um pequeno conjunto de referência, composto por boas soluções e por soluções com elevada diversificação (em relação às restantes). Um conjunto inicial de soluções é criado pelo Método de Geração de Diversificação (DG – Diversification Generation Method). Cada solução neste conjunto é melhorada pelo Método de Melhoria (Imp - Improvement Method) antes da criação do conjunto de referência pelo Método de Actualização do Conjunto de Referência (RSU – Reference Set Update Method), que escolhe para fazer parte deste conjunto as melhores soluções bem como soluções com elevado nível de diversificação. O Método de Geração de Subconjuntos (SG – Subset Generation Method) forma subconjuntos de soluções do conjunto de referência para serem combinados pelo Método de Combinação de Soluções (SC – Solution Combination Method) em novas soluções. A qualidade das soluções assim obtidas é mais uma vez melhorada pelo Método de Melhoria antes do conjunto de referência ser actualizado. O algoritmo continua até que algum critério de paragem seja satisfeito.

O algoritmo pode ser implementado de diversas maneiras de acordo com as estratégias adoptadas em cada um dos seus cinco métodos principais. As estratégias utilizadas nesta tese para cada um dos métodos são adaptadas de [Pacheco 2005; Abdule-Wahab, Monmarché et al. 2006] e resumidamente descritas de seguida. Como função de adaptação foi usado o índice I.

4.2.1 Método de Geração de Diversificação

Este método é responsável pela criação de um conjunto inicial de OS_{size} soluções.

Para cada solução, começamos por gerar aleatoriamente um número de grupos a formar, K, entre 1 e $K_{\max}$, sendo $K_{\max}$ o número máximo de grupos permitido (dado pelo utilizador). São escolhidos aleatoriamente K centros $S = \{c_1, \cdots, c_K\}$. No entanto, em vez de poderem ser escolhidos para centros quaisquer pontos de X, foi introduzido um parâmetro $\alpha \in [0,1]$ que controla o nível de aleatoriedade deste processo, determinando o conjunto de pontos que em cada passo podem ser escolhidos para centros, como proposto em [Pacheco 2005]. Para evitar a repetição na escolha dos centros das várias soluções criadas durante esta fase do algoritmo guardou-se a frequência com que casa ponto foi escolhido como centro, penalizando-se a escolha de pontos com elevada frequência. A penalização é controlada pelo parâmetro β.

Depois de terem sido escolhidos os centros dos grupos, os restantes pontos são atribuídos a estes grupos usando o processo heurístico *greedy* descrito em [Pacheco 2005], com o objectivo de minimizar a soma dos quadrados das distâncias de cada ponto ao centro do grupo a que pretence.

4.2.2 Método de Melhoria

Foi escolhido o método de melhoria apresentado em [Abdule-Wahab, Monmarché et al. 2006], baseado no algoritmo das K-Médias [Gan, Ma et al. 2007] e que utiliza a simplificação proposta por Spath [Spath 1980] para aproximar o incremento em termos de soma dos quadrados das distâncias de cada ponto ao centro do seu grupo resultante de mover o ponto x_i do grupo C_l para o grupo C_j. Em cada iteração deste método cada ponto de X é movido para o grupo que corresponde a um maior decréscimo nesta soma dos quadrados das distâncias. São feitas *MaxIterImp* iterações sempre que o método é utilizado.

4.2.3 Método de Actualização do Conjunto de Referência

Para construir o conjunto de referência, RS, começamos por escolher as melhores b_1 soluções, de entre as OS_{size} soluções criadas inicialmente. São depois adicionadas iterativamente b_2 soluções de acordo com a sua diversidade. As soluções escolhidas são as que maximizam

$$\delta_{\min}(\lambda) = \min\{dif(\lambda, \lambda') : \lambda' \in RS\} \qquad (8)$$

em que $dif(\lambda, \lambda')$ é o número de pontos que são atribuídos a grupos diferentes nas soluções λ e λ'.

Nesta implementação o conjunto de referência é apenas actualizado quando são encontradas soluções de boa qualidade.

4.2.4 Método de Geração de Subconjuntos

Este método gera uma colecção de subconjuntos de soluções do conjunto de referência para serem posteriormente combinadas em novas soluções. Nesta implementação foram considerados todos os pares de soluções do conjunto de referência, isto é, são considerados $C_2^{b_1+b_2}$ pares de soluções.

4.2.5 Método de Combinação de Soluções

Para combinar um par de soluções numa ou mais novas soluções foi considerada uma estratégia do tipo path relinking, descrita em [Pacheco 2005]. A ideia deste tipo de estratégia é de que no "caminho" (série de movimentos simples que permitem alcançar uma solução a partir da outra) entre duas boas soluções deverão existir outras boas soluções. Neste caso um movimento corresponde a trocar um ponto de um grupo para outro. São propostas uma a três soluções escolhidas aleatoriamente neste caminho.

4.3. Resultados

Para ambos casos de estudo considerados, foi apresentada uma pequena análise dos parâmetros envolvidos no algoritmo *Scatter Search*. Apenas 5 experiências foram feitas para cada conjunto de valores dos parâmetros do algoritmo, pelo que os resultados não devem ser generalizados mas devem ser tidos em conta apenas a título indicativo. Em todas as experiências foi escolhido $K_{\max} = 100$, $b_1 = b_2$ e $OS_{size} = 10 \times (b_1 + b_2)$.

Foi possível ver a importância do parâmetro α no controlo da aleatoriedade do algoritmo, uma vez que para $\alpha = 0$ (escolha dos centros totalmente aleatória) os resultados finais apresentavam um elevado desvio padrão, não acompanhado de uma melhoria dos resultados em termos médios. O uso da memória durante a geração do conjunto de soluções iniciais mostrou-se positivo. Ao aumentar a dimensão do conjunto de referência de 4 $(b_1 = 2)$ para 10 $(b_1 = 5)$ conseguimos aumentar a qualidade das soluções com algum esforço computacional adicional. No entanto para ambos os casos de estudo este esforço adicional não foi considerado excessivo. Claro que, numa situação real, esta conclusão dependeria sempre do problema em concreto e do tempo disponível para realizar esta tarefa. O método de melhoria das soluções não melhorou significativamente a qualidade média das soluções para todas as variáveis linguísticas. Ao estudar a evolução do conjunto de referência verificou-se que a segunda parte do algoritmo não produziu boas soluções.

	K-Means++			Scatter Search		
	Tempo (seg.)	Nº Clusters	I	Tempo (seg.)	Nº Clusters	I
Radius	426,3904	4	18,9609	352,8277	3	26,9636
Texture	398,9221	3	16,8796	344,1798	5	30,18262
Perimeter	463,4574	7	830,33	371,4513	3	1286,17
Area	416,6392	6	426500,7	394,1495	3	668728,5
Smoothness	419,2019	3	0,000143	413,9573	4	0,0002044
Compactness	407,7729	3	0,004632	308,5648	3	0,004547
Concativity	418,1248	3	0,080241	305,7193	3	0,080673
Concave Points	405,9813	3	0,002273	470,995	3	0,002743
Symmetry	411,8899	3	0,000921	672,4182	3	0,000932
Fractal Dimension	413,8989	4	0,000167	679,9634	3	0,000255

Tabela 1: Caso de Estudo 1- K-Means++ vs Scatter Search (melhores resultados)

	K-Means++			Scatter Search		
	Time (sec.)	Nr. Clusters	I	Time (sec.)	Nr. Clusters	I
A2	854.5374	5	266.0796	821.3367	3	384.8384
A3	862.5715	4	90.18299	794.977	4	96.4486
A8	810.7072	6	105.4446	1277.734	6	94.76789
A11	735.1273	23	6.97E+28	1112.77	23	6.97E+28
A14	1134.509	6	346890.2	1812.912	4	402917.4
A15	1062.561	6	5.51E+09	2031.943	6	5.57E+09

Tabela 2: Caso de Estudo 2 – K-Means++ vs Scatter Search (melhores resultados)

Em termos médios, foi considerando $\alpha = 0.5$, $\alpha = 0.8$, $b_1 = 5$ e MaxIterImp$= 2$ que se obtiveram os melhores resultados. Os resultados apresentados na Tabela 1 foram obtidos com estes parâmetros. O algoritmo *Scatter Search* desenvolvido foi capaz de obter melhores resultados que o algoritmo K-Means++ para a maior parte das variáveis. Com ambos os algoritmos, foi possível reduzir significativamente o número de funções de pertença das variáveis linguísticas analisadas (Figure 4.24 - Figure 4.33 e Figure 4.52 - Figure 4.57).

5. Caso de Estudo: Um Sistema de Inferência *Fuzzy*

Como foi referido na Introdução, a natureza deste caso de estudo é diferente da dos casos de estudo do capítulo anterior. Neste caso de estudo o objectivo é a redução do número de funções de pertença em variáveis linguísticas pertencentes a um sistema de inferência previamente construído. Pretende-se reduzir a complexidade do sistema sem perder demasiado desempenho.

Este caso de estudo foi desenvolvido no CA3 – UNINOVA [CA3 2006] no âmbito do projecto “MODI- Simulation of a Knowledge Enabled Monitoring and Diagnosis Tool for ExoMars Pasteur Payloads” [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Fonseca et al. 2006; Santos, Martins et al. 2008] para a Agência Espacial Europeia [ESA 2008]. Foram construídos de forma automática dois sistemas de inferência: um para um sistema de alarme para a detecção de comportamentos anormais durante perfurações em Marte e outro para o reconhecimento da dureza do terreno a ser perfurado. Os resultados aqui apresentados utilizam somente o sistema de reconhecimento de terreno.

As variáveis linguísticas de entrada foram criadas automaticamente usando dados recolhidos por sensores durante a fase de aprendizagem [Santos, Fonseca et al. 2008]. Durante a fase de aprendizagem foram realizados furos para diferentes velocidades de translação e rotação em diversos tipos de terreno. Cada variável linguística representa um sensor diferente e cada função de pertença trapezoidal numa dada variável linguística corresponde a um diferente subcenário testado. O resultado da inferência é um dos tipos de terreno possível e o nível de certeza nessa classificação [CA3 2006; Jameaux, Vitulli et al. 2006; Santos, Martins et al. 2008].

5.1. Algoritmo

O algoritmo adoptado baseia-se no algoritmo proposto por [Setnes, Babuska et al. 1998], em que os conjuntos difusos mais semelhantes vão sendo fundidos de forma iterativa até que os restantes conjuntos sejam suficientemente distintos, o que é feito através da imposição de um limite mínimo para a semelhança entre dois conjuntos juntar, minS (Figure 5.4). Este algoritmo pode ser visto como um algoritmo de agrupamento hierárquico.

Viu-se que neste caso de estudo em que o sistema de inferência foi construído previamente seria importante ter em conta medidas de desempenho do sistema de inferência. Assim, em vez de se definir um valor para minS, corremos o algoritmo até todas as funções de pertença serem disjuntas, avaliando o desempenho do sistema de inferência actual, P(M), e comparando-o com o desempenho do melhor sistema encontrado até ao momento, P(BestM). O algoritmo devolve o sistema de inferência com melhor desempenho, de entre os sistemas gerados durante o algoritmo (Figure 5.8). O algoritmo foi definido para qualquer medida de desempenho para um sistema de inferência, P(.). Neste caso foi utilizada a seguinte medida de desempenho (a ser maximizada) :

$$F = \frac{2 \cdot P \cdot MCL}{P + MCL} \tag{9}$$

em que a Precisão (P) do sistema de inferência é o quociente entre o número de amostras bem classificadas sobre o total de amostras e o Nível de Certeza Média (MCL) é a média dos níveis de certeza para as amostras correctamente classificadas.

Se quisermos uma solução de compromisso entre o número de funções de pertença no sistema e o seu desempenho podemos combinar estes objectivos considerando

$$P(M) = \alpha F - \frac{1-\alpha}{n_0} n \tag{10}$$

em que $\alpha \in [0,1]$ é o peso dado a F, n_0 é o número inicial de funções de pertença e n é o número de funções de petença do sistema M a ser avaliado.

O algoritmo está ainda definido para uma medida de semelhança entre dois conjuntos difusos genérica. Neste caso foi usada a medida de semelhança de Jaccard com :

$$S_J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{11}$$

em que $|C| = \int_U \mu_C(x)\,dx$ e $\cap$ e $\cup$ representam a intersecção e a união de conjuntos difusos.

5.2. Resultados

Foram testados 6 tipos de terreno, 3 valores para a velocidade de rotação e 3 valores para a velocidade de translação da broca, obtendo-se assim 54 funções de pertença para cada uma das variáveis linguísticas que representam os diferentes sensores instalados na broca (Figure 5.9).

O algoritmo foi aplicado a cada uma das variáveis linguísticas de forma sequencial. Os resultados estão resumidos nas tabelas abaixo. Como se pode ver na Tabela 3 e pelos gráficos das variáveis linguísticas finais (Figure 5.19) foi possível reduzir de forma muito significativa o número de funções de pertença em praticamente todas as variáveis linguísticas. Foi também possível melhorar o desempenho do sistema de inferência, como se pode ver pela Tabela 4. Por exemplo, a Precisão do sistema (P), aumentou de 72.33% para 85.47%.

	Original	BestP
Rotation Current	54	3
Rotation Voltage	54	5
Rotation Speed	54	3
Thrust	54	17
Torque	54	46
Translational Voltage	54	3
Translational Current	54	3
Translational Speed	54	3
TOTAL	432	45

Tabela 3: Redução do número de funções de pertença

	Original	BestP
P	72.33%	85.47%
MCL	34.49%	44.00%
F	46.71%	58.09%
N	423	45

Tabela 4: Comparação dos sistemas de inferência

6. Conclusões

O objectivo desta tese era desenvolver algoritmos para reduzir o número de funções de pertença numa variável linguística. Este problema foi abordado como um problema de agrupamento.

Foi desenvolvida uma metaheurística Scatter Search para encontrar boas soluções para o problema. Usando dois casos de estudo, esta metaheurística foi comparada com o algoritmo K-Means++. Os resultados obtidos não foram os esperados. A segunda parte do algoritmo Scatter Search não conseguiu produzir boas soluções. No entanto, a primeira parte do algoritmo foi suficiente para obter melhores resultados que os resultados conseguidos com o K-Means++. Com ambos os métodos, foi possível reduzir significativamente o número de funções de pertença em cada variável linguística.

No último capítulo foi apresentado um caso de estudo em que as variáveis linguísticas faziam parte de um sistema de inferência construído de forma automática. Neste caso é importante ter em conta o desempenho do sistema de inferência durante o algoritmo de redução, usando medidas de desempenho adequadas. Os resultados obtidos foram bastante satisfatórios. Não só foi possível reduzir de forma bastante significativa o número de funções de pertença no sistema, mas também foi possível aumentar o seu desempenho.