A Dynamic Adaptive Framework for improving Case-Based Reasoning System Performance

Fernando Orduña Cabrera

Computer Science Department

Universitat Politècnica de Catalunya · BarcelonaTech

Ph.D. Programme in Artificial Intelligence

Ph.D. Thesis presented for obtaining the degree of Doctor

Advisor: Dr. Miquel Sànchez-Marrè

2015

Preface

This thesis document is the document which presents the student Fernando Orduña Cabrera for obtaining the degree of doctor by the Universitat Politècnica de Catalunya · BarcelonaTech, in the Doctoral Programme of Artificial Intelligence.

Acknowledgments

First, I want to thank my supervisor for the support needed to realize this project, the large amount of overtime that he addressed to me. Thanks for your patience and support, and by the enthusiasm and dedication. I also want to thank CONACYT for the support with grant number 205684, also wish to thank the "Faculty Development Program, PRODEP" for their support to the thesis writing scholarship "DSA / 103.5 / 15/6635". And all those who were directly and indirectly involved in this project. Thanks.

Abstract

An optimal performance of a Case-Based Reasoning (CBR) system means, the CBR system must be *efficient both in time and in size*, and must be *optimally competent*. *Efficiency in time* requires that CBR tasks must be carried out in a fast way: retrieval time, adaptation/reuse time, evaluation/revise time and learning/retain time should be as low as possible. From these tasks, usually the retrieval task is taking longer than the other ones. Therefore, the efficiency in time is closely related to an *efficient and optimal retrieval process* over the Case Base/Case Library of the CBR system. *Efficiency in size* means that the Case Library (CL) size should be minimal. Thus, a minimum number of cases must be stored in the CL. Therefore, the efficiency in size is closely related to *optimal case learning policies*, *optimal meta-case learning policies*, *optimal case forgetting policies*, etc. On the other hand, the *optimal competence* of a CBR system means that the number of problems that the CBR system can satisfactorily solve must be maximum.

To improve or optimize all three dimensions in a CBR system at the same time is a difficult challenge because they are interrelated, and it becomes even more difficult when the CBR system is applied to a dynamic domain or continuous domain (data stream). In this thesis work, a dynamic adaptive framework is proposed to improve the CBR system performance coping especially with reducing the retrieval time, increasing the CBR system competence, and maintaining and adapting the CL to be efficient in size, especially in continuous domains.

One of the main contributions of the work is the proposal of a Dynamic Adaptive Case Library (DACL) framework. A DACL is composed of a set of dynamically built case libraries to cope with the heterogeneity and complexity of real domains. It learns cases and organizes them into dynamic cluster structures. The DACL is able to adapt itself to a dynamic environment, where new clusters, meta-cases or prototype of cases, and associated indexing structures (discriminant trees, *k*-d trees, etc.) can be formed, updated, or even removed. DACL offers a possible solution to the management of the large amount of data generated in an *unsupervised continuous domain* (data stream). In addition, we propose the use of a Multiple Case Library (MCL), which is a static version of a DACL, with the same structure but being defined statically to be used

in *supervised domains*. The core of the retrieval process in both MCL and DACL is the matching of the current/query case against a set of prototype cases called meta-cases, to select the case library to search in. In a MCL, the number of meta-cases and its corresponding case libraries is fixed *a priori* according to the different class labels, rather than dynamically like in a DACL. The structure of a DACL/MCL is organized hierarchically at different levels: the meta-case, the prototype of a concrete cluster of cases and the indexing hierarchical structures (discriminant trees, *k*-d trees, etc.) that represent the way that all the cases in a given cluster are organized.

The framework proposed is flexible enough to allow the implementation of many different retrieval techniques, because of the facilities that the Multiple Case Library (MCL) and the Dynamic Adaptive Case Library (DACL) offers for indexing.

The thesis work proposes some techniques for improving the indexation and the retrieval task. The most important indexing method is the *NIAR k-d tree algorithm*, which improves the retrieval time and competence, compared against the baseline approach (a flat CL) and against the well-known techniques based on using standard *k*-d tree strategies. In addition, a partial matching exploration technique is proposed. NIAR *k*-d tree algorithm performs quite similar to the standard *k*-d tree selecting by discrimination the attributes through cycling the list of major attributes, but differs of standard *k*-d tree approach in the technique of selecting the split value. The *Partial Matching Exploration (PME) technique* explores a hierarchical case library with a tree indexing-structure aiming at not losing the most similar cases to a query case. This technique allows not only exploring the best matching path, but also several alternative partial matching paths to be explored. Both techniques: NIAR *k*-d tree indexing structure and Partial Matching Exploration (PME) technique have been evaluated. The results show an improvement in competence and time of retrieving of similar cases. Through the experimentation tests done, with a set of well-known benchmark *supervised databases*, it has been shown that the use of a Multiple Case Library (MCL) embedding a NIAR *k*-d tree, and using the additional strategy of PME to explore the indexing structure provides a very good approach both to improve the time efficiency and the competence in CBR systems.

The dynamic building of prototypes in DACL has been tested in an *unsupervised domain* (environmental domain) where the air pollution is evaluated. Here, the prototypes are incrementally generated and managed by the DACL. The core task of building prototypes in a DACL is the implementation of a *stochastic method for the learning of new cases* and management of prototypes. Therefore, it allows creating and managing a DACL. The stochastic method works with two main moments, the first moment guides the learning of new cases and decides where to store the cases. The second moment evaluates the prototypes selecting or building a new prototype. The proposed method for building prototypes has been conducted using the database acquired in an air pollution environmental domain. The experimental results shown, that using DACL in the environmental domain performs well and helps the experts to identify critical moments where the pollution is dangerous for people.
Finally, the *whole dynamic framework*, integrating all the main proposed approaches of the research work, has been tested in *simulated unsupervised domains*. Several databases from the UCI Machine Learning repository were tested in an *incremental way*, as data streams are processed in real life.

The conclusions outlined that from the experimental results, it can be stated that the dynamic adaptive framework proposed (DACL/MCL), jointly with the contributed indexing strategies and exploration techniques, and with the proposed stochastic case learning policies, and meta-case learning policies, improves the performance of standard CBR systems both in supervised domains (MCL) and in unsupervised continuous domains (DACL).

Contents

List of Figures

List of Tables

1 Introduction

Since the introduction of Case-Based Reasoning (CBR) principles by Schank in (Schank 1974; Schank 1982) in late 70s, CBR has been consolidating as a reliable reasoning paradigm in the Artificial Intelligence field (Richter & Weber, 2013; López, 2013; López de Mántaras, 2005; Aamodt & Plaza, 2004; Kolodner, 1993). Several research works (Orduña and Sànchez-Marrè, 2008a, 2008b) have been proposing techniques/methods to use CBR as a solution for learning of experiences, and as a method to make an interpretation of the expert knowledge, translating the knowledge to CBR systems.

In the CBR literature (see for instance (Fornells et al. 2008)), for methodological purposes it is distinguished between *Data-Intensive CBR systems* (DI-CBR) and *Knowledge-Intensive CBR systems* (KI-CBR). DI-CBR systems share characteristic features like extensive use of learning from examples, spare use of domain knowledge and simpler case and solution representations. They are different from the main features shared by KI-CBR systems, like intensive use of domain knowledge, complex case and solution structures and current developments being now based on ontologies and description logics. Our approach falls within the *Data-Intensive CBR systems* (DI-CBR) class, because in continuous/dynamical domains usually the huge amount of data is slightly more important than the domain knowledge. In our approach, both the case problem description and the case solution are implemented with a list of attribute-value pairs.

In this thesis work, a dynamic adaptive framework is proposed to improve the CBR system performance coping especially with reducing the retrieval time, increasing the CBR system competence, and maintaining and adapting the case library to be efficient in size, especially in continuous domains. The framework proposed works for reasoning and learning both in *supervised domains* and *unsupervised domains*. The proposal is entitled as a *Dynamic Adaptive Case Library Framework (DACL).*

A DACL is composed of a set of dynamically built case libraries. It learns cases and organizes them into dynamic cluster and tree-indexing structures. The DACL is able to adapt itself to a dynamic environment.

The proposal offers a solution to the management of the large amount of data incrementally generated in unsupervised domains. In addition, it is proposed the use of a Multi-Case Library *(MCL)* for supervised domains.

Both DACL and MCL aim to retrieve cases by matching the current/query case against a set of prototype cases called meta-cases. The structure of DACL/MCL is organized hierarchically at different levels: the meta-case and the use of hierarchical indexing structures. Both structures improve the proposed framework. One improvement is showing flexibility enough to allow the implementation of many different retrieval techniques. One of the main techniques proposed is a *NIAR k-d tree* algorithm, which improves the retrieval time and competence. Another technique is the *Partial Matching Exploration (PME) technique*. Those techniques have been evaluated. The results show an improvement in competence and time of retrieving of similar cases. Both techniques combined provide a very good approach to improve the time efficiency and competence in CBR systems.

Other contribution included in the framework is the dynamic building of prototypes in DACL. These prototypes are incrementally generated and managed by DACL. The technique aims to implement a stochastic method for the learning of new cases and management of prototypes. Others techniques are included in the proposal; those techniques implement methods for building representative prototypes. All the techniques proposed have been tested and the results indicates an improvement of the performance of the standard CBR system both in supervised domains with MCL and in unsupervised continuous domains with DACL.

1.1 Motivation

Case-Based Reasoning (CBR) systems solve new problems by retrieving and adapting the solutions to previously solved problems that have been stored in a case library (Richter &Weber, 2013; López 2013; López de Mántaras et. al., 2005, Aamodt and Plaza, 1994, Kolodner, 1993). Systems until now have been used in different fields as it is mentioned in (Orduña and Sànchez-Marrè, 2008b). CBR systems are a good tool for the experts interested in management of knowledge in an automatic way. The work of Orduña and Sànchez-Marrè in (Orduña and Sànchez-Marrè, 2008b) refers to several

fields where CBR systems have been used successfully, but there are domains where CBR systems need to be more efficient. Continuous domains are domains where is necessary to continuously work to improve the CBR systems. Continuous domains are complex because they generate large amount of information in a short time period.

Case Based Maintenance (CBM) is a Case-Based Reasoning subarea where the community of CBR has been working with the aim of finding better methods that improve the performance of the CBR cycle. In the last years, several research works have proposed different methods to handle the information in a CBR system. Some of these contributions are detailed in (Jalali, 2014; Salamó, 2011; Orduña and Sànchez-Marrè, 2008c; Perner, 2006; Iglezakis et al., 2004; Portinale and Torasso, 2001; Smyth and Mckenna, 2001; Leake and Wilson, 2000; Yang and Wu, 2000).

The performance of a CBR system is usually measured along two dimensions: efficiency and competence. *Efficiency* of a CBR system means that the CBR system requires the minimal resources needed to solve any case in the domain of application. The resources are twofold: the *time* required for solving a case, and the *size* of the case library needed to solve a case. Therefore, efficiency is related both to the case solving time and to the size of the case library. The *competence* of a CBR system refers to the range of problems that can be satisfactorily solved, for instance see (Salamó & López-Sánchez, 2011). All these performance dimensions are interrelated. For instance, if the case library size is getting smaller by decreasing the number of cases learnt, then the time efficiency could be decreased but the competence of the CBR system could be reduced. On the contrary, if the case library size is larger by increasing the number of learnt cases, then the time efficiency could be worsened, but the competence of the CBR system could be increased.

The retrieval is one of the important steps in Case-Based Reasoning systems. Several algorithms have been proposed for the indexing of cases, since the original indexing approach of *k*-d trees appeared in the literature. Main approaches propose to use a pre-computed binary search tree to get an average logarithmic time effort in searching. The basic idea of the proposal is to implement a Dynamic Adaptive Case Library for Continuous domains strengthen its structure by the imple-

mentation on its second level of indexing algorithms based on the principle of binary search trees for efficient retrieval according to a given similarity measure *sim*. Even more, a proposed NIAR *k*-d tree algorithm is based on the computation of the average value of the corresponding attribute among the sub-tree cases. An evaluation of algorithms such as the *k*-d trees and NIAR *k*-d trees has been done in (Orduña and Sànchez-Marrè, 2013); to make a comparative of efficiency and performance evaluation.

In some research works, an effort to give an approximation to reduce the complexity of learning in continuous domains is observed (Salamó & López-Sánchez, 2011; Segata, 2010). One of them is the research work (Sànchez-Marrè et. al., 2000), where their proposal consists in a Meta-reasoning by means of a set of meta-cases approach. This research differs from our research proposal given that in their proposal a static structure is presented, consisting in a given number of meta-cases. In their proposal (Orduña and Sànchez-Marrè, 2009), (Orduña and Sànchez-Marrè, 2014a), (Orduña and Sànchez-Marrè, 2014b) the structure is dynamic and the library is able to deal with a variable number of meta-cases, clusters and discriminant trees. In his research presents one rule to know whether a case is similar to one of the given meta-cases. This rule is considered in (Orduña and Sànchez-Marrè, 2009). The Meta-case is the top level of the indexing method proposed where its attributes are used in the learning process. In that proposal, the values of its attributes were considered as previously established by experts.

The DACL proposed in (Orduña and Sànchez-Marrè, 2009) is illustrated in figure 13. DACL learns cases and organizes them into the dynamic cluster structures. The library is able to adapt itself to a dynamic environment, where new clusters, meta-cases, and associated indexing structures (discriminant trees, *k*-d trees, etc.) can be formed, updated, or even removed (Orduña and Sànchez-Marrè, 2015a). DACL offers a possible solution to the management of the large amount of data generated in a continuous domain. With the stochastic learning process of new cases and meta-cases introduced in (Orduña and Sànchez-Marrè, 2013) is feasible to improve the retrieval time and size of the case library. Even more, a fusion of all proposed techniques with DACL proposed in (Orduña and Sànchez-Marrè, 2009) is feasible to improve CBR system performance, especially when facing unsupervised continuous domains with large case bases.

1.2 Issues of the work

In this thesis work, we aim at improving the general CBR system performance, by jointly coping with the time efficiency, size efficiency and competence dimensions, especially when coping with *unsupervised continuous domains*.
We aim at improving the *case library structure* with some new proposals for dynamically structuring the case library in such a way that the performance of the system increases. These new structures should be able to handle a data stream of cases within the unsupervised domains.
In addition, the work has the aim to find *indexing strategies*, which improve the performance of CBR systems. The aim is to search for variations in the building of *k*-d trees, to make them more balanced structures and faster accessing methods.
Moreover, the research work will aim at proposing new *exploration techniques* of *k*-d trees, and in general of hierarchical structures, different from usual ones (like hyberball techniques, etc.) so that the retrieval time could be reduced, even though the accuracy of the system could be slightly worsened.
Another issue is to propose new *case learning strategies* based on reliable facts like, for instance, the relevance concept.
A very important aspect related to unsupervised continuous domains is the *incrementality problem*. General CBR systems assume that the set of cases available for building the case library is fixed. Then they build the memory indexing structures, like for instance, a *k*-d tree, etc. However, when a CBR system is facing an unsupervised continuous domain, the system should build and update the case library structure/s in an incremental way.
The proposal of some new *introspective tasks and strategies* are also another sub goal of our work, in order to update and *maintain* the CBR system in optimal conditions.

In our thesis proposal, the structure of the Dynamic Adaptive Case Library (DACL) can be built-up in an incremental way, and some introspective reasoning tasks can be scheduled to regularly update the indexing structures (NIAR *k*-d trees, etc.).

1.3 Contributions

The main contribution of this thesis is a general *Dynamic Adaptive Framework* for improving Case-Based Reasoning System Performance. This framework is general enough for coping with *supervised domains*, where DACL is named as Multi-Case Library (MCL), because here the structure is not dynamic, and for coping with *unsupervised, usually continuous, domains*, where a stream of cases are needed to be processed and solved.

The global contribution of this research framework can be split in the following research contributions:

1. The DACL: a Dynamic Adaptive Case Library technique to be able to store a big amount of cases and to manage them for *unsupervised domains*, and especially, for continuous domains. The cases will be stored according to different policies/methods. The cases will be stored in the best-matching sub-library. This will be accomplished by means of the use of prototypes of the cases indexed through dynamic tree structures. The prototype will be called Meta-case. The Meta-cases will have associated indexing schemes (discriminant trees, *k*-d trees, etc.). The DACL structure was introduced in (Orduña and Sànchez-Marrè, 2009) (see section 3.1).

2. The MCL: a Multi Case Library will be a similar structure of DACL, but in that situation, the building of the structure will follow a special procedure for static *supervised domains*. The main use of Meta-cases is the same than in a DACL (see section 3.2)

3. New proposed variants of a *k*-d tree structure to improve the efficiency of the CBR system performance. Both AvKd-tree (Average *k*-d tree) and NIAR *k*-d tree (Nearest Instance to the Average Root) algorithms will be proposed to improve the competence in retrieval time and quality of retrieval in CBR. AvKd-Tree and NIAR *k*-d tree will differ of standard *k*-d tree in the technique of selecting the split value for each attribute at the internal nodes. The idea of the algorithms was introduced in (Orduña and Sànchez-Marrè, 2013) (see sections 4.1 and 4. 4.2).

4. The proposal of a new exploration technique for traversing a *k*-d tree, and obtaining the most similar cases of the *k*-d tree exploring alternative paths to the main path. This technique will be named as *Partial Matching Exploration technique* (PME) (see section 4.3)

5. The proposal of a *SMcLM* Stochastic Meta-case Learning Method, which will consider two relevant moments to guide the learning of the new case (*Nc)*: building a new Meta-case (Mc) or storing it in the existing Meta-cases. The aim of this policy will be to learn those cases that accomplish the two moments of the stochastic process. This method has been implemented in the study of the air pollution domain (Orduña and Sànchez-Marrè, 2015a), where the results obtained shows the efficiency of the method. This method is a second method of building representative prototypes, because a first DACL method was described in (Orduña and Sànchez-Marrè, 2009) (see sections 5.1 and 5.2)

6. *Other* techniques for selecting the Meta-cases and building representative prototypes (see section 5.3).

7. The proposal and inclusion of some introspective tasks for global maintenance of the DACL (see section 5.4).

1.4 Thesis Organization

This thesis document is organized as follows:

Chapter 2: Describes and details the topics strongly related and inherent to the DACL Framework: Case-Based Reasoning, Case representation and Case Library organization, Case Base Maintenance and related concepts, Introspective reasoning, stochastic learning and Continuous domains.

Chapter 3: In this chapter, the DACL is detailed. The three-layer structure of a DACL is described: the Meta-cases, the cluster of cases and the corresponding sub-library for each meta-case. Two basic algo-

rithms for building Meta-cases are introduced. The DACL version for supervised domains is also detailed: the Multiple Case Library (MCL).

Chapter 4: The proposed algorithms and techniques for indexing in DACL/MCL are explained. The AvKd-tree algorithm, the NIAR *k*-d tree and the Partial Matching Exploration (PME) technique are detailed.

Chapter 5: The improving of CBR cycle by the use of the stochastic learning strategy is explained. The two moments of the method and the policy of the method are detailed. Other strategies for building Meta-cases are introduced. In addition, some introspective tasks for an optimal maintenance of the DACL are explained. Two kinds of tasks are described. Ones related to a synchronous triggering task for maintaining the structure of the case library, and others asynchronous tasks for the maintenance of the indexing structures.

Chapter 6: The experimental evaluation strategy for DACL/MCL is set. The indexing techniques Avkd-tree and NIAR *k*-d tree in *exact-case search* are evaluated and some results are given. Next, an experimental setting for a MCL in *supervised domains* is set-up, combining the indexing structures proposed (NIAR *k*-d tree) and the Partial Matching Exploration (PME) technique against usual techniques. Afterwards, some experimental results are given. Next, the experimentation description for the stochastic learning method in an *unsupervised continuous domain* is set-up. Then, the results obtained are discussed. Finally, the whole Dynamic Framework, combining all the proposed approaches, is tested for *simulated unsupervised domains* in an *incremental way*.

Chapter 7: In this chapter a summary about the research work of the thesis and the main contributions of the work are outlined. In addition, some future research lines are detailed.

2 State of the Art

The following chapter introduces and analyzes some related concepts to the work of the thesis. Some definitions of the related concepts of Case-Based Reasoning (CBR), and some contributions to CBR are included. The CBR cycle is explained and some research works are considering for explanation. CBR cycle has four main phases, and the organization of knowledge plays a very important role for retrieving and for knowledge generation. Because of it, the case structure and organization are considered. For fast retrieving, it is important to store the knowledge in an efficient method; to do so, different methodologies used in CBR knowledge organization are introduced. The maintenance in CBR aims to improve the performance of CBR systems in several ways. For instance, through the improving of retrieving and storing cases tasks, by the use of policies. Here, for our aim some maintenance topics are introduced. The reasoning in CBR systems is the main task, and several methods for reasoning are handled in CBR; in our proposal, the introspective reasoning is applied and in this chapter, some details are introduced. The retrieval of cases is improved by the use of hierarchical structures, which is the case of using binary trees, such as the *k*-dimensional trees, here detailed. Some improvements to *k*-dimensional trees have been introduced in the literature. Especially, the proposal of using hyperball strategies for quality improvement in a tree retrieval. The improvement of quality it is related to methods used to learn new cases and with the case library organization; for this reason, in this proposal, a stochastic method is introduced and some details about stochastic learning are given in this chapter. Finally, the aspects of continuous domains, which are addressed by our proposal, are outlined.

2.1 Case-Based Reasoning

CBR is a cognitive paradigm based on Schank's Dynamic Memory theory (Schank, 1982), whose main conclusion is that remembering is the base of learning and understanding. The use of experience was proposed in order to understand a new situation. Because of that, it is necessary to have an appropriate indexation scheme for the previous experiences and a "mechanism" to recovery those experiences.

Janet Kolodner in (Sasikumar, 1998) define Case Based Reasoning (CBR) as: "*Case Based reasoning can mean adapting old solutions to meet new demands, using old cases to explain new situations, using old cases to critique new solutions, or reasoning from precedents for interpret a new situation (much as lawyers do) or create an equitable solution to*

a new problem (much as labor mediators do)". For other definition see (Richter and Weber, 2013, López, 2013; López de Mántaras et al., 2005; Aamodt and Plaza, 1994).

CBR is an extremely useful technique when the underlying models to the solution of a problem are not clear or they are not well understood. CBR is an especially appropriate alternative when the number of rules needed to capture by expert's knowledge is unmanageable, or when the domain theory is too weak or incomplete. In cases where an expert is not available, or it is too expensive or their knowledge cannot be articulated verbally, but it is possible to find a set of cases that are representative examples of the domain, then CBR is a very appropriate solution to solve new problems (Bonissone and López de Mántaras, 1998).

CBR systems solve new problems by retrieving and adapting the solutions from previously solved problems that have been stored in a case library. The performance of a case-based reasoning can be measured according to efficiency and competence. Case-Based Reasoning until now has been used in several applications, even though in continuous domains. CBR has been used to solve mostly the learning of experiences. In the dynamic environments field there are very few applications of CBR. Some of these efforts of using CBR can be found in some fields such as the Robotic field, where the goal is to endow a navigation robot with the experience to reduce its navigation time. Some examples of this are: (Urdiales *et al*., 2006; Supic and Ribaric, 2005; Kruusmaa, 2003; Fox, 2000; Ram and Santamaria, 1997; Ram et al., 1997). Some CBR applications for continuous domains are in the industry field, where CBR is applied to improve the expert supervision. For instance, see (Melendez *et al*., 2001). In the research work (Sànchez-Marrè *et al*., 1999) they propose to learn only the relevant cases generated in a continuous domain to avoid the growing of the case library, and a CBR framework for temporal domains is proposed (Sànchez-Marrè *et al*., 2005).

The CBR formalization is summarized in the basic CBR system reasoning cycle, proposed by Aamodt and Plaza in (Aamodt and Plaza, 1994) and López de Mántaras in (López de Mántaras *et al*., 2005) which is depicted in figure 1.

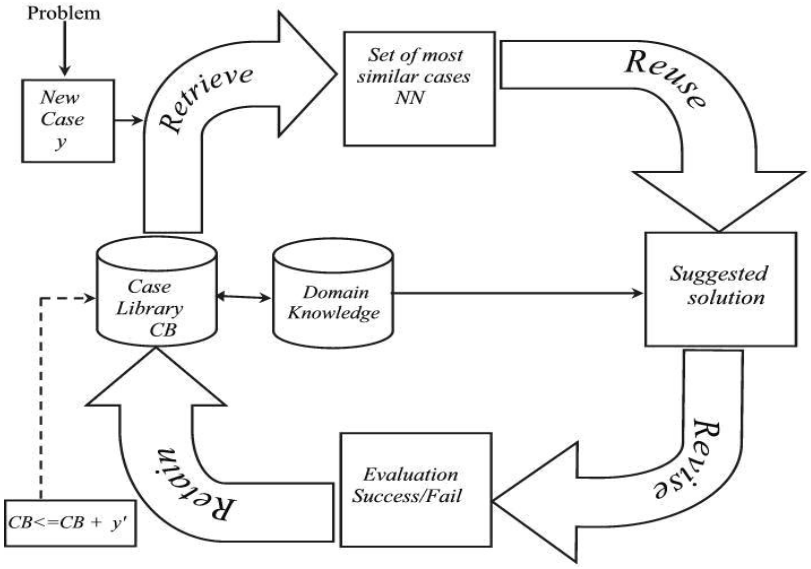

Fig. 1. CBR Cycle

The following processes describe a complete CBR system. Some definitions and descriptions are taken from (Núñez-Rocha, 2004a):

Retrieve the most similar case or cases (Case Retrieval). The retrieval process consists in finding the best match to the new problem among all previous cases. The identification task is based on a comparison between the relevant attributes describing the problem. In this first step of the cycle, there are very sensitive elements making more or less efficient the performing of the task:

- Attribute relevance
- Case base structure
- Similarity evaluation

This is one of the important processes in a CBR system. If you are able to retrieve the most similar case to the new problem, then the proposed solution will have more chances of being successful.

The factors that will determine the possibility to retrieve the best case are:

- Case representation formalism (attribute-value vectors, graphs, free text, etc.)
- Case base organization (flat or hierarchical memory)
- Similarity measure
- Attribute types
- Missing values
- Attribute relevance
- Discretization

These factors influence the retrieval phase in a different way. The case representation formalism wills define the way that retrieval is made. For example, if cases are represented as graphs, finding the similarity between two cases implies a comparison among graphs. If free text is used, the comparison will be based on natural language text, which could be very difficult. Usually, the text are represented by the relevant terms that there are in the case description (*keywords*).

The case base organization has a direct influence in strategy of searching the similar enough cases. It is necessary to make an evaluation about the convenience of having a specific representation, taking into account the advantages and disadvantages of each organization scheme. Some elements to evaluate are: case base size, case representation formalism, case sets to look for (specific groups or case prototypes), retrieval speed and accuracy needed. The type of the attributes involved in case definition affects what type of similarity measures can be used to evaluate the similarity between cases.

The efficiency of the retrieval is directly related to the quality of the data. If a case base has a great quantity of missing values, surely the retrieval will be less efficient than when all attribute values are known. Some strategies for dealing with missing values are:

- Eliminate cases with unknown values
- Not to take into account attributes with missing values
- Substitute missing values with heuristic values
- If in the evaluation of two attributes there are just one missing value, then dissimilarity between those values could be 1, else if both values are missing then the dissimilarity could be 0.

It is very common to find domains where the case description contains one or several attributes irrelevant to retrieval or only representative when they take certain values. Under these circumstances, it is necessary to establish attribute relevance so that the most relevant have a bigger influence on the retrieval process than the irrelevant ones. In other domains, it is preferable to establish the relevance according to the class values (supervised domains), or to give different weights to each case, or to different attribute values.

Reuse the solution and knowledge in that case to solve the current problem (Case Reuse) from the principle "*similar problems have similar solutions*", it is possible to have the solution of a problem if we find one that is similar enough to the new case in the case base. There are two aspects to take into account:

- The differences between the new case and the most similar one, and
- The part of the solution of the recovered case that can be reused in the new case.

A trivial use of reuse, but effective in some applications like a classification task, is to copy the solution (null adaptation) of the most similar case as an answer to the new case. In other applications, it will be necessary to carry out an adaptation process that consists of changing the solution to the previous case according to the differences that the values of the attributes show between the new and previous cases. The techniques used in the adaptation process can be enclosed according to the approaches proposed by Kolodner in (Kolodner, 1993) as: *Null Adaptation, Structural Adaptation and Derivational Adaptation*. Where *Null adaptation* can be a strategy adopted in CBR systems with very simple actions in the solution (like accept / reject, failure diagnosis). In these systems, the previous solution is directly applied without changes to the new case. When an *adaptation* process is directly applied to a previous solution, we have a structural adaptation method. These methods can be divided into the next main techniques: *substitution methods, transformation methods* and *special-purposes heuristic adaptation*. *The substitution methods* provide a solution for the new case with components or appropriate values computed from the solution recovered in the retrieval step. Most of substitution techniques are based on parameter adjustment or parameterized solutions, where the differences between the

values of the new case and the most similar one are used to appropriately guide the modification of the parameters in the solution. *The transformation methods* use either some common sense transformation rules, such as deleting a component, adding a component or adjusting values of a component, as in the JULIA system (Shinn, 1988), or some model-guided repair transformation techniques based on a causal model(s). *The special-purpose adaptation* techniques or critic-based adaptation methods are based on some specific rules of repairing, called critics (Sacerdoti, 1997), such as those used in PERSUADER (Sycara, 1987). Other systems such as CHEFF (Hammond, 1989) and JULIA (Shinn, 1988) use some domain specific adaptation heuristic and some structure modification heuristic. Finally *Derivational adaptation* methods do not operate on the original solutions, but on the method used to derive that solution. The goal is rerunning the same methods applied to derive the previous solution, to re-compute the solution for the new case. This methodology was implemented in the ARIES system, and was named as derivational replay (Carbonell, 1986).

Revise the proposed solution (Case Revision). This step gives to CBR systems a way to evaluate its decisions in the real world, allowing feedback that enables the learning from success or failure. The experts decide if the proposed solution is the most appropriate, generally evaluate the solution applied to the real world. If there is not an expert available, then a simulation of the application of this solution can be performed, or a direct experimentation in the real world. The results will determine the quality of the solution and it will be observed.

Retain the parts of this experience likely to be useful for future problem solving (Case Retention). The information given to the system about the quality of the proposed solution in the revision step is an important part in the learning process of the CBR system. Learning in a CBR environment could be in two different methods such us the *learning by observation and learning by experience*. *Learning by observation* is done when the system starts with a representative set of initial cases. These cases result from the direct observation of the domain or when an expert provides the knowledge. This kind of learning can also be given in the course of the use of the system, when new cases are observed, and are considered as representative or essential situations to describe the real environment of the problem. The expert can also add new cas-

es, if it is possible to describe them even when they have not been observed, and if they are not already in the case base. And the *Learning by experience*: is done after each cycle of the CBR system. After the revision step, it is determined if the solution was successful or not. In the first case, the CBR system memorizes the success, with the system being given the opportunity of storing the new case together with its solution in the case base, and marking it as a correct answer. If the answer was not appropriate, the system must be able to prevent itself from making the same mistake in the future, learning from failure. If the evaluation process concludes that the proposed solution is not the appropriate one, it is important to know what caused the error. It could be due to a bad adaptation of the proposed solution from the most similar case to the new one, or it is possible that within the case base there is not a very similar case. Even so, the system will recover this. Although it seems to be similar enough, it exist a difference. In this case, it is important to inform to the expert if there exists one, and try to generate a representative case of the current situation and to provide an appropriate solution. Another circumstance directly related to the purpose of this work can happen if exists a case in the case base, being more similar to the one recovered. If this is the situation, the similarity measure, the attribute weights, the normalization process, the discretization process, and the way to handle missing values must be revised.

2.1.1 Knowledge Organization

Cases in a CBR system must be represented and organized in such a way that the case representation methodology allows the comparison between cases, and the overall case organization permits the case retrieved in an optimal way.

2.1.1.1 Case Representation

A case is a piece of knowledge about a context, which represents an experience that teaches a fundamental lesson to the reasoner in view to get its goal (Kolodner, 1993). A case should represent a specific knowledge linked to a context, to store a different experience. It should be useful to the reasoner, and it is necessary to be able to compare it with other cases to determine its similarity.

A CBR system is highly dependent on the elected formalism to represent the cases in its case base. Since the core of the system consists of finding a similar previous case to the current one, the methodology used to make the search in the case base will depend on the structure used to represent each case and its solution.

The first task when building a CBR application is to decide how the cases will be represented. This decision will have a direct impact on the strategies to continue in the following design phases of the application such as case base organization, similarity evaluation, recovery of the most similar case to the current one, solution adaptation and system learning.

The most common case representation formalisms are the following:

- *Attribute-Value Vectors* is one of the most common case representations and organization mechanisms. For their simplicity and easy handling, the attribute-value vectors are a broadly used formalism. This formalism assumes that each case (C^i) is defined by a set of m attributes that can be continuous or discrete; C_c^i is an optional class label for the case C^i in a supervised domain.

  $$C^i = \left(C_1^i, C_2^i, \cdots, C_m^i; C_c^i\right)$$

  As an example, consider an automobile buyer who consults a CBR system to find out their best option. The corresponding vectors to the attribute description and value vectors are:

  Attributes:
  < Type, brand, cylinders, power, fuel, color, price, year, seats >
  Cases:
  < sport, BMW, 8, 275, gasoline, blue, 35000, 2000, 4 >
  < sport, AstonM, 12, 450, gasoline, red, 120000, 2002, 2 >
  < sport, Maserati, 12, 380, gasoline, green, 80000,2002,2 >
  < tourism, Citroen, 4, 95, diesel, red, 20000, 2002, 5 >
  < tourism, Renault, 4,65, gasoline, white, 12000, 2002, 5 >
  < tourism, Renault, 6, 85, gasoline, white, 17000,2002,5 >
  < tourism, Renault, 4, 60,gasoline,white,8000,2000,5 >

- *Free Text*. In some applications such as judicial cases (Weber-Lee et al., 1998), text categorization, electronic trade, answers to frequent questions systems (FAQ's) (Lenz and Burkhard, 1997) and medical and technicians reports, attribute- value vectors are not an appropriate option. In these domains, free text using natural language is a good way of representing cases. The case is described by means of sentences trying to include words that are good enough for the case discrimination and representing the problem domain in a faith-full way. In the automobiles context a formalism to case representation could be:

  "A red sports car with 300 horse power and 12 cylinders, with a price less than 250000 and must be BMW"

  The system operation is exposed in (Lenz and Burkhard, 1997). The case base consists of a set of texts containing possible answers to the questions made by the user. The described system takes the user's question expressed in natural language, and retrieves the text that better matches as an answer to the formulated question. The retrieval is carried out evaluating the similarity between the question and the cases in the case base, recovering those that are more similar. For each case, a set of Information Entities (IE) produced by the key words in the text are identified.

- *Conversational CBR*. In recent years, due to Internet expansion, many web-based consultation applications have emerged. In them, the user introduces a brief text ex- plaining his problem. With this text, the system finds the most similar case in its case base. Then, the system presents a set of questions associated to the recovered case. The user can answer to these questions in a quick and direct way. These systems are known as CCBR (Conversational CBR). The system should automatically infer the details that describe the problem starting from the text introduced by the user.

  During the conversation, the system evaluates and shows the most similar cases and their solutions, progressively until finding the most appropriate solution according to the user's opin-

ion. In these systems (Aha et al., 1999), a case x is represented as follows:

1. Problem $X_p = X_d + X_{qa}$ encodes the problem solved by X_s. Where:
description X_d is a portion of free text that partially describes X's problem. Specification X_{qa} is a set of <question, answer> pairs.

2. Solution $X_s = X_{a1}, X_{a2} \cdots, X_{an}$ is a sequence of actions X_{ai} for responding to X_p.

Actions can be free text, hyperlinks or other objects. A case's problem description and specification serve as its index. Questions in case specification can be internally disjunctive (i.e. have multiple answers). Cases are "positive" examples: applying X_s to X_p is assumed to be successful. X serves as a prototype for solving queries whose problem specification is similar to X's. In figure 2, the generic process of problem resolution in a CCBR system is presented.

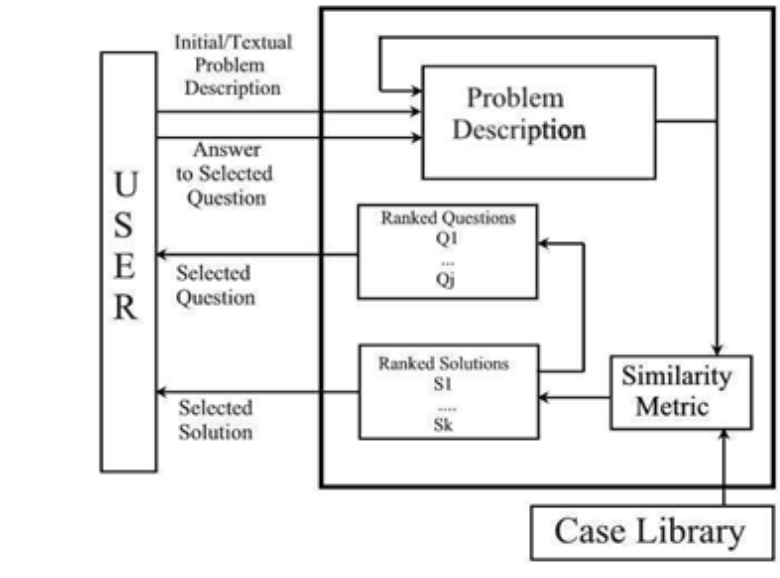

Fig. 2. Conversational CBR Problem Solution Generic Process

- *Graphs*. In previous sections, it was shown how the case representation formalism in a CBR system is linked to the domain. Some domains exist where the objects representing the problem are highly related to each other. This relationship cannot be efficiently represented in any of the previously described models. In those domains, graphs could be used as a case representation formalism.

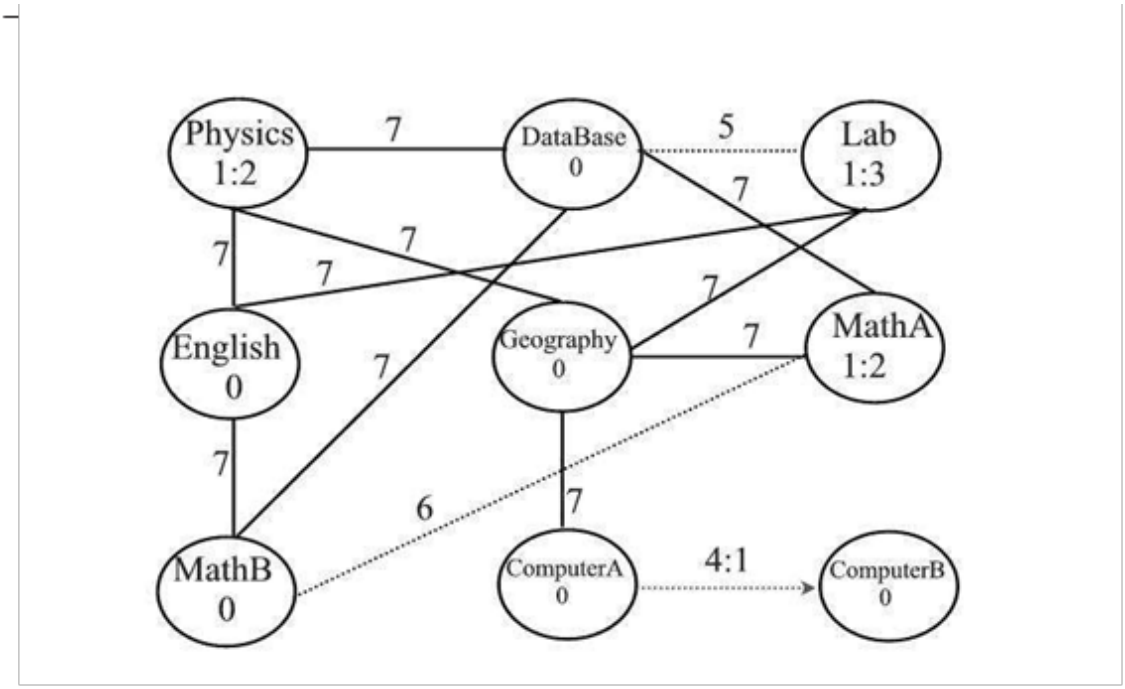

Fig. 3. A graph representation

A graph is defined as a structure $G = < VG, AG >$ where VG is a finite set of vertices and $AG \subseteq VG * VG$ is a set of edges. In general, vertices are used to represent the objects of the domain, and edges express the relationships and restrictions among them, as was used in the planning of timetabling of a school course (Burke et al., 2001). In some domains, edges can represent binary predicates as in CHIRON and CAPER (Sanders and Hendler, 1997). An example of this kind of representation is shown in the figure 3.

By means of this formalism, one could represent elements of the domain that are unlikely to appear in another representation: hard and soft constraints that are indicated by solid or dotted edges respectively. In the notation $X:Y$, X is the label and y represents the value of the attribute; *Physics*, *Lab* and *MathA* are labelled by 1, indicating that they are multiple courses. Values 2, 3, 2 give the times they should be held per week respectively. Other courses labelled 0 (ordinary courses) should be held just once a week. The courses adjacent to edges labelled 7 cannot be held simultaneously. Database should be consecutive to Lab if possible (the edge between them is labelled 5), and *MathA* should not be consecutive to *MathB* if possible (the edge between them is labelled 6). The direct edge between *ComputerA* and *ComputerB* is labelled 4, denoting that *ComputerA* should be held before *ComputerB*. One of the main disadvantages of using this formalism is that when recovery is done, it is evident that this is an isomorphism graph problem (a graph representing a current case compared with graphs representing previous cases), and it is broadly well known that this is a NP-complete problem.

2.1.1.2 Case base Organization

Several methods to organize the Case base can be implemented; the aim of the organization of the cases is that the implementation of method improves the search and retrieving of cases. Hierarchical and flat organizations are main used formalisms, but any other combination of structures is possible.

2.1.1.2.1 Flat Memory

This is the most intuitive organization scheme, since involves storing all the available cases sequentially, in a simple list, array, or file. In a flat memory, the new case is matched against each case in the memory, and the best matches are returned. An algorithm to guide this search is presented in algorithm 1.

This algorithm is very simple, and it is the similarity evaluation heuristic which carries out all the work. The main advantage is that, the entire

library is used to search, and the accuracy will only depend on how good the similarity measure is computed. There are some disadvantages that should be evaluated. The main one is that this scheme tends to be time-consuming in the retrieval step. As the case library gets large, so does the time needed for retrieval. A scheme like this works well in applications where the case base is not very large. However, it is necessary to closely watch over the case base growth, incorporating maintenance schemas, evaluating the relevance of incorporating a new case avoiding redundancy, but being flexible enough to accept a new case that represents a new domain situation that was not present among the existent cases in the case base.

Algorithm 1: linear search in a flat memory

```
1   Input: Set of cases of the Case Library (S),
2            The New Case (Nc)
3   Output: Index of nearest case (r)
4   Begin linear search(S, Nc)
5   Let N = the number of cases of the Case Library;
6   Let BestDissimilarity = the best dissimilarity value be-
7   tween Nc and all the cases until the current case
8   Let CurrentDissimilarity = the dissimilarity value be-
9   tween Nc and the current case (C^i)
10  Let d() = the distance computation function among two
11  cases
12  N = |S|
13  BestDissimilarity = + ∞
14  i = 1
15  while i ≤ N do
16    CurrentDissimilarity = d(C^i,Nc)
17    if CurrentDissimilarity < BestDissimilarity then
18      r = i
19      BestDissimilarity = CurrentDissimilarity
20    endif
21    i = i + 1
22  enwhile
23  return r
24  end linear search
```

2.1.1.2.2 Hierarchical Organization

Through time, a CBR system increases its data by means of the inherent learning in the learning phase. Thus, the number of cases in the case base will increase. In this situation, a flat structure and sequential search is impractical. As the case base grows, there is a need to organize cases hierarchically so that only a small subset needs to be considered during retrieval. This subset, however, must be likely to have the best-matching or most useful cases in its organization (Kolodner, 1993). Next some alternatives are presented to solve the hierarchical organization problem.

Shared Feature Networks. The main idea is as follows; if you can cluster together cases that are similar to one another and figure out which cluster best matches the new situation, then only items in that cluster need to be considered in finding a best-matching case. Hierarchies are formed when clusters are broken into sub-clusters and so on (Kolodner, 1993). Shared-feature networks provide a means of clustering cases so that cases that share many features are clustered together. Each internal node of a shared-feature network holds features shared by the cases below it. Leaf nodes hold cases themselves. The retrieval process in a shared-feature network performs a sort of breadth first search. The new case is matched against the contents of each node at the highest level in the graph. The best-matching node is chosen. If this is a case, the case is returned. Otherwise, if it is an internal node, the same thing is repeated among its descendants. This continues until a case is returned. View the Kolodner approach in (Kolodner, 1993).

Discrimination Networks. In a discrimination network, each internal node is a question that subdivides the set of cases stored underneath it. Each child node represents a different answer to the question posed by its parent, and each child organizes the cases that have its answer. A main difference with shared-feature networks is that discrimination networks put more emphasis on the discrimination than on clustering, exactly the opposite happens in shared-feature networks. To perform better in the recovery, it is important to include the most relevant questions in high levels of the hierarchy, leaving the less important questions for the low levels. A discrimination network algorithm is presented by Kolodner in (Kolodner, 1993).

2.1.1.2.3 *k*-d Trees

The retrieving of similar cases is one of the key steps in the case-based reasoning paradigm (Richter & Weber, 2013; Kolodner *et al*., 1985). The case base must be analyzed to detect a set of potentially useful cases for adaptation purposes. Commonly, there is the distinction between surface and structural similarity (Holyoak and Koh, 1986). Structural similarity computation is normally very expensive, because it means to consider all available knowledge of the domain. On the contrary, the retrieval step should manage the similarity computation of all cases in the case base as fast as possible. Thus, this task can only rely on the comparison of syntactical features, i.e., surface similarity (Gentner and Forbus, 1991). In addition, in the case-based reasoning literature there can be distinguished two different approaches to similarity assessment (Althoff and Wess, 1992); the representational approach (Kolodner, 1980) and the computational approach (Aha, 1991). The former is based on using a structured memory of cases, and the latter is based on the computation of an explicit similarity measure *sim*. This work is based on the computational approach.

When facing a relatively small case base, the similarity computation of all cases could be done in a sequential process, comparing each case in the case base against the current problem (see algorithm 1). This strategy is reasonable for small case bases, as the computational time effort is linear (*O(n)*, being *n* the number of cases) but it is not feasible for larger case bases. Several strategies have been proposed to improve the retrieval step for large case bases. Some approaches use massively parallel computer hardware to speed up the similarity assessment process. Others are based in the pre-computation of efficient indexation schemes which improve the efficiency of the retrieval step. We will propose a new strategy following these later approaches, but we will propose in chapter 5, some incremental indexing methods to tackle the data stream problem.

Also, some indexation schemes were designed to find the *m* most similar cases (*m* nearest neighbors, or *m-NN*) such as (Wess et al. 1993; Friedmann *et al.* 1977), while others focused on the one most similar case (nearest neighbor or 1-NN) (Arya et al. 1993; Bentley 1975). In our work, we took the second option, and the aim of the retrieval process is the most similar case. However, having in mind that the value *m* normally is low, be-

cause only the most similar cases are needed, and depending on the minimum number of cases in the leaves of the indexing tree (bucket size), the 1-NN indexing strategies can also provide the *m-NN* without exploring other subspaces of the indexing tree.

In case-based reasoning applications, sometimes the cases can be represented in a structured way by a set of independent features. In other times more sophisticated case representations are needed when there are attribute dependent cases, and normally they also cover a subspace rather than a point in the problem-solution space. The former are named as point cases, and the later as generalized cases. Most of the indexing approaches in the case-based reasoning community have dealt with point cases (Wess et al., 1993; Arya et al., 1993; Friedman et al., 1977; Bentley, 1975), but also some people has focused on the attribute dependent generalized cases (Bergmann and Tartakovski, 2009). In this thesis, the point case representation will be addressed.

Cases can be considered as points within a multidimensional search space, where each attribute is one dimension that can be explored with an associative search. The idea of our approach is to structure the search space according to the observed statistical distribution of the values of the attributes, and using this computation in advance to improve the case retrieval process according to a given similarity measure sim (Stottler et al., 1989). Our approach is based on the earlier concept of a multidimensional (k dimensions) binary search tree, i.e. a *k*-d tree (Broder 1990; Friedman *et a*l. 1977; Bentley 1975). The rationale behind our approach is to try to get the sub-trees of each internal node as balanced as possible, in order to reduce the number of tree levels, and consequently, increase the retrieval speed. A *k*-d tree is very similar to a discrimination tree/network (Kolodner 1993; Charniak and McDermott 1985). Major differences rely on how to decide which attribute must be the discriminator at each internal node of the tree. In discrimination trees, there are some heuristic approaches to derive which are the most discriminant attributes. For instance, an expert based discrimination list or some other automatic procedures such as using unsupervised feature weighting techniques (Núñez and Sànchez-Marrè 2004b).

One of the most satisfactory proposals for indexing a case base was based on the database query approach applying a *k*-d tree proposed by (Friedman et al. 1977; Bentley, 1975).

The basic idea behind a standard *k*-d tree is to partition a *k*-dimensional attribute space into some simpler subspaces and to search for the nearest neighbors in the corresponding subspaces. These subspaces are *k*-dimensional cubes. A *k*-d tree is a binary search tree very similar to a decision tree. The internal nodes are labeled with attribute names and the edges with partition values. The leave nodes are labeled with a disjoint subset of the cases (bucket of cases). Every node of a *k*-d tree represents a subset of the case base. The root node represents the whole case base. Each internal node partitions is the represented by a set of cases into two disjoint subsets. The left sub-tree contains the subset of cases with a value for the discriminator attribute of the node less or equal than the partition value. The right sub-tree contains the subset of cases with a value for the discriminator attribute of the node greater than the partition value. See figure 4 where a *2*-d tree and its corresponding two-dimensional search space is depicted.

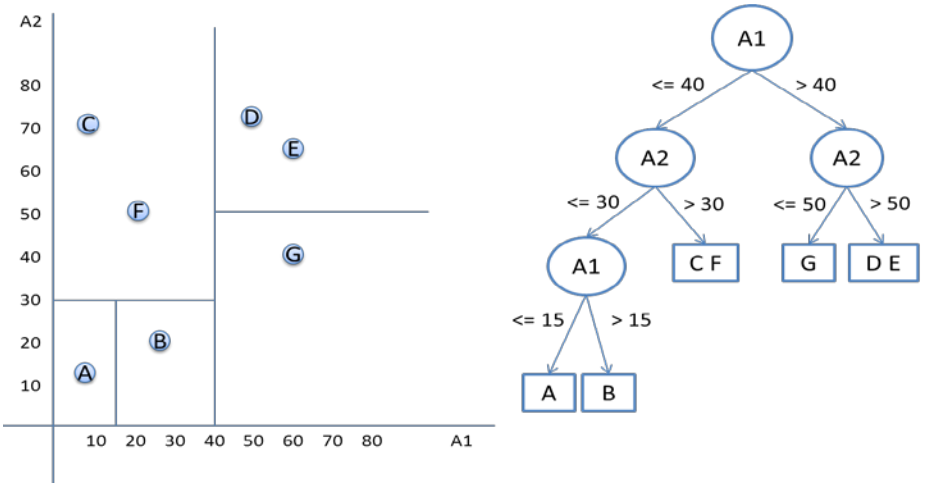

Fig. 4. An example of a *2*-d tree and its corresponding two-dimensional search space

The construction of the *k*-d tree is recursive. Beginning with a root node, the represented set of cases is partitioned according to a chosen discriminator attribute and a chosen partition value. This recursive pro-

cedure ends when a certain termination criterion is met, such as the number of cases in a subset is less or equal than the bucket size.

The retrieval task in a *k*-d tree is a recursive traversal search of the tree, starting from the root node and following the corresponding sub-tree until a leaf is reached. At each internal node, it descends the tree following the branch whose constraint on the discriminator attribute of the node ($\leq$ or $>$) is matched by the attribute value in the query case. When a leaf is reached, the similarity measure *sim* between the query case and the cases in the bucket is computed (see algorithm 2).

Algorithm 2: binary search in a *k*-d Tree

```
1   Input: The root node of the k-d Tree (Node),
2          The New Case (Nc)
3   Output: The most similar case (C^+sim)
4   Begin binarySearch (Node, Nc)
5   Let BucketCases = the cases stored in a bucket of length
6   k in a leaf node
7   Let DissimilarityList = the list of the dissimilarity
8   values computed between the NC and the BucketCases of the
9   corresponding leaf node
10  Let CaseList = the list of the BucketCases of the corre-
11  sponding leaf node
12  Let d() = the distance computation function among two
13  cases
14  Let IndexAttributte = a function which gives the index
15  value corresponding to the attribute stored in that node
16  Let PartitionValueAttributte = a function which gives the
17  partition value corresponding to the attribute stored in
18  that node
19  Let Cases = a function which gives the cases stored in
20  that node
21  Let NumCases = a function which gives the number of cases
22  stored in that node
23  if leaf?(Node) then
24    BucketCases = Cases(Node)
25    numC=NumCases(node)
26    DissimilarityList = ∅
27    for i = 1 to numC do
28      DissimilarityList = add(DissimilarityList, d(C^i,Nc))
29      CaseList = add(CaseList, C^i)
30    endfor
```

```
31      order(DissimilarityList, CaseList)
32      //order the two list in increasing order
33      return(first(CaseList))
34    else    //it is an internal node
35      m = IndexAttribute(Node)
36      if Nc_m ≤ PartitionValueAttribute(Node) then
37        binarySearch(leftTree(Node, Nc))
38      else
39        binarySearch(rightTree(Node, Nc))
40      endif
41    endif
42    end binarySearch
```

The average computational time for retrieving the most similar case is $O(log_2 n)$, being n the number of cases, if the tree is optimally organized, such as in discrimination trees optimally built. For the worst case, the retrieval cost is $O(n)$. Thus, on average, really improves the case retrieval of a sequential linear procedure for computing the similarity value for each one of the cases in the case base.

The different approaches based on the use of *k*-d trees differ in how the discriminator attribute is determined, and how the partition value is selected at each internal node of the tree. The *original k-d tree formulation* (Bentley, 1975) proposed to choose the discriminator attribute cycling over the list of the attributes, and selecting the partition value at random. The *standard k-d tree approach* (Friedman et al., 1977) suggests selecting the attribute with the highest spread (range) in values, and the median of the attribute value distribution.

The application of *k*-d trees for *similarity-based retrieval* in case-based reasoning was first proposed by Wess in (Wess et al. 1993). In that work, authors explored four approaches for selecting the attributes: using the category-utility of CobWeb (Fisher, 1987), using the entropy measure proposed by (Quinlan, 1983), selecting the attribute maximizing the dispersion (interquartile distance) with respect to the similarity measure *sim*, and selecting the attribute which maximizes the average similarity within partitions and buckets. This last strategy was the best one according to the experimentation undertaken by authors.

In addition, in the work by Bergmann and Tartakovski in (Bergmann and Tartakovski, 2009) which was addressing the generalized cases retrieval, they proposed to select the discriminator attribute as the one maximizing the dispersion of projections of cases, and also taking into account the length and intersection of projected case intervals.

All contributions previously cited above have shown that the standard *k*-d tree approach is feasible to be improved, and until now, standard *k*-d tree approach has been improved in different ways as described in the mentioned research works. In our research work, we formulate other proposals to improve the standard *k*-d tree approach. The results obtained shown that AvKdTree, and especially, the NIAR *k*-d tree is an efficient case retrieval method. And the proposed indexing technique is a fast indexing strategy and the generated trees are well-balanced according to the results obtained.

2.1.2 The hyperball strategy with BWB and BOB tests

Usually, there are no identical cases (*exact-case search*) in the case library and the retrieval procedure must look for the most similar ones. One of the most used exploration technique for *similar-case search* (not losing the most similar case) is the technique proposed in (Friedman et al., 1977), where the authors proposed a technique to address the search for the *m* most similar cases in a *k-d* tree. The tree is used as a binary search tree leading to a bucket where *b* specific cases are stored. It is necessary to compute the similarity of each case stored in the bucket using the predefined similarity measure *sim*. If the *m* most similar cases are searched for, a queue containing these most similar cases can be built up. Using this queue, the authors propose to draw a hyperball around the given problem including the *m* most similar cases found in the current bucket. Every case being at least as similar as the examined ones must be within this constructed *k*-dimensional hyperball. By using this hyperball, it is possible to decide which buckets are needed to be examined next. The authors propose two test procedures for the implementation of this idea: Ball-Within-Bounds (BWB) and Bounds-Overlap-Ball (BOB) (see figure 5). These procedures check whether it would be reasonable to explore certain areas of the search space or not. These tests can be carried out without retrieving the respective cases. The geometric bounds of the considered subspaces are

used to compute a *similarity interval* whose upper bound answers the question about to explore or not. For finding the m most similar cases for a given query case, a recursive tree search is applied. The input needed is the query case X_q, the number m of most similar cases, the *k-d* tree represented by its root node, and the similarity measure sim. During the search process a priority queue PQC is continuously updated. It includes the m most similar cases. $PQC[n]$ denotes the nth most similar case information, and $PQS[n]$ stores its actual similarity value. If the recursive search procedure examines a leaf node, the similarity of all included cases is computed and, if necessary, the PQC is updated. If the examined node is an inner node, then the traversing procedure is recursively called for that son node which should include the query case. If this call terminates, it is tested whether it is also necessary to examine the other son node by using the BOB test.

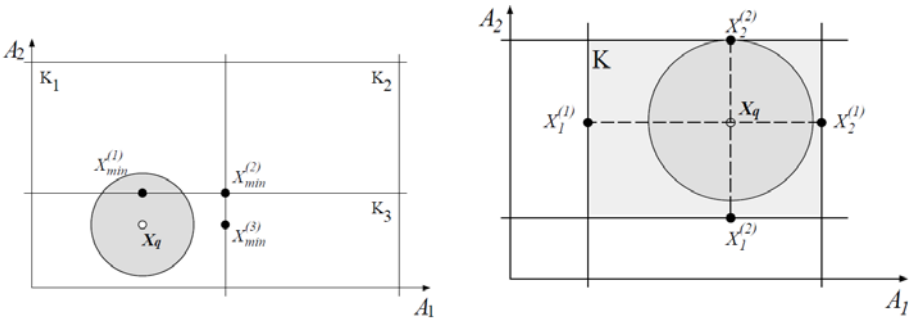

Fig. 5. BOB and BWB test basic ideas, from (Wess et al., 1993)

The BOB test is true if the cases of the actual tree node have to be explored. The inner nodes are correct generalizations of all the cases they represent in the sense that they include the geometric (upper and lower) bounds, for every indexing attribute, which correspond to the respective subspace.

$$BOB \Leftrightarrow Sim(X_{min}, X_q) \geq PQS[m] = Sim(PQC[m], X_q)$$

These geometric bounds are used to compute a similarity interval whose upper bound answers the question about to explore or not. The closest point X_{min} within the actual node's subspace is computed as the projection onto the actual node's geometric bounds (see figure 5). X_{min}

is on the actual node's bounding box on the edge facing the query case X_q. If there is no overlapping in any of the k dimensions between the node's bounding box and the k-dimensional ball around X_q, then X_{min} is a corner of the bounding box. If X_q is within the bounding box then $X_q = X_{min}$ (see figure 6). Before the recursive search procedure ends, the BWB test is applied. This test is true if the k-dimensional ball round X_q is completely within the bounding box the actual tree node (Fig. 6).

$$BWB \Leftrightarrow Sim(X_1^j, X_q) < PQS[m] \ \wedge \ Sim(X_2^j, X_q) < PQS[m] \ \forall j = 1, \cdots, k$$

In this case, no overlapping with other bounding boxes is possible. Thus, the search is finished, and the m most similar cases for the current query case according the similarity measure sim are found.

Wess et al. (Wess et al., 1993) also proposed an improvement of the hyperball with BOB and BWB bounds by using information about the known cases. The basic idea is to describe the subspaces that really include cases more precisely. Therefore, all occurring maximal and minimal values for each attribute are stored. These ideas led them to define the concept of *Minimal Virtual Bounds* (see figure 6).

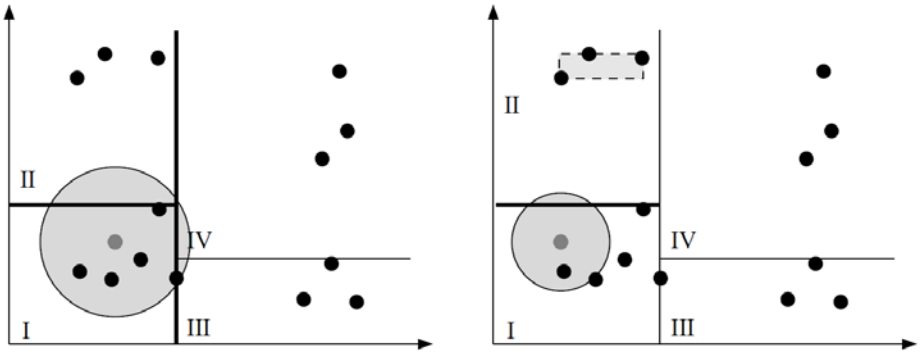

Fig. 6. BOB Tests and Minimal Virtual Bounds, extracted from (Wess et al., 1993).

In the right part of the figure 6, the intuitive idea of how much from the description space need not to be considered during search by looking at the white areas is depicted. A BOB test using minimal virtual bounds recognizes that the bucket II does not include any better cases.

The *Minimal Virtual Bounds* of a tree node for the dimension k are defined as follows:

$$minBounds[k].Upper = max(Case_i[k])$$
$$maxBounds[k].Lower = min(Case_i[k])$$

$\{Case_i[k]\}$ *Denotes the set of all values of attribute k for the cases* $Case_i$ that it is being represented by the tree node.

While minimal virtual bounds lead to an improvement of the BOB tests, an analogous idea, of *Maximal Virtual Bounds*, can be used to improve the BWB tests. For the latter, it is reasonable to describe the searched subspace as precise as possible such that the k-dimensional hyperball around the query case has the maximal chance to be completely within that ball. Therefore, the *Maximal Virtual Bounds* were introduced, as described in figure 8.

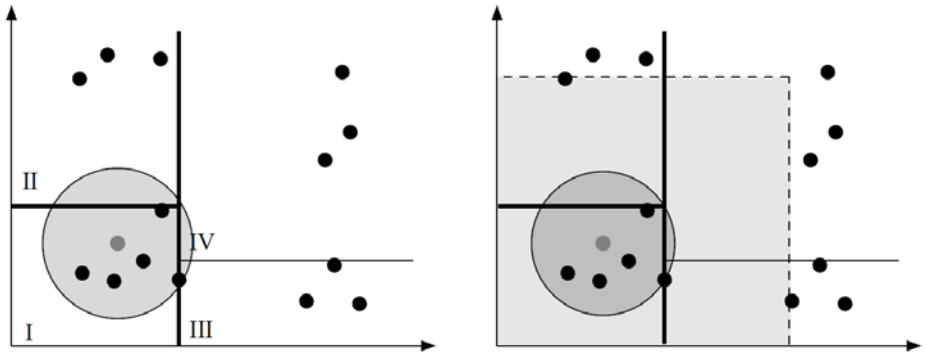

Fig. 7. BWB tests and Maximal Virtual Bounds, extracted from (Wess et al., 1993).

Within such maximal virtual bounds it is guaranteed that no more similar cases can be found within those borders. The computation of the maximal virtual bounds requires more effort because it is not based on the analysis of the cases, but on the analysis of all neighboring subspaces. The virtual bounds can be computed during tree generation.
Within the maximal virtual bounds, it is guaranteed that only cases of the respective subspace itself belong to it. There are no more similar cases outside these boundaries. Thus, the search process is finished.

2.2 Case Base Maintenance

Case Base Maintenance (CBM) is defined by David Leake such as the process of refining a CBR system's case base to improve the system's performance:

"Case base maintenance implements policies for revising the organization or contents (representation, domain content, accounting information, or implementation) of the case base in order to facilitate future reasoning for a particular set of performance objectives".

Maintenance in CBR can mean a number of different things: out-of-date, redundant, or inconsistent cases may be deleted; groups of cases may be merged to eliminate redundancy and improve reasoning power; cases may be re-described to repair inconsistencies. Thus case-base maintenance may involve revising indexing information, links between cases, or other organizational structures and their implementations. Maintaining Case-Base contents may affect a single case or multiple cases. It may revise:

- The case representations used
- Either domain information in the case-base or accounting "information"
- How case representations are implemented
- The case-base at the implementation level, representation level, or the knowledge level

The Leake and Wilson framework of CBM shown in (Leake and Wilson, 1998; Wilson and Leake, 2001) is used in this section to describe the concepts and to understand the state of the art in Case Base maintenance (see figure 8). Those research works presents a first attempt at identifying the dimensions of the Case Base maintenance. They show that characterizations along such dimensions can suggest avenues for future Case Base maintenance research and presents initial steps exploring one of those avenues: identifying patterns of problems that require generalized revisions and addressing them with lazy updating.

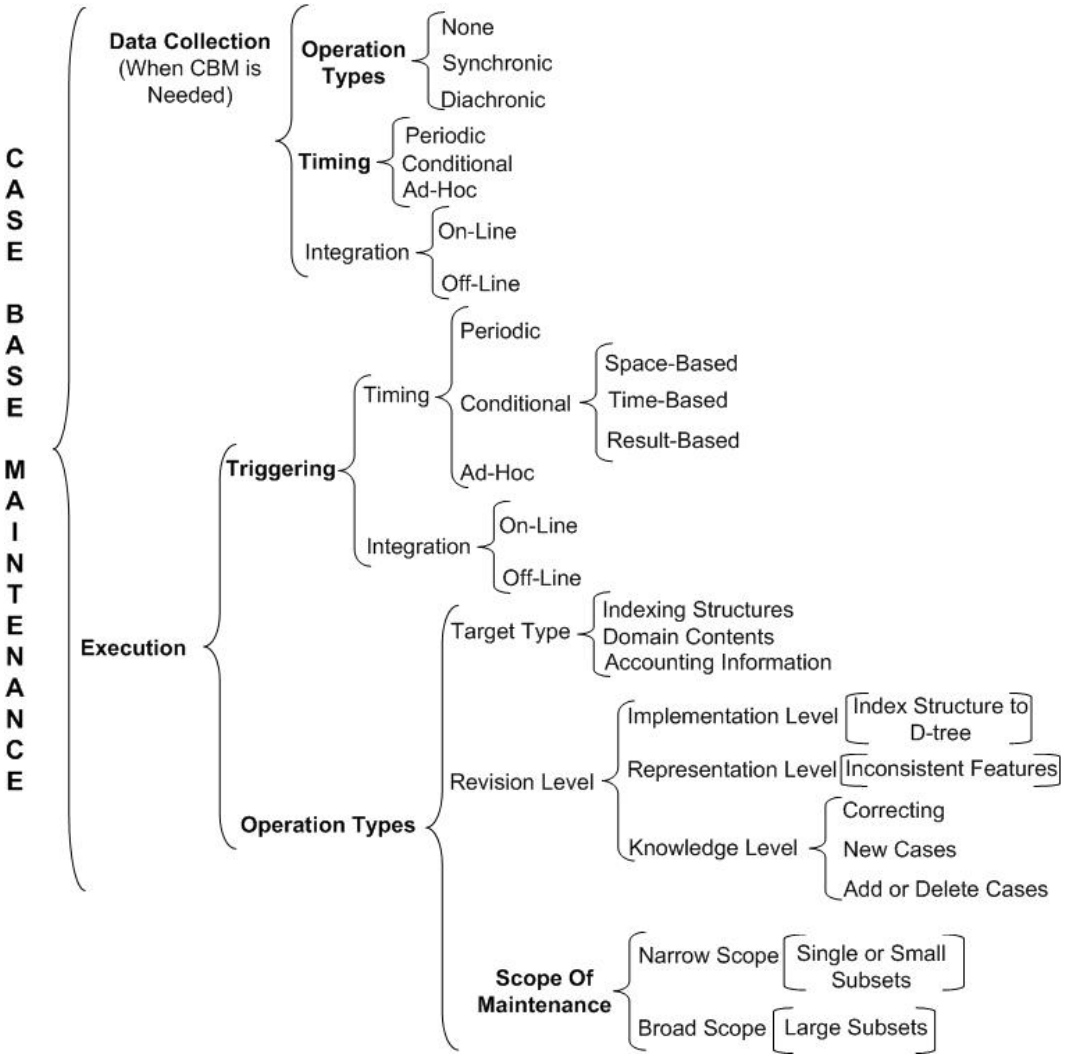

Fig. 8. Case Base Maintenance from Leake's and Wilson Framework
(Leake & Wilson, 1998, 1999) and (Wilson & Leake, 2001)

As CBR systems are deployed in real-world situations, the issue of case maintenance becomes more and more critical. Uncontrolled Case Base growth can cause serious performance problems as retrieval efficiency degrades and incorrect or inconsistent cases become increasingly difficult to detect.

Case Base Maintenance has become an active CBR research area, producing results with important ramifications for both the theory and practice of CBR, some research works are the proposals; (Perner, 2006; Iglezakis et al., 2004; Portinale and Torasso, 2001; Smyth and Mckenna, 2001; Leake and Wilson, 2000; Yang and Wu, 2000). Much significant work in this area focuses on developing methods for reducing the size of the Case Base while maintaining Case Base competence (Barry and Paul, 2001; Smyth and Mckenna, 2001, 1999; Wilson and Leake, 2001; Leake and Wilson, 1998; Smyth, 1998; Smyth and Cunningham, 1996; Smyth and Keane, 1995). The goal of achieving compact competent Case Bases addresses important performance objectives for CBR systems. As an added benefit, compact Case Bases decrease communications costs when Case Bases are used as vehicles for knowledge sharing or are transferred in distributed CBR systems. However, Case Base compactness is only an opinion to a proxy for performance in a CBR system, rather than an end in itself.

Experience with the growing number of large-scale CBR systems has led to increasing recognition of the importance of Case Base maintenance. Many researchers have addressed pieces of the Case Base maintenance problem, considering such issues as maintaining consistency and controlling Case Base growth.

CBM methods aim to improve Competence and Efficiency of the CBR systems. Where *Performance* objectives provide criteria for evaluating the internal behavior and task performance of a particular CBR system for a given initial case-base and sequence of problems solved. The performance objectives may be quantitative or qualitative.

2.2.1 Concepts about efficiency and competence

Performance objectives may change over time to reflect varying external circumstances, which may necessitate changing (maintaining) maintenance policies as well. Performance models which combine competence and efficiency can be used to guide the deletion of redundant cases from a case-base in order to optimize system performance.

Effective maintenance Case Base reasoning depends on the ability to measure and manage case competence as well as case efficiency.

Maintenance policies are described in terms of how they gather data relevant to maintenance, how they decide when to trigger maintenance, whether they react to problems or proactively forestall them, the types of maintenance operations available and how selected maintenance operations are executed.

2.2.1.1 Efficiency

Efficiency means that a CBR system is the most efficient one, it if requires the minimal resources needed to solve any case in the domain. The resources are twofold: the time required solving a case, and the size of the Case Base needed to solve a case. Thus, efficiency has to do both with case solving time and the size of the case base.

Utility problem: The utility problem highlights the link between knowledge base (Case Base) size and the retrieval time needed to select an item of knowledge to use in a particular problem solving situation. Addition of more knowledge results in potentially severe efficiency degradation. Utility metric is used to take into account the cost of maintaining.

2.2.1.2 Competence

Competence means the range of problems that can be satisfactorily solved. During future problem solving, as cases are learned and deleted from the Case Base, the case categories must be updated by recomputing the coverage and reachability of affected cases to adjust the categories accordingly.

Competence-Directed Maintenance Individual knowledge items only contribute to problem solving efficiency. An underlying first principles problem solver is always used to encode basic problem solving competence. Cases contribute to both competence and efficiency.

2.2.1.3 The Foundations of Competence

The following definitions were formulated in the research works (Barry and Paul, 2001; Smyth and McKenna, 2001; Smyth, 1998; Smyth and Cunningham, 1996;

Smyth and Keane, 1995), and are used to give a characterization and overview of the CBM framework. The local competence contributions of individual cases can be characterized by two sets. The coverage set of a case is the set of all target problems that this case can be used to solve. It is not the responsibility of the competence model to explicitly define the ”solves” predicate other than to assume that it exists for any target CBR system. The reachability set of a target problem is the set of all cases that can be used to solve it. It is not possible to enumerate all possible future target problems (T), but by using the case base (C) itself as a representative of the target-problem space, we can efficiently estimate these sets, as shown in the definitions of coverage and reachability.

Coverage of a case: the set of target problems that a given case can successfully solve.

Cases with large coverage sets seem likely to be making large competence contributions. In contrast, cases that are members of large reachability sets seem likely to be less important, as many other cases exist which can solve similar problems. The ability to measure coverage and reachability is the key to under- standing competence in CBR. Of course it should be clear that the coverage and reachability sets depend on the characteristics of particular retrieval and adaptation methods.

Definition 1: Case Coverage

Given a Case Base: $C = \{c^1 \ldots c^n\}$, *and a case* $c^i \in C$, $i \in \{1, \ldots, n\}$

$$Coverage(c^i) = \{c' \in C : Adaptable(c^i, c')\}$$

Reachability of a target problem: the set of cases that can be used to solve a given target problem.

Definition 2: Case Reachability

Given a Case Base: $C = \{c^1 \ldots c^n\}$, *and a case* $c^i \in C$, $i \in \{1, \ldots, n\}$

$$Reachability(c^i) = \{c' \in C : Adaptable(c', c^i)\}$$

Competence Groups. Coverage and Reachability sets provide a measure of local competence only. In order to estimate the true competence contributions of cases, it is necessary to model the interactions between related cases, specifically in terms of how their coverage and reachability sets overlap.

$$RelatedSet(c^i) = CoverageSet(c^i) \cup ReachabilitySet(c^i)$$

$$For\ c^1, c^2 \in C,$$

$$SharedCoverage(c^1, c^2) \Leftrightarrow [RelatedSet(c^1) \cap RelatedSet(c^2)] \neq \emptyset$$

$$For\ G = \{c^1 \dots c^n\} \subseteq C,$$

$$CompetenceGroup(G) \Leftrightarrow \forall c^i \in G, \exists c^j \in G - c^i : SharedCoverage(c^i, c^j)$$
$$\wedge$$
$$\forall c^k \in C - G, \nexists c^l \in G : SharedCoverage(c^k, c^l)$$

First, the related set of a case to be the union of its coverage and reachability sets. When the related sets of two cases overlap, we say that they exhibit shared coverage, and cases can be grouped together into so-called competence groups that are maximal sets of cases exhibiting shared coverage. In fact, every case base can be organized into a unique set of competence groups which, by definition, do not interact from a competence viewpoint i.e., while each case within a given competence group must share coverage with at least one other case in that group, no case from one group can share coverage with any case from another group.

The importance of the competence group concept is that each group makes a unique contribution to competence. Thus, competence groups allow us to partition the case space into non-interacting groups of cases. Each group can be treated independently of all other groups in the case base from a competence viewpoint, and this means that the competence of a case base as a whole can be computed as the sum of the competence contributions of each competence group.

While each competence group makes a unique contribution to overall competence, not every case in a group makes the same (or even a positive) contribution. The final stage of the model involves identifying the so-called footprint cases and measuring their relative competence contributions. By definition, footprint cases are those cases, which make a positive competence contribution to competence, and the set of footprint cases of a group covers the entire group. In contrast, non-footprint cases make redundant competence contributions because they are fully covered by nearby footprint cases.

We will revisit the concept of footprint cases in a later section, where we will provide algorithms for identifying the footprint of a group. The relative competence contribution of an individual case is estimated by the relative coverage (RC) measure, which estimates the competence contribution of an individual case c as a function of the size of the case's coverage set (see next Equation).

RC weights the contribution of each covered case by the degree to which these cases are themselves covered. It is based on the idea that if a case c' is covered by n other cases, then each of the n cases will receive a contribution of $1/n$ from c' to their relative coverage measures.

$$Relative\ Coverage(c) = \sum_{c' \in CoverageSet(c)} \frac{1}{|ReachabilitySet(c')|}$$

2.2.2 Maintenance Data Collection

Maintenance Data Collection gathers, synthesizes, and distills the data about the case base and about system processing. This information will be used to determine whether maintenance operation should be performed or not. It gathers information about:

- *Individual cases*: it might record the number of times a case has been successfully used or the number of times it has failed.
- *The case base*: the case base as a whole could involve, for example, monitoring the size of the case base.
- *Processing*: it might involve noting clusters in input problems or input problems that the system is unable to solve successfully.

There are three approaches to collecting and analyzing data to decide when CBM is needed: None, Synchronic, or Diachronic. The simplest is to do no collection at all:

- A policy with **no data collection** makes maintenance decisions independently of the present or past state of the case base. As such, this type of policy is referred to as nonintrospective. For example, a CBR system that updates its case base by unconditionally adding a case each time it adapts a prior case would need no data collection. This is the approach of most CBR systems. Similarly, a system may drive maintenance according to external information sources. This is valuable for proactive maintenance, for example, to add cases to a help-desk case base in anticipation of future queries.

- **Synchronic** (Policies that consider snapshot information). More sophisticated reasoning is enabled by considering a snapshot of the current case base in part or as a whole. Examination of this information can determine, for example, whether a case is worth adding to a case base because it increases the competence of the CBR system or whether a solution can be discarded without affecting competence (Smyth and Keane 1995). As another example see Reinartz *et al*. contributions in (Reinartz et al,. 2000) where they propose a set of measures that can be computed to assess the overall quality of a case base in order to trigger maintenance.

- **Diachronic** (Policies that consider changes in the case base over time). The most informative approach is to collect data over time, over a sequence of snapshots, in order to identify trends in how Case Base contents and usage are changing. For example, a policy that gathered information about trends in retrieval times to identify the onset of utility problems would be diachronic. Because synchronic and diachronic collection examines the internal state of the case base, both are referred to as introspective.

Regarding the time when the data collection is needed it **Timing** could be: *periodic*, *conditional*, or *ad hoc*. A maintenance policy must specify when data collection is performed. Periodic timing happens at an estab-

lish frequency with respect to the CBR cycle is termed continuous. Conditional data collection is performed in response to a well-defined but non-periodic condition. Ad hoc timing happens under well-defined conditions determined externally to the CBR system.

Integration of the data collection could be: *on-line* or *off-line*. Data collection may operate on-line, during the course of an active reasoning episode, or offline, during a pause in reasoning, such as waiting for user input or when idle between reasoning episodes. The choice between on-line and offline processing may affect the resources that can be devoted to the analysis process, making it important for determining whether a policy is appropriate for time-constrained processing.

2.2.3 Maintenance Execution

Execution is characterized by the timing of maintenance operations and their integration with other system processing. Execution timing is described using data collection (periodic, conditional, or ad hoc). Execution integration is described as on-line or off-line depending on whether maintenance operations are performed during or between reasoning episodes.

Triggering: The results of data analysis serve as or determining whether CBM is necessary. Both the timing and integration dimensions dis- cussed previously apply to this step as well. Maintenance triggering evaluates whether to perform maintenance, selects maintenance actions to use, and may set parameters to guide their future execution (e.g., determining when they will be performed). Triggering can be done periodically, conditionally, or on an ad hoc basis and on-line or off-line.

Conditional triggering can be subdivided into three classes depending on the conditions that determine whether maintenance is triggered:

- Space-based (e.g., filling a limited amount of case storage),
- Time-based (e.g., retrieval time exceeding a threshold), or
- Result-based (e.g., the system failing to solve a given problem or the wrong case being retrieved).

Triggering integration could be on-line or off-line. Most of the maintenance triggering tasks will be undertaken during the CBR cycle (on-line). However, another times it could be triggered off-line.

Operation types: Different maintenance policies revise different types of in-formation (the target type) at different levels (the revision level).

- Target type: Revision operations can focus on four types of targets
- Revision level: Revision operations can make revisions with ramifications at three levels:
    - *affecting only the implementation level* (i.e., changing an indexing structure from a list to a D-tree when the case base exceeds a certain size or changing case representations from lists to vectors),
    - *affecting the representation level* (i.e., reconciling inconsistent feature names or case formats in cases that come from different sources), or
    - *affecting the knowledge level* as well (i.e., correcting an erroneous feature value, generalizing case values, or adding or deleting cases).

Scope of Maintenance (*broad* or *narrow*): A given operation may be applied locally to few items in the case base, or more globally. Operations that affect a single case or a small subset of the case base have *narrow scope*, and operations that affect a large subset or the entirety of the case base have *broad scope*. This dimension is especially useful when characterizing resource-bounded processing.

2.2.4 Categorizing Policies for CBM

The several kinds of policies that can be undertaken in CBM could be dived in the following groups:

- *Policies targeting domain content*: may be organized in policies aiming to adding and deleting cases, and policies at revising internal case content.

- *Standard case learning and manual maintenance*: always adding each new case to the case base.

- *Additional* policies aimed at case retention: based on coverage and reachability, and integrating offline or online, beneficial or detrimental.

- *Policies aimed at interval case content*: are aiming at internal case content.

- *Policies targeting indices*: A number of classification systems using IBL and related techniques IBL_n include policies for eliminating noisy and redundant instances from a set of training examples (cases).

- *Policies targeting* maintenance policies include the capability for meta-maintenance of the maintenance strategies themselves.

2.2.5 Synthetic Analysis of CBM contributions

After an analysis of the related contributions in the literature in Case Base Maintenance, summary tables 1, 2 and 3 with the works and the main aspects of the contributions has been elaborated.

Table 1. Contributions to CBM field part 1

Reference	Brief Resume of Contribution
(Portinale & Torasso, 2001; Portinale *et al.*, 1999; van Someren *et al.*, 1997)	Analyze CBM in a multimodal architecture and presents the policies: Replace Policy and Learning by Failure with forgetting.
(Iglezakis *et al.*, 2004; Reinartz & Iglezakis, 2000; Reinartz *et al.*, 2001) (Smyth & Keane, 1995)	Propose to extends the CBR cycle in 6 steps(2 steps extra in Maintenance), Define quality measures, Describe methods in restore step and the relation of the review steps. Show results of traditional deletion policies, introduce the policies. Coverage and Reachability. Introduce 6 categories of cases.
(Smyth & Mckenna, 2001)	Introduce the terms of Competence and competence of groups. Propose 4 algorithms models for CBR.
(Smyth, 1998)	Presents a variation of the algorithm FootPrint Deletion, the spanning case are not considered.
(Yang & Zhu, 2001; Zhu & Yang, 1999)	Propose a greedy algorithm based on case addition, the algorithm aims is to find a near-optimal CB of size K efficiently. The addition-based policy guaranteed a concise CB. The addition-based policy.

Table 2. Contributions to CBM field part 2.

Reference	Brief Resume of Contribution
(Yang & Wu, 2000)	Presents a retrieval proposal to improve the performance of a CBR. The idea is to build small case bases storage in distributed way. The building of the case bases are guided applying clustering, using his proposal of the algorithm New Condorcet Criteria.
(Munoz-Avila, 2001)	Show the policies: Case Index Refinament; based in the relative importance of a case based in the retrieval. Case Retantion; The contribution of a Retrieval case and the adaption effort. The policies are based in the case index revision, and perform the CB adaptation by derivational Replay.
(Leake & Wilson, 1998; Wilson & Leake, 2001)	Presents a first attempt identifying the dimensions of CBM and a general CBM framework, give a definition of CBM, Characterize CBM policies presenting a basic dimension of it. In 3 sub-steps: Data Collection, Triggering and Execution.
(Leake & Wilson, 2000)	Presents an argument for integrating performance, considering addition and deletion allowing a finer-grained optimization, using finer-grained performance metric to guide the addition and deletion. Shwos a relationship between competence, compacness and adaptation performance.
(Perner, 2006)	Propose a similarity measure and an algorithm that incrementally learns the organizational structure of a case, Organize the cb in a graph subsumtion. Concider the learning of the retrieval knowledge.
(Arshadi & Jurisica, 2004, 2005)	Propose MOE4CBR to improve the prediction accurancy such as a manintenance teachnique implementing sptectral clustering and logic regression to improve the classification specially in high-dimensional biological data set. The technique has the main concepts: Ensemble, Clustering and Feature selection.

Table 3. Contributions to CBM field part 3.

Reference	Brief Resume of Contribution
(Salamó & Golobardes, 2003)	Present two approaches based on deletion policies based on Rough Set Theory, those with the gaol of reduce the CB size. On the idea of remove noisy cases and achieve a good generalization accuracy the approaches are:ACCM and NACCM.
(Salamó & Golobardes, 2004)	Propose a dynamic model thats updates itself, using the data form learning. The knowledge of the process is update applying Reinforcement Learning in the revise phase.
(Yang *et al.*, 2003)	Propose 3 algorithms for maintenance the CB. Combine similarity and diversity reducing the CB size. The algorithms proposed are: Random-based Selection, RelDiversity-Based Selection and Evolution Strategies-Based Selection.
(Ni *et al.*, 2005)	Are structured in maintain the CB adopting the outlier and case sieving techniques. Their results show that the CB gets satisfactory and stably. The algorithm proposed detect erroneous cases from outliers. The algorithm is devided in: Extracting outliers, Analyzing and eliminating and saving.

2.3 Introspective Reasoning

The tenets of metacognition are monitoring, modeling, evaluating and controlling of the cognition (Anderson and Oates, 2007). Metacognition means to have the knowledge and how to use this knowledge to improve performance and reduce as low as possible deficiencies. The main challenge of meta-reasoning is to find answers to most predictable future needs. Introspective reasoning is an invaluable part in meta-reasoning. John McCarty in (McCarthy, 1979) defines introspection as a machine with beliefs of its internal state. Leake and Wilson in (Leake and Wilson, 2008) defines that an introspective strategy must be characterized by properties defining the When, What and How the learning step is done. Research on meta-reasoning and introspective reasoning has been reviewed and analyzed especially in fields such as physiology, social science and especially in artificial intelligence. In this later, some scientific contributions define this process to be successfully implemented in computer science, such intelligence algorithms. Cox in (Cox, 2005) does an extensive review of those topics where discuses cognition about Meta-cognition and establish a relationship between psychology and computer science fields. Anderson in (Anderson and Oates, 2007) extends the definition of metacognition in computer science illustrating terms such as learning, meta-learning and meta-cognition. Afterwards Cox in his Manifesto (Cox and Raja, 2008) explains in detailed fundamental terms such as Ground Level, Object Level and Meta-level. Thus, Meta-reasoning has been extensively studied and characterized such as been demonstrated in literature.

The meta-reasoning has been implemented effectively in the Case-Based Reasoning (CBR) (Arcos et al, 2008; Leake, 2001; Fox and Leake, 2001; Leake et al, 1995; Fox and Leake, 1994). In (Leake et al, 1995) the objective is to improve the adaptation step by introspective reasoning considering the requirements and built a library with adapted cases. The introspective reasoning process implements memory search as form of planning and use operators such as sensors to detect internal states. Their results show a successful adaptation of knowledge. In Arcos et al, 2008 the performance of the CBR cycle through an internal reasoning process that guides the use of its cases is considerable improved. Arcos et al. implements the method such as an introspective reasoner monitor, pursuing the goal to determinate the causes of failures, and then adjusts retrieval and reuse strategies to improve solution quality.

Fox and Leake in (Fox and Leake, 2001; 1994) have proposed some improvements to the CBR scenario, where meta-reasoning is embedded in the cycle. In (Fox and Leake, 1994) proposed to refine the indexing criteria by the implementation of an introspective reasoning framework with the aim of refining reasoning processes, this frame-work is extended and detailed in (Fox and Leake, 2001). Sànchez-Marrè in (Sànchez-Marrè et al., 2000) introduced the idea of retrieval cases in hierarchical case libraries, where a radius is defined by the implementation of a metric measure, where if the distance from the library to the case exceeds the threshold, then the case will not store there. Our proposal shares a similitude to this, but exists some differences between them. In our case the proposal aims to build the required prototypes and make a decision of where going to store the new cases. The core of our proposal is the implementation of a stochastic method working with two moments. That will be detailed in the thesis.

The meta-learning and meta-reasoning have been widely studied; even more, those topics are one of the goals to achieve by the Artificial Intelligence community. In CBR community interesting and efficient methods have been proposed in this sense. Several proposals that have been published have had the aim of implementing meta-reasoning. Some proposals use introspective reasoning as our method to detect some expected behavior or to evaluate the behavior to guarantee the good performance of the CBR system. Our proposal uses an introspective reasoning strategy to evaluate the performance of a set of algorithms in learning phase. This done autonomously by the system, the introspective reasoning strategy involves the implementation of a set of rules to evaluate whether the algorithms gives the best performance to the system. With the strategy implemented we answer the questions formulated in Leake and Wilson, 2008 that consist in ¿When? ¿What? And ¿How the system learns? ¿When? When there exists a strategy that improves the performance. ¿What? The new algorithm. And finally ¿How the system learns? the system learns by the implementation of the new indexing strategy proposed. This is done when the reasoning gathers sufficient information and its results improve the other algorithms, when an algorithm over perform others, then it is considered as the best candidate to be used. All these answers and our proposal are

defined according to introspective reasoning as reflected in (Cox and Raja, 2008; Anderson and Oates, 2007; Cox, 2005).

2.4 Stochastic Learning

The Stochastic method has been widely used in several applications where learning is required. Here data plays the main role in the learning.

One field in artificial intelligence where stochastic learning has been used is in the clustering field. Contribution in this field proposes improved methods to get better results in finding clusters. For instance see (Swee et al., 2011), where authors examine a practical stochastic clustering method that has the ability to find clusters in datasets without requiring users to specify the centroids or the number of clusters. Other application is in the field of robotics and learning, with the proposal of (Zhang *et al.*, 2013). They propose an efficient Stochastic Clustering Auctions for centralized auctioning and homogeneous robot teams. For others contributions see (Rebagliati et al., 2013), (Wang et al., 2012). For the application in Trees see (Akbari and Reza, 2011). There are many other contributions in the application of stochastic methods. One of the main contribution in the CBR field is the proposal of (Finestrali and Muñoz-Avila, 2013) where they studied the problem of explaining events in stochastic environments. They claimed that a system using stochastic explanations reacts faster to abrupt changes in the environment than a system using deterministic explanations. They demonstrated this claim in a CBR system, while playing a real-time strategy game. In chapter 5 some differences between our proposal and their proposal will be discussed.

Chuan in (Tan et al., 2010), implements a stochastic method where building and classification of clusters are the issue. This is done without human interaction. Then, they use the time variable to evaluate the probability of belonging to some cluster. In our method, we use a time variable to indicate the time when the acquired data was taken. Both methods learn cases and store it, but we differ from Chuan in the method of learning the cases. In our case, we talk about representative prototypes. The learning of cases in our proposal is conditioned to accomplish with the maximal acceptable dissimilarity or dispersion.

The following is a revision of the stochastic method according with Taylor and Karlin in (Taylor and Karlin, 1998). The word "stochastic" derives from the Greek and means "random" or "chance". The antonym is "sure," deterministic," or "certain". A deterministic model predicts a single outcome form a given set of circumstances. A stochastic model predicts a set of possible outcomes weighted by their likelihoods, or possibilities.

A coin flipped into the air will surely return to earth somewhere. Whether it lands heads or tails is random. For a "fair" coin, we consider these alternatives equally likely and assign to each the probability $1/2$. However, phenomena are not in themselves inherently stochastic or deterministic. Rather is the choice of the observer to model a phenomenon as stochastic or deterministic. The choice depends on the observer's purpose. Most often, the proper choice is quite clear, but controversial situations do arise. If once fallen the coin is quickly covered by a book, so that the outcome "heads" or "tails" remains unknown, two participants may still usefully employ probability concepts to evaluate what is a fair bet between them. That is, they may usefully view the coin as random, even though most people would consider the outcome now to be fixed or deterministic. As a less mundane example of the converse situation, changes in the level of a large population are often usefully modeled deterministically, in spite of the general agreement among observers that many chance events contribute to their fluctuations.

Scientific modeling has three components: (i) a natural phenomenon under study, (ii) a logical system for deducing implications about the phenomenon, and (iii) a connection linking the elements of the natural system under study to the logical system used to model it. If we think of these three components in terms of the great-circle air route problem, the natural system is the earth with airports at Los Angeles and New York; the logical system is the mathematical subject of spherical geometry; and the two are connected by viewing the airports in the physical system as points in the logical system. The modern approach to stochastic modeling is in a similar spirit. Nature does not dictate a unique definition of "probability," in the same way that there is no nature-imposed definition of "point" in geometry. "Probability" and "point" are terms in pure mathematics, defined only through the properties invested in them by their respective sets of axioms.

In many real life situations, observations are made over a period and they are influenced by random effects, not just at a single instant but throughout the entire interval of time or sequence of times (Athanasios, 1984).

In a "rough" sense, a random process is a phenomenon that varies to some degree unpredictably as time goes on. If we observed an entire time-sequence of the process on several different occasions, under presumably "identical" conditions, the resulting observation sequences, in general, would be different.

A random variable (RV) is a rule that assigns a real number to every outcome of a random experiment, while a random process is a rule that assigns a time function to every outcome of a random experiment.

A random experiment may lead not only to a single random variable, but to an entire sequence of random variables.

$$\{X_i : i = 1, 2, 3 \cdots\} = \{X_1, X_2, X_3 \cdots\}$$

Consider the random experiment of tossing a dice at $t = 0$ and observing the number on the top face. The sample space of this experiment consists of the outcomes $\{1, 2, 3, \cdots, 6\}$. For each outcome of the experiment, let us arbitrarily assign a function of time $t\,\{0 \leq t < \infty\}$ in the following manner; considering the list of last outcomes, where its function time is as follows:

$$\begin{gathered} X_1(t) = -2, \\ X_2(t) = -4, \\ X_3(t) = 2, \\ X_4(t) = 4, \\ X_5(t) = -t/2, \\ X_6(t) = t/2 \end{gathered}$$

The set of functions $\{X_1(t), X_2(t), \cdots, X_6(t)\}$ represents a random process. A random process is a collection of RVs $\{X(s,t)\}$ that are func-

tions of real variable, namely time t where $s \in S$ (sample space) and $t \in T$ (parameter set of index set).

The set of possible values of any individual member of the random process is called state space. Any individual member itself is called a sample function or a realization of the process.

Classification of random processes

Depending on the continuous or discrete nature of the state space S and parameter set T, a random process can be classified into four types:

1. If both T and S are discrete, the random process is called a *discrete random process.* For example, if X_n represents the outcome of the *n*th toss of a fair dice, then $\{X_n, n \geq 1\}$ is a discrete random sequence, since $T = \{1, 2, 3, \cdots\}$ and $S = \{1, 2, 3, 4, 5, 6\}$.
2. If T is discrete and S is continuous, the random process is called a *continuous random sequence*.
   For example, if X_n represents the temperature at the end of the nth hour of a day, then $\{X_n, 1 \leq n \leq 24\}$ is a continuous random sequence, since temperature can take any value in an interval and hence continuous.
3. If T is continuous and S is discrete, the random process is called a *discrete random process*.
   For example, if $X(t)$ represents the number of telephone calls received in the interval $(0, t)$ then $\{X(t)\}$ is discrete random process, since $S = \{0,1,2,3, \cdots\}$.
4. If both T and S are continuous, the random process is called a *continuous random process*. For example, if $X(t)$ represents the maximum temperature at a place in the interval $(0, t)$, $\{X(t)\}$ is a continuous random process. In the names given above, the word 'discrete' or 'continuous' is used to refer to the nature of T.

Specifying a random process.

Let $X_1, X_2, \cdots X_k$ be the k random variables obtained by sampling the random process $X(t, \zeta)$ at times $t_1, t_2, \cdots t_k$:

$$X_1 = X(t_1, \zeta), X_2 = X(t_2, \zeta), \cdots, X_k = X(t_{k2}, \zeta).$$

The joint behavior of the random process at these k time instants is specified by the joint cumulative distribution for the vector random variable $(X_1, X_2, \cdots, X_k)$.

A *stochastic process* is specified by the collection of the kth-order joint cumulative distribution functions:

$$F_{X_1 \cdots X_k}(x_1, x_2, \cdots, x_k) = P[X_1 \leq x_1, X_2 \leq x_2, \cdots, X_k \leq x_k]$$

For any k and any choice at sampling instants $t_1, \cdots, t_k$.

If the stochastic process is discrete-valued, then a collection of probability mass functions can be used to specify the stochastic process:

$$P_{X_1 \cdots X_k}(x_1, x_2, \cdots, x_k) = P[X_1 = x_1, X_2 = x_2, \cdots, X_k = x_k]$$

The **Mean** $m_X(t)$ of a random process $X(t)$ is

$$m_X(t) = E\,[X(t) = \int_{-\infty}^{\infty} x f_{x(t)} x\, dx.$$

In general, $m_X(t)$ is a function of time. Suppose we write $m_X(t) + Y(t)$ then $Y(t)$ has zero mean. Trends in the behavior of $m_X(t)$ are reflected in the variation of the $m_X(t)$ with time.

The **Autocorrelation** $R_x(t_1, t_2)$ of a random process $X(t)$ are reflected in the variation of $m_X(t)$ with time. The autocorrelation $R_x(t_1, t_2)$ of a random process $X(t)$ is the joint moment of $X(t_1)$ and $X(t_2)$

$$R_X(t_1, t_2) = E\,[X(t_1)X(t_2)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} xy f_{x(t_1), x(t_2)} dx\, dy.$$

Note that $f_{x(t_1), x(t_2)}$ is the second order pdf or $X(t)$ and $R_X(t_1, t_2)$ is a function of t_1 and t_2.

The **Autocovariance** $C_x(t_1, t_2)$ of a random proces $X(t)$s is defined as the covariance of $X(t_1)$ and $X(t_2)$:

$$C_X(t_1, t_2) = E[\{X(t_1) - m_X(t_1)\}\{X(t_2) - m_X(t_2)\}]$$

$$= R_X(t_1, t_2) - m_X(t_1)m_X(t_2)$$

In particular, when $t_1 = t_2 = t$ we have

$$VAR[X(t)] = E[(X(t) - m_X(t))^2] = C_X(t, t).$$

Correlation coefficient of $X(t)$ is defined as

$$\rho x(t_1, t_2) = \frac{C_X(t_1, t_2)}{\sqrt{C_X(t_1, t_1)}\sqrt{C_X(t_2, t_2)}}; \quad |\rho x(t_1, t_2)| \leq 1$$

The mean, autocorrelation and autocovariance functions provide only partial description of a random process.

2.5 Continuous Domains

In recent years, advances in hardware technology have facilitated new ways of continuously collecting data. In many applications such as network monitoring, the volume of such data is so large that it may be impossible to store the data on disk. Furthermore, even when the data can be stored, the volume of the incoming data may be so large that it may be impossible to process any particular record more than once. Therefore, many data mining and database operations such as classification, clustering, frequent pattern mining and indexing become significantly more challenging in this context (Aggarwal, 2007). The monitoring of many events in real time produces much information. In recent years data stream field has grown rapidly. This field provides of techniques to deal with large information, but the lead of big amount of data there are some challenges to consider such as the following:

- With *increasing volume of the data*, it is no longer possible to process the data efficiently by using multiple passes. Rather, one can process a data item at most once. This leads to constraints on the implementation of the underlying algorithms.

Therefore, stream mining algorithms typically need to be designed so that the algorithms work with one pass of the data.

- In most cases, there is an inherent *temporal component* to the stream mining process. This is because the data may evolve over time. This behaviour of data streams is referred to as temporal locality. Therefore, a straight-forward adaptation of one-pass mining algorithms may not be carefully designed with a clear focus on the evolution of the underlying data.

Continuous problem domains require different underlying representations and place additional constraints on the problem solving process (Ram and Santamaría, 1997). Ram and Santamaria define three characteristics where the problem domain is continuous, and those are: First, they require *continuous representations*, For example, a robotic navigation task requires representations of continuous perceptual and motor control information. Second, they require *continuous performance*. For example, driving a car requires continuous action. Often, problem-solving performance is incremental of necessity because of limited knowledge available to the reasoning system and (or) because of the unpredictability of the environment; the system can at best execute the "best" short term actions available to it and then re-evaluate its progress. A robot, for example, may not know where obstacles lie until it actually encounters them. Third, these problem domains require *continuous adaptation* and learning. As the problems encountered become more varied and difficult, it becomes necessary to use fine-grained, detailed knowledge in an incremental manner to act, and to rely on continuous feedback from the environment to adapt actions and learn from experiences.

Reasoning about continuous domains is not an easy task. Moreover, this is a domain where CBR can rapidly extend its benefits because data is systematically collected for its analysis. A CBR system that continuously interacts with an environment must be able to create autonomously new situation cases (new concepts or clusters) based on its perception of the local environment in order to select the appropriate steps to achieve the current mission goal (Harris and Slobodan, 2005), but a general framework is still missing. Some systems that use case-based methods in continuous environment are described in (Urdiales *et al*., 2006, Kruusmaa, 2003, Ram *et al*., 1997).

There are two other central problems derived from the continuous nature of some domains. First of all, the *size of the case library* could grow very fast as the CBR system is learning new cases without an extensive improvement in the competence of the system, as pointed out in (Miyashita and Sycara, 1995). Two natural human cognitive tasks appear as the solution to these problems: forgetting (Keane and Smyth, 1995) and sustained relevant learning (Sànchez-Marrè *et al.*, 1999). On the other hand, learning many cases could provoke an *overhead in the case library organization*. As new cases are stored in the case library, it will be necessary to update the case library organization (Meléndez, 2001).

3 The proposed Dynamic Adaptive Framework

When CBR systems are deployed in continuous domains (i.e., *domains where cases are generated from a continuous data stream*), the case maintenance becomes critical. An uncontrolled growth of the case library can cause some serious problems of performance, where the retrieval efficiency and the quality of the retrieval is affected. When the case base has large amount of data, some inconsistent cases could be stored. This condition affects directly in the performance of the library. Sànchez-Marrè *et al*. in (Sànchez-Marrè et al., 1999) proposed to learn only the relevant cases to get a stable library. The learning of relevant cases could be an interesting task for learning in continuous domain. To get a stable library in a continuous domain Orduña and Sànchez-Marrè in (Orduña and Sànchez-Marrè, 2009) proposes a dynamic library able to adapt itself to the continuous changes in the domain. This research work proposes a framework called "DACL/MCL Framework for improving Case-Based Reasoning performance", see (Orduña and Sànchez-Marrè, 2015b; Orduña and Sànchez-Marrè, 2009), as a reasonable solution to learn experiences and control the growing of the library.

The proposed framework aims to improve the *retrieve* and *retain* phases of the CBR cycle. Our proposal is summarized in figures 9, 10 and 11. In figure 9, the CBR cycle is depicted. Considering this figure versus figure 1 in chapter 2, the only difference is in the representation of the case library. While the figure 1 depicts "case library" the figure 9 depicts "DACL/MCL". That means that our proposal works different in the knowledge organization, and have special methods for case retrieval and case learning. Figure 10 makes emphasis in the Stochastic Learning Meta-case Method "*SLMcM*" proposed in (Orduña and Sànchez-Marrè, 2015a). The proposal of *SLMcM*, improves the CBR cycle with methods to build prototypes of the library. The same figure depicts the use of *NIAR k-d tree* algorithm, introduced in (Orduña and Sànchez-Marrè, 2013). *NIAR k-d tree* is an algorithm proposed for indexing the cases in the case base. With this technique, the learned cases are saved in a hierarchical structure that improves the case retrieval quality. The algorithm has been tested in the supervised and unsupervised domains. The results are detailed in chapter 6. The algorithm has been tested both in an *exact-case retrieval process* (typical from database field, where k-d trees were originated) and in a *similar-case retrieval process* (typical from case-

based reasoning field). The results show a good performance in retrieval time and a good competence in the retrieval step. Finally, the figure 11 depicts a complete CBR cycle including the methods proposed. In addition, a new exploration technique, the *Partial Matching Exploration technique (PME)*, has been proposed for facing the similar-case search, and coping with the problem of losing cases in hierarchical indexing structures. This exploration technique has been tested, and the results have shown a very good performance.

Both proposals (NIAR k-d tree and PME) jointly with the Stochastic Learning Meta-case Method (SLMcM) have been integrated within the DACL/MCL framework.

In this chapter, the details of *DACL/MCL* are given and the details of the different levels of *DACL/MCL* are explained. The representation of what is a case is detailed, referred as usual in the literature in CBR field. With the definition of a case then the definition of the prototypes of cases is proposed. Finally, the explanation of what constitute the DACL/*MCL* is provided.

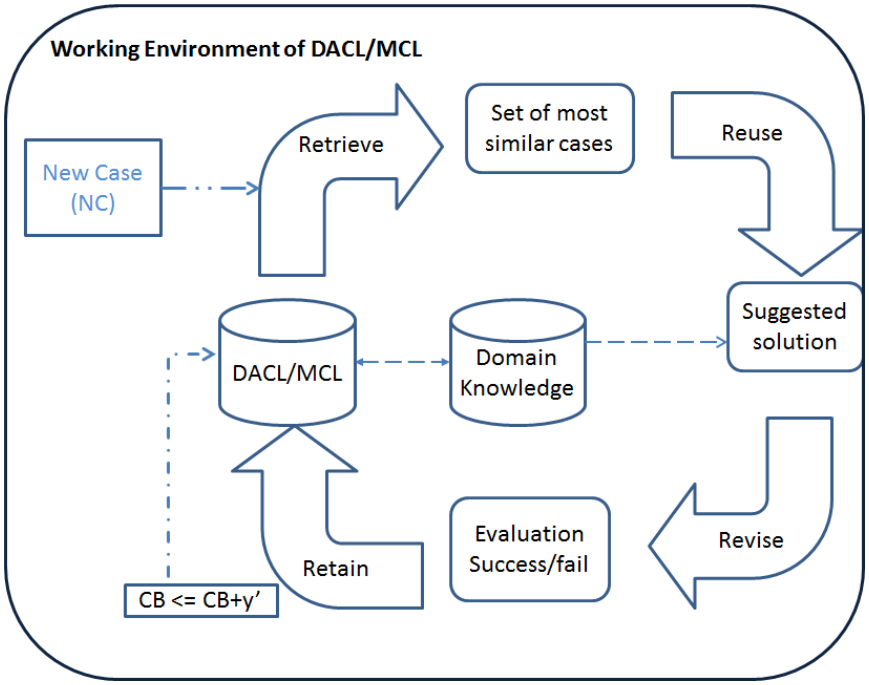

Fig. 9. Environment of DACL/MCL

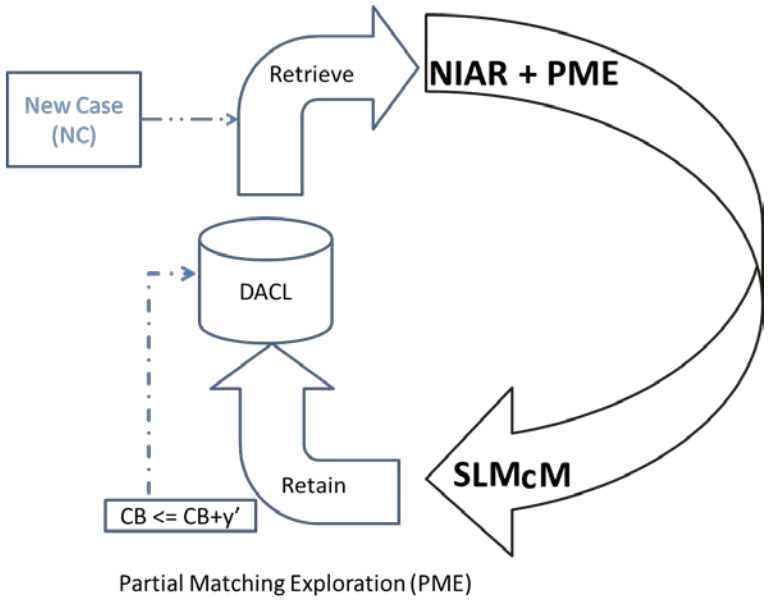

Fig. 10. DACL/MCL Methods

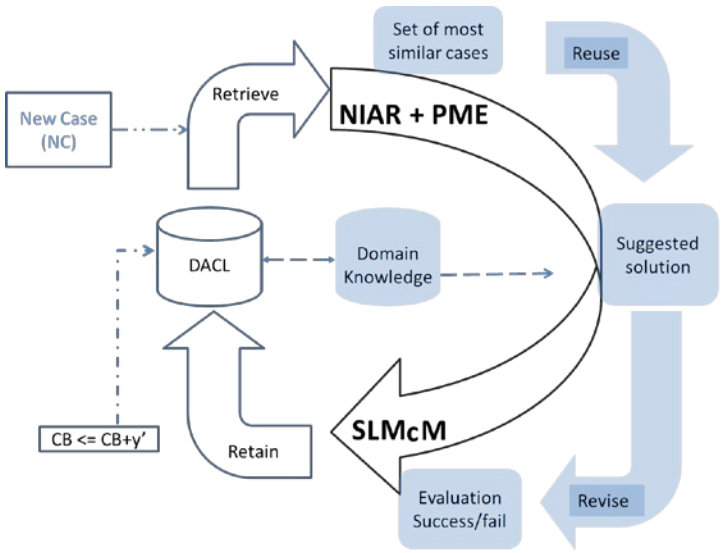

Fig. 11. DACL-CBR

3.1 Dynamic Adaptive Case Library

The continuous domains require software that works efficiently. In a continuous domain is difficult to store all the information generated. In these domains, the amount of data generated could be different in similar situations. Therefore, there are changes produced by the dynamic of the system within the environment. The continuous domain has a dynamic relationship with the dynamic environments. For this reason, it is required to provide them with a dynamic competence model, with the goal of being adapted as much as possible in relation with the changing environment.

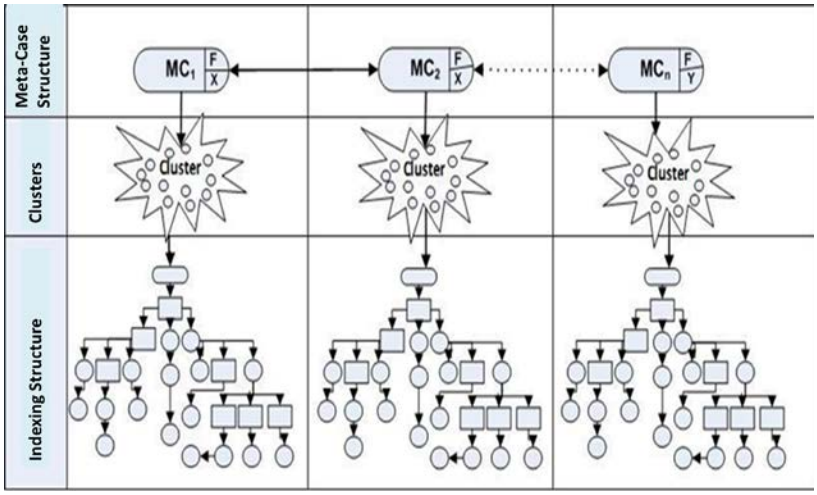

Fig. 12. The Three-level Dynamic Adaptive Case Library structure.

The architecture proposed presents a Dynamic Adaptive Case Library (see figure 12), with the aim of giving a possible solution to the management of the large amount of data generated in a continuous domain. The library will be split in several sub-libraries. Each sub-library is organized hierarchically at three levels:

- The *Meta-case*: The Meta-case is the prototype of a concrete cluster of cases.
- The *clusters*: The set of cases belonging to the same cluster, and that are being represented by the meta-case.
- *Indexing structures*: They implement the way that all the cases are organized in the sub-library. In our proposal, the

cases are organized in a hierarchical indexing structures (*k*-d trees, discriminant trees, etc.), but could be organized with other indexing approaches.

3.1.1 The Case

Most of applications involving CBR approaches in continuous systems define a case as a process state in a time instant (Ram and Santamaria, 1997), or like an average of values in a predefined period of time (Poch *et al*. 1999), that is not always possible and major information is obtained from sequences of data and state transition (Meléndez *et al.*, 2001). In (Melendez *et a*l., 2001) it is proposed to build a case according to the previous definition, with the idea of adding historic signals as temporal series of acquired signals.

Continuous domains present some added difficulties to the building process of a CBR system (Joh, 1997). The first difficulties are presented in the definition of case. The most popular questions raised in the literature about the characteristics of a case, are: which elements of the domain constitute a case? How should continuous cases be represented?, When do cases start and end (in the temporal sense)?, When are two experiences different enough to warrant consideration as independent cases?, what is the scope of a single case?.

We can argue that a case is a snapshot of a situation. This concept could guide the extraction of a case from discrete domains, where the boundaries of a situation (case) are clear. In continuous domains, however, this is not obvious and creates some problems such as cited above. Most of the continuous domains are systems where planning and execution or monitoring and control are interleaved (Portinale and Montani, 2005), (Sànchez-Marrè *et al*., 2005), (Martín and Plaza, 2004), (Veloso, *et al*., 1996). They commonly have a dynamic associated with the process where time has to be considered (Portinale and Montani, 2005), (Sànchez-Marrè *et al.*, 2005). The main approach to identify a case is to establish some regular sampling points to obtain the discrete values for the relevant attributes better characterizing the process.

Considering the literature cited above, in this work, a case has the structure depicted in following case structure, using the concepts of cover-

age and reachability formulated by Keane and Smyth (Keane and Smyth, 1995):

Case#_X: The identifier assigned to the new case.
List of Attributes (att$_1$ ... att$_m$): The list of the attributes witch characterize both the problem description and the corresponding solution.
Problem: A situation detected by the system it is described by a list of attribute-value pairs.
Solution: It is the solution for the problem. The solution is a list of attribute-value pairs showing the solution to the problem.
Distance to Meta-case: It is the distance of the case to its Meta-case model.

Taking the Case as an object, it could incorporates some others values that help to have a better representation of the current new case (Nc).

3.1.2 The Meta-case

The idea of using a Meta-case as a representative case of several similar cases was introduced by Sànchez-Marrè in (Sànchez-Marrè *et a*l., 2000). The aim is to show a formal proposal of how to a Meta-case (Mc) can be built. For our goal, a Meta-case is the prototype of a set of related cases. The centroid value is generated taking into account the whole cases stored in the indexing structure. With the following formula: $Mc_j^i = \frac{1}{n_i}\sum_{k=1}^{n_i} C_j^k \; where$ j = 1, … , m. The average distance (centroid) of the set of cases in that cluster is computed. A case (C^i) and a Meta-case (Mc) are described by m attribute values. That is the first proposal that was introduced in (Orduña and Sànchez-Marrè, 2009). Our proposal of constructing Meta-cases (Orduña and Sànchez-Marrè et. al., 2015a) is where the stochastic methods is introduced and prove it. DACL have as representative case a Mc, this Mc works like a clustering filter where the decision to learn a new incoming case (Mc) is made, this decision concerns to a method to evaluate the $Mc's$ and find the most appropriate Mc where to learn the Nc.

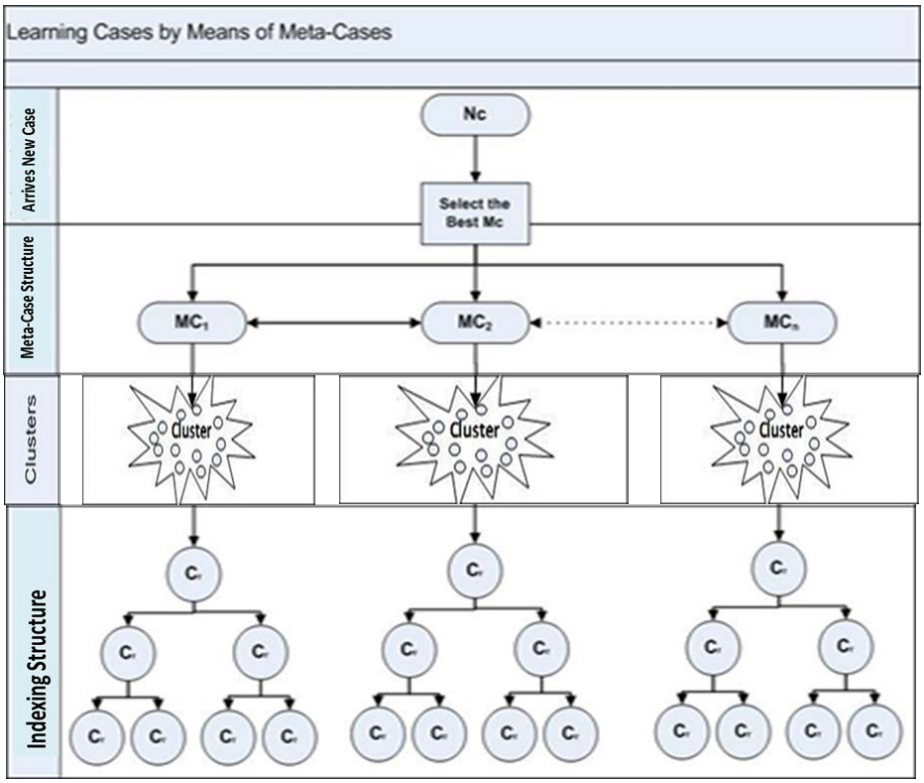

Fig. 13. Learning new cases through the DACL framework

The figure 13 depicts the general way of learning a new case, first the case arrives, and next the method finds the most representative Mc. Once the Mc has been identified, the DACL proceed to store the Nc in the corresponding indexing structure (*k*-d tree, discrimination tree, etc.). Here follows the formalization of this process:

$$C^i = \left(C_1^i, C_2^i, \cdots, C_m^i\right)$$

$$Mc^i = \left(Mc_1^i, Mc_2^i, \cdots, Mc_m^i\right)$$

Where $n_i \; = \; \#Cases\; represented\; by\; the\; Meta - case(Mc^i)$

$$Mc_j^i = \frac{1}{n_i}\sum_{k=1}^{n_i} C_j^k \qquad \text{j} = 1, \ldots, m \tag{1a}$$

If j is a qualitative attribute, and

$$Mc_j^i = mode\left(C_j^k\right) \quad k = 1, \dots, n_i \quad j = 1, \dots, m \tag{1b}$$

If j is a quantitative attribute

The Meta-case structure improves the performance of the retrieval time according to the proposals of Orduña and Sànchez-Marrè in (Orduña and Sànchez-Marrè et. al., 2015a; Orduña and Sànchez-Marrè, 2009). The Meta-case is related to the clustering and learning processes. A Meta-case structure is considered as follows:

Meta-case-id: It is the identification of the Meta-case.
Meta-case centroid: It is computed, for each component, as the average value or mode value of corresponding values of all cases existing in the corresponding cluster. This average or mode is generated by using the formulas 1a or 1b.
IndStr link: The link to the root of the Indexing structure.
McBrother: The link to the nearest Meta-case brother.

Considering the Mc as an object, it could incorporate some other values to help having a better representative prototype.

A CBR system executes the following 4 phases to learn a new experience; retrieve, reuse, revise and retain. In the DACL approach, the retrieve and retain phases need to be addressed and reformulated. Retrieving similar cases regarding to a new case is a process that needs to be done accurately. A new algorithm (DACL Retrieval algorithm) to retrieve the most similar case or cases in the DACL is shown.

Algorithm 3: DACL retrieval algorithm

```
Input: The Case (C)
Output: the retrieved set of cases (Retrieved)
Begin Discriminant_tree(Node Nc)
Let C = the arriving case;
Let Mc = a Meta-case;
Let dMC = the most similar Meta-case;
Let i = total number of Meta-cases;
Let K = total number of attributes of the C;
Let a = an attribute of C where a= {1… K};
Let CL = the cluster that's represents a Meta-case where
CL={1…i}
Let ListMc = a {list to store temporally the distances
estimated};
Let N = The root of the tree;
Let Node C = the representation of the C;
Let att_a = an specific attribute of C, where a = {1… K};
Let Dadnode = a {node that's represents a discriminant
section}
Estimate the distance between the C and the entire Meta-
case using the formula 2;
dMC = Min(ListMc)
Similar = Method_search(C, N)
Begin Method_search(Node C, Node DadNode)
Path = Compare dMC.att_x with C.att_x
if(Path.leaf == Relevance)
  If Path.leaf ==  DadNode then
     Method_search(C, Path.leaf)
  Else
    If Path.leaf == CaseNode then
      If CaseNode.ListCases  == true then
        while ListCases != null
          Apply the formula 3.
        Endwhile
        Retrieved = Max(ListSimil)
      Else
        Retrieved = Path.leaf
      Endif
    Endif
  Endif
Endif
Return Retrieved
End Method_search
```

$$ListMc_{[cl]} = \sum_{Cl=1}^{Cl_i} \left[\frac{1}{att_k} \left(\sum_{att=1}^{a_k} \left(C_{att} - Mc_{Cl_{att}} \right) \right) \right] \quad (2)$$

$$ListMc_{[i]} = \sum_{i=1}^{CasesList!=null} \left[1 - \left(\frac{\sum_{att=1}^{n} e^{w}att\ d\left(C_{A_{att}} - CasesList_{i_{att}} \right)}{\sum_{att=1}^{n} e^{w}att} \right) \right] \quad (3)$$

Where:

$$d\left(C_{A_{att}} - CasesList_{i_{att}} \right) = \frac{\left| quantval(C_{A_{att}}) - quantval(CasesList_{A_{att}}) \right|}{UpperVal(A_{att}) - LowerVal(A_{att})} \quad (4)$$

Retain task aims to maintain (Basic retaining/learning algorithm) a competent Case Library with a high coverage. The Retain process decides whether the new case needs to be stored in the case library, by updating an existing sub-library, building a new sub-library or simply ignoring the case. The process to make a decision is guided by the learning algorithm.

The algorithm basic retaining/learning algorithm works as follows: it receives a solved new case (Nc) and then computes the distance to all the Meta-cases, with the aim to find the closest Meta-cases. Once the best Meta-case is found, it proceeds to compare whether the distance found falls within the α threshold (previously defined by the experts or tuned by trial and error experimentation). Then, it proceeds to store the new case into the current library. Otherwise, if the distance is higher than the α threshold value, a new sub-library must be created containing the new solved case. The last consideration in the algorithm is when the distance of the solved case falls within the ratio of two or more meta-cases. If so, the competence of the most similar Meta-case (MsMc) and the second most similar Meta-case (2MsMc) is computed taking into account the solved case. Moreover, the solved case is stored into the sub-library that better improves its competence with the case.

Algorithm 4: basic retaining/learning algorithm (VirtualMcSel-1)

```
Input: the new case solved (Nc)
       The DACL
Output: The updated DACL
Begin VirtualMcSel-1
Let MsMc = the most similar Meta-case;
Let 2MsMc = the second most similar;
The distance is computed between the new case and the
whole set of Mc
if d(Nc, MsMc) < α and (d(Nc, 2MsMc) < α then
  /* α is the threshold value, predefined by the experts
     or tuned by trial and error experimentation */
  Evaluate the competence of the MsMc and 2MsMc;
  Select the sub-library that get a better competence
  with the new case;
  The selected Meta-case is updated with the new case;
elseif d(Nc, MsMc) < α then
  Update the corresponding sub-library with the new case
elseif d(Nc, MsMc) > α then
  Build a new sub-library, and store the new case
endif
end VirtualMcSel-1
```

3.2 Multiple Case Library (MCL)

In our research, we propose to use a static version of a DACL: a Multiple Case Library (MCL) with the same structure but being defined statically at the building stage of the Multiple Case Library. MCL approach is intended for supervised domains. Main rationale for a MCL is the same than for a DACL: the main problem in hierarchical Case Libraries retrieval task is that sometimes is impossible to reach most similar cases due to an exploration in a wrong area of the hierarchy. This problem could be because the hierarchy of nodes does not correspond to the relevance of the attributes, as for example a bad –unique– discrimination ordering of the attributes. Our proposal to overcome this problem is to split the case library in a set of different smaller case libraries. These smaller case libraries can provide a faster search because they have a smaller number of tree levels, and in addition, they can provide more accurate *similar-case search* because each Case Library corresponds to a different prototype (Meta-case) of cases in the whole domain.

The retrieval process in a MCL starts matching the current/query case against a set of prototype cases, called meta-cases, to select one (or more) case libraries to search in. This approach tries to build several hierarchical structures for suiting different kind of cases, making the CBR system more flexible, accurate and reliable.

In a MCL, the number of meta-cases and its corresponding Case Libraries has been fixed a priori, rather than dynamically like in a DACL structure. One common situation is when the domain or database is *supervised*, and there is a class label for each case. Then, all the cases sharing the same class label form the corresponding cluster, and its prototype is the Meta-case. In other *unsupervised* situations, a previous clustering process or a general dynamic incremental process can give, as a result, the set of Meta-cases to be used for splitting all the cases in the corresponding set of case libraries. For each Meta-case (cluster of cases), a hierarchical structure (*k*-d tree) is constructed to discriminate among all the cases belonging to the same cluster.

Each sub-library is organized hierarchically with the same layers than a DACL:

- The Meta-case: The Meta-case is the prototype of a concrete cluster of cases.
- The *clusters*: The set of cases belonging to the same cluster, and that are being represented by the Meta-case
- The indexing structures (discriminant trees, *k*-d trees, etc.): Represents the way that all the cases in a given cluster are organized in the corresponding sub-library.

In the top level of a MCL could exist several Meta-cases, where each one describes a subtype of cases in the general domain (class type) stored in MCL. In next level, the hierarchical structures/trees are employed as an indexing strategy which aim is to improve both the time retrieval and the accuracy of retrieved case/s. MCL is a very flexible structure because in the second level, any hierarchical strategy could be used. Furthermore, a MCL has the flexibility to mix different strategies. For instance: the second level of any Meta-case, i.e., the hierarchical/tree level, could have a special method implemented within: a standard *k*-d tree, a NIAR *k*-d tree, a binary tree, etc., where this method improves the retrieval especially in this kind of cases (subdomain).

This characteristic endows MCLs to deal with an introspective reasoning method and allows the selection of the best indexing technique for each meta-case. The implementation of possible *different methods* at indexing levels avoids the implementation of the same retrieval algorithm for all data in the whole MCL, always aiming to improve the performance and quality of the system.

This third level of a MCL is where several approaches will be tested in chapter 6: *standard k-d trees, NIAR k-d trees*, the same approaches with a *Partial Matching Exploration strategy (PME)* and the same approaches with the *hyperball with bounds* strategy.

Therefore, an experimental testing will be undertaken to show whether the use and combination of the new approaches proposed against previous well-known used techniques improves the efficiency and competence of CBR systems.

4 Improving the retrieval task

Case retrieval is one important step in the case-based reasoning cycle, especially related to the time efficiency of a CBR system. Several algorithms have been proposed for the hierarchical indexing of cases, since the original indexing approach of *k*-d trees appeared in the literature. Main approaches propose to use a pre-computed binary search tree to get an average logarithmic time effort in searching. The basic ideas of the proposal are indexing algorithms based on the principle of binary search trees for efficient case retrieval according to a given similarity measure *sim*. In next sections, the AvKd-Tree, the NIAR *k*-d tree and the Partial Matching Exploration (PME) technique will be proposed and explained.

4.1 Av*K*d-Tree

An algorithm called *AvKd-Tree* is introduced. *AvKd-Tree* is a proposal with better performance in retrieving time than a standard *k-d tree*, but no better than a *NIAR k-d tree*. *AvKd-Tree* spends least retrieval time than standard *k-d tree* in databases with large amount of data. However, in databases with a considerable amount of data the behavior is similar to a standard *k-d tree*. In the next section, the algorithm will be explained.

The rationale behind *AvKd-Tree* approach is to try to get the sub-trees of each internal node as much balanced as possible, in order to reduce the number of tree levels, and consequently increase the retrieval speed. The approach is to deal the incremental problem in the building of *k-d trees*, especially when facing continuous domains, where a lot of new cases are generated and must be stored progressively in the case base. The indexing strategies should be analyzed and modified to show reasonable time in the updating of the case base indexation.

The use of *k-d trees*, *NIAR*, *AvKd-Tree* embedded in DACL gives facilities to build an *indexing structure* framework with the aim to select the best algorithm for indexing cases that belongs to its representative Meta-case. The selection of the best method is according of the framework proposed. Its behavior is learned following a set of evaluation rules

with the aim to achieve the major efficiency and to gain the best possible performance.

This approach proposal works quite similar to the standard *k*-d tree and *NIAR k-d tree*, selecting the discrimination attributes cycling along the list of attributes, but differs of standard *k-d tree* approach in the technique of selecting the split value for each attribute at the internal nodes. The *proposed partition value is the average value of the attribute values from the instances* in the corresponding node. The technique proposed is explained as follows:

1. Compute the average value Att_i^l of the corresponding attribute i among the instances in the current node l.

   Let be $\{C^j\}_{j=1,n}$ the case base composed of n cases. Each case C^j could be described with the set of m attributes:

$$C^i = \left(C_1^i, C_2^i, \cdots, C_m^i\right)$$

   Let be I_l the set of instances represented by the Node(l):

$$I_l = \left\{C^j \middle| C^j \in Node(l)\right\} \quad l = 1, \ldots, \mathrm{n}$$

   The computation of the average value of attribute i among the instances at node l, Att_i^l is defined by the formula:

$$Att_i^l = \frac{1}{\#I_l}\left(\sum_{j=1}^{\#I_l} C_i^j\right)$$

   Then, finally, the partition value is: Att_i^l

2. Once found the partition value, all the node l instances with lower or equal values than the partition value will be the instances of the left sub-tree, constituting the new node $(2 * l)$. On the other hand, all the node l instances with higher values than the partition value will be the instances of the right sub-tree, constituting the new node $(2 * l + 1)$.

3. Recursively continue generating the tree for the left son node, and for the right son node, until the number of instances of a node is lower or equal than the bucket size (m cases)

The procedure for generating the AvKd-tree is detailed in the Avkd-tree algorithm, following detailed.

Algorithm 5: AvKd-Tree

```
1   Input: case base CB, discriminator attribute i,
2          total number of attributes m, bucket size d
3   Output: AvKd-tree root node
4   begin AvKd-tree(CB, i, m, d)
5     let n, sum, ni, atti, partitionValue,
6       lowerPart, upperPart
7     n = count(CB); {number of cases of CB}
8     if n > d then   //cases do not fit in one bucket
9       sum = 0;
10      for j= 1 to n do
11        sum = sum + consult(CB,j,i) //value i of case j
12      endfor;
13      atti= sum / n;   //Average value of attribute i
14      maxsim = - infinity;
15      partitionValue = consult(CB, atti ,i);
16      lowerSon = ∅; upperSon = ∅;
17      for j= 1 to n do
18          if consult(CB,j,i) <= partitionValue then
19            lowerPart = lowerPart + case(CB,j)
20          else
21            upperPart = upperPart + case(CB,j)
22          endif
23      endfor
24      ni = i mod m + 1;  //next indexing attribute
25      return (makeNode(i,partitionValue,
26                   AvKd-tree (lowerPart,ni,m,d),
27                   AvKd-tree (upperPart,ni,m,d)))
28    else
29      return (makeLeaf(CB,n.d))//make a bucket of n cases
30    endif
31  end AvKd-tree
```

The procedure `count(CB)` computes the number of cases in `CB`. The procedure `consult(CB,j,i)` provides the value of the attribute *i* in the case *j* of the `CB`. The procedure `case(CB,j)` returns the case *j* of `CB`. The procedure `makeNode(i,partitionValue, lowerSon,upperSon))` returns an internal node including the discrimination attribute *i*, the splitting value `partitionValue`, and two

pointers to the nodes of the subtrees `lowerSon` and `upperSon`. The procedure `makeLeaf(CB,n.m)` returns a leaf node with a bucket filled with the cases in `CB`.

The algorithm shown above is a recursively procedure that builds the AvKd-Tree tree proposed.

The average computational time for generating a AvKd-Tree is $O(n * log_2 n)$, being n the number of cases, and $O(n^2)$ for the worst case, improving the effort for generating the *k*-d tree of some of the standard *k*-d trees; its cost does not depend on the number of attributes k.

4.2 NIAR *k*-d Tree

The proposed NIAR *k*-d tree algorithm has two main steps based on the computation of the average value of the corresponding attribute among the sub-tree cases, and selecting for that attribute, the value of the Nearest Instance/case to the Average as the Root (partition value). Several experimental results with some databases have shown that the retrieval step in NIAR *k*-d tree is faster than in the standard *k*-d tree approach. The time efficiency, the depth and breadth in both trees are analyzed. The results obtained depict a significant difference of levels in the trees. The presented approach is implemented within the DACL framework for case-based reasoning.

As explained in the previous sections, the different approaches based on the standard *k*-d tree approach as an associative retrieval procedure differ both in the selection criteria for the discriminating attributes and in the selection of the partition value. All proposed strategies try to improve the retrieval time for query cases through the generation of well-balanced binary trees. A balanced tree is a very compact tree where the number of cases represented by each sub-tree is nearly the same than in the other sub-tree. This guarantees a stable retrieval time independently of the distribution of the query cases.

Our proposal works quite similar to the standard *k*-d tree, selecting the discrimination attributes cycling along the list of attributes, but differs of standard *k*-d tree approach in the technique of selecting the split val-

ue for each attribute at the internal nodes. The proposed partition value is the attribute value more similar to the average value of the attribute values from the instances in the corresponding node. The technique proposed is explained below.

The name *NIAR* means that the attribute value of the Nearest Instance to the Average will be the Root node partition value. This value is the new partition value. To find this split value, the process proposed is as follows:

1. Compute the average value Att_i^l of the corresponding attribute i among the instances in the current node l.

Let be $\{C^j\}_{j=1,n}$ the case base composed of n cases. Each case C^j could be described with the set of m attributes:

$$C^i = \left(C_1^i, C_2^i, \cdots, C_m^i\right)$$

Let be I_l the set of instances represented by the Node(l):

$$I_l = \left\{C^j \middle| C^j \in Node(l)\right\} \quad l = 1, \ldots, \mathrm{n} \tag{5}$$

The computation of the average value of attribute i among the instances at node l, Att_i^l is defined by the formula:

$$Att_i^l = \frac{1}{\#I_l}\left(\sum_{j=1}^{\#I_l} C_i^j\right) \tag{6}$$

2. Find the Nearest Instance attribute value to the Average value ($C_i^{NIARoot}$) of the attribute values from the instances (I_l) in the corresponding node l, which will be the Root partition value.

Once computed the mean of the corresponding attribute, the next step is to find the nearest instance to the mean value obtained. The average value computed very probably does not really exist in the list of instances. It is not a true split value. This average value cannot be taken as the final split value because is a virtual value so far, and thus, the partition value would be virtual and would increase the number of levels of the tree. To reduce the number of levels in the tree, the Nearest Instance to the Average value of the Attribute must be located within all the instances (I_l) of the corresponding node l:

$$NIARoot = Arg\ Max_{j=1,\#I_l} sim^i\left(Att_i^l, C_i^j\right) \quad (7)$$

sim^i is the similarity measure on the dimension of the attribute *i*.

Then, finally the partition value is: $C_i^{NIARoot}$

3. Once found the partition value, all the node *l* instances with lower or equal values than the partition value will be the instances of the left sub-tree, constituting the new node $(2 * l)$, and, all the node *l* instances with higher values than the partition value will be the instances of the right sub-tree, constituting the new node $(2 * l + 1)$,

4. Recursively, continue generating the tree for the left son node, and for the right son node, until the number of instances of a node is lower or equal than the bucket size (m cases)

The procedure for generating the NIAR *k*-d tree is detailed in the algorithm NIAR k-d tree following detailed:

Algorithm 6: NIAR *k*-d tree

```
Input: case base CB, discriminator attribute i,
       total number of attributes m, bucket size d
Output: NIAR k-d tree root node

begin GenNIARtree(CB, i, m, d)
Let n, sum, ni, atti, NIARoot, partitionValue,
    lowerPart, upperPart
  n = count(CB); //number of cases of CB
  if n > d then //cases do not fit in one bucket
    sum = 0;
    for j= 1 to n do
      sum = sum + consult(CB,j,i) {value i of case j}
    endfor;
    atti= sum / n; //Average value of attribute i
    maxsim = - infinity;
    for j= 1 to n do
      if simi(Atti, consult(CB,j,i)) > maxsim then
         maxsim = simi(Atti, consult(CB,j,i));
         NIARoot = j;
      endif
    endfor
    partitionValue = consult(CB,NIARoot,i);
    lowerSon = ∅; upperSon = ∅;
```

```
24      for j = 1 to n do
25          if consult(CB,j,i) <= partitionValue then
26            lowerPart := lowerPart + case(CB,j)
27          else
28            upperPart := upperPart + case(CB,j)
29          endif
30      endfor
31      ni = i mod m + 1; {next indexing attribute}
32      return (makeNode(i,partitionValue,
33                   GenNIARtree(lowerPart,ni,m,d),
34                   GenNIARtree(upperPart,ni,m,d)))
35    else
36      return (makeLeaf(CB,n.d))//make a bucket of n cases
37    endif
38  end GenNIARtree
```

The procedure `count(CB)` computes the number of cases in `CB`. The procedure `consult(CB,j,i)` provides the value of the attribute *`i`* in the case *`j`* of the `CB`. The procedure `simi(Atti, consult(CB,j,i))` computes the similarity measure between *`Atti`* (the average of the attribute *`i`*) and the value of the attribute *`i`* in the case *`j`*. The procedure `case(CB,j)` returns the case *`j`* of `CB`. The procedure `makeNode(i,partitionValue,lowerSon, upperSon))` returns an internal node including the discrimination attribute *`i`*, the splitting value `partitionValue`, and two pointers to the nodes of the sub-trees `lowerSon` and `upperSon`. The procedure `makeLeaf(CB,n.m)` returns a leaf node with a bucket filled with the cases in `CB`.

The algorithm shown above is a recursive procedure that builds the NIAR *k*-d tree proposed. Figure 14 depicts the splitting step of the tree based on the partition value obtained.

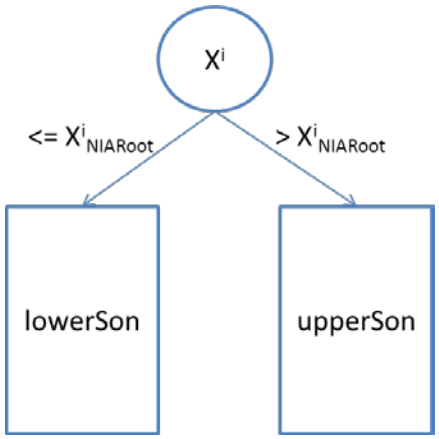

Fig. 14. Splitting step in the generation of a NIAR *k*-d tree

The average computational time for generating a NIAR *k*-d tree is $O(n * log_2 n)$, being n the number of cases, and $O(n^2)$ for the worst case, improving the effort for generating the *k*-d tree of some of the standard *k*-d trees.

4.3 Partial Matching Exploration (PME) Technique

In this section, a partial matching exploration technique (PME) for a tree indexing structure is proposed (figure 15). It is a technique to explore a hierarchical case library, with a tree indexing structure aiming to not to lose the most similar cases to a query case. The basic idea underlying the process is to prevent from the fact that some potentially good (similar) cases could not be reached in the retrieval process. Such an unsuccessful search in the case library will lead to a bad retrieving strategy. Among the causes originating these failures there is a wrong choice at a high node in case library as an effect of the discretization process of the node attribute values or, if the hierarchy of nodes does not correspond to the importance of the attributes, as for example a bad discrimination order of the attributes, *etc*.

The partial matching exploration technique means that not only the best matching path will be traversed, but also several alternative partial matching paths will be explored. The exploration task searches the case library with two exploration techniques: best matching exploration and partial matching exploration. The best matching exploration means that

only the best[1] child (best matching node) will be explored. It is the common search of the case library following the main path through the hierarchical case library. The *partial matching exploration* means that the two best children (best and second-best matching nodes) of the current node will be explored.

At root node of the tree, the *partial matching exploration* is used. That means the two best children of the root node will be explored. The best child is explored again with *partial matching exploration*, and the second-best child is explored with *best matching exploration*, if possible. Summarizing, the nodes on the best matching path (main path) are always searched with *partial matching exploration*, and the nodes on the alternative matching paths are searched with *best matching exploration*. Being k the maximum number of attributes used in the tree, then at most k paths from the root are explored, and at most k^2 nodes are explored. So, the searching time $T(k)$, is upper bounded by a function of the number of attributes used in the tree, usually smaller than the number of cases (n) stored in the Case Library, and does not depend on it, which is usually bigger as the system grows: $T(n,k) \in O(k^2)$.

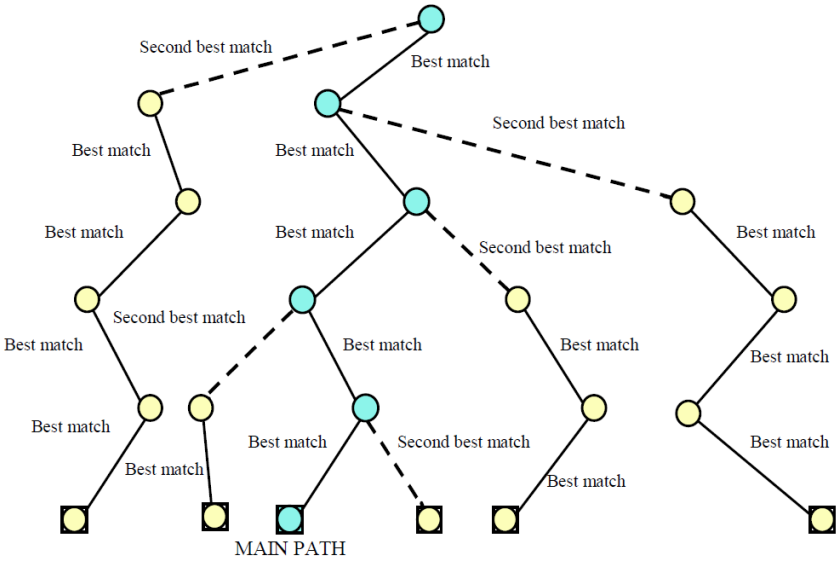

Fig. 15. Partial-matching exploration

[1]The most similar value to the query case value for the attribute of the corresponding node.

Thus, the cases retrieved are all the cases stored in the Case Library differing at most in one attribute's value from the query case (see figure 15). Therefore, this *partial matching technique* allows recognizing *partial matching cases* as *possible similar cases* to the query case.

Below, the Partial Matching Exploration algorithm is described. The algorithm has a main class called PME. This class has two methods named *last* and *second*.

Algorithm 7: Partial Matching Exploration (PME)

```
1   Input: root of the tree
2          Case node
3   Output: the most similar cases collectes through the ex-
4   ploration of the tree
5   Begin PME(case_node, root)
6   if(case_node != root){
7     different = true;
8     break;
9   }
10  if(case_node[root.Attdisc] <= root.Dato[root.Attdisc]){
11    if(!different){
12       root.left = true;
13       root.MainPath = true;
14       nearest[dist_count] = root.ID;
15       Distance[dist_count] = this.Euclidean(case_node,
16  root.Dato);
17       dist_count++;
18     }
19    else{
20       if (root.left != null){
21        root.left = true;
22        root.MainPath = true;
23        nearest[dist_count] = root.ID;
24        Distance[dist_count] = this.Euclidean(case_node,
25  root.Dato);
26        dist_count++;
27        PME(case_node, root.left);
28        }
29       else{
30        root.MainPath = true;
31        main = root;
32        nearest[dist_count] = root.ID;
```

```
      Distance[dist_count] = this.Euclidean(case_node,
root.Dato);
      dist_count++;
      second(nearest, dist_count, root, case_node);
     }
  }
}
else{
  if (root.right != null){
   root.Mainright = true;
   root.MainPath = true;
   nearest[dist_count] = root.ID;
   Distance[dist_count] = this.Euclidean(case_node,
root.Dato);
   dist_count++;
   PME(case_node, root.right);
  }
  else{
   root.MainPath = true;
   main = root;
   nearest[dist_count] = root.ID;
   Distance[dist_count] = this.Euclidean(case_node,
root.Dato);
   dist_count++;
   second(nearest, dist_count, root, case_node);
  }
}
endPM

Method: second
Begin second(nearest,ser,root,case_node){
if (root.dad != null){
root = root.dad;
  if (root.Mainright == true) {
     if (root.left != null) {
       nearest[ser] = last(root.left, case_node);
       Distances[ser] = this.Euclidean(case_node,
root.Dato); ser++;
       second(nearest, ser, root,case_node);
     }
     else
       second(nearest, ser, root, case_node);
  }
  else{
    if (root.right != null) {
```

```
78        nearest[ser] = last(root.right, case_node);
79        Distances[ser] = this.Euclidean(case_node,
80   root.Dato); ser++;
81        second(nearest, ser, root, case_node);
82       }
83       else
84        second(nearest, ser, root, case_node);
85     }}}
86   Method: last
87   Begin last(root, case_node) {
88   if (case_node[root.Attdisc] <= root.Dato[root.Attdisc]) {
89     if (root.left != null) {
90       last(root.left, case_node);
91     }
92     else
93       Aux = root;
94   }
95   else{
96     if (root.right != null) {
97      last(root.right, case_node);
98     }
99     else
100      if(root.left != null)
101        last(root.left, case_node);
102      else
103        Aux = root;
104  }
105  return Aux.ID;
106  }
```

The PME algorithm requires the Nc (case_node) and the root node of the tree to start its process. The first task (lines 2-5) checks if the Nc matches with the root node. If the node does not match, then checks the relevance of the attributes between root and case_node. If the value of attribute in case_node is lower than root (line6),then checks if it has a left son (line 16) and if it has one, then the nearest node going to be root (line 19). Then the Euclidean distance is computed (line 20) and stored in the node. Then PME algorithm is called sending the case_node and the left son (line 23). Else root its labeled as main path and the euclidean distances between case_node and root is estimated (lines 26-31), then the second best match is going to be search calling the method second (line 32).

If the value of attribute in case_node is higher than root (line6) then the right side is checked searching for a right son. And the same dynamic is computed (lines 36-54).

The method called second requires beginning the nearest case, the case_node and the root node. The first question is whether the root has any son, on right side (line 62). If it has a son, then the last method is called with the aim of finding the nearest case (line 64), the Euclidean distance is computed, after this task second method is called with new values (line 67). Then, once computed the procedure in the right side then it is the left sides turn, following a similar criteria (lines 72-81).

The task of the method last has the main task of finding the last node in the path. To start it job it requires the root node and the case_node. The criterion to find the last node is to compare the values of the attributes in the case_node and the root node (line 84). Then the recursive search in left and right sides begins (lines 86, 93, 97). The computing is successful when the last node is returned.

5 Improving the maintenance and learning of the Case Library

The improving of the CBR cycle it is one of the challenges for case base maintenance policies, especially in domains with large amount of cases, and even more, if data are changing. In this chapter, some maintenance policies and methods aiming to improve the maintenance and learning of the CBR cycle are introduced. The first method introduced is related to the improvement of the learning of the DACL proposing a stochastic method to build the representative prototypes (Mc) in a dynamic environment. The proposed method considers two moments for the learning of prototypes. The moments will be detailed. For the supervised domains with DACL, the proposal is to use Multiple Case Library (DACL/MCL). The chapter introduces one additional policy for building the representative prototype.

5.1 The Stochastic Learning Strategy

In previous chapter 3, the details of the DACL framework have been introduced and a method to build Meta-cases has been proposed. The method has the characteristic of building a Meta-case following the strategy of computing a centroid as a Meta-case. This second method of computing a Meta-case considers a Stochastic Method. It has two core moments used in the learning algorithm.

One of the open problems in clustering field is to select the number of clusters; this is relevant in a DACL too. When there are several sub-libraries in the DACL and a dispersion of the cases is generated, then the quality is coming down. On the contrary way, when there exists a few number of sub-libraries and these are compact, the quality its better and the retrieval time is faster. With the following learning policy, DACL is able to learn and classify continuous data precisely, according to the expert's evaluation, as we will explain in the evaluation chapter 6.

The proposed method has the ability to learn Meta-cases (Mc) as is depicted in figure 12. In this work, a *Mc* is described as follows:

The Mc's are designed as the top level of the DACL strategy; this has two aims. The first is when a New Case (Nc) is being considered by the DACL. It has to learn it and store it in the best optimal way or decide not to store it; and second, when the best case has to be retrieved, this strategy should improve time and quality of the process avoiding an exploration in a wrong sub-library.

A Mc is a prototype of the entire cases belonging to the sub-library. The *Mc* in DACL helps to find the most optimal sub-library where the cases have to be learnt. The *Mc* is a generalization of the cases stored in the sub-library. The Mc is built following the next *Mc building process*:

1) A case C^i is defined as $C^i = (C_1^i, C_2^i, \cdots, C_m^i)$ where $C_{j=1,\dots m}^i$, are the attribute values describing the case C^i and where i is the current case of the total of n cases belonging to the corresponding Mc.

2) A Mc^l is defined as $Mc^l = < Mc_1^l \dots Mc_m^l >$ where $Mc_{j=1,\dots,m}^l$, are the computed attribute values of the Meta-case as the average prototype values for continuous/numerical attributes or the most frequent values (mode) of discrete categorical values, according to the following formulas:

If att_j is numerical: $Mc_j^l = \frac{1}{n_l}\sum_{k=1}^{n_l} C_j^k$ and $\#cases(Mc^l) = n_l$

If att_j is categorical: $Mc_j^l = mode(C_j^k) \quad k = 1,\dots,n_l$

The proposed strategy modifies the total number of attributes of the case, where a case C^j is defined by its attributes $< C_1^j \dots C_m^j >$. The strategy adds a new attribute C_{ts}^j, and then, a case C^j is described as $< C_{ts}^j, C_1^j \dots C_m^j >$. This new attribute is a *time stamp* ordering identifier (*ts*). The new attribute is initialized at $ts = 1$ and increases one by one. The increase occurs when a new case is stored in the sub-library.

This C_{ts}^j value is considered like an attribute in the algorithm. The attribute is used in both first and second moment of the stochastic method proposed. It is named as τ. τ plays the role of time, an ordered attribute,

like in the statically normal stochastic method. The value of τ helps to increase the differences of cases adding an increase value between them. The value is taken into account when first moment is computed (see formula 9). The method can be summarized as follows: when a new case arrives, and the algorithm is working to find the *Mc* most similar to the case, the τ value gives a direct difference and make that *Mc*'s be filtered more effectively, because a low τ value indicates "most similar" while a higher τ value indicates a higher separation of the cases.

Other relevant use of the τ value is when the prototype is built; this is depicted in step 2 of the *Mc* building process, previously introduced. Here, the value of the first attribute of the *Mc* is 1. When the *Mc* is updated, the second value is 1.5. At the third step the value is 2, and continues increasing following a constant increase of 0.5 for update iteration. These values are generated computing the step 2 in formula of *Mc* building process.

The learning of new cases and the building of its representative prototype it is one of the aims to achieve in a DACL. To get a successful learning of cases and a good quality of learning *Mc's* is introduced a Stochastic Learning $Mc's$ Method "***SLMcM***".

SLMcM considers two relevant moments to guide the learning. The **first moment** is described as:

$$\rho = \min_{j} arg\, D\left(Nc, Mc^{j}\right) \tag{8}$$

Formula 8 is computed to find the most similar Mc in DACL to the new case Nc. D is the distance function that evaluates the dissimilarity value. And ρ will represent the most similar prototype selected. The similarity of a new case will be assessed against all $Mc's$ by means of the same dissimilarity measure used in the normal assessment of similarity between two cases. In this case, the proposed measure is the *Euclidean distance* when all attributes are numerical, but other heterogeneous measures can be used when both numerical and categorical attributes exist.

The method for finding the most similar Mc can be summarized in following Search similar Mc algorithm.

The Search similar Mc algorithm requires the Nc to start its process. If does not exist a Mc in the library then the Nc is going to be the new Mc (line 4). But if there are some Mc in the library, then a cycle for searching the similar enough Mc begins (lines 5-11).

Algorithm 8: Search similar Mc

```
1  Input: Nc (new case)
2  Output: ρ (most similar Mc)
3  begin search similar Mc
4  Dmin = + ∞
5  currentMc <- firstMc
6  while currentMc != null do
7   if(D(Nc,currentMc) < Dmin) then
8     ρ = currentMc;
9     Dmin = D(Nc,currentMc)
10  endif
11  currentMc = nextMc
12 endwhile
13 return ρ
14 end search similar Mc
```

ρ value is the Mc most similar to the *Nc. This Mc* is the prototype which will store the new case Nc in.
For the retrieval process, the same approach is implemented. When the retrieving of similar cases is executed, the first task is to find where to search. This can be done by finding the most similar Mc. Next task is the retrieving of similar cases. This has to be done following the indexing strategy of the sub-library. In our case, we have implemented the NIAR *k*-d tree jointly with a partial matching exploration technique (Orduña and Sànchez-Marrè, 2015b), as previously explained.

The **second moment** is when the following condition comes true:

$$\sigma(Nc) \leq \sigma(\rho) * (1 + \gamma), \text{where} \quad 0 \leq \gamma \leq 1 \tag{9}$$

γ is described as a virtual threshold added to the computed *Mc* threshold by $\sigma(\rho)$. The computation of $\sigma(Nc)$ and $\sigma(\rho)$ it is done considering all the attributes of Nc and Mc respectively. $\sigma(Nc/\rho)$ It is computed by the summation notation of standard deviation formula.

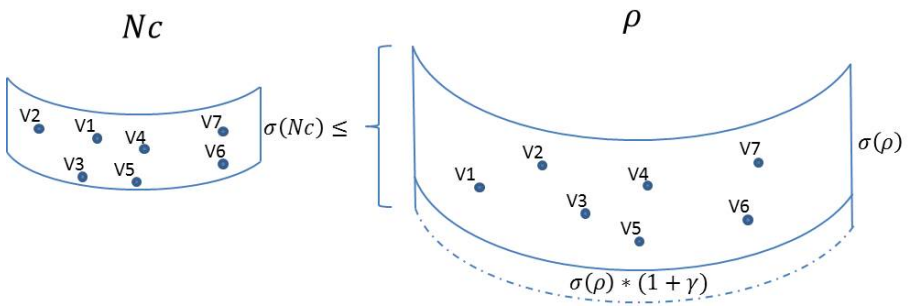

Fig. 16. Mc/ρ virtual threshold representation

Figure 16 represents a virtual size (normal line section) and resize (dot line section) of the Mc threshold; it has built in formula 9, when $\sigma(\rho) * (1 + \gamma)$ it is estimated. This threshold is considered to decide whether a case is going to be stored or not in current Mc. The value $1 + \gamma$ indicates the percentage of the dispersion of ρ that will be considered as a *relaxation of the Mc value*. The value of γ can be less than 1, but, its optimal value has to be found. The implementation of the *relaxation process* considers endowing the Mc with a higher range of learning. The implementations of some evaluation tests have concluded that if the *relaxation process* is not implemented, the learning rate is reduced.

5.2 The Stochastic Learning Policy

In the previous section, it has been introduced the details of the two core-DACL moments used in the learning policy algorithm. The aim of this policy is to learn those cases that accomplish the two moments of the policy, previously detailed in last section. The data are stored in the sub-library that is most similar to the incoming case, according to the first moment in formula 8. If the case does not accomplish with the second moment (formula 9), a new sub-library is build. This policy is executed by the following algorithm.

Algorithm 9: NewMc building

```
1   input: the new case (Nc)
2   begin NewMc policy
3   Let indValue //tested γ value
4   //Find most similar to the incoming case according formu-
5   la 1
```

```
6                          ρ = min_j arg D(Nc, Mc^j)
7   /* ρ represents the sub-library where the Nc is stored and
8   Mc is the prototype of it*/
9   Let desvMc = σ( ρ.Mc)
10  Let desvNc = σ(Nc)
11  if (desvNc <= desvMc * (1 + indValue)) then
12    StoreCases(Nc, ρ.Mc)
13    Update(ρ.Mc)
14    return //the process ends when the case is learned
15  else
16    BuildMc(Nc) /* a new Mc is created, the new Mc takes
17                   the values of the Nc*/
18  endif
19  return
20  end NewMc building
```

At the beginning of the algorithm is defined the relaxation value for the prototypes (line 2). According to an experimental procedure that it has been implemented, the optimal value of $\gamma = 0.1$. Others values were tested but this threshold gave the best results (detailed in experimental evaluation section). Once found the nearest prototype (line 6), then the dispersion of the selected prototype is computed (line 9), those taking into account its variables. In line 10 the dispersion of the new case is computed. Using those dispersions the estimation of building or not a new prototype is computed in line 11. Whether the case is going to be stored in the selected prototype then it is done in line 12. In the step $Update(\rho.Mc)$ line 13, the value of the prototype is updated. This is done considering the values of the Nc saved in its structure (line 22). The update is done implementing the Mc building process, previously explained. When an arriving case does not accomplish the second moment, then a new prototype is created (line 16). This new prototype considers the values of the Nc as $\rho.Mc = Nc$ and store the case in its structure.

5.3 Another Meta-case Learning Strategy

In this section, one additional strategy for the building of meta-cases will be presented.

5.3.1 Building real meta-cases

This criteria uses the same procedures than the standard building of meta-cases described previously, but with the difference that this time, the prototypes of each cluster (virtual meta-cases) are replaced for *real cases*. This means that the prototype of a cluster of classes is no more a virtual case, but a case existing within the set of cases of the cluster. The procedure for computing the real Mc is is described by the computing real Mc algorithm.

Algorithm 10: computing real Mc

```
1   Input: the cases stored in the MCL
2   Let VMc = the usual Virtual Meta-case
3   Let RMc = the Real Meta-case
4   VMcs[n] = Compute the usual set of n virtual MCs;
5   foreach VMc in VMcs do
6       DistCase2VMc[m] = compute the m dissimilarity values
7   between the corresponding VMc and all the m cases repre-
8   sented by the VMc.
9       Take the RMc as the most similar case to the VMc
10  endforeach
11  return RMc
12  end
```

The dissimilarity between the cases and their corresponding Virtual Mc can be computed using the Euclidean distance, when all attributes are numeric, and using a heterogeneous dissimilarity measure (Gower dissimilarity measure (Gower, 1971), *L'Eixample* distance (Sànchez-Marrè et al., 1998), etc.) when there are both numeric and categorical attributes.

With this strategy, the following algorithm is used to learn a real Mc

Algorithm 11: *learning real Mc*

```
1   Require: the new case solved
2   Let MsRMc = the Real Meta-case most similar to the case;
3   Let 2MsRMc = the second most similar Real Meta-case to
4   the case;
5   The distance is computed between the new case and the
6   whole set of Real Meta-cases
7   if d(NC, MsRMc) < α and d(NC, 2MsRMc) ≥ α then
```

```
/* α is the threshold value, predefined by the experts or
computed by trial and error*/
   Update the corresponding sub-library with the new case
elseif d(NC, MsRMc) > α then
   Build a new sub-library
elseif d(NC, MsRMc) < α and d(NC, 2MsRMc) < α then
   Evaluate the competence of the MsRMc and 2MsRMc;
   Select the sub-library that get a better competence
   with the new case;
   The corresponding cluster and Virtual Meta-case is up
   dated with the new case;
endif
End
```

5.4 Introspective tasks for optimal maintenance of the DACL

Continuous domains are domains where cases are generated from a continuous data stream. In these domains, many cases are continuously solved and learned by a CBR system. This means that many cases could be stored in the case library. Thus, the efficiency of the CBR system both in size and in time could be deeply worsened. The proposed Dynamic Adaptive Case Library (DACL) framework is able to adapt itself to dynamic environments by means of a set of dynamic clusters of cases and indexing structures (k-d trees, discriminant trees) associated to each cluster. The prototype of a cluster is the Meta-case. The aim of DACL is to get an optimal and competent case library that works efficiently in a continuous domain.

Another important aspect related to unsupervised continuous domains is the *incrementality problem*. General CBR systems assume that the set of cases available for building the case library is fixed. Then, they build the memory indexing structures, like for instance, a *k*-d tree, etc. However, when a CBR system is facing an unsupervised continuous domain, the system should build and update the case library structure/s in an incremental way.

In previous chapters, we have been proposing several techniques for indexing the cases (*k*-d trees), for exploring the indexing structure and not missing the most similar cases (partial matching exploration), stochastic techniques for the learning of new solved cases and meta-cases. Anyway, a dynamic structure must be able to be constructed in an incremental way.

In our DACL proposal, the structure of the DACL can be built-up in an incremental way, and some introspective reasoning tasks can be scheduled to regularly update the indexing structures (NIAR *k*-d trees, etc.). In this chapter, some introspective tasks will be proposed to get incremental indexing structures of our DACL proposal, and to improve the learning of new cases and meta-cases.

5.4.1 Introspective maintenance of the NIAR *k*-d tree

The indexing structure used in the DACL, i.e, a NIAR k-d tree, can be populated with an initial set of cases at the beginning of the use of a CBR system. This way, several meta-cases, as representatives of

each cluster of cases will be created. In addition, the cases will be stored in their corresponding tree node, according to the splitting criteria of the indexing structure. Regarding the NIAR k-d tree, the splitting value at each node is the nearest value of the cases to the average value of the attribute. Therefore, at the beginning all the average values for all the attributes are computed and the nearest value present in the corresponding cases for those attributes are selected as the splitting values at each node.

The problem is that when the system is continuously learning new cases, it can happen that the splitting value of a node, which should be the nearest value to the average value of the attribute, is not anymore the nearest value. This will be caused by the fact that the average value of the attributes is changing continuously. This situation could provoke that the indexing k-d tree could start to be not well balanced in all its subtrees, worsening the retrieval time. This will be a hard problem if the new values of the attribute, which are arriving at the DACL, are very disturbing.

For our proposal, we will consider that a value of an attribute

$$x_{n+1} \text{ is a disturbance value} \Leftrightarrow |Av(x_n) - x_{n+1}| \geq stdev(x_n)$$

That means the values with high dispersion will be those that are far from the mean value of the attribute. *Av*() is the mean value and *stdev*() is the standard deviation of the distribution of values.

Fortunately, the DACL framework can easily and incrementally compute the new average values for all the attributes, according to the following formula:

$$Av(x_{n+1}) = \frac{n\, Av(x_n) + x_{n+1}}{n+1}$$

Where $Av(x_k)$ is the average mean value of the attribute x according to its first k values (x_1, ..., x_k).

Also the standard deviation can be computed in an incremental way through this formula due to Welford (Welford, 1962):

$$stdev(x_{n+1}) = stdev(x_n) + (x_{n+1} - Av(x_n)) * (x_{n+1} - Av(x_{n+1}))$$

Our proposal is that the DACL system will trigger a maintenance task to rebuild a concrete indexing NIAR *k*-d tree, at asynchronous time periods, when the following condition would be satisfied, since the last time the task was fired:

$$\#\text{Disturbance values} \geq \delta * N$$

where δ is a specified percentage, we propose as initial trial that δ= 0.2 and N is the size of the corresponding sub-library.

This criterion mean that when the number of disturbance values is higher than a specified percentage (for instance the 20%) of the number of cases of the sub-library, the task of rebuilding the indexing NIAR *k*-d tree corresponding to the sub-library will be started. A new NIAR *k*-d tree will be generated with the possible new splitting values at each node of the tree.

5.4.2 Introspective task to improve the learning of new cases

As it was detailed in section 5.3.1, there is the possibility to work with real Meta-cases instead of the virtual Meta-cases. The difference relies in the fact that now the prototypes of each cluster (virtual meta-cases) are replaced for *real cases*. This means that the prototype of a cluster of classes is a case existing within the set of cases of the cluster. Concretely, the real Meta-case will be the nearest real case to the virtual meta-case.

The idea is that the real meta-cases perhaps could be a good solution for *impasse situations*. Impasse situations happen when a new case which must be stored in the DACL is equally similar to more than one virtual meta-case (prototype). Even though a case could be at the same distance to several virtual Meta-cases, perhaps the distance to the corresponding real Meta-cases will not be than same, and the impasse situation could be solved using the following algorithm.

Algorithm 12: retaining/learning algorithm avoiding impasses with real Mc

```
Input: the new case solved (Nc)
Let MsMC = the virtual Meta-case most similar to the
case;
Let 2MsMC = the second most similar virtual Meta-case to
the case;
begin
The distance is computed between the new case and the
whole set of virtual Meta-cases using the formula 1;
if d(NC, MsMC) < α and d(NC, 2MsMC) ≥ α then
    /* α is the threshold value, predefined by the
       experts*/
  Update the corresponding sub-library with the new case
elseif d(NC, MsMC) > α then
  Build a new sub-library
elseif d(NC, MsMC) < α and d(NC, 2MsMC) < α then
  Use the real Meta-cases instead of the virtual Meta-
  cases;
  Select the sub-library with the minimum distance
    between the case and the corresponding real
    Meta-cases;
  The corresponding cluster, the virtual Meta-case, the
    real Meta-case are updated with the new case;
endif
end retaining/learning algorithm avoiding impasses with
      real Mc
```

6 Experimental Evaluation and Results

In the previous chapters, the methods proposed in the thesis have been detailed. In this chapter, the evaluation of the methods is explained, and the results are discussed. The evaluation of DACL/MCL and the policies to improve the CBR reasoning cycle are deeply detailed. In first place, the evaluation of several indexing strategies in *supervised domains* is detailed. In this evaluation, the algorithms NIAR *k*-d tree and AvKd-tree are compared versus the standard *k*-d tree for *exact-case search* such depicted in figure 18, in order to get a feedback from the database field scenario, where *k*-d trees approach was originated. In the evaluation of proposed strategies, it is considered the evaluation of the quality in the retrieval process and the retrieving time. In this scenario, the depth of tree is evaluated too such depicted in figure 18.

After this, the evaluation of a MCL in *supervised domains*, but using a *similar-case search* as a guiding searching experimentation, it is detailed and 12 strategies including a flat case library and both standard *k*-d tree and NIAR *k*-d tree methods for indexing cases are described. In addition, the Partial Matching Exploration (PME) technique is evaluated against the commonly used technique based on the hyperball with bounds technique. To evaluate the case retrieval step, the algorithms for retrieving such as NIAR *k*-d tree, standard *k*-d tree, hyperball with bounds and partial matching are analyzed in detail, see figure 17. Then, the promising results are discussed.

Next, the Dynamic Adaptive Case library (DACL) is analyzed (see figure 17). An environmental domain has been selected to test the evaluation. To achieve the objective of building representative prototypes in DACL, a stochastic strategy/method has been proposed. In this section, the building of representative meta-cases with the proposed stochastic method is evaluated. Finally, a discussion of the results is presented.

The final step in the evaluation scheme has been the testing of the whole Dynamic Adaptive Case Library framework, where the DACL, the NIAR k-d trees, the PME technique have been used to cope with some *simulated unsupervised domains* in an incremental way. Therefore, this way, the hierarchical structures (NIAR k-d trees) are *incre-*

mentally constructed and all the cases in the databases used are processed in a step by step mode.

The figure 17 graphically summarizes the experimentation process. showing the different strategies used to evaluate the different proposals in different domains (supervised and unsupervised). In the figure, the indexing methods implemented are detailed, theexploration ntechnique used, the kind o serach pursued, etc.

For the evaluation of the dynamic proposal, the following figure 17 shows sections that are combined when a set of tests is performed. The first section shows the option of not use or to use the structure MCL. Other part indicates the algorithm that could be used as an option for indexing the data and storing in the MCL or NonMCL. Those indexing strategies are evaluated considering the time used in retrieval the information and the quality in retrieval. The four retrieval algorithms are indicated in the figure 17, but the 12 combinations of index method vs retrieval method used to evaluate the proposal are detailed in the sections of this chapter.
In the figure 17, the unsupervised domains section, shows the evaluations done with the environmental data base and the strategies that have been used, here the DACL + Stochastic learning method was implemented successfully. The second part is under construction, the strategies that going to be used are DACL + NIAR + PME and the others 12 policies used in the supervised domains. In this occasion the 10 data bases form UCI used in the supervised evaluation going to be used, but in unsupervised point of view that means that the data going to be handled as an unsupervised data arriving as a data stream.

Domain	Search	Library Type	Indexing Method	Exploration Strategy

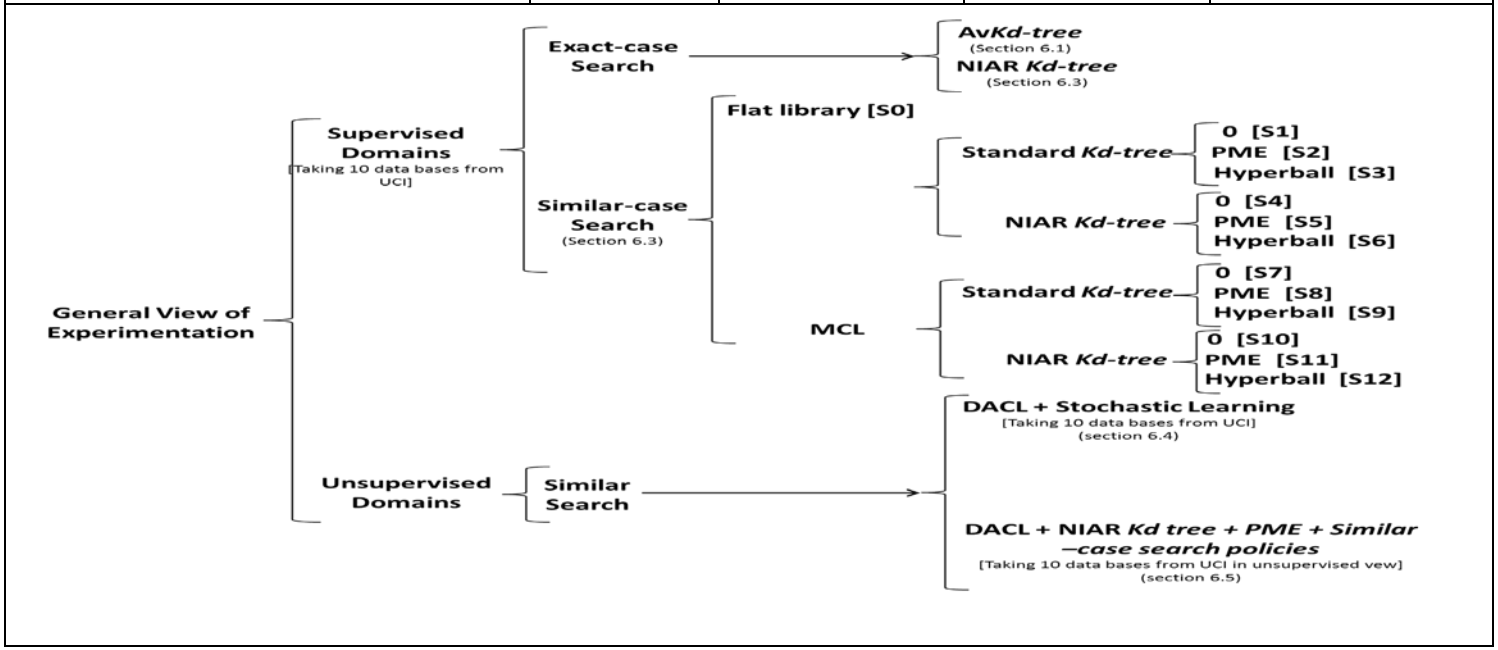

Fig. 17. Experimentation flowchart

6.1 Av*k*d-tree evaluation in exact-case search

For the experimental evaluation, ten databases from UCI Machine Learning repository (Frank and Asuncion, 2010) were selected. The databases selected are depicted in table 4. All the databases have numerical attributes or categorical ordered attributes, which can be transformed to a numerical ordered attribute. In one database (AB) one attribute which was categorical Not ordered was transformed to a numerical one, to test some future usage in the *k*-d trees and AvKd-Tree. In table 4 there is the description of all databases used in the experimental evaluation.

Table 4. Description of databases used in the experimentation. #Inst is to the total number of instances in the database, #Cont means the total number of continuous/numerical attributes in the database, #CatOrd mens the total number of categorical ordered attributes, #CatNOrd means the total number of categorical non ordered attributes and #Classes refers to the total number of different class labels in the database.

Database	#Inst	#Cont	#Cat Ord	#Cat NOrd	#Classes
Abalone	4177	7	0	1	29
Car Eval.	1728	0	6	0	4
Ecoli	336	7	0	0	8
Glass	214	9	0	0	7
Ionosphere	351	34	0	0	2
Pima	768	8	0	0	2
Iris	150	3	0	0	3
Waveform	5000	3	0	0	3
Letter	20000	16	0	0	26
Balance	625	21	0	0	3

We have conducted a test to evaluate the performance of algorithms. The aim of the test was to assess both the retrieval CPU time effort and the depth of the *k-d Tree* and *AvKd-Tree* (the distribution of cases at the tree levels). The testing was done for an exact-case search.

In the experimentation for each scenario, the same cases are tested (test set) in all trees, and the retrieval time is computed. The experimental validation was done randomly sampling the *test set* with a number of cases equal to the 15% of data in each database.

Each experimental validation in one database was conducted in the following way:

- The cases to be retrieved were randomly sampled from the database.
- Case retrieval for each query test case was executed for all tree approaches and the retrieval time was computed.
- The mean time of all query cases retrieval in *test set* was computed for each *k*-d tree approach.
- Searching for accurate results, a multiple validation process was undertaken. Each experiment was repeated twelve times in both trees. Once computed the 12 runs for each database with both approaches, the maximal and minimal time consumed were excluded to get more stable results. Thus, finally only 10 runs were considered.
- With the ten time mean values obtained before, a final time average over all runs was calculated and this value was the best estimation of a general case retrieval time consumed by the CPU.

The whole experimentation was repeated for each one of the databases. The experiments were done in a computer with an Intel Core i7 processor, and 10 GB of RAM.

Table 5. Average CPU Time for case retrieval

Database	***k*-d Tree**	**Av*k*-d Tree**
Glass	16.08	17.43
Ecoli	19.42	19.77
Ionosphere	6.62	7.31
Pima	17.84	23.12
Car	44.47	37.1
Abalone	17.25	16.36
Iris	14.72	9.25
Balance	7.45	5.41
Waveform	15.37	16.01
Letter	13.05	10.12
Mean	***17.227***	16.18

The table 5 depicts the average CPU processing time for a case retrieval using the two new approaches compared with the standard *k*-d tree ap-

proach, and for all the databases tested. The time computed is expressed in *nanoseconds (ns)*.

In figure 18 there is a chart with the respective retrieval time in the approaches for all databases.

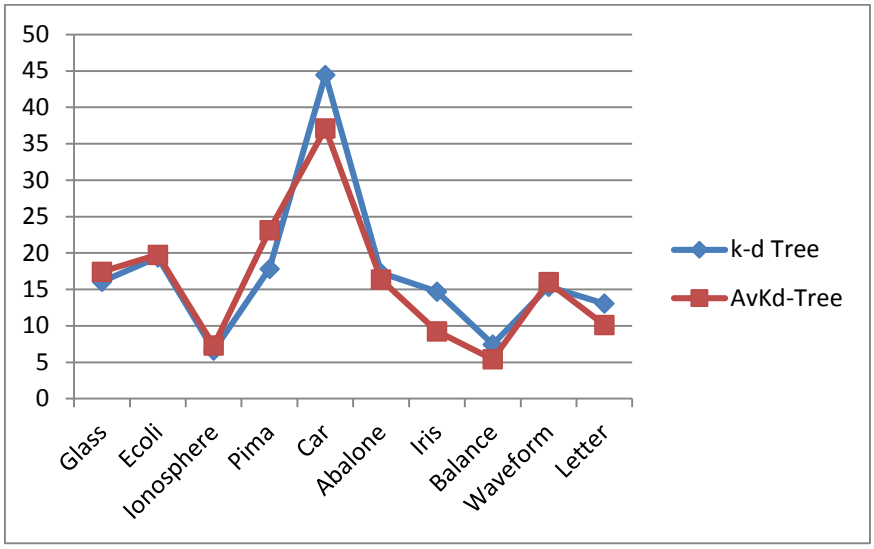

Fig. 18. Comparison of retrieval time in the approaches

The results shown in table 5 and figure 19 depict a difference in time consuming when a search is implemented. It seems that the proposed *AvKd-Tree* strategy is more efficient in retrieving information than standard *k*-d tree in some occasion, but it is clear that Av*k-d tree* strategy is generally more efficient than *Kd-Tree* strategy. The comparison between *AvKd-Tree* strategy and standard *k-d tree* approach gave as a result that *AvKd-Tree* strategy is on average 1% better. The data sets *Abalone, Letter, Waveform and Car* databases are the databases show a higher reduction in the depth of the tree (32, 21 ad 21 levels reduction) despite that they are the largest databases (4177, 20000, 5000 and 1728 respectively).

The second dimension that was used to evaluate the *AvKd-Tree* strategy and *k-d tree* proposal was the evaluation of the *tree depth* and the distribution of cases at the different levels of the trees. The table 6 shows

significant differences in the number of tree levels in the trees, for all the databases. Table 6 shows the depth level of the *AvKd-Tree* strategy and the standard *k*-d tree approach. In all databases, *AvKd-Tree* strategy build the trees with a significant reduction of levels, and the standard *k*-d tree builds the trees with more levels causing a more expensive time in the retrieval.

Abalone, Pima and Iris databases are the databases showing a higher reduction in the depth of the tree (46, 37 and 28 levels were reduced to 14, 16 and 13 levels respectively) despite some are the largest databases. Not surprisingly, most of these databases are the ones were the time retrieval reduction was higher. This fact means that the reduction in time retrieval is directly correlated with the decrease in the number of levels in the trees.

Table 6. Depth of trees generated using in approaches

Data Base	*k*-d Tree	AvKd-Tree
Abalone	28	15
Car	14	13
Ecoli	21	14
Glass	26	16
Ionosphere	19	62
Pima	46	17
Iris	37	12
Balance	15	11

Thus, it is very interesting to try to find out to which factor is due the reduction of levels. A reasonable answer is that the trees would be more balanced, i.e. the number of nodes in the tree will be more uniformly distributed along the different levels of the tree. In order to check this hypothesis a third dimension was investigated: the nodes expanded at each level. This means that the tree should be expanded uniformly in breadth, and not to generate extra levels in the tree increasing the depth of the tree.

Table 7. Distribution of expanded nodes by level in the approaches and compared with the most compact possible binary tree, for the Car database

Level	Most compact tree	*k*-d tree	AvKd-Tree
0	**1**	**1**	**1**
1	**2**	**2**	**2**
2	**4**	**4**	**4**
3	**8**	**8**	**8**
4	**16**	13	**16**
5	**32**	17	**32**
6	**64**	20	**64**
7	**128**	23	**128**
8	**256**	26	**256**
9	**512**	33	**512**
10	1024	33	704
11	2048	41	960
12	4096	56	1152
13	8192	61	0
14	16384	63	0
15	32768	74	0
16	65536	69	0
17	131072	88	0

The table 7 details a comparative by levels indicating the number of expanded nodes at each level, for the Car database. The results show in the first nine levels, the *AvKd-Tree* has expanded the maximum nodes as the most compact tree would do. On the contrary, the *k*-d tree has a different situation: starting at level 4 and going on, it has not expanded all the nodes that would be desirable. For example, at level 4 it has 3 nodes not been expanded. This behavior continues until 37th level where the last 1 node is expanded. This table outlines the fact that the *AvKd-Tree* provides in Iris data base a more structured and breadth-balanced tree than a standard *k*-d tree. Its behavior corresponds to the implemented strategy where a virtual root node is built each time when the Att_l^i value is computed and selected to be the new root. Our *AvKd-*

Tree proposal is better in several evaluations, but not in all evaluations than standard *k-d tree*.

6.2 NIAR *k*-d tree evaluation in exact-case search

All *k*-d trees can only cope with numerical or categorical ordered attributes, where an order relation exists. This means that the splitting process, separating two sets according to the order relation of all cases/instances is possible. *K*-d trees are not initially suitable for unordered categorical attributes.

Experiments on ten databases from UCI Machine Learning repository (Frank and Asuncion, 2010) were conducted. The databases selected are detailed in table 4, in the table there is the description of all databases used in the experimental evaluation.

It was aimed in this experimental setting, to assess both the retrieval CPU time effort, the depth of the generated trees, and the distribution of cases at the tree levels, as we did in the experimentation with *AvKd-Tree* for *exact-case search*. We compared the standard *k*-d tree approach using the median value as the partition value at each internal node against the NIAR *k*-d tree approach. Each experimental test retrieved the same cases in both trees, and computed the retrieval time in both approaches. The experimental validation was done by taking random samples from the *test set* with a number of cases equal to the 15% of data in each database.

Each experimental validation was conducted in the following way:

1. The cases to be retrieved were randomly selected from the database.
2. Case retrieval for each query test case was executed for both tree approaches and the retrieval time was computed.
3. The mean time of all query cases retrieval in *test set* was computed for each *k*-d tree approach.
4. Searching for accurate results, a multiple validation process was performed. Each experiment was repeated twelve times in both trees. Once computed the 12 runs for each database with both approaches, the maximal and minimal time consumed were excluded to get more stable results. Thus, finally only 10 runs were considered.

5. With the ten time mean values obtained before, a final time average over all runs was calculated and this value was the best estimation of a general case retrieval time consumed by the CPU.

The table 8 depicts the average CPU processing time for a case retrieval using the two approaches, and for all the databases tested. The time computed is expressed in *nanoseconds (ns)*. The time reduction percentage is between the two methods is presented for comparative purposes.

Table 8. Average CPU Time for case retrieval

Database	***k*-d Tree**	**Av*k*-d Tree**	**NIAR *k*-d Tree**
Glass	16.08	17.43	11.02
Ecoli	19.42	19.77	9.85
Ionosphere	6.62	7.31	6.18
Pima	17.84	23.12	5.49
Car	44.47	37.1	9.85
Abalone	17.25	16.36	15.98
Iris	14.72	9.25	13.04
Balance	7.45	5.41	5.09
Waveform	15.37	16.01	19.26
Letter	13.05	10.12	27.64
Mean	***17.227***	***16.18***	***10.263***

The result showed in table 8 and figure 19 shows a difference in time consuming when a search is implemented. It seems that the proposed NIAR *k*-d tree strategy is more efficient in retrieving information. The comparison between NIAR *k*-d tree algorithm and standard *k*-d tree approach gives a result that NIAR *k*-d tree is on average 22% faster in the retrieval process than the standard approach according to the experimental results obtained.

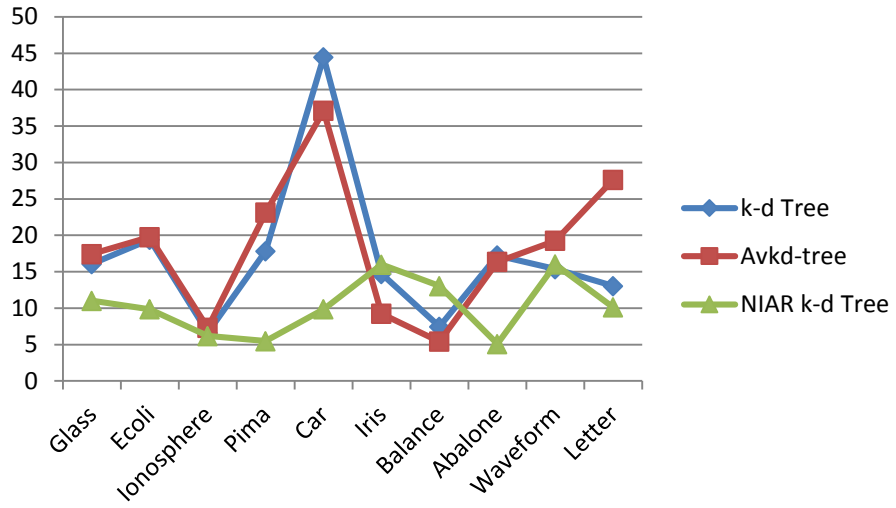

Fig. 19. Comparison of retrieval time

As in the previous section, the second dimension used to assess the performance of the NIAR *k*-d tree proposal was the evaluation of the *tree depth* and the distribution of nodes at the different levels of the tree. The table 9 shows significant differences in the number of tree levels in both trees, for all the databases. It is shown that the NIAR *k*-d tree approach builds trees with a significant reduction of levels, and the standard *k*-d tree builds trees with more levels that cause a more expensive time in the retrieval.

Table 9. Depth of trees generated using both approaches

Data Base	*k*-d Tree	NIAR *k*-d Tree
Waveform	26	13
Glass	14	11
Pima	21	11
Ecoli	26	13
Ionosphere	16	13
Abalone	46	14
Car	37	16
Letter	62	41

Abalone, Letter, Waveform and Car databases are the databases show a higher reduction in the depth of the tree (32, 21 ad 21 levels reduction)

despite that they are the largest databases (4177, 20000, 5000 and 1728 respectively). The other large database is Waveform with a reduction of 13 levels. Not surprisingly, these four databases are the ones were the time retrieval reduction was higher. This fact corroborates the previous findings in the Avkd-tree experimentation, in the sense that the reduction in time retrieval is directly correlated with the decrease in the number of levels in the trees.

Thus, again we hypothesized that the factor causing the reduction of levels was the same. The number of nodes in the tree is more uniformly distributed along the different levels of the tree. In order to check this hypothesis the same third dimension than with Avkd-tree was analyzed: the expanded nodes at each level. This means that the tree should be expanded uniformly in breadth not to generate extra levels in the tree increasing the depth of the tree.

Table 10. Distribution of expanded nodes by level in both approaches and compared with the most compact possible binary tree, for the Abalone database

Level	Most compact tree	*k*-d tree	NIAR *k*-d tree
0	**1**	1	**1**
1	**2**	2	**2**
2	**4**	4	**4**
3	**8**	8	**8**
4	**16**	15	**16**
5	**32**	22	**32**
6	**64**	38	**64**
7	**128**	53	**128**
8	**256**	71	**256**
9	512	92	502
10	1024	124	940
11	2048	149	1271
12	4096	163	829
13	8192	197	122
14	16384	216	2

To illustrate this fact, in table 10 for the database Abalone is detailed the number of nodes expanded at each level for both approaches in comparison with the complete (maximum) number of nodes that will generate the most compact possible tree. The results show why the

NIAR *k*-d tree has a better performance than the standard *k*-d tree in this *exact-case search* scenario.

In the first eight levels, the NIAR *k*-d tree has expanded the maximum nodes, as would do the most compact tree. On the contrary, the *k*-d tree has a different situation: starting at level 4 and going on, it has not expanded all the nodes that would be desirable. For example, at level 4, only 1 node has been missed, but at level 8, 185 nodes have not been expanded. In the 14th level of NIAR *k*-d tree, which is the last level for this approach, the last 2 nodes have been expanded while in the *k*-d tree 216 nodes have been expanded. This table outlines the fact that the NIAR *k*-d tree provides a more structured and breadth-balanced tree than a standard *k*-d tree.

The Wilcoxon Signed-Range Test (Wilcoxon, 1945) was used to determinate differences in the time retrieval from the two approaches NIAR kd tree and the standard kd-tree (data from table 8). The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples, matched samples, to assess whether their population mean ranks differ (i.e. it is a paired difference test). It can be used as an alternative to the paired Student's t-test when the population cannot be assumed to be normally distributed. We used it, in order not to assume any restrictive hypothesis on the sample like normality.

The ***p*** value reported by the statistical test was 0.0078, which is less than the critical value tabulated for a sample of 8 individuals (4 at 95%, 2 at 98%). Thus, at a 95% of confidence or even at a 98% of confidence, the difference between the paired samples is significant. This means that the NIAR *k*-d tree approach provides faster case retrieval time than the standard *k*-d approach in *exact-case search*.

6.3 Testing the Multi Case Library approach for *similar-case search* in supervised domains

The new approaches proposed (a Multiple Case Library (MCL), the partial matching exploring strategy), and the previously proposed NIAR *k*-d tree (Orduña and Sànchez-Marrè, 2013) will be combined to be tested against other commonly used approaches in *similar-case retrieval*.

In the previous section, it was shown that the NIAR *k*-d tree was a promising technique for improving the time efficiency, but it should be tested for *similar-case retrieval*, which is the usual search done in CBR systems, different from common *exact-case search* in databases. In addition, it should be tested regarding to competence of the CBR system and the new proposed partial matching exploration technique should be tested too.

The commonly used approaches in the literature are the use of just one case library, the standard *k*-d tree approach (Broder 1990; Friedman et al. 1977; Bentley, 1975), the hyperball with bounds exploring strategy (Friedman et al., 1977) improved with the virtual bounds technique (Wess et al., 1993). See section 2.1.2.

Through the experimentation tests, it will be showed that the use of a Multiple Case Library (MCL) embedding a NIAR *k*-d tree, and using the additional strategy of partial matching to explore the indexing structure (the tree) provides a very good approach both to improve the *time efficiency* and the *competence accuracy* for case retrieval task in CBR systems.

6.3.1 Experimental Settings

Eight databases from UCI Machine Learning repository (Frank and Asuncion, 2010) were selected in order to test the different combination of approaches and strategies. All databases have numerical attributes or ordered categorical attributes, which can be transformed to an ordered numerical attribute, because all *k*-d trees can only cope with numerical or categorical ordered attributes, where an order relation exists. In table 4, *Abalone* database has one categorical attribute, which was not ordered. It was transformed into a numerical one aiming to be possible to use the standard *k*-d tree and NIAR *k*-d tree approaches to compute the partition value (median value, nearest value to mean value). The complete properties of each database are compiled in table 4.

The experimental setting was done under the following characteristics:

- The above eight databases were tested

- As a baseline, the Flat Case Library was considered which gives the upper bound of the accuracy.

- Twelve different strategies (plus the baseline strategy) were considered combined the different possibilities for the structure of the Case Library (Non MCL/MCL), The Case Library Indexing structure (Standard *k*-d Tree/NIAR *k*-d tree) and the additional exploring strategies (none/Partial Matching/Hyperball with bounds):

  - S0: Flat Case Library
  - S1: Standard *k*-d tree
  - S2: Standard *k*-d tree + Partial matching exploration
  - S3: Standard *k*-d tree + Hyperball with bounds exploration
  - S4: NIAR *k*-d tree
  - S5: NIAR *k*-d tree + Partial matching exploration
  - S6: NIAR *k*-d tree + Hyperball with bounds exploration
  - S7: MCL + Standard *k*-d tree
  - S8: MCL + Standard *k*-d tree + Partial matching exploration
  - S9: MCL + Standard *k*-d tree + Hyperball with bounds exploration
  - S10: MCL + NIAR *k*-d tree
  - S11: MCL + NIAR *k*-d tree + Partial matching exploration
  - S12: MCL + NIAR *k*-d tree + Hyperball with bounds exploration

- For each database and for each strategy, a 10-fold cross validation was performed, taking sequentially (10 runs), each fold as a *test set* and the other nine folds as the *training set*.

- The *average time retrieval* of one case (in µs) and the *average success* on label classification of the cases (in percentage), as an estimation of the competence accuracy, was computed for each database and for each strategy, as an average quantity among the 10 runs of the cross validation.

6.3.2 Experimental Results

The tables 11, 12, 13 and 14 show both the average CPU processing time, and the average percentage of success in predicting the

correct class label of the cases, for a case retrieval using the different strategies for all the databases tested. In addition, two new columns labelled as "Average" show the average values across all the databases to give an estimation of each strategy (see table 14). Finally, the last two columns added give the "Average Reduction of each strategy regarding the baseline strategy" (S0, Flat Case Library) across all the databases (see table 14).

Table 11. Average CPU Time (in µs) for case retrieval and average Success (in %) in class label prediction for all the strategies, and for all the tested databases.

MCL / Non MCL	Case Library	Additional Strategy		Iris		Glass		Balance	
				%Succ	Time	%Succ	Time	%Succ	Time
Flat Memory			E0	**93.7**	790.67	**70.8**	1752.05	**78.6**	2588.86
Non MCL	Standard K-d Tree	None	E1	39.7	1.09	30.9	1.04	14.1	0.9
		Partial Matching	E2	79.7	19.91	**50.8**	28.04	46.2	113.83
		Hyperball with BWB,BOB bounds	E3	75.7	549.64	48.5	809.55	45.6	909.65
	NIAR K-d Tree	None	E4	59	1.46	31.5	1.54	13.2	1.69
		Partial Matching	E5	81	15.43	45.7	21.43	48.2	20.27
		Hyperball with BWB,BOB bounds	E6	72.9	533.17	47.6	309.94	39.6	333.53
MCL	Standard K-d Tree	None	E7	74.9	1.04	43.2	0.96	38.4	1.09
		Partial Matching	E8	82.3	12.13	39.6	13.23	66.8	46.14
		Hyperball with BWB,BOB bounds	E9	77.7	137.36	42.4	234.3	68.4	319.99
	NIAR K-d Tree	None	E10	79	1.26	45.1	1.2	21.3	5.78
		Partial Matching	E11	**91.8**	10.92	45.7	11.92	**74.9**	42.76
		Hyperball with BWB,BOB bounds	E12	68.5	473.85	38.5	226.4	61.6	1132.39

Table 12. Average CPU Time (in µs) for case retrieval and average Success (in %) in class label prediction for all the strategies, and for all the tested databases (continued from table 11).

MCL / Non MCL	Case Library	Additional Strategy		Pima		Ionosphere		Abalone	
				%Succ	Time	%Succ	Time	%Succ	Time
Flat Memory			E0	67.4	4077.85	86.9	1455.07	50	110040.14
Non MCL	Standard K-d Tree	None	E1	39	0.93	67	0.78	36	1.67
		Partial Matching	E2	51.9	123.12	63.8	633.68	30.7	144.58
		Hyperball with BWB,BOB bounds	E3	52.6	724.25	61.8	1296.78	30	1083.87
	NIAR K-d Tree	None	E4	47.2	2.05	63.3	1.52	35.8	2.13
		Partial Matching	E5	53.4	27.7	65.6	50.57	30.3	36.38
		Hyperball with BWB,BOB bounds	E6	50.5	424.16	61.1	432.07	32.5	738.67
MCL	Standard K-d Tree	None	E7	43.3	1.05	67.2	0.74	44.4	1.87
		Partial Matching	E8	60.5	74.46	67.9	352.95	46.7	62.48
		Hyperball with BWB,BOB bounds	E9	58	441.77	73	544.2	48.5	479.12
	NIAR K-d Tree	None	E10	55.2	7.55	66.4	5.97	36.9	7.44
		Partial Matching	E11	63.1	69.9	72.8	142.8	48.2	86.16
		Hyperball with BWB,BOB bounds	E12	59.7	1382.62	67.5	1184.2	46.6	1403.82

Table 13. Average CPU Time (in µs) for case retrieval and average Success (in %) in class label prediction for all the strategies, and for all the tested databases (continued from table 12).

MCL / Non MCL	Case Library	Additional Strategy		Car		Ecoli	
				%Succ	Time	%Succ	Time
Flat Memory			E0	86.8	19127.2	80.4	2245.03
Non MCL	Standard K-d Tree	None	E1	67.6	0.88	40.2	3
		Partial Matching	E2	56.1	365.47	62.1	76.97
		Hyperball with BWB,BOB bounds	E3	59.2	968.28	63.3	1403.44
	NIAR K-d Tree	None	E4	68.7	1.83	40.5	1.4
		Partial Matching	E5	51	26.16	23.1	21.73
		Hyperball with BWB,BOB bounds	E6	58	502.37	52.5	492.78
MCL	Standard K-d Tree	None	E7	69.8	1.07	78.3	1.13
		Partial Matching	E8	70.4	171.69	77.7	16.51
		Hyperball with BWB,BOB bounds	E9	73	486.54	72	230.2
	NIAR K-d Tree	None	E10	69.3	6.51	72.6	4.51
		Partial Matching	E11	71.3	58.33	79.5	39.25
		Hyperball with BWB,BOB bounds	E12	71.8	1197.87	67.5	870.6

Table 14. Statistics of tables 11, 12 and 13.

MCL / Non MCL	Case Library	Additional Strategy		Average		Average Reduction regarding baseline E0	
				%Succ	Time	%Succ	Time
Flat Memory			E0	76.83	17759.6		
Non MCL	Standard K-d Tree	None	E1	41.81	1.29	35.02	99.99274
		Partial Matching	E2	55.16	188.2	21.67	98.94029
		Hyperball with BWB,BOB bounds	E3	54.59	968.18	22.24	94.54842
	NIAR K-d Tree	None	E4	44.9	1.7	31.93	99.99043
		Partial Matching	E5	49.79	27.46	27.04	99.84538
		Hyperball with BWB,BOB bounds	E6	51.84	470.84	24.99	97.34882
MCL	Standard K-d Tree	None	E7	57.44	1.12	19.39	99.99369
		Partial Matching	E8	63.99	93.7	12.84	99.4724
		Hyperball with BWB,BOB bounds	E9	64.13	359.19	12.7	97.97749
	NIAR K-d Tree	None	E10	55.73	5.03	21.1	99.97168
		Partial Matching	E11	68.41	57.76	8.42	99.67477
		Hyperball with BWB,BOB bounds	E12	60.21	983.97	16.62	94.45951

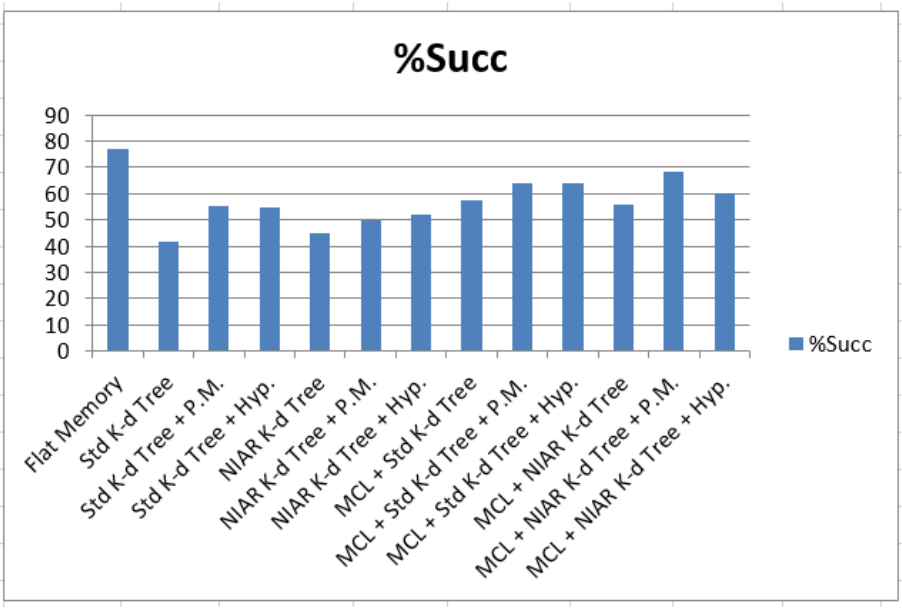

Fig. 20. Comparison of averaged success (%) across all databases for the class label prediction in all the strategies

Figure 21 shows the average CPU retrieval time for one case across all the databases depicted in tables 11, 12 and 13, for each strategy minus the baseline Flat Case Library, in a graphic. The baseline is not depicted in order not to distortion the graphic, because its value is much higher than the other ones, and then, the differences among the other strategies could not be appreciated.

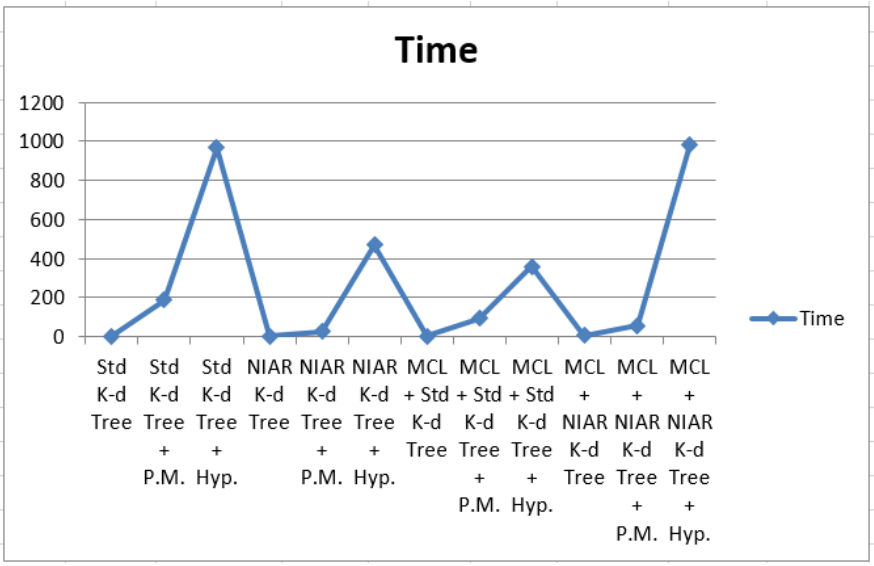

Fig. 21. Comparison of averaged CPU retrieval time (µs) across all databases for the class label prediction in all the strategies minus S0 (baseline Flat Case Library)

6.3.3 Discussion of the results

6.3.3.1 Non-MCL strategies

Regarding the strategies which use only one case library (non-MCL strategies, i.e., S1 to S6), *the use of an additional exploration technique* (partial matching or hyperball with bounds) leads to an *increase between 8-14% in the accuracy* regarding not using them, as it is shown in figure 22. Therefore, they seem *useful for increasing the accuracy*.

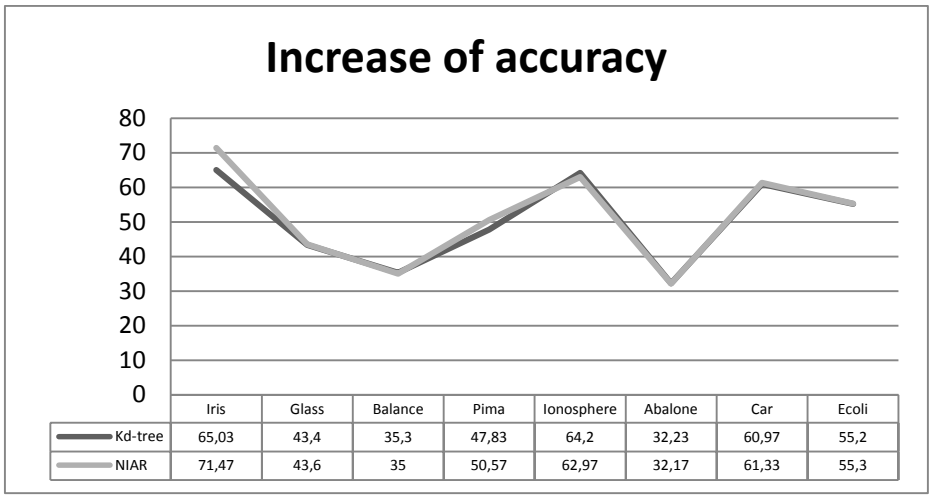

Fig. 22. Increase of accuracy

The NIAR *k*-d tree approach without any additional exploration strategy (S1) compared with the standard *k*-d tree without any additional strategy (S4) is slightly better in accuracy on average, but slightly worse in time. Anyway, these differences are not significant.

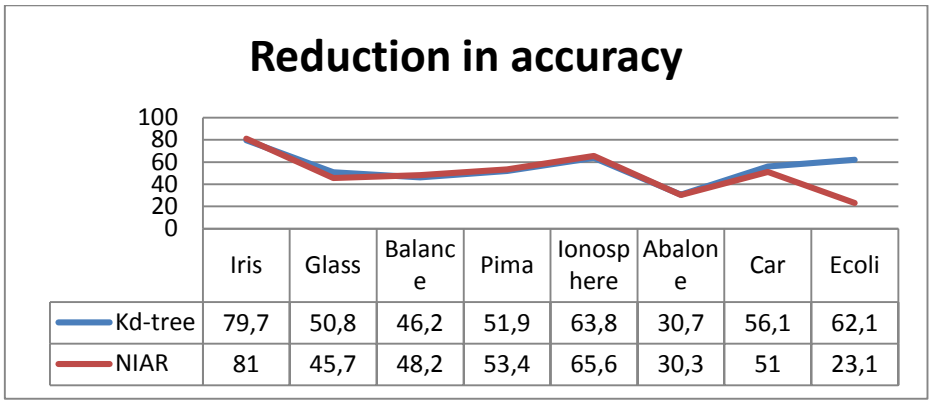

Fig. 23. Reduction of accuracy

Comparing the same additional exploration technique with a standard *k*-d tree (S2 or S3) and with a NIAR *k*-d tree (S5 or S6) there is a 5% approx., reduction in accuracy (see figure 23). This means that for a

single Case Library approach, it seems to be a better option to use the standard *k*-d tree than a NIAR *k*-d tree.

Regarding the time performance, *all these non-MCL strategies* (S1 to S6) *make an important reduction in time* (more than 97%) regarding the baseline strategy (S0), which is a linear sequential search over the Case Library, see following table 15.

Table 15. Time performance.

Average S0,S6		Average Reduction regarding baseline S6	
%Succ	Time	%Succ	Time
76.83	17759.6		
51.84	470.84	24.99	97.34882

Anyway, what can be observed is that the partial matching exploration strategies (S2, S5) have *significative better results in time than the corresponding hyperball with bounds strategies* (S3, S6), independently if they use a standard *k*-d tree or a NIAR *k*-d tree. That means the *partial matching exploration technique* is a very good option to reduce the time retrieval.

Therefore, *among all strategies using only one Case Library* (S1 to S6), and taking into account both the accuracy and time features, probably *the best strategy would be the standard k-d tree with partial matching exploration* (S2, with an average of 55.16% of accuracy and with an average of 188 µs for case retrieval). In addition, the NIAR *k*-d tree with partial matching exploration could be also a good option (S5, with an average of 49.79% of accuracy and with an average of 27 µs for case retrieval).

6.3.3.2 MCL Strategies

Regarding the strategies which use a Multiple Case Library (MCL strategies, i.e., S7 to S12), *the use of an additional exploration*

technique (partial matching or hyperball with bounds), as in the non-MCL strategies, leads to *an increase between 9-17% in the accuracy* regarding not using them as it is depicted in table 16. Therefore, it can be confirmed that these techniques *increases the accuracy*.

Table 16. Increase of the accuracy.

Average S7,S11		Average Reduction regarding baseline S7,S11	
%Succ	Time	%Succ	Time
76.83	17759.6		
57.44	1.12	**19.39**	99.99369
68.41	57.76	**8.42**	99.67477

Moreover, *all MCL strategies* (S7 to S12) *improve the accuracy of its corresponding non-MCL strategies* (S1 to S6) by 8-19%.

The *k*-d tree approach without any additional exploration technique (S7) compared with the standard NIAR *k*-d tree without any additional exploration technique (S10) is slightly worse (2%) in accuracy on average, and slightly worse in time.

Comparing the same additional exploration technique with a standard *k*-d tree (S8 or S9) and with a NIAR *k*-d tree (S11 or S12) there is a different behavior. With the *partial matching exploration*, there is a 4% approx. *increase in accuracy using the NIAR k-d tree*. On the contrary, with the *hyperball with bounds technique*, there is a 4% approx. *reduction in accuracy using the NIAR k-d tree*. This means that for a Multiple Case Library approach, it seems to be a better option to use the NIAR *k*-d tree with partial matching exploration. This strategy reaches a 68.41% of accuracy see following figure 25, the best among all alternatives to the baseline, just only 8.4% lower than the 76.8% of accuracy of the baseline strategy (S0), which is the maximum possible accuracy.

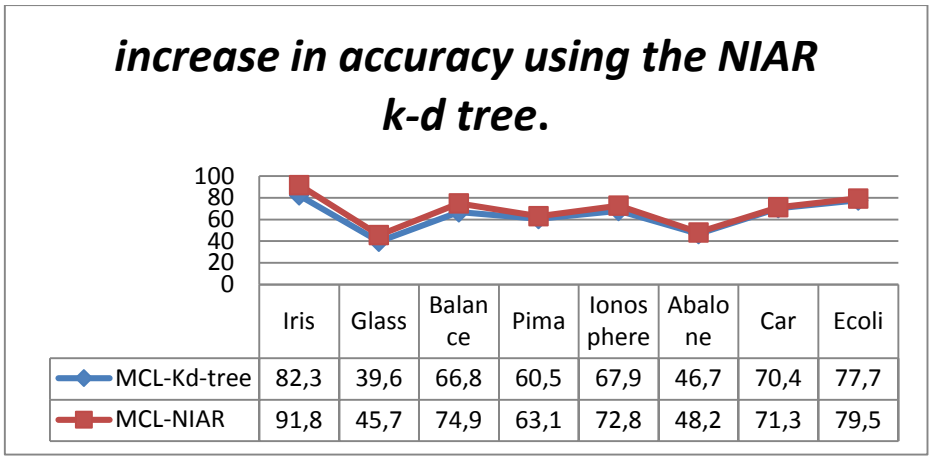

Fig. 24. Increase of accuracy in NIAR

Regarding the time performance with the MCL strategies, the same situation as with the non-MCL strategies can be observed. *All MCL strategies* (S7 to S12) *make an important reduction in time* (more than 97%) regarding the baseline strategy (S0). Notwithstanding, the time employed is lower in strategies using an additional exploration technique (S8, S9, S11, S12) than its corresponding non MCL strategies (S2, S3, S5, S6). S7 and S10 have slightly higher time retrieval, but it is almost the same value.

Again, it can be observed that the *partial matching exploration* techniques (S8, S11) have significate better results *in time* than the corresponding hyperball with bounds techniques (S9, S12), independently if they use a standard *k*-d tree or a NIAR *k*-d tree. Thus, it confirms that the *partial matching exploration technique* is a very good option to reduce the time retrieval.

Therefore, *among all strategies using a Multiple Case Library* (S7 to S12, depicted in table 17), and taking into account both the accuracy and time features, probably *the best strategy is clearly the NIAR k-d tree with partial matching exploration technique* (S11, with an average of 68.41% of accuracy and with an average of 57 µs for case retrieval).

Table 17. Accuracy of MCL

Average S0		Average Reduction regarding baseline S7-S12	
%Succ	Time	%Succ	Time
76.83	17759.6		
		19.39	99.99369
		12.84	99.4724
		12.7	97.97749
		21.1	99.97168
		8.42	99.67477
		16.62	94.45951

6.4 Testing the Dynamic Adaptive Case Library (DACL) approach and Stochastic Learning policies in unsupervised domains

Intra-cluster validity measures are usual techniques proposed to evaluate the quality of obtained clusters in a clustering process, but other techniques can be used. These measures are structural validation measures about the intra-cluster compactness and inter-cluster separation dimensions. In our work, to evaluate the quality of the clusters, associated to each sub-library, obtained through the stochastic method introduced in section 5.2, we combine some methods in two strategies called *Policy One (pol1)*, *Policy Two (pol2)*. The policies are summarized as follows:

Policy One: the policy has the aim to measure the distance between *Mc*'s. The distance measure depicts the prototype distributions in DACL. If the prototypes are too close, the building of a new prototype might be considered to avoid an overlapping of prototypes. Neverthe-

less, in the stochastic proposal the learning of a new case is obliged to comply with the two moments of the method described previously. This will ensure that the prototypes do not overlap. The metric used to evaluate this dissimilarity is the Euclidean Distance if all attributes are numerical or some heterogeneous similarity measures (Gower similarity coefficient (Gower, 1971), *L'Eixample* distance (Sànchez-Marrè et al., 1998), etc.) to assess both the numerical and the categorical attribute dissimilarities.

Policy Two: the policy is structured in four steps. The aim is to estimate the compactness of the clusters associated to the sub-libraries and compare the similarity with the prototype. To achieve that, the following formulas are computed.

$$Dav^j = \frac{1}{m}\left(\sum_{i=1}^{m} d\left(Mc^j, C^i\right)\right) \tag{10}$$

$$Mc^j = \sqrt{\sum_{k=1}^{n}\left(Mc_k^j\right)^2} \tag{11}$$

$$SM^j = \sum_{i=1}^{m} d\left(Mc^j, C^i\right) \tag{12}$$

$$Mt^j = {Mc^j}/{SM^j} \tag{13}$$

Where n = # attributes describing the cases and metacases, and m = #cases(Mc^l)

Formula 11 computes an average of distances of all cases of the sub-library and its prototype (Dav). Dav depicts how compact is the sub-library. For a better quality of a DACL, the sub-libraries must be as compact as possible indicating that cases in the sub-library are similar enough to the prototype. In a DACL, a compact sub-library improves the learning rate.

Formula 12 computes the magnitude of the prototype. With the magnitude, the relation between the attributes of the *Mc* is evaluated. A higher value indicates that the pollution is higher, according to the environmental experts. With low values in attributes, better environmental conditions are met.

Formula 13 express the accumulation of the distances between the total of cases in the sub-library and the prototype. Where *j* indicates the cur-

rent *Mc*, and *i* the current case. This measure it is an indicator of how compact is the sub-library.

In *formula 14*, this measure and the magnitude in formula 12 are used to estimate the similarity of the prototype with the cases that its represents. When the value of *formula 14* is low, the prototype and *Mc* have more in common. This particularity indicates that the attributes of the *Mc* are closely similar to the attributes of the cases in the sub-library. Therefore it is a desired a low value.

6.4.1 The Domain and Experimentation Description

The strategy here proposed implements the use of the data obtained in the monitoring of environmental conditions in the city of Obregón, Sonora State, Mexico for a period of one year. Obregón is one of the municipalities of the northwestern state of Sonora, Mexico. Its capital is Ciudad Obregón. The municipality has an area of 3,312.05 km² and with a population of 784,342 inhabitants according to the National Institute of Statistics and Geography (INEGI) by its name in Spanish (INEGI, 2010).

The environmental condition aims to evaluate the air quality in the city, according to the international norms of the World Health Organization (WHO). The conditions considered are: PB(mB), Temperature, Relative Humidity, RS(w/m2), PM@10(ug/m3), PM2@5(ug/m3), Ozone (PPB), SO2 (PPB), NO(PPB), NO2(PPB), NOX(PPB), CO(PPM). The evaluations were done each minute, following international norms.

The aim of the set of tests is to evaluate the learning task of new prototypes for a dynamic adaptive case library (DACL). The Stochastic Method proposed was thought especially to guide the learning of the environmental conditions and storing the conditions as cases in a sub-library according to the proposed policies. The main motivation is to collaborate with environmental experts, providing a method that helps them to evaluate the behavior of the air conditions, according to the norms of WHO. Nowadays, the experts evaluate the information using others tools, but in this process they need to take care of handling the information, especially when they elaborate reports and evaluate information to build conclusions. This task is complicated by the large

amount of information generated in this domain. With proposed DACL policies, this process is dynamic, adaptive and automatic.

In one day, different environmental conditions could happen. This, actually depends on nature, but the human activities have a direct influence in the air quality. The people activities in the city are the industry, manufacturing, and the major activity, which is the agriculture. One fact that affects the air quality is the burning of sheaf at the end of seasons. The burning of sheaf has a direct influence on the seasonal behavior, where a concentration of pollutants is observed when the burning is done. Wheat cultivation is one of the main agricultural activities in the region of Cajeme. In the winter, the low levels of atmospheric pressure limit the dispersion of atmospheric pollutants. Other activity considered normal is the traffic at peak hours. These two activities are considered as the main cause of the air pollution in the city. For one year, the environmental conditions were monitored, and a database was created.

The analysis of a big amount of data is complicated to be done by the experts, which usually consider the use of spreadsheets. Therefore, the use of special analysis tools is required. For instance, the acquired information in one day is of 1440 information units (cases), considering the 12 attributes plus the time stamp attribute. These cases could be very different, because in one day different environmental conditions could happen. The natural conditions are modified by the human activities, and patterns of environmental behaviors could be found. One instance of this is the peak hours when people is going to pick up the children from school, or going to lunch at middle day, and finally, when day ends and people goes to house to rest. The last description of the human behavior could be found in the analysis of data. In addition, when a burning of sheaf happens, it could be found, too. To get conclusions of human activities related with air quality, the experts should split the information considering the time of each activity, and make inferences of the human affectation in air quality. The split of information with spreadsheets is easily done, considering time of activities, but when the split is done considering environmental conditions, the task is complex. The complication here was to find a method for creating and organizing the stored cases in sub-libraries. This way, the environmental conditions were detected. According to the experts, the ex-

pected result of the method is that *should be easy to found the relation of human behavior and air quality*.

The stochastic method proposed is the tool developed to answer the requirement of the experts.

6.4.2 Discussion of Results

The implementation of the proposed method has been carried out using the database acquired in the year of evaluation.

The *Nc* arrives to the DACL. Then, following the stochastic steps that have to be accomplished, according to previous detailed algorithm, the *Nc* is processed. When the new case arrives (*Nc*), the first task is to evaluate the first moment of the proposal, which consists in finding the most similar prototype. When this prototype is found, the second moment consists of checking the learning value. In this step, the learning rate is evaluated and the experimentation is done with the experimental values. Finally, the decision is made.

Previously it has been introduced the core of the proposal; the stochastic method. At the beginning of the algorithm is defined the relaxation value for the prototypes. Here is proposed that $\gamma = 0.1$, because is the best value found in the experimentation. Several values were used in the evaluation: 0.1, 0.2, 0.3, 0.4 and 0.5. According to environmental experts these values were considered fine for evaluation. These values are tested as the variable *indValue* in the following condition:

$$\text{if } (\text{desvMc} \leq \text{desvMc}*(1+\boldsymbol{indValue})).$$

This condition is part of the second moment of the stochastic method. The table 18 depicts the results of the prototypes has been created using them.

Table 18. Number of Prototypes and the γ policy evaluation values. For each γ policy value there is the number of cases of each prototype.

#Prototypes	γ: Policy Evaluation Values				
	0.1	***0.2***	***0.3***	***0.4***	***0.5***
1	472	695	883	1077	1320
2	305	594	556	362	119
3	295	150			
4	354				
5	13				

Table 18 shows the results of the implementation of the stochastic method. These results are from one day of acquiring of data. The behavior of more days of analysis takes the behavior of adding data to the current prototypes. The comments of the environmental experts about the results obtained in the evaluation were the following. The best proposal is the one built with γ=0.1 because it has more prototypes. The second could be when γ=0.2. However, when γ is evaluated with others values the results are out of expected bounds.

The five prototypes generated with γ=0.1 are highly representative of the environmental conditions taking into account the human activities. The first prototype built has 472 cases, where the first case is acquired at minute 00:01 and the last case learned is acquired about the minute 07:08. Between these minutes, usually the human activity is reduced to be sleeping, and the movement of cars is low. Usually, the traffic increase when the people decide to go schools. Normally, the children school starts at 07:30 hrs. Therefore, a peak hour is considered between 07:00 am and before of 13:00, because the pick-up from the school of the little children starts at 12:30 hrs. According to the table, the second prototype represents this human behavior. Third prototype considers the thirst human activity that starts around 13:00 hours ending about at 18:00 hrs. At this period, the people usually are at work. After this, between the time of 18:00 and 23:00, the people travel to home or maybe going to other places. Then the activity of the day ends for some people. Finally, the fifth prototype represents when the environmental condition represents the accumulative values of the day. Then the nature has a break and helps us cleaning the environment. This is done during the first hours of the new day.

When γ=0.2 the algorithm generates three prototypes. In this situation, the prototypes represent the behavior of the environment in three phases. First is between 0-12 hours; this behavior covers the human activities of the first and second period, according the prototype when the γ=0.1. Between the 12-22 hours covers the behavior when people takes a break to lunch and goes to pick up children to school, and finally when people ends the working day. In the 22-24 hours, the people travel to home or maybe going to other places. Having in consideration this arguing, the second γ policy evaluation value could be acceptable, but is more interesting and realistic the first one according the argumentation of the environmental experts.

Anyway, when γ experiments the other values of 0.3, 0.4 and 0.5, the obtained results are out of a reasonable behavior. One explanation of this fact is that the evaluation threshold is extended too much, and the relaxation learning policy is higher, mixing different prototypes in only one mixed prototype, which does not represent a clear different environmental situation, but a merge of several conditions. While in first evaluation of γ=0.1, the relaxation is more demanding, and only cases that accomplish the stochastic moments are learned.

A normal behavior of nature could be described as follows. At the end of the day, when the human activity is reduced and considering all the night and the first minutes (00:01) of the new day, the nature have a break, and at this time, the nature activity is more effective. Then, the improving of the air quality is higher. When the human activity begins, the air quality starts to going down. At late hours, the concentration of contamination increases and is reduced until the human activity is reduced. Therefore, the cycle ends and begins again.

Once the prototypes have been built, the following task is to evaluate the ***first policy***. The policy aims to measure the separation between *Mc*'s. The following table shows the results in the evaluation of the prototypes when γ=0.1. The prototypes evaluated have been selected having in mind the evaluation of γ, and the comments of the environmental experts.

Table 19. Distance measures between the Mc's obtained for γ=0.1

	Mc 1	Mc 2	Mc 3	Mc 4
Mc 1	0	30.967	30.666	53.225
Mc 2	30.967	0	21.120	26.185
Mc 3	30.666	21.120	0	37.394
Mc 4	53.225	26.185	37.394	0

The normalization of the data between 0 and 1 is an usual procedure in data mining. However, in this special case, data are computed and represented on its natural values, with the aim of following the International and Mexican norms, as it is depicted in table 19.

Table 19 shows the distance evaluation between prototypes. The table shows blocks of different colors. For the evaluation, we concentrate in the cases situated in the white blocks. According to the results depicted in table, all prototypes are separated for a good distance between them. This is an indication that the prototypes are well structured, and the use of the stochastic steps works fine. The prototypes never overlap.

The second evaluation is the implementation of the ***second policy***. This is done through the implementation of the formulas 11, 12, 13 and 14 for the prototypes. Previously, it has been introduced the aim and meaning of the formulas, which is to evaluate the quality of the learning (or the building of new prototypes), starting with an empty case-base. The table 20 depicts the results of the implementation of each formula for the prototypes.

Table 20. Results of the different formulas assessment

	Mc '1	***Mc*** '2	***Mc*** '3	***Mc*** '4	***Mc*** '5
Formula 11	41,93658	24,88025	21,41534	61,14482	11,46441
Formula 12	840,1143	1320,691	1802,095	2336,38	2860,231
Formula 13	19794,06	7588,476	6317,526	21645,27	149,0373
Formula 14	23,56115	5,745838	3,505657	9,264447	0,052107

Formula 11 computes an average while formula 13 computes the sum of distances between the cases and the prototype. These two formulas help to *view the compactness of the sub-library*. Especially in formula 13, it is possible to assess how far the cases to its prototype are. Here, a

reduced value is desired, because it will be an indicator of a good compact sub-library (hard sub-library). A high value indicates that the cases are far from its prototype. Then, the class/cluster is not too compact. Thus, it is soft. Taking into account results, the *prototypes one and four* are the prototypes that could be soft, because both have the higher values, and where prototype 1 has more than *Mc* 4. Prototype 4 has the higher separation of its cases. These prototypes represents the first and last hours of the day, where the *human activity it is reduced*. *Prototypes two and three* are the harder prototypes, and are the prototypes which represents when the human activity is high. The harder prototypes are the cases more nearest to its prototype.

The evaluation of the *magnitude of each prototype* is computed, and the results are depicted in table 20 as results of formula 12. According to the experts, in the morning the air quality is good, but with the human activity, the air quality is going worst. This behavior is expected in data. In the evaluation of the prototype, could be seen how the values for each prototype increases. The increase of magnitude shows us that the pollution has been increased.

Finally, in formula 14, a similarity of the prototype and the evaluation of distances are computed. This similarity gives an idea of *how representative is the prototype of all the cases that are stored/summarized on it*. To have a full evaluation of the results the following table 21 shows the evaluation of the dispersion of the data in the prototypes.

Table 21. Standard Deviation of Prototypes

Mc'1	93.83733
Mc'2	34.00873
Mc'3	29.97352
Mc'4	107.1802
Mc'5	158.8435

Table 20 in formula 11, 13 and 14 shows a pattern as result of the iplementation of the stochastic method. Table 21 shows the standard deviation evaluation, which depicts the hard prototypes and soft prototypes. In hard prototypes, the cases tend to be very close to the ***Mc***, which is the situation of ***Mc'2*** and ***Mc'3***. Others prototypes are spread out over a large range of values. This behavior is depicted in both tables.

Table 20 and 21 complements the evaluation of the prototypes, but especially in the evaluation of prototype 5 the result is low, the magnitude is high, and the distance is low, but the number of cases is reduced. In this case, is essential the evaluation of the dispersion results that is the higher value and magnitude is higher too. It is positively surprising that the pattern of air quality in the city matches with the citizen daily behavior.

6.5 Testing incrementally the whole DACL Framework

An exhaustive experimentation combining all the main contributions of this thesis work: Dynamic Adaptive Case Library (DACL), Incremental NIAR *k*-d Tree building, PME technique, Stochastic Learning and Relevant Case learning policies, have been carried out. This experiments provided the confirmation that the proposed DACL framework is especially suitable to cope with *incremental data streams*.

It is not easy to cope with an unsupervised database, where in an incremental way, a lot of cases are being generated, and must be processed by the CBR system. This way, the system has not so much cases in its Case Library like in a non-incremental processing scenario. This means that it is pretty more difficult to achieve good accuracy percentages, because at the beginning of the processing, normally the precision will be lower than when working in a non-incremental scenario.

Fortunately, the experimentation done has outlined that the DACL approach is able to satisfactorily cope with unsupervised and incremental scenario. As showed later the DACL approach has been able to detect and construct the same number of meta-cases (prototypes)

6.5.1 Experimental Settings

The same 10 databases (Abalone, Balance, Car evaluation, Ecoli, Glass, Ionosphere, Iris, Letter, Pima and Waveform), from the UCI repository, used in section 6.3 were also used here. In table 4 there is the description of main features of all these databases.

The experimental setting was done under the following characteristics:

- The above ten databases were tested

- As a baseline, the Flat Case Library was considered which gives the upper bound of the accuracy.

- Twelve different strategies were tested. These combinations are the result of the crossing of 3 Meta-case Selection strategies (VirtualMcSelection, RealMcSelection and VirtualRmaxMcSelection) , two incremental maintenance strategies for the NIAR k-d Tree (IncMaintTree and IncMaintPercTree) and 2 strategies for the Learning of cases (AllCaseLearning and RelCaseLearning):

  - S1: VirtualMcSelection + IncMaintTree + AllCaseLearning
  - S2: VirtualMcSelection + IncMaintTree + RelCaseLearning
  - S3: VirtualMcSelection + IncMaintPercTree + AllCaseLearning
  - S4: VirtualMcSelection + IncMaintPercTree + RelCaseLearning
  - S5: RealMcSelection + IncMaintTree + AllCaseLearning
  - S6: RealMcSelection + IncMaintTree + RelCaseLearning
  - S7: RealMcSelection + IncMaintPercTree + AllCaseLearning
  - S8: RealMcSelection + IncMaintPercTree + RelCaseLearning
  - S9: VirtualRmaxMcSelection + IncMaintTree + AllCaseLearning
  - S10: VirtualRmaxMcSelection + IncMaintTree + RelCaseLearning
  - S11: VirtualRmaxMcSelection + IncMaintPercTree + AllCaseLearning
  - S12: VirtualRmaxMcSelection + IncMaintPercTree + RelCaseLearning

- For each database and for each strategy, 10 execution runs were done *sampling randomly the cases* to get different ordering of the cases. In addition, one more run was done with the *original ordering* of each database.

- For each execution and for each database several statistics were computed: the average accuracy (in percentage); the incremental evolution of the accuracy (accumulated percentage), the average time retrieval (time in µs), the evolution of the incremental time retrieval (accumulated time in µs); The detection of meta-cases-prototypes; the incremental evolution of the number of cases of the whole Case Library; the incremental evolution of the number of cases of each discovered Meta-case/prototype.

6.5.2 Experimental Results

Regarding the discovering of Meta-cases and protoypes, all the databases have been processed and the different meta-cases (prototypes) were found to correspond accurately with the real hidden class labels. In the table 22, the list of the used databases is shown along with some experimentation details, like the number of classes and instances of each database and the number of meta-cases built (discovered when using different values of the β parameter (ranging from 0.74 to 5.9).

Table 22. List of tested databases with details on the discovered Meta-cases

Number of Metacases built in the dynamic learning of prototypes (DACL)														
DB	**#Classes**	**#Inst**	**β=4.55**	**β=0.9**	**β=3.45**	**β=0.425**	**β=5.8**	**β=5.9**	**β=5**	**β=0.74**	**β=1.1**			
Pima	2	768		Mc1=753 Mc2=15										
Ionosphere	2	351			Mc1=37 Mc2=314									
Iris	3	150				Mc1=50 Mc2=68 Mc3=32								
Waveform	3	5000					Mc1=1411 Mc2=686 Mc3=1675 Mc4=1228							
Balance	3	625						Mc1=298 Mc2=304 Mc3=23						
Car	4	1728	Mc1=616 Mc2=598 Mc3=285 Mc4=229											
Glass	7	214							Mc1=152 Mc2=17 Mc3=1 Mc4=2 Mc5=11 Mc6=30					
Ecoli	8	336								Mc1=144 Mc2=10 Mc3=102 Mc4=1 Mc5=21 Mc6=46 Mc7=2 Mc8=10				
Abalone	29	4177									Mc1=132 Mc2=38 Mc3=4 Mc4=63 Mc5=173 Mc6=16 Mc7=9 Mc8=26	Mc9=31 Mc10=100 Mc11=101 Mc12=33 Mc13=73 Mc14=85 Mc15=8 Mc16=78	Mc17=72 Mc18=24 Mc19=72 Mc20=228 Mc21=64 Mc22=42 Mc23=43 Mc24=20	Mc25=91 Mc26=82 Mc27=55 Mc28=1 Mc29=139 Mc30=151 Mc31=1 Mc32=136 1 Mc33=761

In the following two sections more details about the performance of the proposed methodology when tested on the Iris and Balance database correspondingly, can be found. These databases were selected in order not put all resulting tables for all databases. Anyway, the results on all databases were very equivalent.

6.5.2.1 Evaluating the Accuracy in Iris and Balance databases

After running 10 different tests on each database the average precisions values found are reported in the following table (table 23).

It is important to mention that the algorithm's learning process updates its learning base each time a new meta-case appears, which has as a result that the initial precision value is lower and grows as more meta-cases are identified.

Table 23. Mean precision values for Iris and Balance databases

	#CL	#MCs	#NCL	Precision	β
iris	#CL1=50 #CL2=68 #CL3=32	3	3	87%	0.425
Balance	#CL1=298 #CL2=304 #CL3=23	3	3	68%	5.9

6.5.2.2 Analyzing the Iris database

In figures 25 and 26, the results of the experiments run on the Iris database are presented graphically, with the setting of $\beta = 0.425$. These figures show the evolution of the detection of the Meta-Cases, for several random executions.

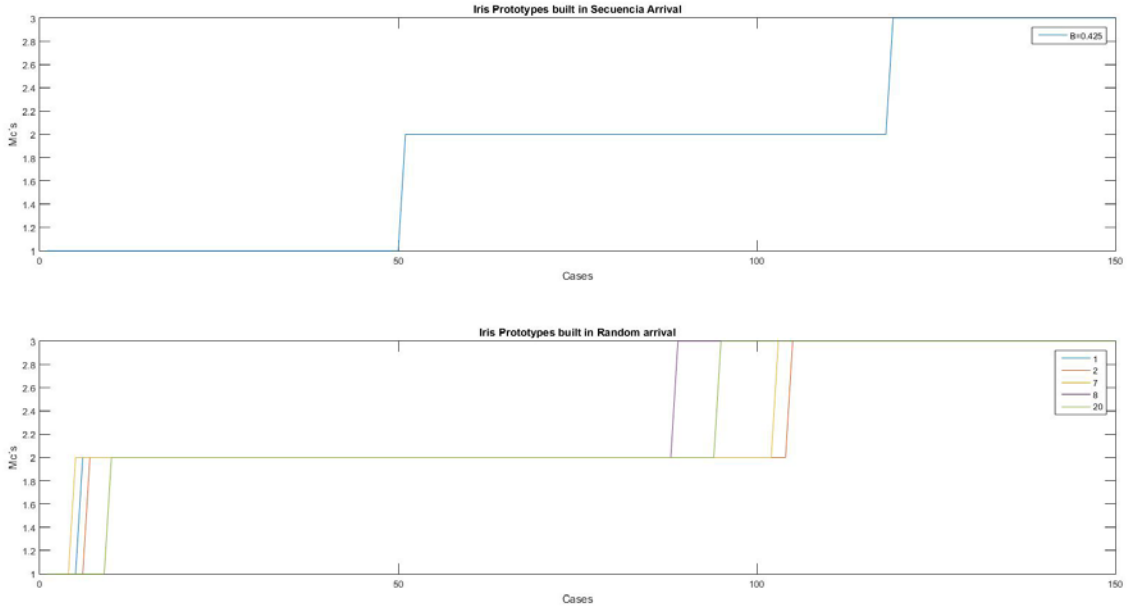

Fig. 25. Iris meta-cases for sequential and random (1, 2, 7, 8, 20) arrival

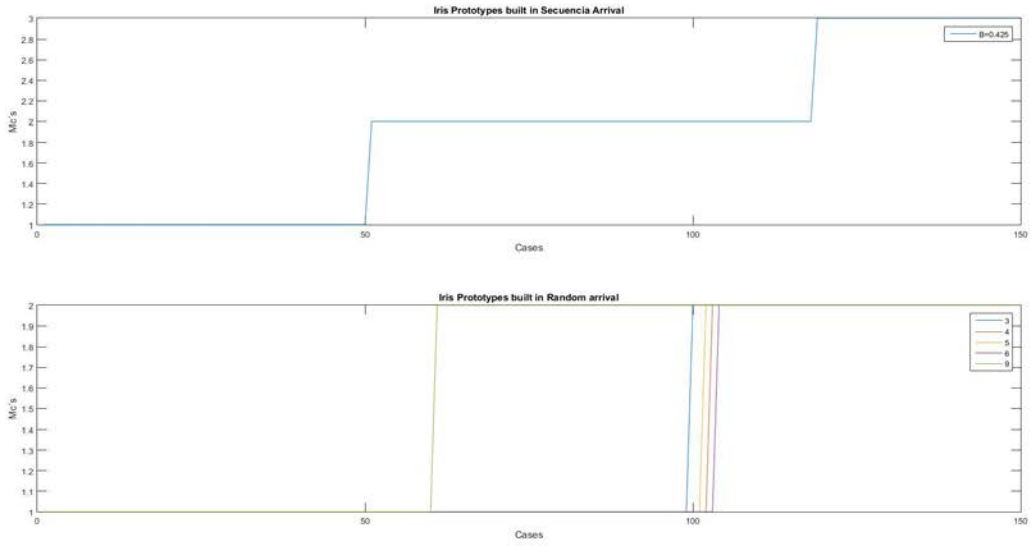

Fig. 26. Iris meta-cases for sequential and random (3, 4, 5, 6, 9) arrival

6.5.2.3 Analyzing the Balance Database

In figure 27, 28 and 29, the results of the experiments run on the Balance database are presented graphically, with the setting of $\beta = 0.61$. These figures show the evolution of the detection of the Meta-Cases, for several random executions.

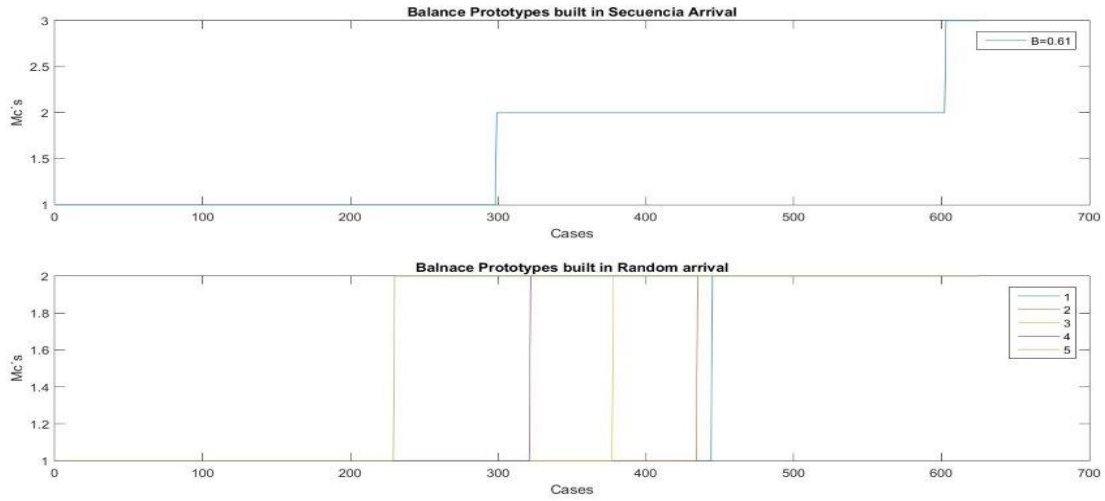

Fig. 27. Balance meta-cases for sequential and random (1-5) arrival

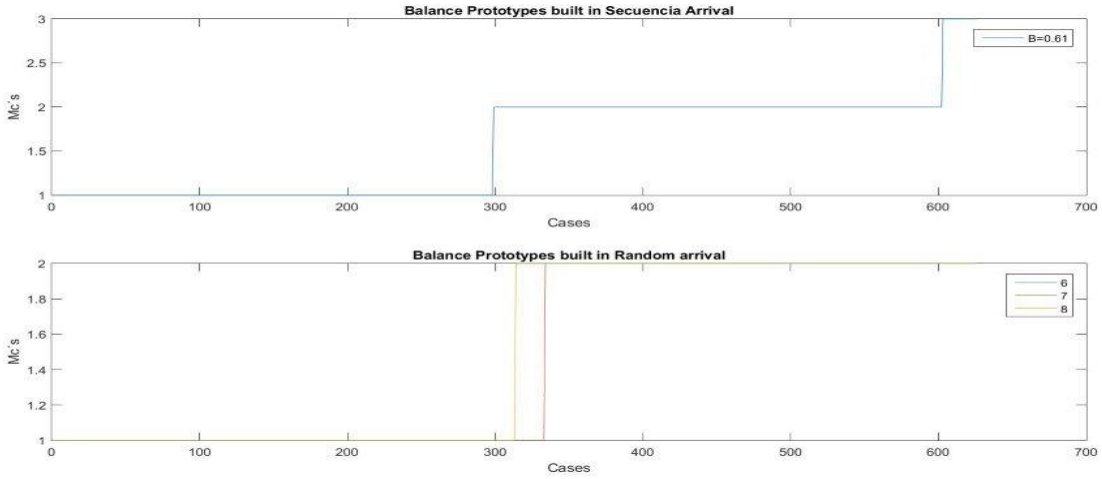

Fig. 28. Balance meta-cases for sequential and random (6, 7, 8) arrival

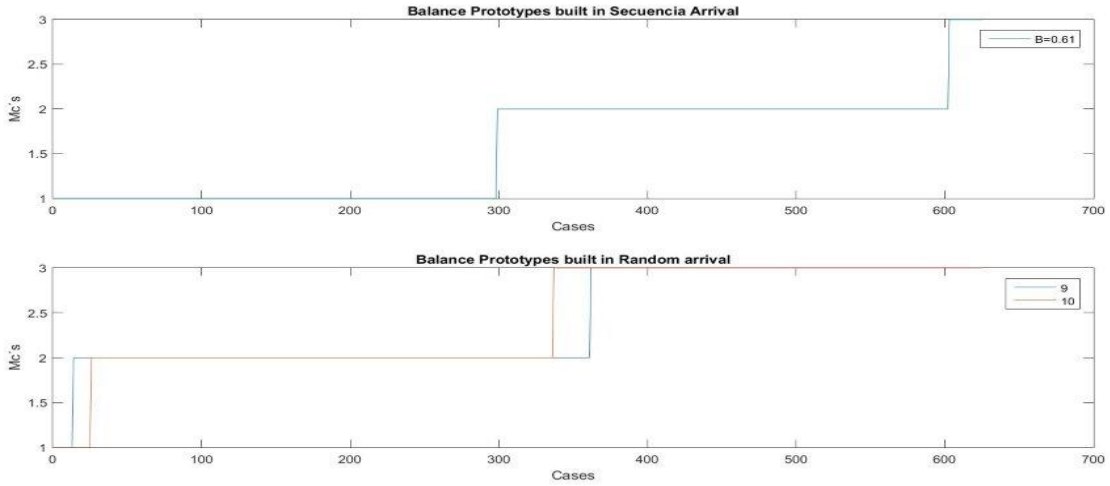

Fig. 29. Balance meta-cases for sequential and random (9, 10) arrival

6.5.3 Discussion of Results

The results in the 10 databases tested showed a good time performance and accuracy average on several execution runs, even though the case library was *incrementally populated*, they achieved good accuracy values, just a little bit worse than compared with the non-incremental processing of the cases, as it was tested in section 6.3. From the experimentation done we can conclude that the DACL framework is a promising approach to cope incrementally with data streams.

This approach has been able to *discover the Meta-cases* (prototypes) in a very good way, and the number of protoytpes is equal to the existing number of the "hidden" class labels in the databases.

Moreover, the *average final precision* is just a little bit worse than the average precision obtained with the same DACL approach, but just using the usual non-incremental way. In such non-incremental scenarios, the Case Library is seeded up with some training set of cases, and tested with a testing set, and of course, it should have better accuracy results, because at the beginning, this configuration has many more cases in the Case Library than the corresponding incremental configuration.

From the experimentation carried out, it seems that we can conclude that the DACL approach is a promising proposal for facing unsupervised domains (data streams), and in general, it improves the CBR system performance.

7 Conclusions and Future Work

7.1 Conclusions

An efficient and competent CBR system for a continuous domain requires techniques that continuously improve its efficiency and competence. The big amount of data generated in this kind of systems needs to be carefully managed to avoid an information overload. The proposal of the DACL Framework and the proposed indexing schemes (AvKd Tree, NIAR *k*-d tree) and exploration technique (Partial Matching Exploration) provides a solution to keep efficient and competent a continuous domain.

DACL Framework is focused on the retrieval and retain CBR steps. DACL Framework stores the information in sub-libraries. The sub-library concept is built by the prototype of cases (meta-case), which is the representation of all the cases stored in the corresponding sub-library (cluster), and a hierarchical indexing strategies (discriminant tree, *k*-d tree, etc.) to organize the information and to improve the retrieval. For *supervised domains*, DACL framework has been evaluated using databases from UCI Machine Learning Repository. The result shows that DACL, with proposed indexation and exploration technique, improves the computational effort spent and the competence of the CBR system.

The aim of the DACL framework is to dynamically adapt itself to the changes of the domain, maintaining the efficiency, both in time and size, and the competence of the system as good as possible. Until now, the evaluation of retain and retrieve steps, from a computational time efficiency point of view, have been done. The results are promising.

The use of a hierarchical indexation scheme substantially reduces the number of cases for which similarity computation must be done. The seminal work in the indexation of data to improve search processes was formulated for database queries applying a *k*-d tree (Bentley 1975, Friedman et al. 1977).

Several contributions reviewed in previous chapters have shown that the standard *k*-d tree approach was improved in different ways. In our

research work, another proposal to improve the standard *k*-d tree approach was formulated in (Orduña and Sànchez-Marrè, 2013): *the NIAR k-d tree approach*. In this thesis has been proposed both the use of a *partial matching exploration technique* and a *Multiple Case Library* (MCL) structure to improve both the time and the accuracy in the case retrieval task through a hierarchical structure (*k*-d tree) or through a set of several hierarchical structures.

These proposed approaches have been combined and compared against well-known approaches in the literature like the standard *k*-d trees, NIAR *k*-d trees, the hyperball with bounds exploration techniques, etc. Twelve strategies have been defined and tested (as detailed in chapters 5 and 6).

All strategies have been subjected to an experimental evaluation aiming to find the faster and more accurate retrieving strategy. The *time efficiency* and the success in the assignment of the right class labels in the databases, as a measure of the *competence* of the system, were analyzed.

From the experimental testing, it can be drawn the use of a *Multiple Case Library using NIAR k-d trees* in its second layer structure, and *using the partial matching exploration technique* is the best strategy among the twelve tested. This strategy reaches on average a 68.41 % of accuracy, only an 8.4% less accuracy than the baseline strategy (linear search in a flat case library). Regarding the retrieval time, this strategy on average spends 57 µs, while the baseline strategy spends 17759 µs for one case retrieval (311 times faster).

This approach is faster than other retrieving strategies, because the computational time of the partial matching exploration technique depends on the number of indexing attributes (*k*) and not on the number of cases (*n*) as other approaches do, which can be very large depending on the size of the databases. In addition, the use of a Multiple Case Library implies that the indexing tree structures used are smaller, as they contain much less cases than when using only one case library. In addition, the accuracy of the retrieval process is also improved because the different case libraries of the MCL let to model in a better way (more accurate) the different prototype of cases (meta-cases) which can be

found in a domain/database. Not all the cases in the same domain/database are similar enough to be characterized and indexed in the same way. With a set of case libraries, the different kind of cases can be modelled in a different and suitable way.

In a complementary research work, we have been testing the Dynamic and Adaptive version of a MCL, a DACL framework (as explained in chapter 3), with several stochastic policies to learn dynamically the different meta-cases to manage the continuous domains (data streams) (Orduña et al., 2015a). A stochastic learning method has been introduced. The aim of the proposal is to learn the new cases and store it in the structure of the similar enough prototype. If current prototypes do not accomplish with the requirements of the method to learn the new case, then a new prototype have to be computed. This is done considering the new case such as the new prototype. The data have been acquired from different environmental sensors. Thirteen attributes has been used to evaluate the air quality of Cd. Obregon Sonora during a period of one year. In the opinion of the environmental experts and according with the results, the method here depicted can be considered trustworthy for organize the cases of the air pollution. The stochastic field here boarded follows the requirement of the statically stochastic methods. Especially in first and second moment, those are the core of this proposal. With this Stochastic method DACL Framework is able to learn new cases and build new prototypes. The learning of new cases is executed according to the most similar prototype. And finally, the building of new prototypes is done according to the proposed method.

All the Dynamic Adaptive case Library framework has been tested with simulated unsupervised domains where the cases were incrementally processed, and all the DACL structures were incrementally created and updated.

7.2 Future Work

During the research work of this thesis of the Meta-cases other ideas and research lines arose. Some of them are:

- The proposal of new policies for building Meta-cases.

- The implementation of entropy methods in building the prototypes.

- Building a CBR system, where the system gain capabilities through deeper introspective reasoning tasks and Meta-Cognition. With this capabilities the CBR system could implement any strategy learned, even the capability of evaluate different strategies and decide the best one for each sub-library. This should be triggered in the learning of new cases or Meta-cases.

- In a future work, discrimination trees will be used at the second layer of a MCL to test whether they can be more accurate. The idea is to decide the discriminating order of the attributes using *unsupervised feature weighting* techniques (Núñez & Sànchez-Marrè, 2004). As higher weights are obtained, a higher position in the discrimination list used for the creation of the discrimination tree is obtained by the attribute.

- Other future research line is to test *different similarity measures*, especially taking into account the *relevance of the different features* with the assignment of weights. In the current experimentation, the Euclidean measure with equal weights has been used, but as showed in previous studies (Núñez et al., 2004), other similarity measures could help to improve the competence accuracy of the CBR system, due to a better similarity assessment between the query case and the cases in the Case Library.

There is too much work to do in this field, and our work is just only a small piece for the improvement of Case-Based Reasoning system performance.

Appendix A

Related Publications

- **Journals**

✓ Orduña Cabrera, Fernando, Sànchez-Marrè Miquel, Ramírez Treviño Alberto and Villa Ibarra Martin (2015b). A Stochastic Learning Approach for Building Prototypes in Air Quality Evaluation Using a Dynamic Adaptive Case Library. Submitted to Journal of Engineering Applications of Artificial Intelligence, Elsevier Science, September 2015.

✓ Orduña Cabrera, Fernando and Sànchez-Marrè Miquel (2015a). Embedding k-d Trees and Exploration Techniques within a Multiple Case Library to Improve Case Retrieval. Submitted to Journal of Knowledge-Based Systems, Elsevier Science, August 2015. In revision process.

- **Conferences / Book Chapters**

✓ Orduña Cabrera, F. and M. Sànchez-Marrè (2013). Using NIAR k-d Trees to Improve the Case-Based Reasoning Retrieval Step. *12th Mexican International Conference on Artificial Intelligence (MICAI 2013). Proceedings Part II. Springer Verlag, Lecture Notes in Computer Science LNAI*, vol. 8266. Advances in Soft Computing and Its Applications, pp. 314-325, 2013. ISBN 978-3-642-45110-2. DOI 10.1007/978-3-642-45111-9_28.

✓ Orduña Cabrera, F. and Sànchez-Marrè, M. (2009). Dynamic Adaptive Case Library for Continuous Domains. In *Proc. of 12th International Conference of the Catalan Association of Artificial Intelligence (CCIA'2009). Frontiers in Artificial Intelligence and Applications Series*, Vol. 202, pp. 157-166, 2009. ISBN 978-1-50750-061-2.

- **Technical Reports**

✓ Orduña Cabrera, F. and Sànchez-Marrè M (2008c). Case base maintenance: Terms and directions. Technical Report LSI-08-21-R, Universitat Politècnica de Catalunya · BarcelonaTech.

Other Publications

- **Conferences / Book Chapters**

✓ Orduña Cabrera, F., M. Sànchez-Marrè, M., J.M. García Gorrostieta and S. González López (2008d). Ergonomic Advice through Case-Based Reasoning to Avoid Dangerous Positions Adopted Using the Computer. Artificial Intelligence Research and Development. In Proc. of 11th International Conference of the Catalan Association of Artificial Intelligence (CCIA'2008). Frontiers in Artificial Intelligence and Applications Series, Vol. 184, pp. 186-194, 2008. ISBN 978-1-58603-925-7

✓ F. Orduña Cabrera and M. Sànchez-Marrè (2007). Related Topics for an Embodied Procedural Reasoning Architecture. In Proc. of 1st International Conference on Industrial, Mechatronics and Manufacturing Engineering (CIMM 2007), pp. 145-150. Instituto de Ingeniería y Tecnología, Chihuahua. Ciudad Juárez, Chhiuahua, México. 3-5 de Octubre de 2007.

- **Technical Reports**

✓ Orduña Cabrera, F. and Sànchez-Marrè M (2008b). Bioinformatics: a promising field for case-based reasoning. Technical Report LSI-08-20-R, Universitat Politècnica de Catalunya · BarcelonaTech.

✓ Orduña Cabrera, F. and Sànchez-Marrè M (2008a). An approach for an architecture to embodied procedural reasoning. Technical Report LSI-08-1-R, Universitat Politècnica de Catalunya · BarcelonaTech.

References

1. Aamodt A. and E. Plaza (1994). Case-based reasoning: fundamental issues, methodological variations and system approaches. AI Communications 7(1):39-59.
2. Aha, D. W. (1991). Case-Based Learning Algorithms. In Ray Bareiss, editor. Proceedings: Case-Based Reasoning Workshop. Morgan Kaufman Publishers.
3. Akbari Javad Torkestani and Mohammad Reza Meybodi (2011). Learning automata-based algorithms for solving stochastic minimum spanning tree problem. journal Applied Soft Computing. volume 11, number 6, pages 4064-4077.
4. Althoff, K-D. and Wess, S. (1992) Case-Based Reasoning and Expert System Development. In Franz Schmalhofer, Gerhard Strube and Thomas Wetter, editors. Contemporary Knowledge Engineering and Cognition. Springer-Verlag, Berlin.
5. Anderson L. Michael and Oates Tim (2007), A Review of recent Reseach in Meta-reasoning and Meta-learning. AI Magazine, AAAI, Volume 28, Number 1.
6. Arcos Josep Lluís, Oguz Mulayim and David Lake, (2008) Using Introspective Reasoning to Improve CBR System Performance. Workshop AAAI 2008, Meta-reasoning: Thinking about thinking, Chicago Illinois.
7. Arya, Sunil and David M. Mount (1993). Algorithms for Fast Vector Quantization. Proc. of DCC '93: Data Compression Conference, pp. 381--390, IEEE Press.
8. Arshadi, N. & Jurisica, I. (2004). Advances in Case-Based Reasoning, chap. Maintaining Case-Based Reasoning Systems: A Machine Learning Approach, 17–31. springerlink.
9. Arshadi, N. & Jurisica, I. (2005). Data mining for case-based reasoning in high-dimensional biological domains. IEEE Trans. on Knowl. and Data Eng., 17, 1127–1137.
10. Athanasios Papoulis (1984). Probability, Random Variables, and Stochastic Processes, Second Edition. McGraw-Hill.
11. Aggarwal, C.C. (2007). Data Stream: Models and Algorithms. Springer, edited by Aggarwal C. Cahru.
12. Bentley, Jon Louis (1975). Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9): 509-517.
13. Bergmann, R., and Tartakovski, A, (2009) Improving KD-Tree Based Retrieval for Attribute Dependent Generalized Cases. In Proceedings of the Twenty-Second International FLAIRS Conference, Association for the Advancement of Artificial Intelligence. Pp. 319-324.

14. Bonissone, P. & de M`antaras, R.L. (1998). ”Case-Based Reasoning”. In: Handbook of Fuzzy Computation. Institute of Physics Publishing, Bristol and Philadelphia (E. H. Ruspini, P. P. Bonissone, Wiltold Pedrycz. eds).
15. Broder, A.J. (1990) Strategies for Efficient incremental Nearest Neighbor Search. Pattern Recognition, 23:171-178.
16. Burke, E.K., MacCarthy, B.L., Petrovic, S. & Qu, R. (2001). Casebased reasoning in course timetabling: An attribute graph approach. In ICCBR, 90–104.
17. Carbonell, J. (1986). Machine Learning: An Artificial Intelligence Approach, chap. Derivational analogy: A theory of reconstructive problem solving and ex¬pertise acquisition. Morgan Kaufman Publishers, Michalski, R. and Carbonell,
18. Chaimontree, Santhana and Atkinson, Katie and Coenen, Frans (2010). Clustering in a Multi-Agent Data Mining Environment.Agents and Data Mining Interaction, Lecture Notes in Computer Science, volume 5980, pages 103-114.
19. Charniak, E., McDermott, D. Introduction to Artificial Intelligence. Addision-Wesley, 1985.
20. Cox T. Michael (December 2005), Metacognition in Computation: A selected research review. Artificial Intelligence, Volume 169, Issue 2, Pages 104–14.
21. Cox T. Michael and Raja Anita (2008), Meta-reasoning: A Manifesto, Workshop AAAI:2008, Meta-reasoning: Thinking about thinking, Chicago Illinois.
22. Doyle Dónal, Pádraig Cunningham, Derek Bridge, Yusof Rahman, (2004). “Explanation oriented retrieval”, ECCBR 2004, LNAI 3155, pp. 157–168.
23. Ferguson Alex, Derek Bridge (2000), “Partial orders and indifference relations: being purposefully vague in case-based retrieval”, EWCBR 2000, LNAI 1898, pp. 74–85.
24. Finestrali, Giulio and Muñoz-Avila, Héctor (2013). Case-Based Learning of Applicability Conditions for Stochastic Explanations. Case-Based Reasoning Research and Development, volume 7969, pages 89-103.
25. Fisher, D (1987). Cobweb: Knowledge Acquisition via Conceptual Clustering. Machine Learning, 2:139-172.
26. Fornells A., E. Armengol and E. Golobardes (2008). Retrieval Based on Self-explicative Memories. Proc. of European Conference on Case-Based Reasoning (ECCBR 2008), pp. 210-224. Springer-Verlag.
27. Fox Susan and David Leake (1994), Using Introspective Reasoning to Guide Index Refinement in Case-Based Reasoning. Proceedings of the Sextennth Annual Conference of the Cognitive Science Society.

28. Fox Susan and David Leake (2001), Introspective reasoning for index refinement in case-based reasoning, Journal of Experimental and Theorical Artificial Intelligence, pp 63-88, Vol. 13.
29. Frank, A. and Asuncion, A. (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
30. Friedman, Jerome H. and Bentley, Jon Louis and Finkel, Raphael Ari, (1977). "An Algorithm for Finding Best Matches in Logarithmic Expected Time," ACM Trans. Math. Software, 3:209--226, September.
31. Gentner, D. and Forbus, K. D. (1991). MAC/FAC: a Model of Similarity-Based Retrieval. In Proceedings of the 13th Annual Conference of the Cognitive Science Society, pp.: 504-509.
32. Gower, J. (1971). A General coefficient of similarity and some of its properties. Biometrics, 27:857–887.
33. Hammond, K.J. (1989). Case-based planning: viewing planning as a memory task. Academic Press Professional, Inc., San Diego, CA, USA.
34. Haris S. and R. Slobodan (2005). Autonomous Creation of New Situation Cases in Structured Continuous Domains. Springer-Verlag, pp. 537—551.
35. Holyoak, K.J. and Koh K. (1986). Analogical Problem Solving: Effects of Surface and Structural Similarity in Analogical Transfer. In Midwestern Psychological Association, editor, Proceedings of the Conference of the Midwestern Psychological Association.
36. Iglezakis, I., Reinartz, T. & Roth-Berghofer, T.R. (2004). Advances in Case-Based Reasoning, chap. Maintenance Memories: Beyond Concepts and Techniques for Case Base Maintenance, 227–241. springerlink.
37. Joh D.(1997). CBR in a Changing Environment. Proc. of the 2nd Int. Conference on Case-Based Reasoning (ICCBR'97). Springer-Verlag, pp. 53—62.
38. Keane M. T. and B. Smyth (1995). Remembering to Forget: A Competence-Preserving Case Deletion Policy for Case-based Reasoning systems. Procc. of IJCAI 1995, Morgan Kaufmann, 377-382.
39. Kolodner, J.L (1993). Case-Based Reasoning, Morgan Kaufmann.
40. Kolodner, J.L. (1980) Retrieval and Organizational Strategies in Conceptual Memory. Ph.D. Thesis, Yale University.
41. Kolodner, J.L., Simnpson, R.L. and Sycara, K. (1985). AProcess Model of Case-Based Reasoning in Problem Solving. In IJCAI, editor, Procc. of IJCAI 1985, pp. 284-290. Morgan Kaufmann Publishers, Los Angeles, California, USA.
42. Kruusmaa M. (2003). Global Navigation in Dynamic Environments Using Case-Based Reasoning. Autonomous Robots 14, pp. 71--91.
43. Leake B. David, Kinley Andrew and Wilson David (1995), Learning to improve Case Adaption by Introspective Reasoning and CBR, In Proceed-

ings of the First International Conference on Case-Based Reasoning, pp 229-240, Springer-Verlag.
44. Leake David and Wilson Mark (2008), Extending Introspective Learning from Self-Models, Workshop AAAI: 2008, Meta-reasoning: Thinking about thinking, pp. 143-146, Chicago Illinois.
45. Leake, D.B. & Wilson, D.C. (1998). Categorizing case-base maintenance: Di-mensions and directions. In Advances in Case-Based Reasoning:Proceedings of the Fourth European Workshop on Case-Based Reasoning, 196–207, Springer-Verlag.
46. Leake, D.B. & Wilson, D.C. (2000). Remembering why to remember: Performance-guided case-base maintenance. In In Proceedings of the Fifth Eu-ropean Workshop on Case-Based Reasoning, 161–172, Springer Verlag.
47. Lenz, M. & Burkhard, H. (1997). Cbr for document retrieval. Second International Conference on Case-Based Reasoning, Case-Based Reasoning Research and Development ICCBR-97 , 1266/1997.
48. López, B. (2013). Case-Based Reasoning: a Concise Introduction. Morgan & Claypool, 2013.
49. López de Mántaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, and M. T. Cox, K. Forbus, M. Keane, A. Aamodt and I. Watson. Retrieval, (2005). reuse, revision and retention in case-based reasoning. The Knowledge Engineering Review 20(3), 215-244.
50. López-Rubio Ezequiel (year 2011). Stochastic approximation learning for mixtures of multivariate elliptical distributions, journal Neurocomputing, volume 74, number 17, pages 2972-2984.
51. Martín F. J. and E. Plaza (2004). Ceaseless Case-based Reasoning. In Procc. of 7th European Conference on Case-Based Reasoning (ECCBR'2004), pages 287-301, LNAI-3155, Madrid, Spain. September.
52. Mccarthy John (1979). Ascribing mental qualities to machines, In Philosophical Perspectives in Artificial Intelligence, Humanities Press, pp. 161-195.
53. Meléndez J, J. Colomer and J. Ll. de la Rosa (2001). Expert Supervision Based on Cases. Proc. of 8th IEEE International Conference on Emerging Technologies and Factory Automation, Vol. 1, pp. 431—440.
54. Miyashita K. and K. Sycara (1995). Improving system performance in case-based iterative optimization through knowledge filtering. Procc. of IJCAI 1995, Morgan Kaufmann, pp. 371—376.
55. Munoz-Avila, H. (2001). Case-base maintenance by integrating case-index revision and case-retention policies in a derivational replay framework. Computational Intelligence, Blackwell Publishers, Inc., 17, 280–294.

56. N. Segata, E. Blanzieri, S. Delany, P. Cunningham, Noise reduction for instance- based learning with a local maximal margin approach, Journal of Intelligent Information Systems (2010), doi:10.1007/s10844-009-0101-z.
57. Ni, Z.W., Liu, Y., Li, F.G. & Yang, S.L. (2005). Case base maintenance based on outlier data mining. Proceedings of the Second International Conference on Machine Learning and Cybernetics, 5, 2861– 2864.
58. Núñez -Rocha, H.F. (2004a). Feature Weighting in Plain Case-Based Reasoning. Ph.D. thesis, Technical University of Catalonia, Software Department.
59. Núñez, H. and Sànchez-Marrè, M (2004b). Instance-based Learning Techniques of Unsupervised Feature Weighting do not perform so badly!. In Proceedings of 16th European Conference on Artificial Intelligence (ECAI'2004), pp. 102-106. IOS Press, València, Spain.
60. Orduña Cabrera, Fernando and Sànchez-Marrè Miquel (2015a). Embedding k-d Trees and Exploration Techniques within a Multiple Case Library to Improve Case Retrieval. Submitted to Journal of Knowledge-Based Systems, Elsevier Science, August 2015.
61. Orduña Cabrera, Fernando, Sànchez-Marrè Miquel, Ramírez Treviño Alberto and Villa Ibarra Martin (2015b). A Stochastic Learning Approach for Building Prototypes in Air Quality Evaluation Using a Dynamic Adaptive Case Library. Submitted to Journal of Advanced Engineering Informatics, Elsevier Science, September 2015.
62. Orduña Cabrera, F. and M. Sànchez-Marrè, (2013). Using NIAR k-d Trees to Improve the Case-Based Reasoning Retrieval Step. Proc. of 12th Mexican International Conference on Artificial Intelligence (MICAI 2013). Lecture Notes in Computer Science LNAI, vol. 8266, pp. 314-325.
63. Orduña Cabrera, F. and M. Sànchez-Marrè, M (2009). Dynamic Adaptive Case Library for Continuous Domains. In Proc. of 12th International Conference of the Catalan Association of Artificial Intelligence (CCIA'2009). Frontiers in Artificial Intelligence and Applications Series, Vol. 202, pp. 157-166.
64. Orduña Cabrera, F., Sànchez-Marrè M. (2008a). An approach for an architecture to embodied procedural reasoning. Research Report LSI-08-1-R. Universitat Politècnica de Catalunya.
65. Orduña Cabrera, F., Sànchez-Marrè M. (2008b). Bioinformatics: a promising field for case-based reasoning. Research Report LSI-08-20-R. Universitat Politècnica de Catalunya.
66. Orduña Cabrera, F., Sànchez-Marrè M. (2008c). Case base maintenance: Terms and directions. Research Report LSI-08-20-R. Universitat Politècnica de Catalunya.
67. Perner, P. (2006). Case-base maintenance by conceptual clustering of graphs. In Engineering Applications of Artificial Intelligence, 381–393.

68. Poch M., I. R. Roda, M. Sànchez-Marrè, J. Comas, U. Cortés and J. Lafuente (1999). A Multi-paradigm Decision Support System to improve Wastewater. Treatment Plant Operation. AAAI Workshop on Environmental Decision Support System and Artificial Intelligence, Technical Report WS-99-07, AAAI Press, pp. 68--73.
69. Portinale L. and S. Montani. (2005) Case-Based Representation and Retrieval with Time Dependent Features. Proc. of 6th International Conference on Case-Based Reasoning (ICCBR05), LNAI- 3620, Springer, pp. 353-367.
70. Portinale, L. & Torasso, P. (2001). Case-base maintenance in a multimodal reasoning system. Computational Intelligence, Blackwell Publishers, Inc., 17, 263–279.
71. Quinlan, J.R (1983). Learning efficient Classification procedures and Their Application to Chess Endgames. In R. Michalski, J. Carbonell and T. Mitchell, editors. Machine Learning. Tioga Press.
72. Ram A. and J. C. Santamaría (1997). Continuous Case-Based Reasoning. Artificial Intelligence 90, pp. 86-93.
73. Ram A., R. C.Arkin, K. Moorman and R. J. Clark (1997). Case-based reactive navigation: a method for on-line selection andadaptation of reactive robotic control parameters. Systems, Man, and Cybernetics Part B 3, pp. 376--394.
74. Rebagliati, Nicola and Rota Bulò, Samuel and Pelillo, Marcello (2013). Correlation Clustering with Stochastic Labellings. Similarity-Based Pattern Recognition. volume 7953, pages 120-133, year 2013.
75. Reinartz, T. & Iglezakis, I. (2000). On quality measures for case base maintenance. In In Proceedings of the 5th European Workshop on Case-Based Reasoning, 247–259, SpringerVerlag.
76. Reza Akbari, Koorush Ziarati: A rank based particle swarm optimization algorithm with dynamic adaptation. J. Computational Applied Mathematics 235(8): 2694-2714 (2011)
77. Richter, M.M. and Weber R.O. (2013). Case-Based Reasoning: a textbook. Springer-Verlag, 2013.
78. Rong Pan, Qian Yang, Lei Li, (2004). “Case retrieval using nonlinear feature-space transformation”, ECCBR 2004, LNAI 3155, pp. 361–374.
79. Salamó, M. & Golobardes, E. (2003). Hybrid deletion policies for case base maintenance. American Association for Artificial Intelligence.
80. Salamó, M. & Golobardes, E. (2004). Dynamic case base maintenance for a case-based reasoning system. In Advances in Artificial Intelligence, IBERAMIA, 93–103, American Association for Artificial Intelligence.
81. Salamó, M., M. López-Sánchez: Adaptive case-based reasoning using retention and forgetting strategies. Knowledge Based Systems. 24(2):230-247. 2011.

82. Sacerdoti, E. (1997). A structure for plans and behavior. In Proc. of Workshop on Case-Based Reasoning (DARPA).
83. Sanders, K. & Hendler, J. (1997). The case for graph-structured representations. Procc. Of the 2th International Conference on Case-Based Reasoning, 245–254.
84. Sànchez-Marrè M., U. Cortés, M. Martínez, J. Comas and I. Rodríguez-Roda (2005). An Approach for Temporal Case-Based Reasoning: Episode-Based Reasoning. Proc. of 6th International Conference on Case-Based Reasoning (ICCBR'2005). LNAI-3620, pp. 465-476.
85. Sànchez-Marrè M, U. Cortés, I. Rodríguez-Roda and M. Poch (2000) Using Meta-cases to Improve Accuracy in Hierarchical Case Retrieval. Computación y Sistemas (4), pp. 53.
86. Sànchez-Marrè M., U. Cortés, I. Rodríguez-Roda, and M. Poch (1999). Sustainable case learning for continuous domains. Environmental Modelling and Software 14(5):349-357, 1999.
87. Sànchez-Marrè, M., Roda, I. R., and Comas, Q. (1998). L'Eixample distance: a new similarity measure for case retrieval. 1st Catalan Conference on Artificial Intelligence (CCIA'98). ACIA Bulletin 14-15:246-253.
88. Sànchez, Miquel, Cortés, Ulises, Béjar, Javier, De Gràcia, Joan, Lafuente, Javier and Poch, M. (1997). Concept Formation in WWTP by Means of Classification Techniques: A Compared Study. Applied Intelligence.
89. Sasikumar, M. (1998). Case based reasoning, by janet kolodner and morgan kaufmann,. User Modeling and User-Adapted Interaction, 8, 157–160.
90. Schank, R. (1982). Dynamic memory: a theory of learning in computers and people. Cambridge University Press.
91. Shinn, H. (1988). Abstractional analogy: a model of analogical reasoning. In Proc. of Workshop on case-based reasoning (DARPA).
92. Someren Van, M., Surma, J. & Torasso, P. (1997). A utility-based approach to learning in a mixed case-base and model-based reasoning architecture. In ICCBR '97: Proceedings of the Second International Conference on Case-Based Reasoning Research and Development, 477–488, Springer-Verlag, London, UK.
93. Smyth, B. & Keane, M.T. (1995). Remembering to forget: A competence preserving case deletion policy for case-based reasoning systems. Proc. of 14th Int. Joint Conference on Artificial Intelligence (IJCAI 95), pp. 377–382, Morgan Kaufmann.
94. Smyth, B. & Mckenna, E. (2001). Competence models and the maintenance problem. Computational Intelligence, Blackwell Publishers, Inc., 17, 235–249.
95. Stottler, R.H., A. L. Henke and King, J.A (1989). Rapid retrieval Algorithms for Case-Based Reasoning. In Proceedings of the 11th International Conference on Artificial Intelligence IJCAI-89, pp. 233-237. Detroit, Michigan, USA.

96. Swee Chuan Tan, Kai Ming Ting, and Shyh Wei Teng. A Comparative Study of a Practical Stochastic Clustering Method with Traditional Methods. AI 2010, LNAI 6464, pp. 112–121, 2010.
97. Sycara, K. (1987). Finding creative solutions in adversarial impasses. In Pro-ceedings of the Ninth Annual Conference of the Cognitive Science Society.
98. Tan, Swee Chuan and Ting, KaiMing and Teng, ShyhWei (2011). A Comparative Study of a Practical Stochastic Clustering Method with Traditional Methods. AI 2010: Advances in Artificial Intelligence, volume 6464, pages 112-121.
99. Taylor Howard M. and Samuel Karlin. An introduction to Stochastic Modeling. 1998.
100. Todd K. Leen and Robert Friel and David Nielsen (2012). Approximating distributions in stochastic learning. Journal Neural Networks. Vol. 32, number 0, pages 219 - 228.
101. Urdiales C., E.J. Pérez, J. Vázquez-Salceda, M. Sànchez-Marrè and F. Sandoval (2006). A Purely Reactive Navigation Scheme for Dynamic Environments using Case-Based Reasoning. Autonomous Robots 21, pp. 65-78.
102. Veloso M., H. Muñoz-Avila and R. Bergmann (1996). Case-based planning: selected methods and systems. AI Communications 9(3)128-137.
103. V. Jalali, D. Leake:Adaptation-Guided Case Base Maintenance. AAAI 2014: 1875-1881.
104. Wang, Fei and Li, Ping and König, ArndChristian and Wan, Muting (2012). Improving clustering by learning a bi-stochastic data similarity matrix. Journal of Knowledge and Information Systems. volume 32, number 2, pages 351-382.
105. B. P. Welford (1962). Method for Calculating Corrected Sums of Squares and Products. Technometrics, Vol. 4, No. 3 (Aug., 1962), pp. 419-420.
106. Wess, S., Althoff, K-D. and Derwand, (1993). Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning, editors, Stefan Wess, Klaus-Dieter Althoff, and M. M. Richter, Proccedings of European Workshop on Case-Based Reasoning, EWCBR-93, pp. 167--181,Springer-Verlag.
107. Wilcoxon, Frank (1945). Individual comparisons by ranking methods. Biometrics Bulletin 1(6): 80–83.
108. Wilson, D.C. & Leake, D.B. (2001). Maintaining case-based reasoners: Dimensions and directions. Computational Intelligence, Blackwell Publishers, Inc., 17, 196–213.
109. Xi-Ren Cao (2009). Stochastic learning and optimization—A sensitivity-based approach, journal Annual Reviews in Control, volume 33, number 1, pages 11-24.

110. Yang, L.Z., Ha, M.H. & Wang;, X.Z. (2003). Diversity-based case base maintenance. Proceedings of the Second International Conference on Machine Learning and Cybernetics, 3, 1591–1596.
111. Yang, Q. & Wu, J. (2000). Keep it simple: A case-base maintenance policy based on clustering and information theory. In In Proc. of the Canadian AI Conference, 102–114.
112. Yang, Q. & Zhu, J. (2001). A case-addition policy for case-base maintenance. Computational Intelligence, Blackwell Publishers, Inc., 17, 250–262.
113. Yang, Q. & Wu, J. (2000). Keep it simple: A case-base maintenance policy based on clustering and information theory. In In Proc. of the Canadian AI Conference, 102–114.
114. Zhang, Kai and Collins, Emmanuel G., Jr. and Barbu, Adrian (2013). An Efficient Stochastic Clustering Auction for Heterogeneous Robotic Collaborative Teams. Journal of Intelligent and Robotic Systems, volume 72, number 3-4, pages 541-558.
115. Zhu, J. & Yang, Q. (1999). Remembering to add: Competencepreserving case addition policies for case base maintenance. In In Proceedings of the International Joint Conference in Artificial Intelligence (IJCAI , 234–241, Morgan Kaufmann.