Learning of Knowledge-Intensive Similarity Measures in Case-Based Reasoning

Vom Fachbereich Informatik
der Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

von

Diplom-Informatiker Armin Stahl

Prüfungskommission:

Vorsitzender:	Prof. Dr. Stefan Deßloch
Berichterstatter:	Prof. Dr. Michael M. Richter
	Prof. Dr. Ralph Bergmann
Dekan:	Prof. Dr. Hans Hagen

Tag der wissenschaftlichen Aussprache: 10. Oktober 2003

D 386

Acknowledgements

This work was developed during my time as a member of the "Artificial Intelligence – Knowledge-Based Systems" group at the University of Kaiserslautern, Germany. Many people influenced my work during this time and it is nearly impossible to name all of them. However, I would like to name a few persons I am particularly indebted to.

First of all, I would like to thank Prof. Dr. Michael M. Richter for enabling me to work in his group and for serving as a supervisor of this thesis. I will always remember the time in his group as a very comfortable, inspiring and fruitful period of my life. Further, I would like to thank Prof. Dr. Ralph Bergmann for introducing me into the field of Case-Based Reasoning, his great job as co-referee of this thesis and all his support, advice and very helpful discussions during my scientific work in the research group of Prof. Richter.

Of course, such a thesis is also influenced by a lot of other people. I am indebted to my former students, Philipp Dopichaj, Daniel Reidenbach, and Christian Bartelt for their support during the development of this thesis. Special thanks go to Thomas Gabel for his great job during the implementation of the concepts of this thesis, for the very fruitful discussions and for proofreading earlier versions of this thesis. Without his support this work would not have been possible in its final form.

I would also like to thank all my former colleagues for the harmonious working atmosphere in our group and many inspiring discussions. Especially I would like to thank my office neighbour Sascha Schmitt for all our interesting dialogs about many topics beyond Case-Based Reasoning and our exciting conference travels. Further thanks go to the members of the Case-Based Reasoning group, Ralph Bergmann, Wolfgang Wilke and Ivo Vollrath. During our common system adminstration job, Ivo teached me a lot of technical things that I don't want to miss today. Thanks go also to all other members of the group: Fawsy Bendeck, Eros Comunello, Harald Holz, Jochem Hüllen, Sigrid Goldmann, Boris Kötting, Dirk Krechel, Wolfgang Lenski, Rainer Maximini, Kerstin Maximini, and Martin Schaaf. I am also very indebted to Petra Homm, Edith Hüttel and Willi Klein for all their adminstrative support.

I owe a special thank to Sandra Ohla for proofreading the final version of this thesis. Last but not least, I would like to thank my parents, Christel and Gerhard Stahl, for all their support.

Kaiserslautern, August 2004

Abstract

Retrieving information from some set of data is a classical research topic in computer science. However, depending on the requirements of the particular application field, different approaches for realising retrieval functionality have been developed. On the one hand, traditional database management systems suppose users are beware of their information needs and that they are able to express these needs exactly by using a standardised query language. Here, the retrieval objective is the efficient selection of data records matching exactly the users' queries. On the other hand, in other application fields such well-formulated queries cannot be presumed, either because one has to deal with unstructured data (e.g. documents in the World Wide Web), or because users are not beware of their actual information needs.

One such application field is a problem-solving technique developed in Artificial Intelligence (AI), called *Case-Based Reasoning (CBR)*. Here, collected data records—called *cases*—represent information about problems solved in the past, and the basic idea is to reuse this knowledge when solving new problems. Therefore, cases *useful* for the current problem-solving episode have to be retrieved from all collected cases. The major problem of this retrieval task is that users usually are not able to express exact retrieval criteria describing which cases are useful and which are not. However, they should be able to describe their current problem situation. The selection of corresponding useful cases is then left to the CBR system which retrieves cases to be used for solving the problem by employing so-called *similarity measures*. Basically, a similarity measure represents a heuristics for estimating the a-priori unknown utility of a case. As typically for heuristics, the quality of a similarity measure can be improved by incorporating as much as possible knowledge about the particular application domain. However, the definition of such *knowledge-intensive similarity measures* leads to the well-known knowledge acquisition problem of AI. Unfortunately, the difficulty to acquire and formalise specific domain knowledge often prevents the usage of these actually very powerful kinds of similarity measures in commercial applications.

The objective of this thesis is the development of a framework and algorithms based on *Machine Learning* methods in order to facilitate the definition of knowledge-intensive similarity measures in CBR. The basic idea of this framework is to extract the mandatory domain knowledge form special training data that can be acquired more easily than the actual knowledge itself.

Contents

1. Introduction

In recent years *Case-Based Reasoning (CBR)* has become a very popular technique for developing knowledge-based systems. In some real world application domains it has even emerged to one of the commercially most successful approaches compared to other techniques developed in *Artificial Intelligence (AI)* research.

One of the major reasons for this success of CBR is a fundamentally new paradigm for realising knowledge-based systems. While the most traditional techniques aim at building systems able to solve all problems of the respective domain exactly, CBR does not focus on the completeness and exactness of the problem-solving process. These actually desirable properties have been relaxed in order to decrease the effort required for implementing knowledge-based systems. Hence, the correctness of a solution delivered by a CBR system usually cannot be ensured. However, a CBR system should provide at least a good approximation of the correct solution. Whether a particular CBR system is suitable for a particular real world application or not, of course, strongly depends on the quality of this approximation process, which itself strongly depends on the domain knowledge the CBR system is provided with. However, similar to other AI techniques, the acquisition and formalisation of domain knowledge is still the most difficult task when developing CBR systems.

The objective of this thesis is the development of a framework and algorithms for facilitating the acquisition and formalisation of one particular kind of domain knowledge CBR systems rely on, namely knowledge required for defining so-called *similarity measures*. Although the quality of this knowledge crucially influences the competence of a CBR system, up to now, a well-founded methodology for acquiring it is still missing.

1.1. Motivation

The new paradigm the CBR approach is based on, might be described as "controlled inexactness". Instead of modelling a complete domain theory, for example, by using rules, CBR exploits single situation-specific knowledge chunks called *cases*, which are easier available than generalised knowledge about the domain. In the traditional sense of problem-solving, a case represents a particular problem with a corresponding solution. Such a kind of knowledge is also named *experience* (Bergmann, 2002).

In order to reason from cases for solving new problems, CBR systems rely on knowledge required to select *useful cases* with respect to the current problem situa-

tion. In CBR, this selection of useful cases is based on a problem-solving assumption that can be summarised by one short phrase:

The core assumption of CBR: *Similar problems have similar solutions.*

This assumption's basic idea is to solve a current problem by reusing solutions that have been applied to *similar* problems in the past. Therefore, the current problem has to be compared to problems described in cases. Solutions contained in cases that represent very similar problems are then considered to be candidates for solving the current problem, too.

To enable a computer system to judge the similarity between two problems, CBR systems employ so-called *similarity measures* that represent a mathematical formalisation of the very general term "similarity". Similarity measures usually do not describe the dependencies between problems and corresponding solutions in detail, but only represent a form of a heuristics. Thus, the selection of actually useful cases—and so the strict correctness of the output—cannot be guaranteed in general. Nevertheless, by tolerating this inexactness one is able to develop powerful knowledge-based systems with significantly less effort and costs compared to the more traditional AI techniques relying on a complete and correct domain theory.

But of course, the relation between tolerated inexactness and development effort leads to a crucial trade-off. A CBR system will only be accepted by the users, if the inexactness can be restricted to a tolerable degree. Therefore, the quality of the employed similarity measure is a very important aspect of every CBR system. However, similarity measures used traditionally commonly represent only simple distance metrics that do not consider much knowledge about the domain. Here, similarity is rather interpreted as a "similar appearance".

In many CBR applications developed in the past, a more precise definition of similarity has not been introduced. On the one hand, in the addressed application domains the quite simple heuristics of "similar appearance" was mostly sufficient to obtain acceptable results. On the other hand, due to the novelty of the CBR approach, many other research issues had to be treated first, for example, the development of efficient retrieval algorithms. In recent years CBR research has reached a level where the basic questions have been answered. Now, research focuses more on an extension of the applicability of CBR on other, more complicated domains. Therefore, it becomes important to precise the definition of similarity in order to enable CBR systems to select useful cases also in domains, where the heuristics of "similar appearance" is no longer sufficient.

Although such *knowledge-poor similarity measures* are often sufficient to obtain acceptable results, the definition of more sophisticated measures may clearly improve the quality of CBR systems. In general, the utility of cases completely depends on the underlying domain and application scenario. Hence, considering as much domain knowledge as possible during the similarity assessment should lead to much better

heuristics for selecting useful cases. Such *knowledge-intensive similarity measures* might help decreasing the inexactness during problem-solving in domains where CBR already has been applied successfully. Moreover, they might also allow to apply CBR in domains where this technique could not be applied before, because knowledge-poor measures led to unacceptable results.

The definition of knowledge-intensive similarity measures is a very difficult task and leads again to knowledge acquisition problems like typical for other common AI techniques. Unfortunately, no clear methodology or general applicable approaches to support the modelling of such measures in an intelligent way have been developed, yet. Nowadays, the mandatory knowledge usually has to be acquired manually, for example, by interviewing domain experts. Further, the acquired knowledge has to be formalised by using complex mathematical representations. Thus, the definition of accurate similarity measures is still a complicated and time-consuming process when implementing CBR systems.

It is important to point out that a clear distinction between knowledge-poor and knowledge-intensive similarity measures is not possible. Both kinds of measures can be defined by using the same representation formalisms. The only difference lies in the amount of domain-specific knowledge encoded into the measures. So, each approach that facilitates the definition of measures that contain more domain knowledge without increasing the modelling effort significantly will probably contribute to a more widespread use of knowledge-intensive similarity measures resulting in more powerful CBR systems.

1.2. Objectives and Limitations of this Thesis

As discussed in the previous section, knowledge-intensive similarity measures allow to approximate the utility of cases much better than traditional knowledge-poor measures. However, this improvement of a CBR system's competence is coupled with the drawback of a significantly increased implementation effort, which is often not acceptable in commercial applications. Unfortunately, no clear methodology of how to define knowledge-intensive similarity measures efficiently has been developed, yet. The state-of-the-art procedure can rather be characterised as a creative process that requires particular skills and experiences from a human knowledge engineer. If such an experienced knowledge engineer is not available or too expensive, the only possibility is using standardised knowledge-poor similarity measures.

1.2.1. Objectives

The objective of this work is the development of a general framework towards a more methodological approach for acquiring and formalising domain knowledge required for the definition of knowledge-intensive similarity measures. In order to support a

human knowledge engineer during this process, or even to automate the knowledge acquisition process partially, the presented framework is based on the application of machine learning strategies. The basic idea is to extract the required knowledge from some special training data that can be acquired easier than the actual knowledge itself. The fundamental goals used as guidelines during the development of our framework can be summarised as follows:

General Applicability: In the last years several approaches towards learning of similarity measures have been developed. The problem of these approaches is that they mostly focus on very specific application situations. Firstly, they usually allow to learn single representation elements of similarity measures only, but are not applicable to support the definition of the entire similarity measure. Secondly, they are often restricted to particular application tasks, e.g. classification tasks. However, nowadays CBR addresses a much wider spectrum of application scenarios coupled with different properties and requirements. Due to these differences, here, existing approaches for learning similarity measures are mostly infeasible. Thus, one important objective of this work is the development of a clear methodology for learning similarity measures that is suited for various application scenarios in which CBR is applied today.

Integration into the CBR Process Model: In CBR research a generic process model introduced by Aamodt and Plaza (1994) is commonly accepted. This process model describes the basic steps of problem-solving when applying CBR. To point out the general applicability of our methodology for learning similarity measures, we aim to integrate it into this generic process model. Therefore, several basic steps of the process model have to be refined or extended, respectively.

Implementation of Exemplary Learning Algorithms: Besides the development of a general methodology describing basic concepts and processes required to enable learning of similarity measures, another goal of this work is the development of particular learning algorithms. These algorithms are required for implementing the actual learning functionality as part of the entire framework. We intend to realise this learning functionality by adapting well-researched machine learning algorithms to the particular requirements of the learning scenario targeted. Further, a first experimental evaluation will be necessary to validate the basic learning capability of these algorithms.

Prototypical Implementation of the Introduced Concepts: Moreover, the implementation should not be restricted to particular learning algorithms, but should also include other software modules required for applying the presented framework. So, this work should result in a first prototype in order to show

the applicability of the presented methodology and the corresponding concepts. Because such a prototype relies on basic CBR functionality, it will be realised in form of an extension of an existing commercial CBR tool.

1.2.2. Limitations

Since this work can be seen as a first step towards a more general application of learning techniques in order to facilitate the definition of knowledge-intensive similarity measures, this thesis cannot address all issues of this research topic. Thus, we have to point out some basic assumptions and limitations to be considered when employing the presented framework in real-world applications.

Focus on Particular Case Representation: Depending on the requirements of the application domain, very different approaches to represent cases are used in CBR. In this thesis, we focus on the formalism used most commonly, namely *attribute-value based case representations*, which is sufficient in most domains, and thus, is also employed in most commercial applications.

Focus on Particular Types of Similarity Measures: For attribute-value based case representations, a particular method for defining similarity measures has proven its worth. The basic idea of this approach is the so-called *local-global principle* which states that the entire similarity assessment can be subdivided into a set of local parts and a global part. Here, the local parts reflect the contribution of individual attributes to the entire similarity assessment. Therefore, for each attribute a so-called *local similarity measure* has to be defined. The global part is responsible for estimating the actual utility of complete cases. It is based on the outcome of the local similarity measures and so-called *attribute weights* that reflect the relative importance of the individual attributes for a case's utility. Although sometimes other approaches to represent similarity measures are used, in this work we focus on similarity measures defined according to the local-global principle. On the one hand, this is a very powerful approach and on the other hand, it is supported by the most commercial CBR tools.

Prototypical Implementation: As already mentioned, one objective of this work is an implementation of the presented framework by extending a commercial CBR tool. However, it must be pointed out that the resulting prototype is only intended to represent a demonstration and test environment for showing the principle applicability of the concepts and algorithms introduced. This prototype won't be suited for a direct application of our framework in industrial CBR practice. Therefore, also other issues (e.g. computational performance or comfortable user interfaces) would have to be considered in detail, what would go beyond the scope of this work.

Limited Evaluation: In the scope of this thesis we restrict the necessary experimental evaluation to two selected application scenarios with a clear practical relevance. Nevertheless, the corresponding experiments should be sufficient to show the capabilities of our framework.

1.3. Overview

This thesis is divided into four parts. The first part introduces the basic terminology and discusses the problems addressed by this work. In the second part, the framework and particular algorithms are presented. Scenarios how to apply the framework in practice are discussed. The third part describes a prototypical implementation of the framework and a corresponding experimental evaluation. Finally, in the fourth part an overview on related work and a discussion of the achieved results is presented.

Part I: Foundations After this introduction, in Chapter 2 we start with a short summary of Case-Based Reasoning. We discuss some of the characteristics that are responsible for the great success of CBR. Later, in Chapter 4, we will see that these aspects also motivate the development of a framework for learning similarity measures. We also give a rough overview of the commonly accepted process model of CBR and discuss the role of domain knowledge. After summarising popular application fields, we investigate the relationship between CBR and machine learning in more detail. In Chapter 3 we focus on one important aspect of CBR, namely similarity measures used to select cases that might be useful for solving a given problem. First, we discuss the role of similarity measures and their mathematical foundations in detail. After that, we review some traditional approaches for representing similarity measures and introduce the more sophisticated representation formalism presumed in this thesis. Finally, we show how such similarity measures are commonly defined, and point out some crucial drawbacks of this state-of-the-art procedure.

Part II: The Framework In Chapter 4, we motivate why machine learning approaches seem to be promising for supporting the definition of similarity measures. After that, we present the basic concepts of our framework. We introduce an alternative type of knowledge—called *utility feedback*—that can be be used as a foundation for defining similarity measures. Further, we show how this knowledge, in principle, might be exploited by machine learning algorithms in order to learn accurate similarity measures. In Chapter 5, we introduce two concrete learning algorithms that are suited to perform the learning process described before. Finally, in Chapter 6 we discuss several scenarios how the presented framework might be applied in practice.

Part III: Implementation and Evaluation In Chapter 7, we shortly describe the prototypical implementation of our framework. Here, we do not focus on the actual software code (e.g. class hierarchies), but more on the basic concepts and the implemented user interfaces. In order to show the capabilities of the learning algorithms developed, Chapter 8 discusses some evaluation experiments carried out with the implemented prototype.

Part IV: Related Work and Conclusions In Chapter 9, we give an overview on other work that is related to the topic of this thesis. After a short summary, in Chapter 10, we then discuss this works's outcome and contribution and point out some open questions and further research issues.

1.4. Expectations on the Reader

In principle, we do not expect particular previous knowledge from the reader of this thesis. However, although we give a short overview on CBR in Chapter 2, some experiences in this field would certainly help to understand the motivation of this work. For a more detailed description of CBR the reader is referred to respective literature, e.g. (Bergmann, 2002; Lenz et al., 1998). Readers who are familiar with the CBR technology, and in particular with the similarity measure representation presumed, might also start with the actual description of our framework directly. Concerning the learning algorithms described in Chapter 5, some basic knowledge in Artificial Intelligence and Machine Learning might be helpful. A good overview on Artificial Intelligence is given, for example, by Norvig and Russell (1995), while Mitchell (1997) gives a good introduction into the fundamental concepts of Machine Learning.

Part I.

Foundations

2. Case-Based Reasoning

The desire for computer systems being able to support human experts during complex problem-solving tasks is a traditional topic of *Artificial Intelligence (AI)* research. In order to enable a computer system to give reasonable support when solving problems in a complex application domain, it is indispensable to provide it with specific knowledge about that domain. Several methodologies to realise such knowledge-based systems have been developed, for example, *rule-based* and *model-based* approaches. In the last years an alternative approach called *Case-Based Reasoning (CBR)* has become very popular. A huge number of applications have been developed yet, addressing a wide range of domains like classification, diagnosis, decision support, configuration or planning. In contrast to some other traditional AI technologies, CBR has made the step from research to successful commercial application very quickly. This success can be explained by the new problem-solving paradigm that avoids some crucial problems of the traditional approaches and takes care of some characteristic properties of problem-solving.

In this chapter, we want to give a brief overview on Case-Based Reasoning. We start with a motivation of the problem-solving paradigm laying the foundation of the CBR technology. After that, a short introduction to the basic concepts of CBR is given, followed by an overview of some typical application fields. We also introduce the basic terminology and some important definitions required in the scope of this thesis. Because the aim of this work is to extend the capability of CBR systems to learn domain knowledge, the last section of this chapter focuses on the relation between CBR and *Machine Learning (ML)*.

2.1. Motivations for CBR

The underlying paradigm of CBR introduced in Section 1.1 can be discussed from two different points of view. On the one hand, it may be interpreted as the foundation of a cognitive model of human reasoning. On the other hand, it gives an idea of how to implement efficient problem-solving functionality for realising knowledge-based computer systems. In the following, we want to discuss these two aspects of the CBR paradigm in more detail.

2.1.1. Cognitive Model

The roots of CBR are founded in the research field of *cognitive science*. Here, researchers try to understand and explain the procedures that human beings use to solve the problems of their daily life. Therefore, cognitive models on how the problem-solving processes might be executed within the human brain have been developed on the basis of evidences obtained through psychological experiments.

One observation during the development of these models of human reasoning was that human beings obviously use two different types of knowledge during problem solving:

Generalised rules and procedures: This kind of knowledge represents general expectations that human beings possess about the world around them. A special type of generalised knowledge is propagated by the *script-theory* (Schank and Abelson, 1977). According to this model, a *script* is a special structure in memory which describes general expectations about a typical situation. By using generalised knowledge human beings are able to reason and act in the world. However, the accumulation of such knowledge also requires a certain understanding of the coherences in that world.

Situation-specific knowledge: Problems with the script-theory showed that human beings use an additional kind of knowledge to act in the world appropriately. The *dynamic memory* model (Schank, 1982) introduces specific knowledge about typical situations as an additional knowledge resource. In contrast to generalised knowledge, the acquisition of such specific knowledge requires no deeper understanding of the underlying coherences. While the generation of rules implies an inference process, specific knowledge is only stored in memory in form of independent knowledge chunks.

According to the dynamic memory model, general knowledge and situation-specific knowledge may be recalled to solve current problems. Which knowledge is actually used depends on the particular situation. If no appropriate generalised knowledge is available, particular situation-specific knowledge containing knowledge about a situation that is similar to the current one might be reused.

To validate the developed models and to illustrate how situation-specific knowledge might be used to solve problems more efficiently than without this kind of knowledge, some of the models have been implemented by using computer systems. An example for such an implemented cognitive model based on the dynamic memory theory is CYRUS[1] (Kolodner, 1993).

The prototypical implementations of cognitive scientists can be interpreted as the first CBR systems. However, the focus of these systems was not a real support

[1] Computerized Yale Retrieval and Updating System

for solving important real-world problems, but an illustration of the underlying cognitive models. Nevertheless, the ideas of these systems have been taken up by AI research in order to build knowledge-based systems for practical usage.

2.1.2. Approach for Building Knowledge-Based Systems

Motivated by the cognitive models, the CBR paradigm has founded a new research direction in AI[2]. Here, the focus of research is not an explanation of human reasoning, but the implementation of knowledge-based computer systems being able to solve real-world problems.

Most AI approaches to problem-solving developed so far are based on the formalisation and application of generalised knowledge about the addressed application domain. In contrast, the paradigm that propagates the use of situation-specific knowledge to solve problems offers three potential functional benefits compared to the traditional methods (Leake, 1994).

Increased Problem-Solving Efficiency

Depending on the problem task, the process of finding a solution is often a complex procedure since the solution has to be generated by combining a (possibly huge) number of operations or components. This situation typically occurs in synthetic problem domains like planning or configuration. Here, the generation of solutions by employing generalised knowledge (e.g. rules) often leads to difficulties due to the high computational complexity. This complexity is caused by the huge number of theoretical possibilities to compose a solution. Storing already generated solutions in order to reapply them for problems occurring in the future may increase the problem-solving effort significantly compared to generating the solution again from scratch. Besides such an increase of problem-solving efficiency with respect to computation time, an increase of the generated solution's accuracy is possible, too. Consider a situation where the application of generalised knowledge may lead to several valid solutions for a given problem. In such a situation reusing a well-tried solution available in form of situation-specific knowledge can implicitly guarantee high quality of the system's output. In order to be able to consider solution quality when applying a pure rule-based approach would require to encode the quality criteria by introducing additional rules. However, quality criteria are often very vague and therefore difficult to formalise.

[2]The first scientific conferences that address CBR as a methodology to build knowledge-based systems have been organized around 1990. Today, CBR plays a significant role in AI research.

Handling of Incomplete Domain Knowledge

One major drawback of the traditional rule-based and model-based approaches to problem-solving is the need of a complete and correct domain theory. Unfortunately, in practice one is often confronted with poorly-understood domains where such a domain theory is not available. Here, even human domain experts do not know all the interdependences within the domain. When solving problems in such domains, they often make decisions not based on generalised knowledge but on experiences they made in similar situations in the past. If a knowledge-based systems is able to process such situation-specific experience knowledge, it may provide reasonable solution proposals even if it does not posses a complete and consistent model of the underlying domain. Of course, such a procedure is couple with the drawback that the proposed solution's correctness cannot be guaranteed in general. However, if it is not possible to guarantee the consistency of a domain theory this holds for the other approaches as well.

Simplification of Knowledge Acquisition

Even if a complete and consistent domain theory exists in principal, another drawback of traditional AI approaches is the well-known *knowledge acquisition bottleneck*. In order to enable a rule-based or model-based system to reason within the domain theory, the domain knowledge has to be formalised by using the formal knowledge representation required by the employed reasoning techniques (e.g. rules). However, generalised domain knowledge usually does not exist in such an explicit form. Instead it is mostly only implicitly available in form of human domain experts who use informal methods when being asked to describe their knowledge. Sometimes the human experts even have serious difficulties to describe their knowledge instead of applying it only. Therefore, a sophisticated and time-consuming knowledge acquisition process is usually a prerequisite when developing knowledge-based systems that employ traditional AI approaches. In a first step, the knowledge has to be acquired, for example, by performing interviews with domain experts. In a second step, the acquired—but still informal—knowledge has to be formalised by transforming it into the assumed knowledge representation.

One of the major functional benefits of Case-Based Reasoning is the possibility to reduce the knowledge acquisition effort. Of course, the CBR approach also relies on domain knowledge to be able to provide solution proposals for given problems. However, the use of situation-specific knowledge as core knowledge resource from which to reason, simplifies the still necessary knowledge acquisition phase significantly. In contrast to generalised domain knowledge, specific knowledge is often explicitly available in form of documented solutions applied to problems already treated in the past. In such a situation, knowledge acquisition is reduced to an appropriate formalisation of the already documented knowledge. Due to the underlying reasoning

mechanism of the CBR approach even this formalisation step is usually easier than more formal AI approaches. However, it is a natural desire to reduce the remaining knowledge acquisition effort for developing CBR systems further by providing "intelligent" computer support.

2.2. CBR in a Nutshell

In the last years Case-Based Reasoning research has led to the development of numerous concepts, techniques, and algorithms to implement the CBR paradigm in practice. This section gives a brief overview of the main aspects of CBR as an AI technology. Since a detailed description would go beyond the scope of this thesis, we restrict the summary only on the fundamental aspects relevant for the work described in the subsequent sections. More detailed information about the CBR technology can be found, for example, in publications by Kolodner (1993); Aamodt and Plaza (1994); Lenz et al. (1998); Bergmann et al. (1999a).

First, it is shown how situation-specific knowledge can be used to solve problems. In order to illustrate the general functionality of CBR systems, an overview of a commonly accepted process model of CBR is given. In the following, the role of knowledge in this process model is pointed out in more detail. Finally, basic terms and definitions representing the foundation for the subsequent parts of this work are introduced.

2.2.1. Case-Based Problem-Solving

In CBR situation-specific knowledge originally was seen as *experience* about particular problem-solving episodes observed in the past. Typically, such an episode, called *case*, is represented by a pair consisting of a problem description and a description of the corresponding solution.

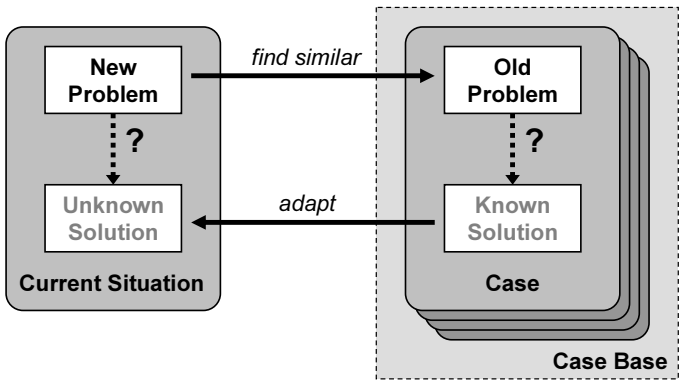


Figure 2.1.: Case-Based Problem-Solving

According to the basic CBR paradigm (see Section 2.1), case knowledge can be used to solve a new, still unsolved problem (see Figure 2.1). For this, the description of the new problem is compared to the problem parts contained in the available cases. The set of all available cases is also called *case base*. The case with the problem description being most *similar* to the new problem then is selected as a candidate to solve the current problem. For this, the solution part of the candidate case is tried to be reused to solve the new problem. If the solution cannot be applied directly, it has to be *adapted* in order to fit the new situation.

2.2.2. The Case-Based Reasoning Cycle

The general procedure when applying CBR, is commonly described by the classical *Case-Based Reasoning cycle* introduced by Aamodt and Plaza (1994) (see Figure 2.2).

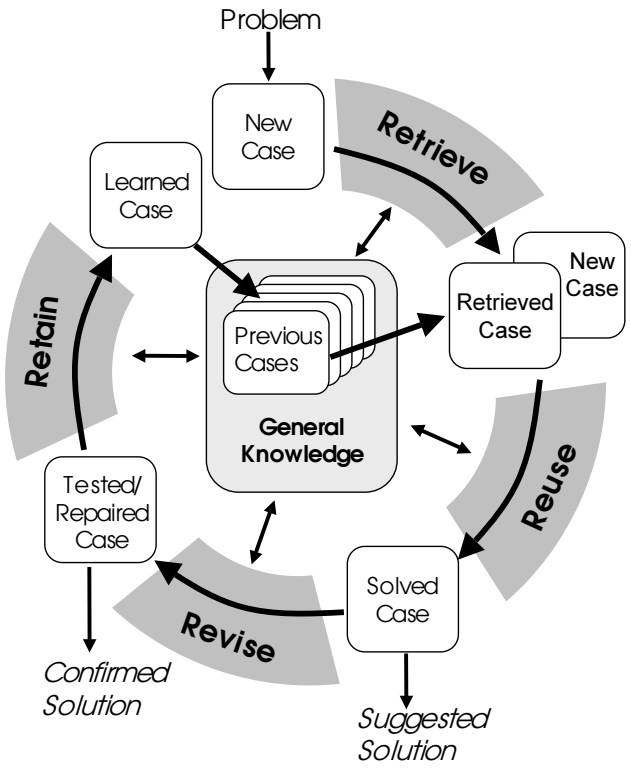


Figure 2.2.: The Case-Based Reasoning Cycle by Aamodt & Plaza

The starting point of a new problem solving-process is a given problem for which a solution is required. This problem can be characterised as a *new case* for which the solution part is still unknown. This new case—often also called *query*—is then processed in four consecutive steps.

Retrieve

The first step is the retrieval of one or several cases considered to be useful to support solving the current problem. The hope is to retrieve one or several cases that contain solutions that can be easily reused in the current problem-solving context. Because it is usually very unlikely that the case base already contains a problem description that matches the new problem exactly, a method to estimate the *utility* of available cases is required. According to the basic CBR assumption (cf. Section 1), here the concept of *similarity* is used. This means, the task of the retrieval phase is to select cases whose problem descriptions are similar to the current problems' description. The underlying assumption is that these cases contain solutions being very similar to the searched—but still unknown—solution of the current problem.

To realise this retrieval task, CBR systems employ special *similarity measures* that allow the computation of the similarity between two problem descriptions. Because the interpretation of this similarity strongly depends on the particular domain, similarity measures are part of the *general knowledge* of the system.

Depending on the size of the case base, the information amount contained in single cases, and the complexity of the used similarity measure, the retrieval step is often a challenging task with respect to computation time. In order to manage this complexity, a large number of different retrieval strategies have been developed, e.g. see Lenz (1999); Wess (1993); Schaaf (1996); Schumacher and Bergmann (2000).

Reuse

After selecting one or several similar cases, the reuse step tries to apply the contained solution information to solve the new problem. Often a direct reuse of a retrieved solution is impossible due to differences between the current and the old problem situation. Then the retrieved solution(s) have to be modified in order to fit the new situation. How this *adaptation* is performed strongly depends on the particular application scenario. An overview of existing approaches to adaptation is, for example, given by Wilke and Bergmann (1998).

In general, adaptation methods require additional general knowledge about the application domain. Because this leads to additional knowledge acquisition effort, many CBR systems used today do not perform case adaptation automatically, but leave this task to the user. Then, of course, the quality of the retrieval step influences the problem-solving capabilities of the entire CBR system primarily. Even if automatic adaptation is provided, the quality of the retrieval result will strongly influence the efficiency of the system due to its impact on the required adaptation effort.

After adapting the retrieved case automatically or manually to fit the current situation, a *solved case* is obtained containing a *suggested solution* for the current problem.

Revise

Depending on the employed adaptation procedure, the correctness of the suggested solution often cannot be guaranteed immediately. Then it becomes necessary to revise the solved case. How such a revision is performed, strongly depends on the particular application scenario. For example, it might be possible to apply the suggested solution in the real world to see whether it works or not. However, often a direct application of an uncertain solution is impossible due to the corresponding risks (e.g. medical diagnosis systems). Then the revision has to be performed manually by a human domain expert or by alternative methods such as computer simulation.

If the revise step fails, the case has to be repaired or a new attempt to generate a valid solution has to be carried out. This new attempt can be realised in different ways. One possibility is to apply another adaptation alternative (if existing). Other possibilities are adapting another retrieved case or executing a new retrieval in order to obtain hopefully more useful cases.

Usually, the focus of the revise phase lays on the detection of errors or inconsistencies in the suggested solution and the initiation of further problem-solving attempts. Unfortunately, up to now less research has been carried out to enable CBR systems to recognise the reasons for failed problem-solving processes. If the system would be able to detect these reasons, it could react and, for example, adapt its internal knowledge resources in order to avoid similar failures in the future. Such a behaviour is often characterised as *introspective reasoning* or *introspective learning* and is discussed in more detail in Section 2.4.

Retain

If the solved case has passed the revise step successfully, a *tested/repaired case* will be available representing a new experience that might be used to solve similar problems in the future. The task of the CBR cycle's last step is to retain this new case knowledge for future usage. Therefore, the new case may be added to the case base. However, it has been shown that a general storage of all generated cases is not always useful. In order to enable better control of the retain process, various approaches for selecting cases to be retained have been developed (Aha, 1991; Smyth and Keane, 1995b; Leake and Wilson, 2000; Ferrario and Smyth, 2000; Reinartz et al., 2000). These approaches often imply a reorganisation of the entire case base when adding a new case, for example, by removing other cases.

Generally, the capability to acquire new case knowledge during a CBR system's lifetime principally adds these systems to the class of *learning systems*. However, many CBR systems developed so far do not exploit this concept of the CBR cycle at all. This holds especially for the commercially employed systems. Further, the original idea of the CBR cycle focuses on learning case knowledge. Nevertheless, the

possibility to learn also general knowledge when processing the CBR cycle should be considered, too (see Section 2.4).

2.2.3. Knowledge Container

As other approaches to building knowledge-based systems, the CBR approach also relies on specific domain knowledge to be able to draw reasonable conclusions. Although the CBR paradigm propagates the use of situation-specific knowledge, the exclusive usage of knowledge contained in cases is usually insufficient to build powerful CBR applications. Hence, already the classic CBR cycle (cf. Section 2.2.2) introduced additional general knowledge to be used during the different phases of the reasoning process. According to Richter (1995) one can generally distinguish four different *knowledge containers* (see Figure 2.3) within a CBR system, namely the *vocabulary*, the *case knowledge*, the *similarity measure*, and the *adaptation knowledge*.

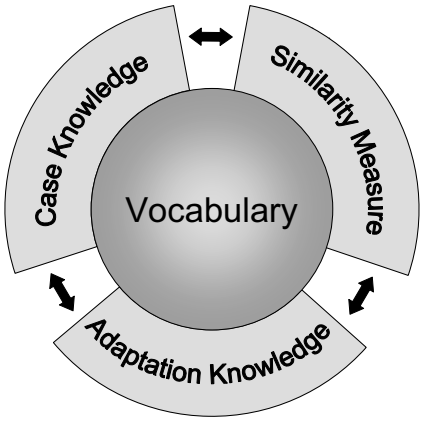


Figure 2.3.: Knowledge Containers

Vocabulary

The vocabulary represents the central foundation of the other knowledge containers. It defines which information is considered to be important in the respective domain and how this information can be expressed. Therefore, it introduces information entities and structures to represent domain knowledge and can be characterised as the "language" used to talk about the domain. Although the vocabulary does not directly define knowledge used for problem-solving, it is also an important knowledge resource. When defining the vocabulary, one has to decide whether an available information item about the domain is important for problem-solving or not. Usually, this decision can only be made by a domain expert based on her/his expertise.

Generally, various approaches to represent the vocabulary are known. Which knowledge representation is the most appropriate one depends on the domain addressed. In Section 2.2.4 we will introduce the representation formalism that we presume in the scope of this thesis.

Case Knowledge

The use of case knowledge is obviously a property of CBR that distinguishes this approach from most other AI approaches to problem-solving. Case knowledge is situation-specific knowledge typically obtained in problem-solving situations of the past. Since an already performed problem-solving process can be characterised as a kind of experience, case knowledge is often also denoted as *experience knowledge*. In contrast to general knowledge, one important characteristic of situation-specific knowledge is that is has not to be understood exactly to be exploited for problem-solving.

The traditional CBR approach assumes that a case consists of two major parts:

Problem Part: This part of a case contains information characterising the problem situation occurred in the past. Due to the basic assumption of CBR, it is crucial that this description in particular includes information relevant to decide whether two problems are *similar* or not.

Solution Part: This part contains information used to reproduce the solution applied successfully in the past when being confronted with new problem situations.

Nevertheless, the solution part can also include additional information that might improve or simplify the reuse of the experience, for example, information about

- the way how the solution was obtained,
- the solution's quality,
- constraints restricting the solution's application,
- alternative solutions.

In some domains a strict differentiation between the problem and the solution part is impossible or inaccurate. Instead, sometimes a case might be characterised as an arbitrary situation-specific knowledge chunk. In Section 2.3 some application fields are discussed where CBR has been successfully applied even though the traditional problem-solution distinction is not obvious.

In order to be able to reuse case knowledge efficiently, it is important to consider the characteristics of the particular application domain. Because CBR has been applied successfully in many different domains, several formalisms to represent case knowledge have been developed, for example:

- *Attribute-value based representations* are a very flexible and commonly used approach to represent case knowledge in a structured way.
- *Object-oriented representations* can be seen as an extension of the "flat" attribute-value based representation. They are useful in situations where additional structural information is required.
- In planning domains *representations based on first order logic* are commonly used.
- To represent locality information (e.g., in architecture) sometimes also *graph representations* are suitable.

In the scope of this thesis we presume an attribute-value based representation, which we will define formally in Section 2.2.4.

Similarity Measure

To be able to reuse case knowledge accurately, additional general knowledge is required. In order to select cases to be reused in a particular problem situation, CBR systems employ special *similarity measures*. The task of these measures is to estimate the *utility* of cases with respect to the current problem-solving task. Unfortunately, the actual utility of cases cannot be determined until problem-solving is finished, or in other words, utility is an *a-posteriori* criterion. The reason for this is the general problem that the underlying utility functions are usually only partially known. In order to be able to approximate the utility of cases before the actual problem solving process, CBR systems rely on specific similarity knowledge encoded in form of similarity measures. Hence, similarity measures can be characterised as an *a-priori* criterion or a *heuristics* used to approximate the unknown utility functions.

It is obvious that the knowledge contained in these measures is crucial for the retrieval step of the CBR cycle, and so for the entire problem-solving capability of the system. The amount and quality of the available similarity knowledge strongly affects the outcome of the retrieval phase.

Depending on the concrete interpretation of the term *similarity*, modelling similarity measures is a more or less difficult task when implementing case-based systems. Chapter 3 deals with this issue in detail. It also introduces approaches how to represent similarity measures in CBR systems.

Adaptation Knowledge

Adaptation knowledge just as similarity knowledge is general knowledge needed to allow efficient reuse of situation-specific knowledge contained in cases. While the

similarity measure is used to select cases to be reused, adaptation knowledge is required to control the actual reuse of previously selected cases. It defines how a retrieved solution can be adapted to fit the special circumstances of a new problem situation. Generally, one can distinguish between two basic approaches to perform solution adaptation:

Transformational Adaptation: Here, the cases' solution part represents a concrete solution generated in the past. During adaptation the retrieved solution has to be transformed to a new solution fulfilling the current situation's requirements by adding, modifying or deleting solution parts.

Generative Adaptation: Instead of storing the actual solution it is also possible to store the process by which the solution was generated in the past. This information can be reused to generate an accurate solution in a similar situation efficiently. Therefore, a generative problem-solver tries to replay the known solution way as far as possible. If some solution steps cannot be replayed, alternative solution steps have to be generated from scratch. This strategy is also denoted as *derivational analogy* (Cunningham et al., 1993).

The major difference between these two basic approaches is the way how adaptation knowledge has to be provided and how it is employed. One the one hand, generative adaptation requires a generative problem-solver that is, in principle, able to solve a given problem without the use of cases. Hence, this problem-solver requires a complete and consistent domain theory. This general domain knowledge is also used to perform adaptation of retrieved cases. Some approaches realise case adaptation by using constraint satisfaction techniques (Purvis and Pu, 1995).

On the other hand, transformational adaptation is performed without a generative problem-solver. Thus, it requires another formalism to represent and to apply adaptation knowledge within the CBR system. Which concrete representation formalism is appropriate depends on the application domain. A common approach are *adaptation rules* (Bergmann et al., 1996; Leake et al., 1995; Hanney and Keane, 1996) or *adaptation operators* (Schmitt and Bergmann, 1999b). Another approach for describing adaptation knowledge within the case representation are *generalised cases* (Bergmann et al., 1999b; Bergmann and Vollrath, 1999; Mougouie and Bergmann, 2002). A detailed overview of adaptation techniques is given, for example, by Wilke and Bergmann (1998); Fuchs et al. (1999); Hanney et al. (1996).

Relations between Knowledge Containers

Richter (1995) pointed out that the four knowledge containers should not be seen as completely independent. Generally, it is possible to shift knowledge between the separate containers in order to adapt CBR systems to the specific conditions of

the addressed application domain. Two basic motivations for shifting knowledge between knowledge containers are conceivable:

Facilitating Knowledge Acquisition: This situation occurs if it is easier to acquire knowledge of one container compared to another one. A typical example is a knowledge shift between case and adaptation knowledge. On the one hand, if it is easy to describe valid adaptation possibilities (e.g. through rules) one may save case knowledge acquisition effort. On the other hand, if it is really difficult to acquire or formalise general adaptation knowledge, additional case knowledge might replace general adaptation knowledge.

Improving Problem-Solving Efficiency: Shifting knowledge between containers might also increase the efficiency when solving problems. For example, knowledge about provided adaptation possibilities can significantly improve the quality of the case retrieval. If the similarity measure would "know" valid adaptation possibilities it could use this knowledge to retrieve the best "adaptable" cases instead of only retrieving the most similar ones. How to realise a shift between adaptation and similarity knowledge to enable such an *adaptation guided retrieval* (Smyth and Keane, 1993, 1995a; Leake et al., 1996a) is discussed in detail in Section 6.5.

Finally, it can be summarised that a proper management of the knowledge contained in the four knowledge containers is necessary to ensure powerful CBR applications. Such a management includes the accurate distribution of available knowledge over the containers just as the maintenance of the knowledge to guarantee its quality continuously (Roth-Berghofer, 2002; Heister and Wilke, 1997). Although numerous techniques and algorithms to process the knowledge have been developed yet, the management of the knowledge has come into focus of CBR research just recently.

2.2.4. Basic Terms and Definitions

As mentioned in the previous section, the foundation to describe information and knowledge about a particular application domain is the vocabulary. In this section some general assumptions and corresponding definitions about the basic structure of the vocabulary used to describe case knowledge in the scope of this work are introduced.

Basic Case Structure

Contrary to the case structure of the traditional CBR approach that distinguishes between a problem and a solution part, in this work we assume a case structure that can be used in a broader sense. According to Bergmann (2002) we distinguish between the following two components of cases:

Characterisation Part: The case characterisation part contains all information required to decide whether a case can be reused in a certain situation. That means this part of the case can be seen as an index used to estimate the utility of cases.

Lesson Part: The lesson part describes all additional information that might be useful for the actual reuse of the case. Note that the lesson part may also be empty. In this situation, the information contained in the case characterisation part is already sufficient to reuse the case.

From the traditional point of view the problem part corresponds to the case characterisation part and the solution part corresponds to the lesson part of cases. In the following, the representation formalism used to describe the two mentioned case components is introduced.

Attribute-Value Based Representation

In the scope of this work we assume attribute-value based case representations. The basic element of this representation formalism are attributes:

Definition 2.1 (Attribute) An *attribute* A is a pair (A_{name}, A_{range}) where A_{name} is a unique label out of some name space and A_{range} is the set of valid values that can be assigned to the attribute, also called *value range*. Further, $a_{name} \in A_{range} \cup \{undefined\}$ denotes the current value of a given attribute A identified by the label A_{name}.

The special attribute value *undefined* may be used, if an attribute value is unknown or irrelevant. In principle, the value range of an attribute may contain an arbitrary (possibly infinite) collection of elements of a basic *value type*. Examples of basic value types are

- the numeric type *Integer*
- the numeric type *Real*
- symbolic types
- temporal types like *Date* and *Time*
- etc.

Usually, the range of allowed values is not defined directly within the attribute declaration but by the declaration of a specialised value type. This approach simplifies the definition of identical value ranges for several attributes by assigning type

names to attributes. In order to simplify the notation, however, in this work we assume that each attribute possesses its own value range.

To describe the set of allowed attribute values efficiently, three possibilities to define attribute ranges can be used:

1. By *specifying only the basic value type* all values of this type are allowed to be assigned to the attribute (e.g., all Integer values).

2. When using numeric types, a set of allowed values can easily be defined by the *specification of an interval* (e.g., Real values of the Interval $[0, 1]$).

3. The most flexible way, which is also the only feasible way for the definition of symbolic types, is an *explicit enumeration of all allowed values* (e.g., an enumeration of colours {red, yellow, green}).

As already mentioned above, we assume that cases consist of a case characterisation and a lesson part:

Definition 2.2 (Case Characterisation Model, Space of Case Char. Models) A *case characterisation model* is a finite, ordered list of attributes $D = (A_1, A_2, \ldots, A_n)$ with $n > 0$. The symbol $\hat{D}$ denotes the *space of case characterisation models*.

Definition 2.3 (Lesson Model, Space of Lesson Models) A *lesson model* is a finite, ordered list of attributes $L = (A_1, A_2, \ldots, A_n)$ with $n \geq 0$. The symbol $\hat{L}$ denotes the *space of lesson models*.

Now, we are able to introduce the basic definitions for a formal description of cases using an attribute-value based representation:

Definition 2.4 (Case Model, Space of Case Models) A *case model* is a pair $C = (D, L) = ((A_1, A_2, \ldots, A_n), (A_{n+1}, A_{n+2}, \ldots, A_m)) \in \hat{D} \times \hat{L}$ with $m \geq n$. The symbol $\hat{C}$ denotes the *space of case models*.

Note that we assume a non-empty case characterisation part in opposite to the lesson part of cases that might contain no information at all.

Definition 2.5 (Case, Case Characterisation, Lesson) A *case* according to a given case model $C \in \hat{C}$ is a pair $c = (d, l)$ where $d = (a_1, a_2, \ldots, a_n)$ with $n > 0$ and $l = (a_{n+1}, a_{n+2}, \ldots, a_m)$ with $m \geq n$ are vectors of attribute values and $a_i \in A_{i_{range}} \cup \{undefined\}$ is the value of the attribute A_i. Further, the vector d is called the *case characterisation* and the vector l is called the *lesson* of c.

Definition 2.6 (Case Space, Case Characterisation Space, Lesson Space) The set of all valid cases according to a given case model $C \in \hat{C}$ is the *case space* $\mathbb{C}_C$ of C. Moreover, the symbol $\mathbb{D}_D$ denotes the *case characterisation space* according to

a case characterisation model $D \in \hat{D}$, and the symbol $\mathbb{L}_L$ denotes the *lesson space* according to a lesson model $L \in \hat{L}$.

Compared with an object-oriented programming language, the case model corresponds to the concept of a *class* and a single case corresponds to an *instance* of that class.

Note that attributes just as attribute values of a case model C, or a case c respectively, can be accessed either by referring the index $i = 1 \ldots n$ within the case model C or by the unique name of the attribute. In this thesis, we use the following notation to access attributes and attribute values:

- access to attributes of a case model C: $C.A_i$ or $C.A_{name}$
- access to attribute values of a case c: $c.a_i$ or $c.A_{name}$

Further, we define a special kind of cases called *queries*:

Definition 2.7 (Query) Given a case model $C \in \hat{C}$, a *query* is a special case $q = (d, l) \in \mathbb{C}_C$ with an empty lesson part l, i.e., for all attributes $A_i \in l$ holds $q.a_i = undefined$.

In a CBR system one has to deal, of course, not only with single cases and queries but with as set of cases leading to the following definition:

Definition 2.8 (Case Base) A case base CB for a given case model C is a finite set of cases $\{c_1, c_2, \ldots, c_m\}$ with $c_i \in \mathbb{C}_C$.

Example of an Attribute-Value Based Representation

To illustrate the definitions introduced in the previous section, in Figure 2.4 an example of an attribute-value based case representation is shown. Here, cases represent descriptions of personal computers and might, for example, be used to realise a CBR-based product recommendation system (see Section 2.3.4). The corresponding case model consists of several attributes with adequate value ranges representing the technical properties of PCs.

2.3. Application Fields

The CBR approach has already proven its general applicability in many different application domains. The capability to build powerful knowledge-based systems without the need of a complete formalisation of the underlying domain has also led to a great commercial success of CBR recently (Bergmann et al., 1999a; Lenz et al., 1998; Stolpmann and Wess, 1999). In particular, in domains not requiring

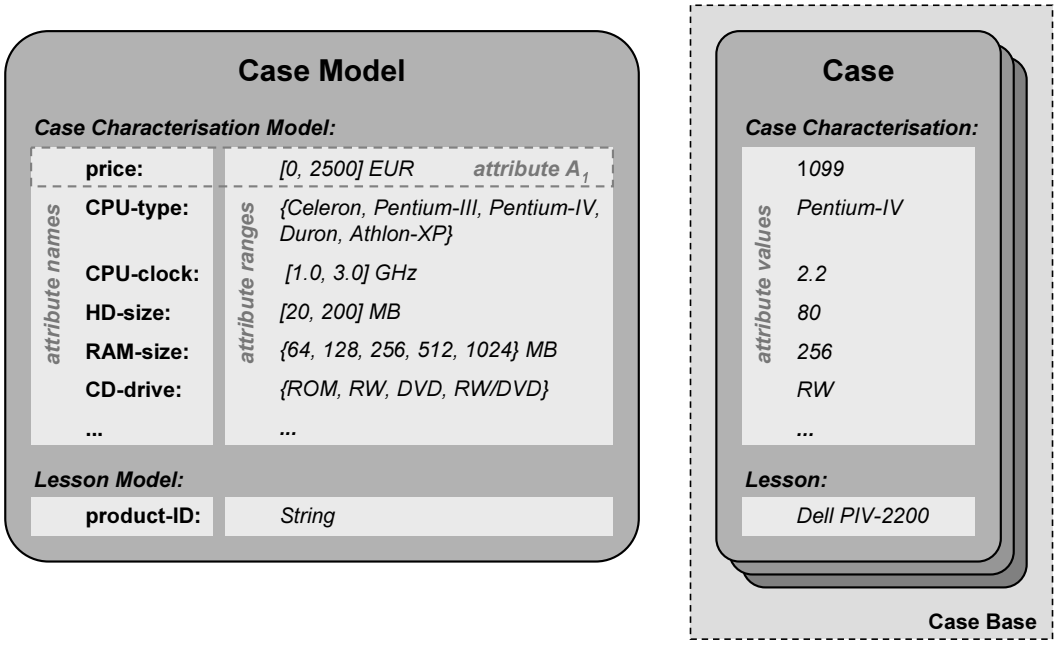


Figure 2.4.: Example: Attribute-Value based Case Representation

necessarily case adaptation, a huge number of commercial applications have been developed yet. However, in very complex domains, where sophisticated adaptation functionality and knowledge is essential, existing applications are still restricted to research prototypes.

In this section, an overview of some typical application fields of CBR is given, comprehending commercially successful domains as well as domains still in focus of research only.

2.3.1. Classification

One of the first application fields where CBR systems have been applied successfully is *classification*.Here, the origin is a technique called *Nearest-Neighbour (NN) classification* (Dasarathy, 1991; Mitchell, 1997), that can be characterised as a quite simple CBR approach.

The goal of classification is to predict the class membership of given entities. The basic idea of NN-classification is using information about entities for which the class membership is already known. In order to classify a new entity, its description has to be compared to the descriptions of the known entities. From an abstract point of view, each entity can be characterised as a point in some problem space defined by the properties used to describe the entity (see Figure 2.5). To predict the unknown class of the new entity, its nearest neighbours within the problem space have to be determined by using some *distance metric*. Finally, the information about the class

membership of these NNs is used to predict the unknown class of the given entity. Depending on the concrete realisation, the class of the actual NN or a weighted voting using the classes of k-NNs is used as prediction.

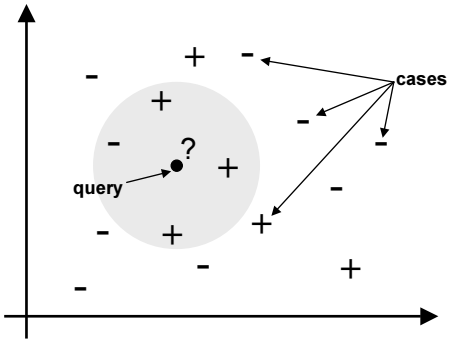


Figure 2.5.: Nearest-Neighbour Classification

In the example shown in Figure 2.5 the entities belong either to the class "+" or "-". Using a voted *4*-NN classification the prediction for the shown query would be that it belongs to class "+" because 3 of the 4 NNs belong to this class.

From the CBR point of view it is obvious that this approach requires no sophisticated adaptation methods as long as the number of cases exceeds the number of possible classes significantly. Although a weighted voting can be characterised as a simple form of *compositional adaptation*[3] (Wilke and Bergmann, 1998), the case retrieval is the central task in classification domains. This is one reason for the very good suitability of CBR for classification. Another reason is the easy capability to handle noisy and incomplete data, leading to problems when deploying traditional approaches to classification like *decision trees*[4]. Due to these advantages, numerous commercial and prototypical case-based classification systems have been developed.

2.3.2. Help Desk

Another application field, in which CBR is already an established technology, are so called *help desk systems*. These systems are used to provide support in *diagnosis* situations in technical or medical domains (Althoff, 1992; Althoff et al., 1998). A case-based help desk system used to diagnose computer problems is described by Göker et al. (1998). A diagnosis task is similar to a classification task, however, it is also coupled with some significant differences.

In a diagnosis situation one wants to find out the reason for an observed malfunction of a complex system like a technical machine, software, or a biological organism.

[3]One speaks about compositional adaptation if a solution is composed of solutions obtained from several cases.

[4]Special techniques to handle these problems (e.g. tree pruning, determination of probable values for unknown attributes) have been developed here, too.

Of course, the reason of such a malfunction can also be interpreted as a form of class membership. However, in a diagnosis situation one is also interested in a way to remedy the malfunction, also called *therapy*. Another important difference to classification is the fact that it is often difficult to obtain the necessary information about the current problem situation, also called *symptoms*. For example, in medical diagnosis one has to perform sophisticated examinations (e.g., blood examination) to acquire this information. Therefore, one concern of diagnosis systems is to minimise the amount of required information about symptoms while ensuring high diagnosis accuracy.

2.3.3. Knowledge Management

In the last years a new interdisciplinary research field called *Knowledge Management* has found a growing attention in the business world. The issue of knowledge management is a systematic collection, storage, sharing, and maintenance of knowledge that is relevant for business processes of organisations and companies (Liebowitz, 1999). By managing such knowledge in a systematic and computer-aided way, one expects to increase the efficiency of the business processes in order to gain competitive advantages.

When comparing the issues of knowledge management with CBR, one can recognise that both disciplines have a lot in common. Principally, CBR can be considered as a technique to manage a specific form of knowledge, namely situation-specific experience knowledge. Hence, the CBR methodology can also be characterised as *Experience Management* (Bergmann, 2002). However, when applying CBR functionality to provide "intelligent" computer-aided techniques for knowledge management, some specific characteristics of this application domain have to be taken into consideration:

- The experience knowledge used in business processes does often not comply with the problem solution distinction assumed by the traditional CBR assumption.
- The CBR functionality usually has to be integrated into the existing IT-infrastructure and superior organisational processes.
- Knowledge management systems are commonly used by many users who possess very different experiences and demands when working with the system.
- The users of knowledge management systems are often enabled to enter new knowledge into the system by themselves due to their expertise in the underlying domain. However, they are usually not familiar with the internal knowledge representation of the systems.

In particular, in the research field *Software Engineering* the deployment of CBR concepts in order to improve software development processes has become very popular (Tautz and Althoff, 1997). Here, experiences collected in software development processes of the past are reused in order to improve the efficiency of ongoing processes. One such approach of Software Engineering is called *Experience Factory* (Basili et al., 1994; Althoff et al., 1997) that establishes a framework for managing all kinds of experience knowledge being are relevant for software development projects.

2.3.4. Electronic Commerce

Today, the internet is becoming increasingly a widely used platform for selling information, services, and goods. *Electronic Commerce (eCommerce)* offers both, companies and customers new opportunities for trading. The major advantages of the new medium are, in principle,

- the availability around the clock,
- the irrelevancy of the physical location of the business partners,
- easier possibilities to provide or take up personalised services,
- the chance to save both, money and time.

However, in practice these principle advantages lead to big challenges for the development of the required software tools. Besides security and performance issues, intelligent methods to automate the traditional sales processes are required (Segovia et al., 2002).

Product Recommendation Systems

Traditional sales processes are often characterised by an intensive interaction between the seller and the customer and can typically be divided into three phases: *pre-sale*, *sale*, and *after-sale* (Wilke, 1999). Particularly, the automation of the presales phase in an electronic shop requires the development of intelligent software agents able to take the place of human shop assistants, which advise customers during the selection of adequate products or services.

Recently, CBR turned out to be a suitable and commercially very successful approach to implement such *product recommendation systems* (Wilke, 1999; Wilke et al., 1998; Bergmann et al., 2002; Burke, 1999). Here, descriptions of available products are considered to be the cases and the wishes and demands of customers are interpreted as the problems to be solved. This means, the task of the CBR system is to find a product description that fits the individual wishes and demands

of a particular customer as well as possible (Schmitt, 2003). Because the perfect product is usually not available or even does not exist at all, the system has to find an available product that is most similar to the perfect product.

Depending on the products to be sold, different CBR techniques have to be employed to handle this task. On the one hand, if the products are fixed and cannot be changed (e.g., books or standard music CDs) a similarity-based retrieval might be sufficient to select appropriate products. On the other hand, if the products allow modifications to customise them with respect to the particular customer demands (e.g., cars or personal computers), adaptation functionality might be required to adapt given base products. Depending on the extend of the provided modification possibilities, different adaptation strategies are suitable, for example, simple rule-based modifications (Schmitt and Bergmann, 1999a) or more sophisticated configuration procedures (Stahl and Bergmann, 2000).

The E-Commerce scenario is also associated with several specific characteristics similar to the characteristics of the knowledge management scenario discussed in the previous section. Generally, the following aspects have to be considered when implementing intelligent customer support systems based on CBR:

- The absence of a clear distinction between problem and solution parts of case knowledge. Instead, cases might be seen as solutions only, namely product descriptions.
- Intelligent customer support systems are used by many different users (here, customers) who may have implicit and individual preferences with respect to the offered products.
- The language of the customers used to describe their needs and wishes may differ from the language used by the providers to describe their products.
- Customers may have problems to describe their needs and wishes exactly and sometimes they even do not know them in detail a-priori.
- The interaction between the sales agent and the customer has to be minimised, i.e., suitable products have to be offered as fast as possible to achieve best customer satisfaction (Schmitt, 2003).

Collaborative Filtering

In the last years another technique called *collaborative filtering* has become very popular to realise product recommendation systems. This approach has obvious similarities with the CBR idea, even if it possesses also its own characteristics. A detailed comparison of the two approaches is given, for example, by Hayes et al. (2001).

Using collaborative filtering, the product recommendations for a particular customer are based on recommendations of other customers who are believed to have similar preferences. Therefore, the set of products preferred by the individual customers has to be compared to determine correlations in the customer's interests. After detecting groups of customers who have all shown an interest in similar products, the set of these preferred products can be used to recommend new products to the members of these groups. For example, in Figure 2.6 three customers have shown an interest in products B, C, and D and rejected product E. That might be an evidence they have a similar taste with respect to the offered products. Because of Customers 1 and 2 have also preferred the product F, Customer 3 might be interested in this product, too.

	A	B	C	D	E	F
Customer 1	-	+	+	+	-	+
Customer 2	+	+	+	+	-	+
Customer 3	?	+	+	+	-	?

Figure 2.6.: Product Recommendation based on Collaborative Filtering

One of the major strengths of the collaborative filtering approach is that it allows to build powerful recommendation systems without needing formal representations of the products being recommended. This is in particular an advantage, if it is very difficult to extract product properties being relevant for the buying decision of the customers. Typical examples are books, music CDs, or films where interest in products mainly depends on subjective tastes of the customers. However, traditional collaborative filtering approaches have problems to provide good quality recommendations for customers who want to buy a type of product for the very first time.

Recent work has shown that it may be useful to combine CBR and collaborative filtering in order to profit of the advantages of both when implementing product recommendation systems (O'Sullivan et al., 2002). From the CBR point of view, a profile of a customer together with her/his preferred products can be considered to be a case. In order to advise a product to a customer her/his profile has to be compared to the profiles of other known customers. The products preferred by very similar customers according to the given profiles are then candidates to be recommended to the current customer.

2.3.5. Configuration and Design

A traditional application field of AI systems are *configuration* and *design* tasks. Here, knowledge-based systems are used to support the generation of complex entities (e.g., technical systems or buildings in architecture) that have to fulfill a given

specification. The common procedure for generating such entities is a composition and adaptation of different primary entities, called *components*. Because usually not all compositions theoretically possible represent valid entities in the underlying domain, special *constraints* have to be considered during the configuration procedure. For example, the general functionality of a technical system has to be guaranteed.

Although only a few commercial successful applications have been developed yet, several research prototypes have already proven the suitability of CBR to build efficient configuration systems. A detailed description of a case-based configuration system to be applied in the E-Commerce scenario is described by Stahl (2000); Stahl and Bergmann (2000); Stahl et al. (2000).

2.3.6. Planning

As configuration and design, *planning* also belongs to the traditional *synthetic tasks*[5] of AI. Generally, a planning problem is given by an *initial state*, a *goal state* and a set of *planning operators*. The objective of the planning process is to reach the goal state by consecutively applying operators on the initial state. A series of planning operators is called a *plan*.

Similar to the configuration scenario, CBR can be used, in principle, to build efficient planning systems. Here, the basic idea is to reuse existing plans in order to improve the generation of new plans in similar situations. However, because of the complexity of planning tasks, adaptation is a crucial aspect of every case-based planning system. This is one of the reasons why case-based planning is still in focus of research only.

2.4. CBR and Machine Learning

Case-based Reasoning is not only an approach to problem-solving but also provides the possibility to build computer systems that are able to *learn* new knowledge during usage. The original learning capability of CBR is operationalised by the retain phase of the CBR cycle (see Section 2.2.2). Here, new knowledge acquired during problem-solving is added to one knowledge container (see Section 2.2.3), namely the case base.

In this section a more general overview of aspects concerning learning in CBR is given. After a short discussion of the foundations of learning, approaches to realise learning functionality in CBR systems are discussed more detailed.

[5]Synthetic tasks require the active generation of a solution in contrast to *analytic tasks* like classification, where a solution consists of an analysis of a given problem only.

2.4.1. Foundations

To be able to talk about learning possibilities of CBR systems, first we have to clarify the interpretation of a *learning computer system*. In the field of *Machine Learning* learning systems are commonly defined as follows (Mitchell, 1997):

Definition 2.9 (Learning System) A computer system is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

So, for example, a case-based system used for a classification task might improve its performance measured by classification accuracy by collecting experiences obtained when classifying new entities.

This general definition states nothing about the way how to achieve the described learning ability. In a CBR system we assume that the improvement of the system's performance is based on changes in the employed knowledge rather than on changes in the employed base algorithms. Therefore, it is useful to introduce a specialised definition for learning CBR systems:

Definition 2.10 (Learning CBR System) A CBR system is said to *learn*, if its performance at tasks of some class of tasks T, measured by a given performance measure P, improves through changes in the knowledge containers triggered by experience E.

To achieve an improvement of a CBR system's performance, two different kinds of changes in the knowledge containers can be distinguished:

1. Changes in one container might be performed independently from the other containers, by *adding or removing knowledge* items. Such modifications obviously change the amount of knowledge available for problem-solving.

2. *Shifting knowledge* from one container to another container might be useful, too. This means, the amount of available knowledge is not changed, however, parts of the knowledge are represented in a different form.

In Section 2.2.3 the general possibility to shift knowledge from one container to another one has already been discussed. There, the focus was laid on the development phase of a CBR application. However, shifting knowledge as a result of learning is usually performed automatically during the CBR system's lifetime.

In Machine Learning another aspect concerning the application of acquired knowledge is important. Basically, two contrary learning methods are distinguished (Mitchell, 1997):

Lazy Learning: Lazy learning methods defer the generalisation required to solve problems beyond the presented training data until a new problem is presented. Such a procedure leads to a reduced computation time during training but to an increased computational effort when solving new problems.

Eager Learning: On the contrary, eager learning methods perform the mandatory generalisation before new problems are presented to the system by constructing a hypothesis about the appearance of the unknown target function of the domain. After constructing a hypothesis of the target function based on some training data, new problems might be solved very efficiently with respect to computation time. However, eager methods cannot consider the current problem during the generalisation process.

2.4.2. Collecting and Maintaining Case Knowledge

Obviously, learning according to the traditional CBR cycle is a form of lazy learning. Here, the training data—given in form of cases—is only stored during the training phase. How to use this data to solve new problems is not decided until such a new problem is presented to the system.

Several algorithms to realise this original learning approach of CBR have been developed very early (Aha, 1991). These *case-based learning (CBL)* algorithms, which mainly focus on traditional classification tasks, can be summarised as follows:

CBL1: This is the most simple algorithm. Here, all presented cases are stored in the case base.

CBL2: The aim of this algorithm is to avoid storage of irrelevant cases. Only cases classified incorrectly using already stored cases are added to the case base. However, the success of this strategy also relies on the cases' presentation order. Hence, this strategy might cause classification failures in future problem-solving situations.

CBL3: This modification of the CBL2 algorithm also removes those cases from the case base that decrease the overall classification accuracy of the system. Therefore, CBL3 keeps track of the frequencies with which cases contribute to correct classifications. Cases coupled with significantly low frequencies are removed from the case base. However, removing cases might cause classification failures in the future, too.

The CBL3 algorithm can also be seen as a kind of maintenance technique because it administrates the case knowledge in order to preserve high classification accuracy with time when storing new cases. In the last years a lot of other approaches to case base maintenance have been developed. Such work can also be seen as a contribution to improve the retain phase and so the learning facilities of CBR. The aim of many of these techniques is to minimise the size of the case base while preserving the problem-solving competence (Smyth and Keane, 1995b; Smyth and McKenna, 1998; Leake and Wilson, 2000; Roth-Berghofer, 2002). Others also try to discover and eliminate inconsistencies within the case base (Reinartz et al., 2000)

similar to the CBL3 algorithm's objective that was originally introduced to handle noisy training data. Recent approaches now also consider other application fields like knowledge management or E-Commerce. Here, cases might be stored by several different users of the system which leads to the need of *collaborative maintenance* (Ferrario and Smyth, 2000).

A detailed investigation of theoretical aspects concerning case-based learning algorithms is given by Globig (1997). Here, it is discussed which concepts, in principle, can be learned by a case-based learning algorithm when dealing with classification tasks.

2.4.3. Learning General Domain Knowledge

According to the knowledge container model (see Section 2.2.3), cases are not the only domain knowledge employed in CBR systems. The additional general knowledge contained in the vocabulary, the similarity measure, and the adaptation knowledge is also important to ensure efficient problem-solving. In principle, it should also be possible to learn this general knowledge, too, even if the traditional CBR cycle does not operationalise this facility explicitly. In the following, we give a brief overview of the state-of-the-art concerning the learning of general knowledge in CBR. To motivate the work described in this thesis we discuss the aspect of learning similarity measures more detailed.

Learning the Vocabulary

As discussed in Section 2.2.3, the vocabulary represents the basis for all other domain knowledge incorporated in a CBR system. Therefore, high quality of this knowledge is absolutely essential to ensure reasonable problem-solving capabilities. For example, choosing the wrong attributes for characterising case knowledge will prevent accurate retrieval results, even if the similarity measure is quite sound.

Due to the fundamental character of the vocabulary, the development of strategies to learn it is a really hard task. Nevertheless, basically, two operations to improve an initially given vocabulary can be distinguished:

- Many works in Machine Learning have shown that *removing irrelevant attributes* can increase the accuracy of classifiers (e.g., decisions trees, nearest-neighbour and neural-network classifiers) significantly.

- In particular, the CBR approach often requires the *introduction of additional attributes* to ensure reasonable retrieval results. Typically, these *virtual attributes* are used to represent important relations between other, already given attributes.

Virtual attributes are very important when applying CBR, because they may simplify the definition of adequate similarity measures significantly. They provide a possibility to avoid non-linear similarity measures by shifting the non-linearity to the definition of the vocabulary. For example, to classify rectangles with respect to the property "quadrat", it is important to consider particularly the ratio between the height and width, although the height and width do already describe a given rectangle completely. Without a virtual attribute that makes this relation explicit, the similarity measure would have to consider this crucial relation to enable correct classifications. This is an example for the possibility to shift knowledge between the vocabulary and the similarity measure.

Although, *feature selection* is a classic topic of Machine Learning, approaches to support the definition of an accurate case representation when developing a CBR application are very rare. Today, the acquisition of the vocabulary is usually still a creative process that can only be carried out appropriately with intensive help of domain experts. Nevertheless, in the future existing feature selection strategies developed to improve classifiers might be adapted to apply them in the more general CBR context.

Unfortunately, suitable approaches to facilitate the determination of crucial virtual attributes are rare. In the field of Machine Learning the branch of *constructive induction* aims on constructing accurate representations from given raw data. Due to the high combinatorial complexity[6] only learning strategies guided by human domain experts might be feasible.

Learning the vocabulary only seems to be suitable during the development phase of a CBR application. Because the representation of all other knowledge relies on the defined vocabulary, changing the vocabulary always necessitates maintenance of the other knowledge containers. Such a maintenance procedure is a complex and time-consuming task that cannot be automated completely (Heister and Wilke, 1997). Thus, one usually tries to avoid changes in the vocabulary during the lifetime of CBR applications as far as possible.

Learning the Similarity Measure

High quality case knowledge is meaningless until it cannot be retrieved in the right situation. Therefore, the similarity measure is obviously a very crucial aspect of every CBR system. As already discussed in Section 2.2.3, when defining similarity measures, one has to consider the specific circumstances of the particular application domain. Similar to the vocabulary, the definition of similarity measures is mostly a creative process requiring a certain understanding of the application domain. Hence, usually only domain experts are able to provide the knowledge to be encoded into the similarity measure.

[6]Note that new virtual attributes might be defined by arbitrary functions over existing attributes!

However, domain experts usually are not familiar with the formal representation of similarity measures used by the employed CBR tools. This leads to additional development effort because CBR experts are required to translate the informal knowledge provided by the domain expert into the formal "language" of the CBR system. In addition, domain experts often are able to detect similarities between different problem situations without being able to explain precisely the reasons.

Generally, approaches towards learning similarity measures might improve the similarity definition process. On the one hand, they might reduce the effort required to model the measures. On the other hand, learning might ensure high quality measures even in poorly understood domains or in situations where experienced domain experts are not available (e.g., because the associated personal costs cannot be justified).

In contrast to the vocabulary, similarity measures contain knowledge that is directly reuse-related. This means that changes in this knowledge container generally do not require changes in other containers. This allows a redefinition of similarity measures without major problems during the lifetime of a CBR application in order to adapt the system to changes in the environment. So, due to the relative independence of similarity knowledge, learning strategies might also facilitate periodical modifications of the similarity measures to ensure or even improve the problem-solving capabilities of a CBR system.

Basically, the following arguments motivate the development of methods for learning reuse-related knowledge:

- From the *cognitive point of view* (see Section 2.1.1) it is an obvious step to extend the learning facilities of CBR towards learning general knowledge, too. Experimental studies have shown that human beings do not only memorise situation-specific experiences but also improve their background knowledge with increasing problem-solving routine (Suzuki et al., 1992).

- When recalling the *functional benefits* of CBR discussed in Section 2.1.2, it is obvious that methods to learn general knowledge might increase the applicability of CBR. Firstly, improving reuse-related knowledge would, of course, increase the problem-solving efficiency, too. Secondly, if reuse-related knowledge could be acquired by learning approaches, the applicability of CBR in poorly understood domains might be increased additionally. And last but not least, learning would obviously also simplify the knowledge acquisition that would lead in turn to lower development costs for developing CBR applications.

- The introduction of well-founded learning strategies might establish a more *engineering-like procedure of similarity measure construction*. The creative procedure representing the state-of-the-art still complicates a widespread employment of the CBR technology.

- Particularly, in *non-traditional application fields* like Knowledge Management and E-Commerce (see Sections 2.3.3 and 2.3.4) in some situations learning approaches are probably the only feasible way to acquire accurate reuse-related knowledge. This issue will be discussed in Chapter 6 in more detail.

Although the advantages of incorporating learning strategies to acquire reuse-related knowledge are obvious, and although several researchers have already postulated the need for such strategies (Richter, 2003, 1992; Leake, 1994; Leake et al., 1997b; Aha and Wettschereck, 1997), general approaches to learning similarity measures are still very rare. A detailed review of existing work will be given in Chapter 9.

One reason for the absence of general feasible approaches might be the "unusual" learning task. Due to the very specific formalisms used to represent similarity knowledge (see Section 3.3) and due to the specific role of this knowledge, applying common Machine Learning strategies directly is difficult. The objective of this work is the development of a framework and particular algorithms enabling CBR systems to learn similarity knowledge.

Learning Adaptation Knowledge

The most aspects discussed concerning learning of similarity measures also hold for the second container containing reuse-related knowledge, namely adaptation knowledge.

However, one difference can be noticed: In contrast to similarity knowledge, adaptation knowledge usually is described in form of a common representation formalism, namely rules. Although several approaches to learn rules have been developed in Machine Learning, only few strategies to learning adaptation knowledge in the CBR context can be found in literature.

For example, Wilke et al. (1996) present a general framework for learning adaptation knowledge but they do not discuss concrete learning algorithms. Further, Hanney and Keane (1996, 1997) have developed a general approach to learn adaptation rules from case knowledge and Leake et al. (1996b) present an approach that learns adaptation knowledge in form of "adaptation cases".

3. Similarity Measures

When recalling the basic assumption of CBR (cf. Section 1.1), it is obvious that the term *similarity* is crucial here. In order to employ the assumption for building knowledge-based systems, a formalisation of the term similarity, that can be processed by computer systems, is required. In practice, so-called *similarity measures* are used to compute the similarity between queries and cases.

In this chapter, the role of similarity measures for case-based problem-solving is discussed in detail. First, the basic task of similarity measures, namely the retrieval of useful cases, is recapitulated. After describing fundamental properties of similarity measures, approaches to represent them in computer systems are introduced. Further, two different views on similarity measures regarding the role of domain knowledge are discussed. The last section deals with the state-of-the-art procedure of modelling similarity measures in CBR systems and the drawbacks caused by this procedure.

3.1. The Task: Retrieving Cases

When initiating a case-based problem-solving process, the first phase is the retrieval of useful cases providing solutions to be reused easily for solving the problem at hand (see Section 2.2.2).

3.1.1. Exact Match Vs. Approximation

The need for retrieving information is not an exclusive issue of CBR. Retrieval mechanisms are also a central topic in traditional *Database Management Systems (DBMS)* and in the research field of *Information Retrieval (IR)*. However, the retrieval goals are quite different:

- In traditional database applications queries[1] are used to express hard conditions with respect to the expected retrieval results. This means, only an *exact match* between the query and a particular data set leads to the retrieval of this data set.

[1] In DBMS queries are usually expressed in a standardised query language like SQL (Structured Query Language).

- In Information Retrieval as well as in CBR, queries are processed in a more flexible manner. Here, in principle, all data sets *approximately matching* a given query may appear in the retrieval result. So, the retrieval goal is the selection of a certain number of data sets matching the query as exactly as possible.

The main difference between these two different retrieval strategies lays in the expectations of the users. In a DBMS the users usually know exactly in what kind of data sets they are interested in. Hence, they are able to define a query that describes their information need explicitly. On the one hand, if the corresponding retrieval result is empty, the users will accept this fact because it gives them the information that the searched data is not available at all. On the other hand, the users are usually interested in all matching data sets, even if their number is really huge.

In IR and CBR systems the users are in a different situation. Here, mostly they either do not precisely know what they are searching for, or they are not able to express their information needs explicitly. Moreover, they are often reliant on some results even if the returned information is slightly different to the query. So, on the one hand, if the database contains no data matching their query exactly, the users will still be interested in alternative data sets being quite close to the given query. On the other hand, the users are mostly not interested in arbitrary large retrieval results. They rather expect that the system provides a ranking of the retrieved data corresponding to its relevance regarding the query. Such a ranking allows the users to decide more easily which information might be useful for them and which not.

To estimate the relevance of data sets both IR and CBR systems try to estimate a form of mathematical distance between the query and the data sets. However, the difference between IR and CBR approaches is the amount of employed background knowledge and the way how this knowledge is used to compute such distances. While IR approaches are based on keyword matching between textual documents, CBR systems usually deal with more structured data (cf. Section 2.2.4) allowing more sophisticated retrieval approaches. In the following, we focus on the specific retrieval task of CBR systems.

3.1.2. Approximating Utility of Cases

The basic task of the retrieval phase in CBR is the selection of cases being useful for the current problem-solving process. In the following, we assume the existence of some case model $C = (D, L) \in \hat{C}$ according to Definition 2.4. This leads us to the following formalisation of the term *utility*:

Definition 3.1 (Utility Function) A function $u : \mathbb{D}_D \times \mathbb{C}_C \longrightarrow \mathbb{R}$ is called *utility function* on the case space $\mathbb{C}_C$.

Such a utility function assigns a value from the set of Real numbers to a case c and a case characterisation d (the query). This value represents the utility of c with respect to d. Of course, a utility function depends on the underlying case model C. However, for one case model C there might exist an arbitrary number of different utility functions $u_1, u_2, \ldots, u_n$. A utility function induces the following preference relation:

Definition 3.2 (Preference Relation Induced by Utility Function) Given a case characterisation d, a utility function u induces a *preference relation* $\succeq_d^u$ on the case space $\mathbb{C}_C$ by $c_i \succeq_d^u c_j$ iff $u(d, c_i) \geq u(d, c_j)$.

Generally, the utility of cases and so the underlying utility function may be influenced by several different aspects, for example, by

- the underlying domain and the application scenario addressed,
- the provided problem-solving functionality of the CBR system employed,
- the knowledge contained in the different knowledge containers of the CBR system,
- the preferences of all users, individual users, or groups of users,
- the point of time of the problem-solving situation, etc.

Such influences on utility functions will be discussed in more detail in Chapter 6.

The basic problem of utility functions is that they are usually only partially known and thus, cannot be computed a-priori, i.e. before a particular case has been reused. In CBR similarity measures are used to approximate the unknown utility function. This means, the quality of the case retrieval—and so the outcome of the first phase of the CBR cycle—strongly depends on the quality of the used similarity measures. If the similarity measure does not approximate the underlying utility function sufficiently, the retrieval phase might select cases not reusable at all, although useful cases are principally available in the case base. Because the case retrieval lays the foundation of the entire CBR cycle (cf. Section 2.2.2), insufficient retrieval results will strongly restrict a CBR system's problem-solving competence and performance.

Today, in most commercial CBR applications the importance of the similarity measure is even increased. Here, the benefit of the applications is mostly completely determined by the quality of the case retrieval. Because existing commercial applications mostly do not provide case adaptation mechanisms, only the quality of the available case knowledge and the capability to select the most useful cases is responsible for the quality of the output.

3.1.3. Similarity-Based Retrieval

To estimate the utility of a given case c for a given query q, the case characterisations of c and q have to be compared by a similarity measure generally defined as follows:

Definition 3.3 (Similarity Measure, General Definition) A *similarity measure* is a function $Sim : \mathbb{D}_D \times \mathbb{D}_D \longrightarrow [0, 1]$.

Assume a given query q, a case base CB, and a similarity measure Sim. The aim of a similarity-based retrieval can be illustrated as shown in Figure 3.1. By computing the similarity between q an the case characterisations of the cases contained in CB, the retrieval mechanism has to identify a list of cases, called *retrieval result*, ordered by the computed similarity values. The number of cases to be retrieved may be specified by one of the following parameters:

- An integer value specifying the *maximal number of cases* to be retrieved.
- A real value specifying a *similarity threshold*. This threshold defines the least similarity value required for some case c to appear in the retrieval result.

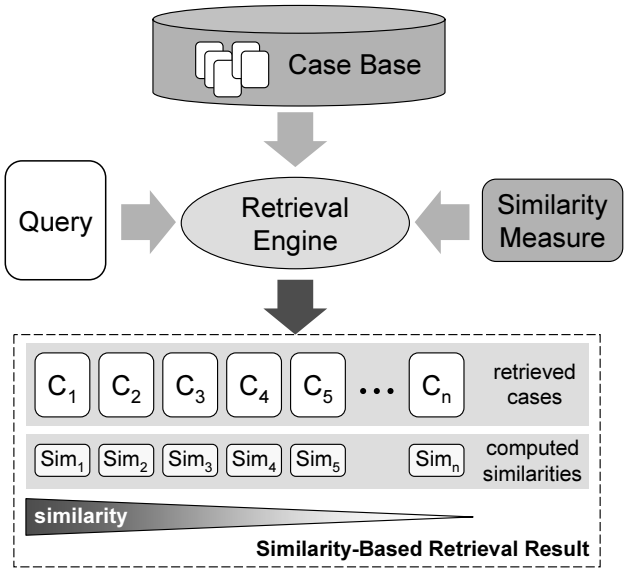


Figure 3.1.: Similarity-Based Retrieval

Now we are able to introduce a formal definition for a similarity-based retrieval result. Note that this definition states nothing about the particular retrieval algorithm used to obtain this retrieval result.

Definition 3.4 (Similarity-Based Retrieval Result) Let $C = (D, L) \in \hat{C}$ be a case model, $q = (d_q, l_q) \in \mathbb{C}_C$ be a query, $CB = \{c_1, c_2, \ldots, c_m\}$ be a case base containing cases $c_i = (d_i, l_i)$, and Sim be a similarity measure. A *similarity-based retrieval result* based on Sim is an ordered list $RR^x_{Sim}(q) = (c_1, c_2, \ldots, c_n)$ with $Sim(d_q, d_i) \geq Sim(d_q, d_j)$ for all $0 \leq i \leq j \leq n$ fulfilling the following conditions:

- either $x > 0$ is an integer value and it holds $|RR^x| = x$ if $x \leq |CB|$ or $|RR^x| = |CB|$ if $x > |CB|$. The retrieval result is then called *size limited* and x is called *maximal retrieval result size*,
- or $x > 0$ is a real value and it holds $Sim(d_q, d_{retrieved}) \geq x$ for all $c_{retrieved} \in RR^x$. The retrieval result then is called *threshold limited* and x is called *similarity threshold*,
- or x is undefined and it holds $|RR| = |CB|$. The retrieval result then is called *unlimited*.

Sometimes it cannot be guaranteed that retrieval results are always complete according to the following definition:

Definition 3.5 (Completeness of Retrieval Results) Assume $RR^x_{Sim}(q)$ to be defined as in Definition 3.4. RR^x is *complete* with respect to the given case base CB if it holds $Sim(d_q, d_{retrieved}) > Sim(d_q, d_{not\_retrieved})$ for all $c_{retrieved} = (d_{retrieved}, l_{retrieved}) \in RR^x$ and for all $c_{not\_retrieved} = (d_{not\_retrieved}, l_{not\_retrieved}) \in CB \setminus RR^x$.

Whether the completeness[2] of retrieval results can be ensured or not depends on the applied retrieval algorithm. Over the years, several retrieval algorithms have been developed to guarantee efficient retrieval of similar cases (Lenz, 1999; Wess, 1993; Schaaf, 1996). For example, Schumacher and Bergmann (2000) present a specific retrieval strategy that abandons the completeness property due to performance issues. Such specialised algorithms are in particular necessary, if the retrieval has to deal with huge case bases containing several thousands of cases. However, in this work we imply the completeness of retrieval results as long as not pointed out differently. This is not a crucial restriction since many retrieval algorithms applied in practice ensure completeness.

3.2. Mathematical Foundations

After having formulated the general task of similarity measures in the previous section, this section deals with some basic mathematical foundations concerning similarity measures. First, different ways to represent similarity formally are presented. Further, the relation between distance measures and similarity measures is investigated. After that, some general properties of similarity measures are discussed.

[2]Note that unlimited retrieval results are inevitably complete.

3.2.1. Formalising Similarity

Besides the formalism introduced in Definition 3.3, other mathematical ways to represent similarity can be defined. For example, Richter (1992) enumerates the following possibilities:

1. A binary predicate $SIM(x, y) \subset \mathbb{D}_D^2$ meaning "x and y are similar".

2. A binary predicate $DISSIM(x, y) \subset \mathbb{D}_D^2$ meaning "x and y are not similar".

3. A ternary relation $S(x, y, z) \subset \mathbb{D}_D^3$ meaning "y is at least as similar to x as z is to x".

4. A quaternary relation $R(x, y, u, v) \subset \mathbb{D}_D^4$ meaning "y is at least as similar to x as u is to v".

5. A function $sim(x, y) : \mathbb{D}_D \times \mathbb{D}_D \longrightarrow [0, 1]$ measuring the degree of similarity between x and y. This possibility corresponds to our formalism introduced in Definition 3.3.

6. A function $d(x, y) : \mathbb{D}_D \times \mathbb{D}_D \longrightarrow \mathbb{R}^+$ measuring the distance between x and y.

Obviously, these formalisms contain in increasing order more and more information about the similarity of case characterisations. On the one hand, the two binary predicates SIM and $DISSIM$ are quite simple formalisms that cannot be used to compute ranked retrieval results. Hence, they are insufficient to be used in CBR systems. On the other hand, the two relations S and R are in principle already sufficient to rank cases. For example, S allows to define the concept "y is most similar to x" for some set of case characterisations $M \subset \mathbb{D}_D$:

$$(\forall z \in M) \quad S(x, y, z)$$

The relation R can be used to induce the relations S by $S(x, y, z) \leftrightarrow R(x, y, x, z)$. While S and R contain only *ordinal* information about similarity, the two functions sim and d also contain *cardinal* information. Although ordinal information is mostly sufficient, CBR implementations usually use similarity or distance functions. Of course, these functions also induce the similarity relations, for example sim induces S and R as follows:

$$S(x, y, z) \leftrightarrow sim(x, y) \geq sim(x, z)$$

$$R(x, y, u, v) \leftrightarrow sim(x, y) \geq sim(u, v)$$

Generally, distance measures are a dual notation to similarity measures because a given distance measure d can be transformed to a similarity measure sim via an accurate function f:

$$sim(x, y) = f(d(x, y))$$

Popular candidates for f are, for example, $f(z) = 1 - (z)/(1 + z)$ for unbounded d or $f(z) = 1 - (z)/(max)$, if d attains a greatest element max. Due to this dualism, in the following we restrict the discussion on similarity measures only.

3.2.2. Properties of Similarity Measures

First, some basic properties of similarity measures are described. After that, it is discussed how soundness of similarity measures might be defined.

Basic Properties

In general, we do not assume that similarity measures necessarily have to fulfill general properties beyond the basic Definition 3.3. Nevertheless, in the following we introduce some definitions of basic properties typically be fulfilled by similarity measures.

Definition 3.6 (Reflexivity) A similarity measure is called *reflexive* if $Sim(x, x) = 1$ holds for all x. If it additionally holds $Sim(x, y) = 1 \rightarrow x = y$, Sim is called *strong reflexive*.

Reflexivity is a very common property of similarity measures. It states that a case characterisation is maximal similar to itself. From the utility point of view, this means, a case is maximal useful with respect to its own case characterisation. Therefore, similarity measures might violate the reflexivity condition, if a case base contains sub-optimal cases. Similarity measures are usually not strong reflexive, i.e. different cases may be maximal useful regarding identical queries. For example, different solution alternatives contained in different cases might be equally accurate to solve a given problem.

Definition 3.7 (Symmetry) A similarity measure is called *symmetric*, if it holds $Sim(x, y) = Sim(y, x)$ for all x, y. Otherwise it is called *asymmetric*.

Symmetry is a property often assumed in traditional interpretations of similarity. However, in many application domains it has been emerged that an accurate utility approximation can only be achieved with asymmetric similarity measures. The reason for this is the assignment of different roles to the case characterisations to be compared during utility assessment. Usually, the case characterisation representing the query has another meaning than the case characterisation of the case to be rated.

Definition 3.8 (Triangel inequality) A similarity measure fulfills the triangle inequality, if $Sim(x, y) + Sim(y, z) \leq 1 + Sim(y, z)$ holds for all x, y, z.

The triangle inequality is usually demanded for distance measures only and is required to ensure the property of a metric. However, due to the dualism of similarity and distance measures it can also be formulated for similarity measures by applying an accurate transformation.

Definition 3.9 (Monotony, General Definition) Let $C = (D, L) \in \hat{C}$ be a given case model and Sim be a similarity measure according to Definition 3.3. Further, assume the existence of an order relation $<_{\mathbb{D}_D}$ defined over $\mathbb{D}_D$. Sim is called monotonic, if it holds $Sim(x, y) \geq Sim(x, z)$ for $x <_{\mathbb{D}_D} y <_{\mathbb{D}_D} z$ or $z <_{\mathbb{D}_D} y <_{\mathbb{D}_D} x$.

The monotony property, which can be characterised as a kind of "compatibility" to the ordering on $\mathbb{D}_D$ (if existing), is an important aspect when modelling similarity measures in practice. We will discuss this issue in more detail in Section 3.3.

Soundness of Similarity Measures

Like utility functions, which induce the preference relation introduced in Definition 3.2, a similarity measure also induces a preference relation:

Definition 3.10 (Preference Relation Induced by Similarity Measure) Given a case characterisation d, a similarity measure Sim induces a *preference relation* $\sqsubseteq_d^{Sim}$ on the case space $\mathbb{C}_C$ by $c_i = (d_i, l_i) \sqsubseteq_d^{Sim} c_j = (d_j, l_j)$ iff $Sim(d, d_i) \leq Sim(d, d_j)$.

These preference relations can now be used as a foundation to define correctness criteria for similarity measures. According to Bergmann (2002), the soundness of a similarity measure Sim can be defined on different levels of generality:

1. Total soundness w.r.t. the complete domain, if Sim orders all possible cases correctly according to a given utility preference relation.

2. Total soundness w.r.t. a given case base CB, if Sim orders all cases contained in CB correctly according to a given utility preference relation.

3. Partial soundness w.r.t. a given case base CB, if Sim orders the "most useful" cases of CB correctly according to a given utility preference relation.

Because these general soundness criteria are difficult or even impossible to measure in practice, in this work we will focus on measurable and more expressive criteria. Basically, we are interested in retrieving the most useful cases regarding to some utility function u. However, depending on the concrete application scenario, minor retrieval errors can be tolerated. This leads to the following definitions:

Definition 3.11 (Best-n List) Suppose a list of cases $CL = (c_1, c_2, \ldots, c_n, \ldots, c_m)$ partially ordered according to some preference relation $\succeq$, i.e. $\forall i, j\ c_i \succeq c_j$ holds. The list $CL^{best-n} = (c_1, c_2, \ldots, c_n, \ldots, c_r)$ so that $\forall c_i \in CL^{best-n}$, $\forall c_j \in CL \setminus CL^{best-n}\ c_i \succ c_j$ holds, is called *best-n list* of CL where n is a parameter to be determined. Further, $\forall n \leq i, j \leq r$ it holds: $c_i \not\succ c_r$ and $c_i \not\prec c_r$.

This definition states that the best-n list for some list of cases CL consists of the n most preferred cases $(c_1, c_2, \ldots, c_n)$ of CL extended by all cases of CL being indistinguishable from c_n w.r.t. the underlying preference relation $\succ$. Figure 3.2 illustrates this exemplarily for a best-4 list. Here, the best-4 list consists of the 4 cases c_1, c_2, c_3, c_4 preferred mostly, and three additional cases c_5, c_6, c_7, since these cases are indistinguishable from c_4 w.r.t. $\succ$.

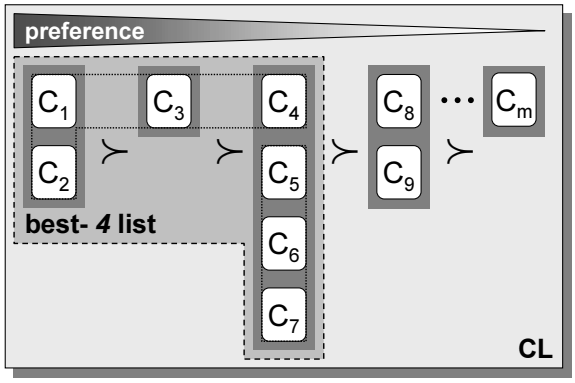


Figure 3.2.: The best-n List

Definition 3.12 (n-in-m Soundness of Retrieval Results) Let $q = (d, l)$ be a query, CB be a case base and Sim be a similarity measure. Further, let $RR_{Sim}(q)$ be the corresponding complete, unlimited retrieval result according to Definitions 3.4 and 3.5, and let $CB_u(q)$ be the partially ordered CB induced by the preference relation $\preceq_d^u$ according to Definition 3.2. $RR_{Sim}(q)$ is called *n-in-m sound* with $n \leq m$ iff the best-m list $RR_{Sim}^{best-m}(q)$ of $RR_{Sim}(q)$ includes the best-n list $CB_u^{best-n}(q)$ of $CB_u(q)$, i.e. iff it holds: $CB_u^{best-n}(q) \subseteq RR_{Sim}^{best-m}(q)$.

Informally spoken, a retrieval result is n-in-m sound, if the set of m most similar cases contains the n most useful cases. A quite hard criterion is the 1-in-1 soundness because it requires that the most useful case is also returned as most similar case. The best-n list criterion is required because we are not dealing with totally ordered lists but only with partial orders.

The n-in-m soundness of retrieval results can be used to evaluate the quality of the underlying similarity measure in practice. Therefore, one has to validate whether a given similarity measure leads to n-in-m sound retrieval results for a particular set

of queries. Because the soundness can mostly not be achieved for every query, one might measure the percentage of n-in-m sound retrieval results with respect to all considered retrievals.

3.3. Representing Similarity Measures

To be able to implement a similarity-based retrieval (see Definition 3.4) in a CBR system, one has to represent the necessary similarity measure in a way that a computer system can process it. Generally, accurate representations for similarity measures strongly depend on the used case representation and the defined vocabulary. In this section we introduce representation formalisms suitable for the attribute-value based case representation (cf. Section 2.2.4) presumed in this work. We review some common similarity and distance measures being used in many traditional CBR applications. After that, we introduce the *local-global principle* and show how this principle allows a decomposition of similarity measures to enable efficient modelling of similarity aspects when developing CBR applications. We also discuss the different components of similarity measures defined according to the local-global principle in more detail.

3.3.1. Traditional Measures

The similarity measures employed in many traditional CBR systems are often quite simple. They have not been developed especially for the purpose to be used in the scope of CBR, but they are founded on common mathematical principles and are also used in other scientific fields. In the following, we review some of these traditional measures.

Traditional Measures for Binary Attributes

The most simple attribute-value based case representation consists only of binary attributes, i.e. attributes with a binary value type, for example, the values $\{0, 1\}$. The following distance and similarity measures have been proposed for such case representations, where $\bar{x}, \bar{y}$ represent the attribute vectors to be compared:

$$dist_H(\bar{x}, \bar{y}) = \frac{1}{n} \cdot \mid \{i \mid x_i \neq y_i\} \mid \qquad \text{(Hamming Distance)}$$

The Hamming distance computes a distance value proportional to the number of attributes with different values. It fulfills the properties strong reflexivity, symmetry and the triangle inequality introduced in Section 3.2.2.

$$sim_H(\bar{x}, \bar{y}) = \frac{1}{n} \cdot \mid \{i \mid x_i = y_i\} \mid \qquad \text{(Simple Matching Coefficient, SMC)}$$

The simple matching coefficient (SMC) represents a similarity measure fulfilling the same properties as the Hamming distance. While both the Hamming Distance and the simple matching coefficient treat all attributes equally, the following generalisation of the SMC introduces weights to express the importance of individual attributes for the similarity assessment:

$$sim_{H,\vec{w}}(\bar{x},\bar{y}) = \sum_{i=1,\ x_i=y_i}^{n} w_i \qquad \text{with} \quad w_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} w_i = 1$$

(Weighted SMC)

A second generalisation of the SMC allows to assign equal and unequal attributes a different influence on the computed similarity by the introduction of an additional parameter $0 < \alpha < 1$:

$$sim_{H,\alpha}(\bar{x},\bar{y}) = \frac{\alpha \cdot sim_H(\bar{x},\bar{y})}{(\alpha \cdot sim_H(\bar{x},\bar{y})) + (1-\alpha)\cdot(1-sim_H(\bar{x},\bar{y}))} \quad \text{(Non-linear SMC)}$$

On the one hand, when choosing a value $\alpha < 0.5$, the influence of unequal attributes is weighted higher. On the other hand, a value of $\alpha > 0.5$ leads to higher weighting of equal attributes.

Another popular similarity measure for binary attributes is the Tversky contrast model (Tversky, 1977). This measure is again a generalisation of the SMC and allows to treat every value combination differently by introducing three parameters α, β and γ:

$$\begin{aligned} sim_{T,f,\alpha,\beta,\gamma}(\bar{x},\bar{y}) \quad = \quad & \alpha \cdot f(\{i \mid x_i = y_i = 1\}) - \\ & \beta \cdot f(\{i \mid x_i = 1 \wedge y_i = 0\}) - \\ & \gamma \cdot f(\{i \mid x_i = 0 \wedge y_i = 1\}) \end{aligned} \quad \text{(Tversky Contrast Model)}$$

Here, the function f measures the set of attributes with a particular value combination, for example, f might simply return the number of these attributes.

Traditional Measures for Numeric Attributes

Of course, case knowledge often also contains numeric data. The most traditional measures used to handle numeric attributes are distance measures in the mathematical sense. Popular examples of such distance measures are the following:

$$dist_{|\cdot|}(\bar{x},\bar{y}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \mid x_i - y_i \mid \qquad \text{(City Block Metric)}$$

$$dist_{Euklid}(\bar{x},\bar{y}) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - y_i)^2} \qquad \text{(Euclidean Distance)}$$

$$dist_{Max}(\bar{x}, \bar{y}) = max_{i=1}^{n} \mid x_i - y_i \mid \qquad \text{(Maximum Norm)}$$

Basically, these three metrics can be generalised by introducing a parameter p, leading to the Minkowski norm:

$$dist_{Minkowski,p}(\bar{x}, \bar{y}) = (\frac{1}{n} \cdot \sum_{i=1}^{n} \mid x_i - y_i \mid^p)^{\frac{1}{p}} \qquad \text{(Minkowski Norm)}$$

To express the different importance of different attributes, all mentioned metrics allow the introduction of attribute weights leading to the weighted Minkowski Norm:

$$dist_{Minkowski,p,\vec{w}}(\bar{x}, \bar{y}) = (\sum_{i=1}^{n} w_i \cdot \mid x_i - y_i \mid^p)^{\frac{1}{p}} \qquad \text{(Weighted Minkowski Norm)}$$

3.3.2. The Local-Global Principle

When dealing with more complex case representations consisting of attributes with various different value types, the previously described traditional similarity and distance measures are not appropriate. Instead one needs a more flexible similarity measure that can be adapted on a particular attribute-value based case representation. The foundation of such a similarity representation is the so-called *local-global principle*. According to this principle it is possible to decompose the entire similarity computation in a local part only considering *local similarities* between single attribute values, and a global part computing the *global similarity* for whole cases based on the local similarity assessments. Such a decomposition simplifies the modelling of similarity measures significantly and allows to define well-structured measures even for very complex case representations consisting of numerous attributes with different value types. In the following the different elements required to define similarity measures according to the local-global principle are discussed more detailed.

3.3.3. Local Similarity Measures

Local similarity measures represent knowledge about the utility of cases on a very low and detailed level. Basically, they are used to express the influence of each single attribute on the utility estimation. In this work, we assume the existence of a local similarity measure for each attribute of the case representation[3] according to the following general definition:

[3]In CBR systems local similarity measures are often not defined particularly for attributes, but for data types that may be assigned to several attributes. However, for simplicity reasons in this work we assume that each attribute has its own data type and so its own local similarity measure.

Definition 3.13 (Local Similarity Measure, General Definition) A *local similarity measure* for an attribute A is a function $sim_A : A_{range} \times A_{range} \longrightarrow [0, 1]$, where A_{range} is the value range of A.

This general definition still states nothing of how to represent local similarity functions in practice. Basically, an accurate representation strongly depends on the basic value type of the attribute. Therefore, in the following we introduce some representation formalism for different value types used commonly.

Local Similarity Measures for Discrete Value Types

CBR applications often have to deal with symbolic data represented by attributes with symbolic, discrete value ranges. For unordered symbol types, i.e. if no additional information about the relation between the defined symbols is available, the only feasible way to represent local similarities is an explicit enumeration in form of a lookup table:

Definition 3.14 (Similarity Table) Let A be a symbolic attribute with the value range $A_{range} = (v_1, v_2, \ldots, v_n)$. A $n \times n$-matrix with entries $s_{i,j} \in [0, 1]$ representing the similarity between the query value $q = v_i$ and the case value $c = v_j$ is called a *similarity table* for A_{range}.

A similarity table represents a reflexive measure, if the main diagonal consists of similarity values $s_{ii} = 1$ only. Further, we call a similarity table symmetric if the upper triangle matrix is equal to the lower triangle matrix, i.e. if for all i, j $s_{ij} = s_{ji}$ holds.

Figure 3.3 shows an example of a similarity table for the attribute casing of the personal computer example domain. This table represents the similarities between different kinds of computer casings, for example, it expresses that a mini- and a midi-tower are quite similar. However, the degree of similarity between these casings also depends on which value occurs as query, i.e. the similarity table is asymmetric. The semantics here is, that customers will probably be less satisfied with a mini-tower when demanding a midi-tower (sim(midi-tower, mini-tower) = 0.7), than in the opposite case (sim(mini-tower, midi-tower) = 0.9). The underlying assumption is that bigger casings would be tolerated due to the advantage of the greater number of extension slots.

A similarity table represents a very powerful representation because of the possibility to define separate similarity values for all possible value combinations. Nevertheless, the effort required to define such a measure increases quadratically with the number of values to be considered.

Generally, similarity tables can be used for all discrete value types where the value range is defined by an explicit enumeration of a finite set of values, i.e. the values need not necessarily be symbols. However, for an enumeration of numbers it

q \ c	laptop	mini-tower	midi-tower	big-tower
laptop	1.0	0.2	0.1	0.0
mini-tower	0.3	1.0	**0.9**	0.5
midi-tower	0.2	**0.7**	1.0	0.7
big-tower	0.1	0.4	0.6	1.0

Figure 3.3.: Similarity Table

is usually more accurate to use the available ordinal information in order to reduce the modelling effort.

Local Similarity Measures for Numeric Value Types

For numeric attributes similarity tables are usually not suitable, either because the value range contains an infinite number of values or because other representations simplify the similarity definition dramatically. In order to reduce the modelling effort, one can profit from the implicit ordering of numbers. A commonly used method is to reduce the dimension of the similarity measure by defining it on the difference between the two values to be compared. In contrast to the general 2-dimensional similarity functions, this approach results in a 1-dimensional function only:

Definition 3.15 (Difference-Based Similarity Function) Let A be a numeric attribute with the corresponding value range A_{range}. Under a *difference-based similarity function* we understand a function $sim_A : \mathbb{R} \longrightarrow [0,1]$ that computes a similarity value $sim_A(\delta(q,c)) = s$ based on some *difference function* $\delta : A_{range} \times A_{range} \longrightarrow \mathbb{R}$. Typical difference functions are

- the linear difference $\delta(q,c) = c - q$,
- or the logarithmic difference $\delta(q,c) = \begin{cases} ln(c) - ln(q) & \text{for} \quad q,c \in \mathbb{R}^+ \\ -ln(-c) + ln(-q) & \text{for} \quad q,c \in \mathbb{R}^- \\ undefinied & \text{else} \end{cases}$

The foundation of such difference-based similarity functions is the assumption that the decrease of similarity stands in some relation with increasing difference of the values to be compared. The identification of this relation and its formalisation by choosing an appropriate similarity function is the crucial task when modelling local similarity measures for numeric attributes. Typically, an accurate similarity function can be defined by combining some base functions f_1, f_2 for negative and

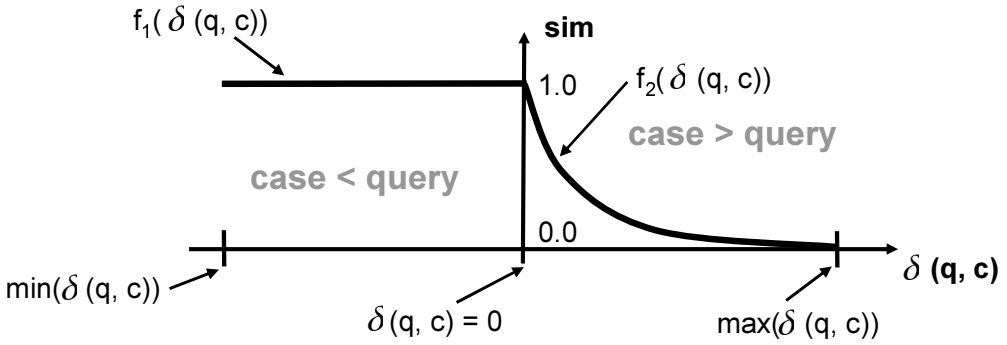


Figure 3.4.: Difference-Based Similarity Function

positive distances:

$$sim_A(q,c) = \begin{cases} f_1(\delta(q,c)) & : \ c < q \\ 1 & : \ c = q \\ f_2(\delta(q,c)) & : \ c > q \end{cases}$$

Here, we assume that the overall similarity function is reflexive due to $sim_A(q,c) = 1$ for $c = q$. Further, the base function f_1 is usually a monotonic increasing and f_2 a monotonic decreasing function. If $f_1(\mid \delta(q,c) \mid) = f_2(\mid \delta(q,c) \mid)$ holds, then sim_A is symmetric. The described structure of a difference-based similarity function is illustrated in Figure 3.4.

Figure 3.5 shows examples of base functions typically used, namely threshold, linear, exponential and sigmoid functions. All these base functions provide specific parameters to define the particular decrease of similarity:

$$sim(\delta(q,c)) = \begin{cases} 1 & : \ \delta(q,c) < \theta \\ 0 & : \ \delta(q,c) \geq \theta \end{cases} \qquad \text{(Threshold Function)}$$

$$sim(\delta(q,c)) = \begin{cases} 1 & : \ \delta(q,c) < min \\ \frac{max - \delta(q,c)}{max - min} & : \ min \geq \delta(q,c) \geq max \\ 0 & : \ \delta(q,c) > max \end{cases} \qquad \text{(Linear Function)}$$

$$sim(\delta(q,c)) = e^{\delta(q,c) \ \cdot \ \alpha} \qquad \text{(Exponential Function)}$$

$$sim(\delta(q,c)) = \frac{1}{e^{\frac{\delta(q,c) \ - \ \theta}{\alpha}} \ + \ 1} \qquad \text{(Sigmoid Function)}$$

When modelling local similarity measures in that way, the utility approximation can be influenced by the following parameters that have to be defined during the similarity assessment process:

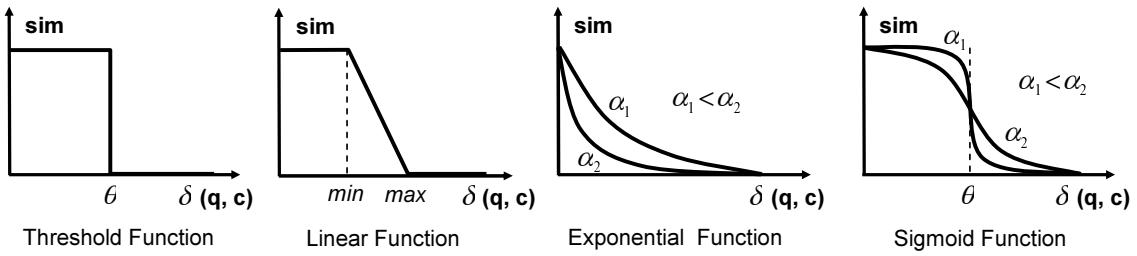


Figure 3.5.: Base Functions for Difference-Based Similarity Functions

- the difference function δ
- the base functions f_1 and f_2
- the parameters required by the chosen base functions

Difference-based similarity functions represent a comfortable way to model the similarity for numeric attributes and are sufficient to fit the requirements of most application domains. However, sometimes it might also be necessary to define more flexible functions according to the general Definition 3.13. Then one has to model a 2-dimensional function that is, of course, a more complex task compared to modelling a 1-dimensional function.

Local Similarity Measures for Structured Value Types

Generally, the similarity definition can be simplified, if it is possible to encode information being relevant for the similarity assessment into the used data type itself. This is in particular useful for symbolic types. Therefore, CBR systems often support the usage of the following structured symbolic types:

Ordered Symbol: Here, the user is allowed to define an order on the symbols to express relations among them.

Taxonomy Symbol: By arranging symbols in a taxonomy tree generalisation or specialization relations, respectively, can be expressed.

When dealing with such structured data types, the knowledge contained in the type definition is usually directly relevant for the similarity assessment. Hence, the underlying structure can be used to simplify the similarity definition. This is in particular useful for types containing a large number of symbols because then similarity tables are difficult to handle.

A way to take advantage of ordered symbols is to treat them like numbers. In the simplest case one assigns an Integer index to each single symbol according to its position in the order. In order to express more flexible “distances” between symbols

these indexes can also be manipulated by the user, i.e. the first symbol might get the index 1 while the index of the second symbol can be set to an arbitrary value greater 1, and so on. These index values can be used to compute the similarity in the same way as for a numeric type by defining an accurate difference-based similarity function.

A powerful approach to profit from the structure of a taxonomy tree during the similarity assessment is described by Bergmann (1998). We refrain from a description of this approach because it is not directly relevant for the work described here.

3.3.4. Attribute Weights

The second important part of similarity measures defined according to the local-global principle are attribute weights. They are used to express the different importance of individual attributes for the entire utility approximation.

Definition 3.16 (Attribute Weights) Let $D = (A_1, A_2, \ldots, A_n)$ be a case characterisation model. Then a vector $\vec{w} = (w_1, w_2, \ldots, w_n)$ with $w_i \in [0, 1]$ and $\sum_{i=1}^{n} w_i = 1$ is called *weight vector* for D, where each element w_i is called *attribute weight* for A_i.

Basically, the following approaches to define attribute weights can be distinguished:

Global Weights: This is the most general weight model where the importance of attributes is defined globally, i.e. the defined weights are valid for the entire application domain. Here, the influence of attributes on the utility approximation is constant for all cases and queries that may occur.

Case Specific Weights: This is a more fine-grained weight model that allows the definition of different attribute weights for different cases. This means, when comparing a query with a given case, a specific weight vector for this particular case is used to perform the similarity computation. A special form of this weight model are *class specific weights* used for classification tasks. Here, the weight vector to be used is determined by the class membership of the particular case.

User Weights: Another approach is the use of specific weights for each new retrieval task, i.e. the weights are acquired together with the query. Such a weight model is in particular useful in domains where the users might have individual preferences with respect to the utility of cases. For example, a product recommendation system in eCommerce might allow customers to input attribute weights in order to express the importance of particular product properties for her/his buying decision.

Different weight models can also be combined. For example, user weights are often not used exclusively, but they are combined with a global weight vector defining the general importance of attributes from the application domain point of view. Such a combination can be realised easily by a multiplication and a final normalisation of the used weights, for example, global weights $\vec{w}_{gobal}$ and user weights $\vec{w}_{user}$. Here, we assume that it holds $\vec{w}_{global} \not\perp \vec{w}_{user}$:

$$\vec{w}_{final} = (w_{final_1}, w_{final_2}, \ldots, w_{final_n}) = \frac{(w_{global_1} \cdot w_{user_1}, \ldots, w_{global_n} \cdot w_{user_n})}{(\vec{w}_{global}) \cdot (\vec{w}_{user})}$$

Although the effort to model attribute weights, at least global weights, is significantly lower compared to the effort required to model local similarity measures, defining attribute weights is a crucial and difficult task. On the one hand, small deviations in the defined weights may lead to very different retrieval results due to their great influence on the entire similarity computation. On the other hand, the determination of concrete weight values is usually more an act of instinct than a well-founded decision process. Although domain experts are often able to state that one attribute is more important than another one, mostly they have difficulties to determine the particular quantitative difference of the importance by assigning exact weight values.

3.3.5. Global Similarity Measure

In order to obtain a final similarity value when comparing a query and a case, the last element of the described similarity representation is the so-called *global similarity measure*. This measure is represented by an *aggregation function* computing the final similarity based on the local similarity values computed previously and the attribute weights defined:

Definition 3.17 (Global Similarity Measure, General Definition) Let $D = (A_1, A_2, \ldots, A_n)$ be a case characterisation model, $\vec{w}$ be a weight vector, and sim_i be a local similarity measure for the attribute A_i. A *global similarity measure* for D is a function $Sim : \mathbb{D}_D \times \mathbb{D}_D \longrightarrow [0, 1]$, of the following form:

$$Sim(q, c) = \pi(sim_1(q.a_1, c.a_1), \ldots, sim_n(q.a_n, c.a_n), \vec{w})$$

where $\pi : [0, 1]^{2n} \longrightarrow [0, 1]$ is called *aggregation function* that must fulfill the following properties:

- $\forall \vec{w} : \pi(0, \ldots, 0, \vec{w}) = 0$
- π is increasing monotonously in the arguments representing local similarity values

Basically, the described similarity computation that is founded on the local-global principle presumes a specific property of the underlying case model:

Definition 3.18 (Linear Independence of Case Model) If the utility of cases is not influenced by non-linear relationships between attribute values, we call a case model *linearly independent*.

In Section 2.4.3 we have already mentioned the possibility to introduce *virtual attributes* in order to guarantee the linear independence of a case model. The definition of the global similarity measure makes clear why shifting non-linear dependencies to the definition of the vocabulary is important. Because the aggregation function does not directly access attribute values, but only similarity values computed by the local similarity measures, this function cannot consider non-linear relations between attributes during the similarity computation. Further, it is assumed that the global similarity measure is monotonic in the sense that the global similarity only decreases, if at least one local similarity decreases.

In principle, the aggregation function π can be arbitrarily complex. However, in practice usually quite simple functions are used, for example:

$$\pi(sim_1, \ldots, sim_n, \vec{w}) = \sum_{i=1}^{n} w_i \cdot sim_i \qquad \text{(Weighted Average Aggregation)}$$

$$\pi(sim_1, \ldots, sim_n, \vec{w}) = (\sum_{i=1}^{n} w_i \cdot sim_i^p)^{\frac{1}{p}} \qquad \text{(Minkowski Aggregation)}$$

$$\pi(sim_1, \ldots, sim_n, \vec{w}) = max_{i=1}^{n}(w_i \cdot sim_i) \qquad \text{(Maximum Aggregation)}$$

$$\pi(sim_1, \ldots, sim_n, \vec{w}) = min_{i=1}^{n}(w_i \cdot sim_i) \qquad \text{(Minimum Aggregation)}$$

3.4. Semantics of Similarity Measures

As already described in Section 3.1, the purpose of similarity measures is an approximation of the cases' utility for particular problem situations. Hence, a similarity measure can also be characterised as a *heuristics* to select useful cases. Basically, such a heuristics can be more ore less sophisticated, depending on the amount of domain knowledge exploited. In this section two different interpretations of the semantics of similarity measures are discussed: the traditional interpretation of similarity in CBR, and a new view on the role of similarity measures inspired by application fields recently addressed by CBR.

3.4.1. Knowledge-Poor Similarity Measures

In traditional CBR applications the term similarity has often been interpreted as a kind of *similar look*. Similarity measures defined according to this semantics consider only syntactical differences between the entities to be compared. Popular examples are the traditional similarity and distance measures like the Hamming distance, the simple matching coefficient, the Euclidean distance, and other simple distance metrics already discussed in Section 3.3.1. If any, such measures consider only little domain knowledge during the similarity assessment. For example, the Hamming distance measures exclusively syntactical differences, while weighted measures at least consider the different importance of individual attributes. The generalised versions of the simple matching coefficient can be adapted on domain specific requirements, but the provided parameters represent very superficial knowledge about the domain only.

In the following, we call those measures *knowledge-poor similarity measures (kpSM)*. While such measures can be defined easily, the drawback of them is that they do not consider the particular coherences of the addressed application domain. Only measuring syntactical differences between descriptions of problem situations often leads to bad retrieval results due to an insufficient approximation of the cases' utility.

3.4.2. Knowledge-Intensive Similarity Measures

Because in many application domains an exclusive measuring of syntactical differences between problem descriptions is insufficient for obtaining reasonable retrieval results, many CBR applications developed in the last years employ more "intelligent" similarity measures. The general need for a more goal-directed retrieval functionality is discussed by Bergmann et al. (2001). By encoding more specific domain knowledge about the utility of cases into the similarity measure, both the efficiency and the competence of a CBR system can be improved significantly. On the one hand, the retrieval of a more useful case might lead to decreased adaptation effort and so to faster problem-solving. On the other hand, the competence to solve a given problem might be increased. For example, the competence of a retrieval-only system is strongly influenced by the ability to retrieve the most useful cases.

Besides attribute weights, particularly local similarity measures (see Section 3.3.3) provide the possibility to represent detailed knowledge about the attributes' contribution to the utility approximation. A similarity table defined for a symbolic type represents specific knowledge about the relationships between the defined symbols. A traditional knowledge-poor similarity measure would typically only perform an exact-match comparison between the values of a symbolic type.

Similarity functions can be used to compute the similarity between numbers based on domain specific criteria instead of only relying on information about the mathematical distance of the values. This even holds for the difference-based similarity

functions introduced in Definition 3.15. Here, the domain knowledge has to be represented in form of accurate 1-dimensional similarity functions. By selecting, combining and parameterising suitable base functions it is possible to consider the particular characteristics of the domain and the specific application scenario. In the following, we call similarity measures that try to approximate the utility of cases through the intensive exploitation of domain knowledge *knowledge-intensive similarity measures (kiSM).*

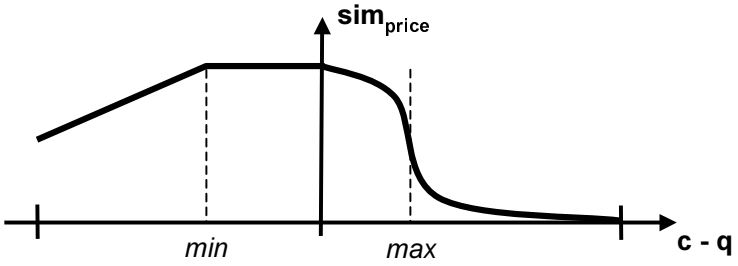


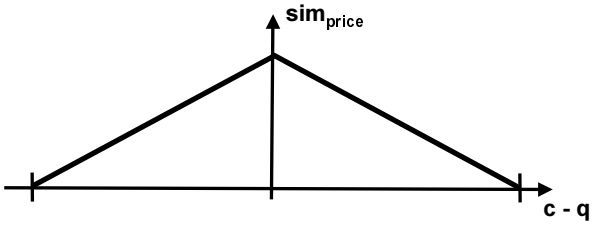
Figure 3.6.: Example: Knowledge-Intensive Local Similarity Measure

A typical example of knowledge-intensive similarity measures are asymmetric similarity measures. Consider again a product recommendation scenario and a numeric attribute price representing the price of the offered products. For this attribute a similarity function like shown in Figure 3.6 might typically be used to approximate the utility of products regarding the price demands of customers. Basically, the semantics of this measure can be interpreted as follows: "I want to have a product that costs *not more* than x EUR". However, the shown function encodes further knowledge for a flexible interpretation of this semantics. The *max* parameter expresses a threshold for the maximal exceedance of the given price limit probably tolerated by most customers. Only when reaching or even exceeding this price difference, the similarity decreases clearly. Products cheaper than the given price limit lead to maximal similarity, because customers are usually happy when getting cheaper products (provided that the required functionality is still fulfilled). However, if the price difference under-runs the *min* parameter, the similarity decreases slightly because some customers would perhaps be deterred by the fear to get a product with low quality. So, the described similarity function represents a lot of knowledge about expectations on the customers' buying behaviour to be encoded in the two parameters *min* and *max*, and the actual steepness of the decreasing parts of the function.

Another example of a knowledge-intensive similarity measure was already shown in Figure 3.3. There, the presented similarity table encodes utility knowledge about computer casings to be used in a product recommendation system, too.

Of course, a clear line between knowledge-poor and knowledge-intensive similarity measures cannot be drawn. On the one hand, also traditional measures might con-

sider domain knowledge encoded in attribute weights or special parameters like the ones introduced by the generalised forms of the simple matching coefficient. On the other hand, the introduced local similarity measures do not necessarily represent domain knowledge. For example, the similarity table and the similarity function shown in Figure 3.7 can be interpreted as knowledge-poor similarity measures. While the similarity table only implements an exact-match comparison, the similarity function represents a simple distance metric.



q \ c	red	green	blue
red	1.0	0.0	0.0
green	0.0	1.0	0.0
blue	0.0	0.0	1.0

Figure 3.7.: Example: Knowledge-Poor Local Similarity Measures

From the cognitive point of view, there are certain evidences that both knowledge-poor and knowledge-intensive similarity measures play an important role during human problem-solving processes. Suzuki et al. (1992) investigated the similarity judgements of human beings when solving the puzzle of Hannoy. Their results show that similarity judgements of persons are based on very different knowledge depending on their level of experience in the puzzle of Hannoy. On the one hand, beginners, who do not have certain experience in solving the puzzle, obviously base their similarity judgements between different states of the puzzle on syntactical differences mainly. On the other hand, experienced persons estimate the similarity by considering much more complex criteria reflecting particular knowledge about the puzzle.

3.5. Modelling Similarity Measures: State-of-the-Art

After having discussed the foundations on representing similarity measures, this section deals with the procedure of how to model similarity measures in practice. When reviewing the CBR literature, one can observe that no clear methodology how to define similarity measures efficiently has been developed, yet. One reason might be, that the representation of similarity measures strongly depends on the kind of case representation to be used. In order to fit the needs of the different application fields several different approaches to represent cases (e.g. attribute-value based representations, object-oriented representations, graph representation, first order logic, etc.) are used and so a methodology for each of these representations would be required. However, one could expect that methodologies for representations used most

commonly—like the attribute-value based representation—have been developed already. A first step into this direction is the INRECA-II[4] methodology (Bergmann et al., 1999a, 1998), that can be characterised as a generic "cookbook" for developing CBR applications. Concerning the process of defining similarity measures for attribute-value based case representations, this methodology identifies the following three elementary sub-processes:

1. **Similarity Characterisation**: Basic characteristics for all required local similarity measures have to be determined. For example, it has to be figured out whether a local similarity function has to be symmetric or asymmetric.

2. **Similarity Definition**: According to the determined characteristics, accurate local similarity measures have to be modelled, for example, by combining and parameterising basic similarity functions.

3. **Similarity Integration**: Finally, all elements of the similarity representation have to be integrated in order to obtain a global similarity measure. This means, accurate attribute weights have to be defined and a suitable amalgamation function has to be selected.

However, the INRECA-II methodology states nothing about the concrete procedure how to acquire the necessary domain knowledge efficiently and how to translate it easily into the representation formalisms introduced in the previous sections. During the development of CBR applications, the definition of knowledge-intensive similarity measures is still a task that requires a lot of experience. On the one hand, the particular application domain has to be understood at least partially to be able to identify aspects influencing the cases' utility. On the other hand, experience how to express the acquired knowledge by modelling both, an accurate case representation and an accurate similarity representation is necessary to be able to implement a powerful CBR system. Due to the lack of a more detailed methodology, defining similarity measures is still a challenging task and requires respective skills of a knowledge engineer.

3.5.1. Employing CBR Tools

Nowadays, the most CBR applications are being developed by employing commercial CBR shells. These systems provide the basic mechanisms that represent the foundation of every CBR application, for example, efficient case retrieval mechanisms. Moreover, they provide comfortable graphical user interfaces (GUIs) for modelling,

[4] *Information and Knowledge Re-engineering for Reasoning from Cases*; Esprit contract no. 22196, contract partners are AcknoSoft(France), Daimler Benz AG (Germany), tecInno GmbH (Germany), Interactive Multimedia Systems (Ireland) and the University of Kaiserslautern (Germany).

entering and managing all required domain knowledge (see Section 2.2.3). This comprehends the modelling of the case structure by defining attributes and corresponding value ranges, as well as the definition of general background knowledge in form of rules or similarity measures. Basically, the similarity definition process strongly depends on the employed CBR tool. In this section we shortly introduce the CBR shell used in the scope of this work. A review of industrial CBR tools is given by Althoff et al. (1995).

CBR-Works

The CBR shell CBR-Works has been developed in the research projects INRECA[5], WIMO[6], and INRECA-II at the University of Kaiserslautern in co-operation with tecInno GmbH (now empolis knowledge management GmbH). The system is written in the programming language Smalltalk and has been employed in numerous research projects and several commercial applications. The main advantage of CBR-Works is the provision of a large set of different modelling tools with comfortable graphical user interfaces. This enables a rapid development of powerful CBR applications. In the following, a short summary of the essential features of CBR-Works is given. A more detailed description is given by Schulz (1999).

- Cases can be modelled by employing *object-oriented representations*. In order to simplify the definition of the case structure comfortable GUIs are provided.
- To define value-ranges for attributes *numerous basic data types* are provided. This includes structured data types like ordered symbols or taxonomies in order to simplify the representation of the required similarity measures.
- The system provides a powerful *rule-interpreter* and GUIs for defining rules used to express general domain knowledge. The rule-interpreter can, for example, be used to perform case adaptation.
- *Flexible representations and GUIs for defining similarity measures* simplify the modelling of knowledge-intensive similarity measures.
- The system includes *interfaces for entering queries and browsing retrieval results* delivered by the internal retrieval engine.
- The *import of case data from relational databases* is supported.

[5] *Induction and Reasoning from Cases*; ESPRIT contract no. 6322, project partners were AcknoSoft (France), tecInno GmbH (Germany), Interactive Multimedia Systems (Ireland) and the University of Kaiserslautern (Germany).

[6] *Modelling and Acquisition of Knowledge for Case-Based Reasoning*; founded by the government of Rhineland-Palatinate, Germany, project partners were the University of Kaiserslautern, tec:inno GmbH and INSIDERS GmbH.

- The system *includes a generic HTTP-Server* making the development of WWW[7]-based applications very easy.

Because this work deals in particular with similarity measures we now focus on the tools of CBR-Works used for representing this part of the domain knowledge. Certainly, the CBR-Works tools represent only one possible way to model similarity measures. However, tools of other established commercial CBR shells provide similar modelling facilities.

3.5.2. Modelling Similarity Measures in CBR-Works

Basically, the formalisms and GUIs used to model local similarity measures in CBR-Works strongly depend on the particular data type. First, we describe how similarity measures for simple numeric types can be represented.

Modelling Similarity Functions

In CBR-Works the similarity measures for numeric types are represented by difference-based similarity functions according to Definition 3.15. In order to simplify the modelling of similarity functions, the system provides two kinds of editors. Firstly, there is the *standard editor* shown in Figure 3.8 that enables the user to select and parameterise a base function. The base functions provided are commonly useful in many application domains.

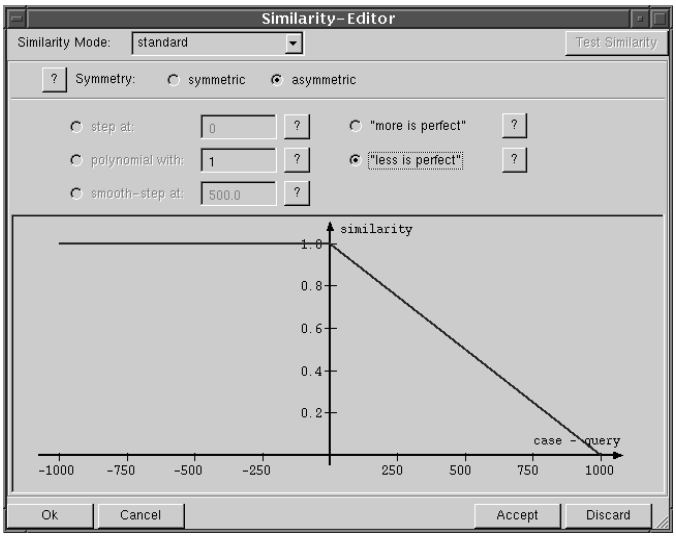


Figure 3.8.: Standard Editor for Similarity Functions

[7]World Wide Web

Secondly, the *advanced editor* shown in Figure 3.9 allows a more flexible definition of similarity functions. Here, the user can define several points of the desired function used to interpolate the entire function linearly. This editor also allows the selection of a logarithmic difference function instead of the linear difference function used by default.

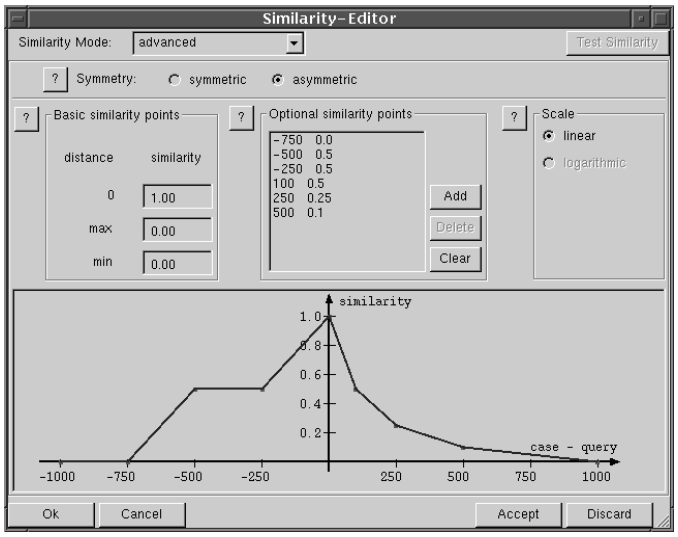


Figure 3.9.: Advanced Editor for Similarity Functions

Modelling Similarity Tables

To represent similarity measures for symbolic types, CBR-Works uses similarity tables (see Definition 3.14). The respective editor is shown in Figure 3.10. When choosing the "symmetric" option, the values of the upper triangle matrix are copied to the corresponding fields of the lower triangle matrix automatically. This simplifies the definition of symmetric similarity tables.

Support for Structured Data-Types

When dealing with structured data types, CBR-Works takes the defined structure into account to simplify the modelling process. Typical examples of such structured data types are ordered symbols or taxonomies. Although these types are basically of a symbolic nature, the additionally defined structure can be used to avoid the time-consuming definition of similarity tables (see also Section 3.3).

For example, similarity measures for taxonomies can be modelled by using the editor shown in Figure 3.11. Here, the user must only determine some basic parameters used to define the concrete semantics of the particular taxonomy. Internally,

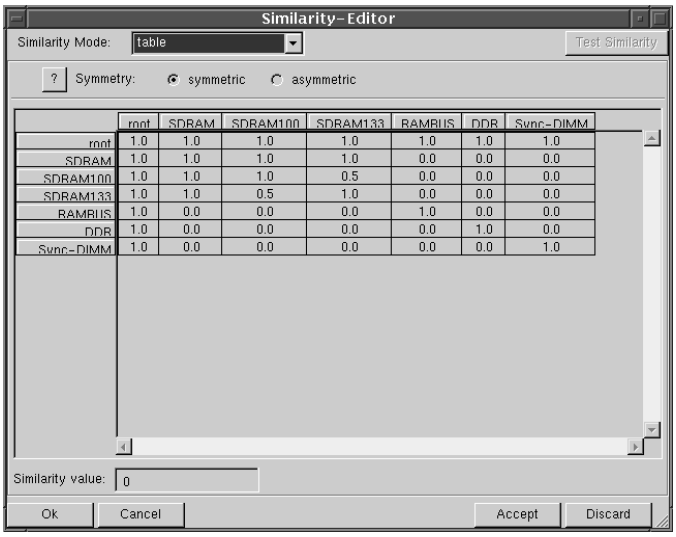


Figure 3.10.: Editor for Similarity Tables

the similarity representation is actually a similarity table where the entries are computed based on the taxonomy structure and the defined parameters according to the approach presented by Bergmann (1998).

Figure 3.12 shows the editor for ordered symbols. Here, the similarity representation is based on a mapping between defined symbols and numbers and an accurate similarity function.

Because the internal similarity representation for structured data types is still founded on similarity functions and similarity tables, we abandon a more detailed description of the provided modelling facilities.

Defining Weights and Global Similarity Measures

Concerning the definition of attribute weights, CBR-Works supports global weights and user defined weights (see Section 3.3). While global weights have to be defined together with the attributes, user weights can be entered together with a query.

CBR-Works allows to choose between several amalgamation functions (cf. Section 3.3). Moreover, it provides the possibility to consider the structure of an object-oriented case representation when defining the global similarity. The approach used here is very similar to the similarity computation of taxonomy types. Concerning details of this functionality, the reader is referred to Bergmann and Stahl (1998) due to the restriction on flat attribute-value based representations in this thesis.

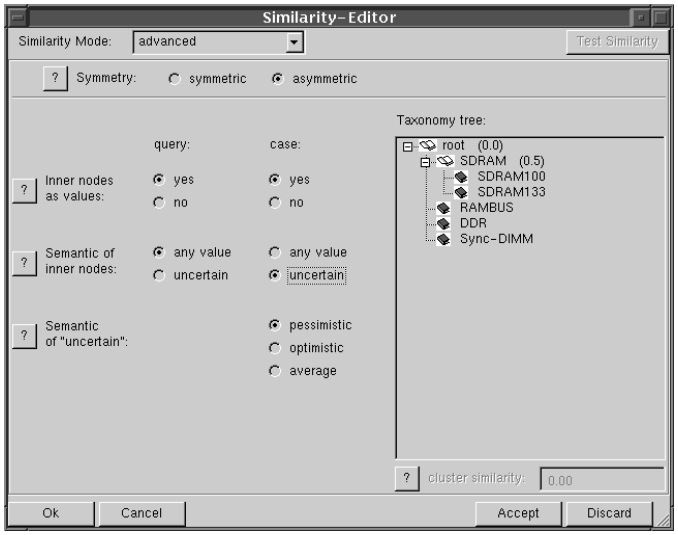


Figure 3.11.: Similarity Editor for Taxonomies

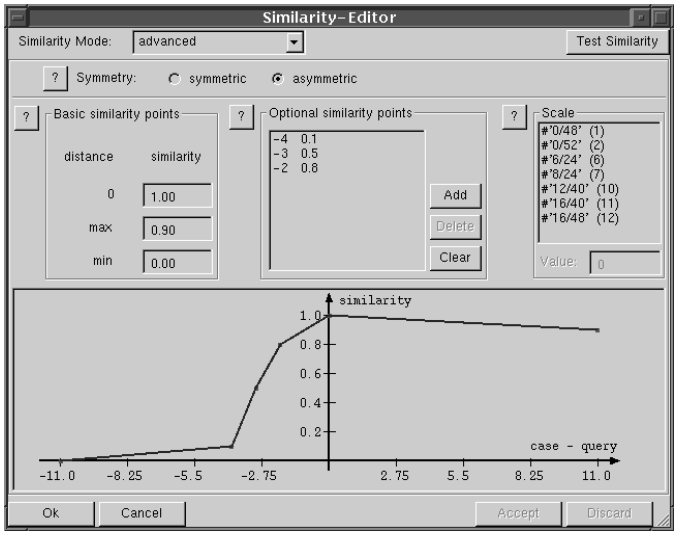


Figure 3.12.: Similarity Editor for Ordered Symbols

3.5.3. Drawbacks

The similarity definition process commonly applied nowadays can be characterised as a *bottom-up* procedure. This means, the entire similarity assessment is based on the acquisition of numerous single knowledge entities about the influences on the utility function. These knowledge-entities have to be encoded separately into suitable local similarity measures, attribute weights, and the global similarity measure. Because this knowledge is very specific and detailed (e.g. a local similarity measure concerns only one single aspect of the entire domain), it could also be characterised as *low-level knowledge* about the underlying utility function. Of course, to be able to acquire such general domain knowledge, at least a partial understanding of the domain is mandatory. The basic assumption of this procedure is that thorough acquisition and modelling of this low-level knowledge will lead to an accurate approximation of the complete utility function. However, in certain situations this bottom-up procedure to defining similarity measures might lead to some crucial drawbacks:

- The procedure is very time-consuming. For example, consider a symbolic attribute with 10 corresponding values. This will require the definition of a similarity table with 100 entries, if it is impossible to use a structured data type!
- In some application domains a sufficient amount of the described low-level knowledge might be not available. Possible reasons are, for example, a poorly understood domain, or the fact that an experienced domain expert who could provide the knowledge is not available or too expensive.
- Even if an experienced domain expert is available, s/he is usually not familiar with the similarity representation formalisms of the CBR system. So, the provided knowledge may only be available in natural language. This informal knowledge then has to be translated into the formal representation formalisms by an experienced knowledge engineer who possesses the required skills leading to additional costs.
- Due to the complexity of the representation, even experienced knowledge engineers often make definition failures by mistake. Unfortunately, the recognition of such failures is very difficult.
- Unfortunately, the bottom-up procedure does not consider the utility of whole cases directly. Instead, the final utility estimation is completely based on the ensemble of the individual low-level knowledge entities. Nowadays, the overall quality of the completely defined similarity measure is mostly even not validated in a systematic way. Existing approaches (e.g. leave-one-out tests and measuring classification accuracy) only measure the overall performance of the CBR system, that is also influenced by other aspects, for example, the

quality of the case data. So, one often blindly trusts the correctness of the similarity values computed by the defined measure.

- Due to the complexity of the bottom-up procedure its application is usually restricted to the development phase of the CBR application. This means, all similarity knowledge is acquired during the development phase and is assumed to be valid during the entire lifetime of the application. However, in many domains changing requirements and/or changing environments require not only maintenance of case knowledge, but also maintenance of general knowledge (Roth-Berghofer, 2002).

- Knowledge about the actual utility of cases might not be available at all during the development phase. For example, when applying personalised similarity measures Stahl (2002b) in an eCommerce or knowledge management scenario, the knowledge can only be provided by the users themselves during the usage of the system. However, here the bottom-up procedure is not feasible.

- Sometimes the required knowledge about the cases' utility might already be available in a formal but quite different representation form. For example, when supporting case adaptation the utility of cases strongly depends on the provided adaptation possibilities. Hence, to obtain an accurate similarity measure one has to transfer adaptation knowledge into the similarity measure. When using the bottom-up procedure this is a very time-consuming task. The adaptation knowledge has to be analysed manually and the knowledge considered to be relevant has to be encoded into the similarity measure.

Part II.

The Framework

4. Framework for Learning Similarity Measures

As described in the previous chapter, in CBR similarity measures are used to approximate the a-priori unknown utility of cases regarding their reuse in a particular problem situation. In order to obtain similarity measures that ensure accurate retrieval results, in many present application fields a consideration of specific domain knowledge is required. The definition of such knowledge-intensive similarity measures is certainly one of the most crucial, but unfortunately also one of the most challenging tasks when developing CBR applications. Since the manual definition of similarity measures often raises certain problems (cf. Section 3.5.3), in this chapter a framework for an alternative strategy for modelling similarity measures is presented. The central idea of this framework is the application of machine learning techniques in order to facilitate the entire similarity assessment process.

First, a short motivation for the chosen strategy is given and it is shown how the framework can be integrated into the well-known CBR cycle (cf. Section 2.2.2). Then an alternative way to acquire the required domain knowledge to be encoded into similarity measures is introduced, followed by the description of a general approach that utilises this alternative way for learning similarity measures. In the last section it is discussed how the presented learning approach can be combined with the manual similarity assessment process in order to profit from the advantages of both.

4.1. Aim of the Framework

As discussed in Section 2.4.1, CBR is commonly considered to be a technique to realise learning systems that improve their problem-solving competence from experiences gained during the usage of the system. According to Definition 2.10, we call a particular CBR system a learning system if it improves its performance by changes in the content of one or several of its four knowledge containers (cf. Section 2.2.3).

Although the original interpretation of the traditional CBR cycle (Aamodt and Plaza, 1994) considers the theoretical possibility to modify not only the case knowledge container, but also general knowledge contained in the three other containers, this opportunity is almost neglected. In practice general knowledge is usually acquired and formalised once during the development time of the application and

considered to be fixed during the lifetime of the application. Nevertheless, if an update of general knowledge is required, the traditional CBR process model does not specify how to realise it. Instead, in practice such maintenance operations are commonly performed completely manually by human knowledge engineers.

4.1.1. Motivations

When recalling the motivations of CBR discussed in Section 2.1, it seems to be obvious to extend the learning capabilities of CBR systems towards learning of general knowledge, too.

Cognitive Point of View

From the cognitive point of view, it is clear that human beings do not only memorise situation-specific knowledge but also modify their general knowledge when making new experiences. This holds in particular also for similarity judgements (Novick, 1988). For example, Suzuki et al. (1992) report about cognitive experiments showing that similarity judgments of human beings depend on their expertise in the respective domain, here solving the Tower of Hannoy puzzle. It seems that experts extensively use their domain knowledge when being asked to judge the similarity between different states of the puzzle. Ohnishi et al. (1994) further present the hypothesis that two types of similarities should be distinguished:

- A *shallow similarity* only considering the superficial differences between the entities to be compared, i.e. objects are considered to be similar if they "look similar". This kind of similarity judgement seems to be used mainly by novices who do not possess a deeper understanding of the domain.
- A goal-related *deep similarity* based on domain knowledge. This kind of similarity judgement takes the actual problem into account and is used mainly by domain experts who infer the similarity by applying general rules about the coherences in the domain. A typical property of deep similarities is that they are often asymmetric (cf. Section 3.2).

In our terminology, shallow similarities can be characterised as knowledge-poor similarities and deep similarities can also be called knowledge-intensive similarities (see Section 3.4). When transferring the described cognitions on CBR research, one could imagine an "inexperienced" CBR system only provided with an initial case base and an initial, "shallow" similarity measure (e.g. a simple distance metric). When solving new problems with such a system, in principle, the system should be able to profit from the new experiences in two ways. On the one hand, it can store new cases, and on the other hand, it could obtain a "deeper" understanding of similarity by encoding more domain knowledge into the initial similarity measure.

Functional Point of View

Besides the cognitive point of view, also the functional point of view supplies good motivations to think about the learning of similarity measures in CBR. In Section 2.1 we have outlined three potential functional benefits of CBR compared to other AI techniques: an increased problem-solving efficiency, the capability to handle incomplete domain knowledge, and a simplification of knowledge acquisition. All three aspects also motivate the development of strategies to learn not only case knowledge but also general knowledge:

- Generally, expertise depends not only on experiences, but also on factors affecting how experience is reused (Leake, 1994). However, CBR systems usually do not refine the knowledge they use to retrieve or adapt prior cases. By doing so and not only relying on static pre-defined knowledge, CBR systems might be enabled to increase their problem-solving efficiency and competence more effectively.

- When being confronted with poorly understood domains, methods to learn general domain knowledge might decrease the problem of the necessity to handle incomplete domain knowledge. Instead of relying only on imperfect knowledge manually acquired during development time, the system might be enabled to extend its knowledge during the lifetime of the application.

- Last but not least, the introduction of learning procedures could, of course, also simplify the knowledge acquisition. On the one hand, machine learning techniques might facilitate the initial knowledge acquisition and formalisation. On the other hand, such techniques could also simplify the maintenance of general knowledge in domains requiring modifications frequently.

These three aspects show that methods for learning general knowledge might further improve the basic advantages of the CBR approach. In particular, facilities for learning similarity measures might also help to avoid the drawbacks that arise when defining similarity measures manually discussed in Section 3.5.3. For example, learning approaches can help to hide the complex formalisms required to represent and process similarity measures inside a CBR system from the human knowledge engineer. Instead of defining these representations directly, they might be tuned by a learning procedure more or less automatically from data that can be handled easier by human experts.

Although generally applicable approaches have not been developed yet, in the past several researches already pointed out the need for more general learning strategies in CBR. For example, Leake (1994); Leake et al. (1996b, 1997b) emphasise the need for *multistrategy learning* in CBR. The desire for a more accurate similarity assessment based on learning strategies is discussed by Leake et al. (1997a). Aha and

Wettschereck (1997) demand learning in CBR should go beyond the classification of feature vectors. Instead, alternative applications scenarios and extended learning capabilities should be investigated.

4.1.2. Basic Assumptions

Before developing an actual approach to extend the learning capabilities of CBR on learning similarity measures, it is important to point out some basic assumptions. Because nowadays Case-Based Reasoning is a wide research field dealing with various domains and applications scenarios, basically it is not possible to develop a general approach applicable in all domains and scenarios. Therefore, in the following several basic assumptions are being made to circumscribe the application situations addressed by the framework to be developed in this thesis:

- In this work a flat attribute-value based case representation as introduced in Section 2.2.4 is presumed. Although the developed approaches can also be applied easily to object-oriented representations this opportunity will not be discussed explicitly.

- Further, we restrict the scope of this thesis to the use of similarity measures based on the local-global principle and the representation formalisms introduced in Section 3.3. We also suppose the existence of an initial default similarity measure. This measure might be a knowledge-poor measure such as illustrated in Figure 3.7.

- When applying the local-global principle, an accurate similarity assessment strongly depends on the underlying case structure, i.e. the definition of suitable attributes. In this work it is presumed that an accurate case characterisation is already available before applying the learning framework.

- The system's initial case base should contain a reasonable amount of cases. Generally, it is not possible to determine a specific number of cases representing a "reasonable amount" for every domain, so in the following we formally assume the existence of at least two cases.

In summary, we build upon a functional CBR system supporting attribute-value based case representations at least able to perform a similarity-based retrieval on an initial set of cases according to an initial similarity measure. Considering the described assumptions, now we are able to informally define the basic objective of the learning framework to be developed:

Definition 4.1 (Objective of Learning Framework) The *objective of our framework for learning similarity measures* in a CBR system is the introduction of a

process that enables the system to improve its performance, measured by some performance measure P, by modifying its internal similarity measure representation in order to influence the outcome of the retrieval phase.

4.1.3. Refining the CBR Cycle

When investigating approaches to realise the discussed learning framework, a central question is the integration of a potential learning strategy into the well-known process model of CBR. This is in particular a crucial aspect if learning should not only be used to support the initial knowledge acquisition process but should also enable the CBR system to improve its performance during usage. Then the system must be able to execute the learning task more or less independently from its actual tasks. Such a learning functionality is often called *introspective reasoning* (Fox and Leake, 1995b) or *introspective learning* (Zhang and Yang, 1999), respectively.

To integrate the desired learning functionality into the traditional CBR cycle consisting of the four well-known phases—retrieve, reuse, revise, retain—two basic possibilities can be distinguished:

1. The *extension* of the existing process model by introducing an additional phase.

2. The *refinement* of one or several phases to integrate the new functionality into the already established phases.

When reviewing the original interpretation of the traditional CBR cycle it becomes clear that the second possibility seems to be more accurate. Aamodt and Plaza (1994) have already discussed that the retain phase could be used to update general knowledge of the CBR system. Concerning the update of similarity measures the possibility to refine case indexes has been mentioned. This can be interpreted, for example, as an adjustment of feature weights. However, it does not specify how this update can be realised concretely. Further, this early model of CBR does not explicitly consider the use of knowledge-intensive similarity measures as introduced in Chapter 3.

Basically, the retain phase is not the only phase of the CBR cycle responsible for the capability to learn new knowledge. Before memorising a new case, the correctness of this new knowledge item has to be validated during the revise phase. So, the revise phase has a significant influence when learning new case knowledge, because it selects cases considered to be candidates for extending the knowledge base. In the following we show that this holds as well when learning similarity measures.

Figure 4.1 illustrates how the traditional CBR cycle can be modified to integrate the possibility to learn similarity measures. These modifications are discussed in more detail in the following sections.

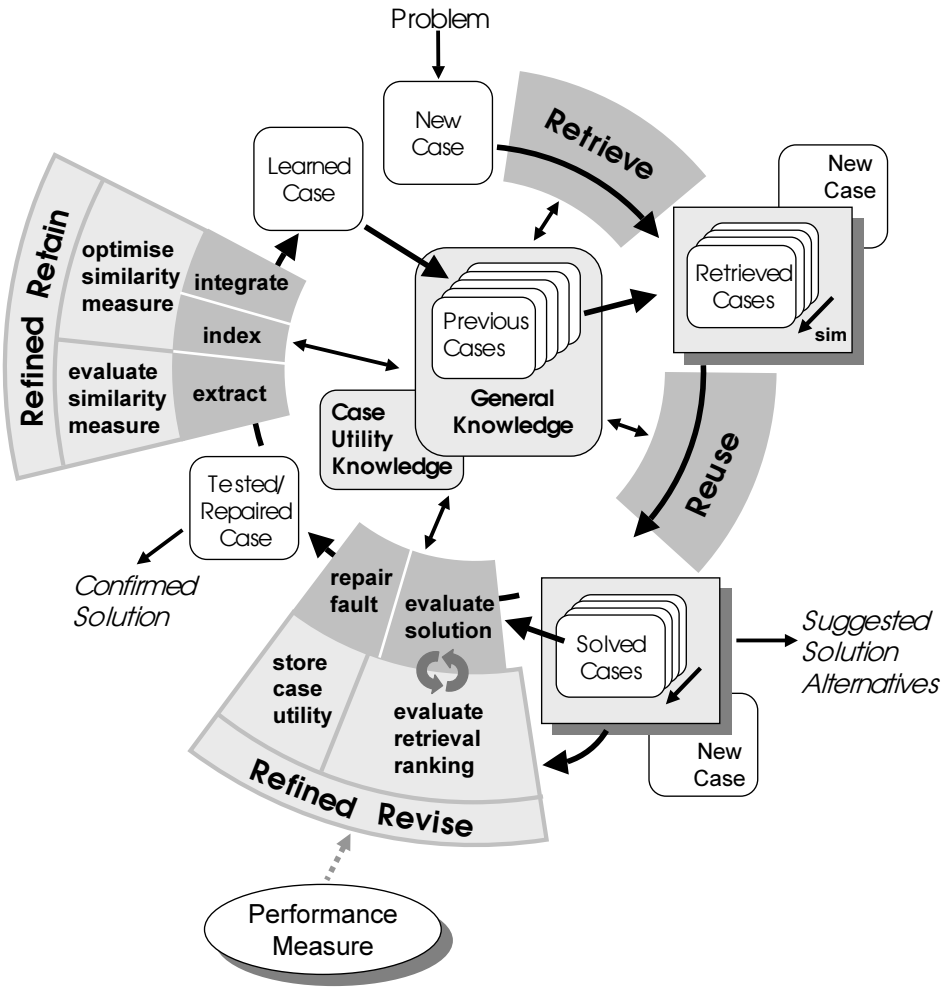


Figure 4.1.: Refining the CBR Cycle for Learning Similarity Measures

Extended Use of Retrieved Cases

In the traditional view of CBR, the retrieve phase provides one or several cases used to generate exactly one solution during the reuse phase. This solution is then proposed for solving the current problem and has to be evaluated during the revise phase. However, in many application domains where CBR has been employed successfully this traditional view is not always suitable. Here, it is not desired that the CBR system generates exactly one solution, but several independent alternatives for solving the given problem. A popular example are product recommendation systems in eCommerce. Such systems usually generate a collection of solutions, here product recommendations, that are presented to the user, here the customer.

So, for the development of our learning framework we suppose that the retrieval phase always provides a list of retrieved cases ordered by the computed similarities

(cf. Definition 3.4). If case adaptation is supported, this list is processed during the reuse phase where several solution proposals might be generated by adapting several retrieved solutions independently from each other. Basically, two ways to generate solution alternatives can be distinguished:

Ad hoc: If it is feasible with respect to computation time, the reuse phase might perform adaptation for a fixed number of cases immediately. The resulting list of solution proposals, still ordered as determined in the retrieval phase, is then directly passed to the revise phase.

On demand: If case adaptation is computational expensive, only the most similar case may be adapted first. The generated solution is then passed to the revise phase where it has to be evaluated. If the evaluation fails, because the solution cannot be applied to solve the current problem or due to poor solution quality, two ways for proceeding are possible. On the one hand, the faulty solution might be repaired during the revise phase. On the other hand, the revise phase could trigger the adaptation of the next similar case in anew execution of the reuse phase to obtain an alternative solution proposal.

Both approaches lead to the suggestion of several solution alternatives—when applying the on demand approach, at least if the most similar case could not be reused successfully—after the reuse phase. In the following we only assume the possible existence of such a list of suggested solution alternatives but we do not care about the approach used to generate it. It is only assumed that solution alternatives are ordered according to the similarity of the underlying cases.

Refining the Revise Phase

According to the original process model that assumes the existence of only one solved case after the reuse phase, the revise phase can be subdivided into two subsequent tasks:

1. **Solution Evaluation:** In a first step the proposed solution, i.e. the outcome of the reuse phase has to be evaluated. This evaluation might be based on feedback from a teacher, on the results obtained through application in the real world, or on the outcome of a model-based simulation.

2. **Fault Repair:** When recognising faults in the suggested solution during evaluation, the solution has to be repaired to obtain a valid solution. Basically, it might be repaired manually by the user or it might be repaired by the system based on additional general knowledge.

To enable a CBR system to learn similarity measures we propose a refinement of the revise phase as illustrated in Figure 4.1. Besides the two described traditional

tasks that ensure the generation of a valid solution, we introduce two additional tasks:

1. **Evaluate Retrieval Ranking:** This task can be characterised as a superior control process for the common solution evaluation task. It initiates the evaluation of several solution alternatives and processes the obtained evaluation results. The foundation of the evaluation might be internal general knowledge or an external *performance measure* in form of a teacher, the real world, or a model.

2. **Store Case Utility:** This task is responsible for storing the results of the retrieval ranking evaluation for further processing. Basically, these results represent knowledge about the utility of cases with respect to the given query.

Generally, one could also argue that storing of evaluation results belongs more to the retain phase of the CBR cycle. However, we decided to assign this task to the revise phase. On the one hand, the decision whether to store particular results or not might be influenced by the performance measure, for example, by a human teacher. On the other hand, the retained knowledge is not directly used by the phases of the CBR cycle that are relevant for problem-solving. It is more an intermediate knowledge buffer that collects knowledge to be used only during the retain phase and thus it does not directly contribute to solving problems.

Basically, the refined revise phase consists of two parallel processes. On the one hand, the traditional revision process that only evaluates and repairs a single solution. On the other hand, a parallel process that evaluates the outcome of the retrieval phase based on the results obtained during several solution evaluations. While the evaluation of the retrieval ranking relies on the solution evaluation process, the traditional revision of a single solution can be initiated independently. This means, the retrieval evaluation can be interpreted as an optional process to be performed if desired.

Refining the Retain Phase

The aim of the retain phase is to select knowledge entities to be integrated into the knowledge resources of the CBR system in order to improve its problem-solving competence and/or efficiency during future usage. Therefore, the traditional retain phase identifies the following three tasks:

1. **Extract:** This task is responsible for the extraction of relevant knowledge entities from the current problem-solving episode to be retained for future usage. Such knowledge entities might be represented by found solutions, solution methods, justifications, etc.

2. **Index:** The objective of this task is to determine indexes to be used for retrieving the learned case. This may be interpreted as the selection of an accurate vocabulary used to characterise the case but it might also be interpreted as the determination of accurate attribute weights.

3. **Integrate:** During the final task the extracted knowledge has to be integrated into the knowledge base of the system. This process might comprehend an update of the case base, the index structure, and of other general knowledge.

Although this traditional interpretation of the retain phase, in principle, already considers the modification of general knowledge and even an adjustment of attribute weights, it seems to be necessary to introduce two additional tasks for realising our learning framework:

1. **Evaluate Similarity Measure:** Here, the quality of the currently used similarity measure is estimated based on the case utility knowledge acquired in the previous revise phase.

2. **Optimise Similarity Measure:** This task can be seen as a specialisation of the index and integrate task of the traditional retain phase but with focus on learning similarity measures. During this task, machine learning or optimisation methods, respectively, are being used to optimise the current similarity measure regarding the available case utility knowledge. This optimisation might be triggered by the outcome of the prior evaluation of the current similarity measure.

Similar to the refined revise phase, the tasks additionally introduced in the refined retain phase have not necessarily to be executed during every pass of the cycle. Instead, in certain application scenarios all described extensions of the traditional CBR cycle might only be relevant during explicit knowledge acquisition or maintenance phases. For example, if the performance measure is supplied by a human domain expert playing the role of a teacher, the refined revision phase can only be executed in situations where this expert is available. During problem-solving situations where the system is used by a "standard user" who does not possess the required expertise, the introduced retrieval ranking evaluation might be skipped.

In the next sections it is shown how the discussed refinement of the CBR cycle can be implemented. First, it is described how to realise the refined revise phase in order to obtain the required case utility knowledge. The following section deals with the implementation of the refined retain phase.

4.2. Assessing Case Utility

When assessing similarity knowledge by applying the bottom-up procedure (see Section 3.5), one has to identify the factors determining whether a case is useful

in a particular problem situation or not. However, knowledge about this influences is very specific and very difficult to acquire because a good understanding of the underlying domain is mandatory. We already called this type of similarity knowledge *low-level knowledge* to be represented in form of attribute weights and accurate local similarity measures.

In this section it is shown that it is also possible to assess similarity knowledge on a higher, less detailed level. Instead of identifying general influences on the cases' utility it is also possible to determine the utility of particular cases for particular problem situations without giving detailed explanations of the reasons. After discussing the abstract foundation of this approach, in the following it is described how to acquire such *high-level knowledge* about the underlying utility function.

4.2.1. Abstract Point of View

As already discussed in Section 3.1, the aim of a similarity measure is the approximation of a particular utility function. In general, similarity functions and utility functions are defined over different domains (cf. Definition 3.1 and 3.3). While utility functions compare case characterisations with complete cases, similarity measures compare two case characterisations only. This means, the actual utility of a case might either be determined exclusively or additionally by its lesson part. However, similarity measures do not consider the lesson part because it is not available in a new problem situation and therefore it cannot be compared to lesson parts contained in available cases.

From an abstract point of view, the relation between utility functions and similarity measures can be described like illustrated in Figure 4.2. The left side of the figure represents a utility function u defined over the case space $\mathbb{C}_C$ and the case characterisation space $\mathbb{D}_D$. The right side of the figure represents the corresponding similarity measure sim defined over the case characterisation space $\mathbb{D}_D \times \mathbb{D}_D$ only. When dealing with the special situation that the case model C contains an empty lesson model (see Section 2.2.4), u would be defined over the same space as sim. In the following we do not discuss this possibility explicitly because it can be seen as a specialisation of the described, more general case.

Basically, one is interested in the structure of the utility function to be able to reproduce it with an accurate similarity measure as good as possible. If no general domain knowledge is available, existing case knowledge only will determine certain points of the utility function. In our example we consider two cases c_1 and c_2. These two cases determine two points of the utility function—$u(d_1, c_1)$ and $u(d_2, c_2)$ illustrated as black dots—that represent the cases' utility with respect to their own case characterizations d_1 and d_2. Usually these utility values will be very high or even maximal when assuming cases that represent successful experiences. These points of the utility function have to be mapped to corresponding points of the similarity measures, here illustrated through dark grey dots. Because similarity

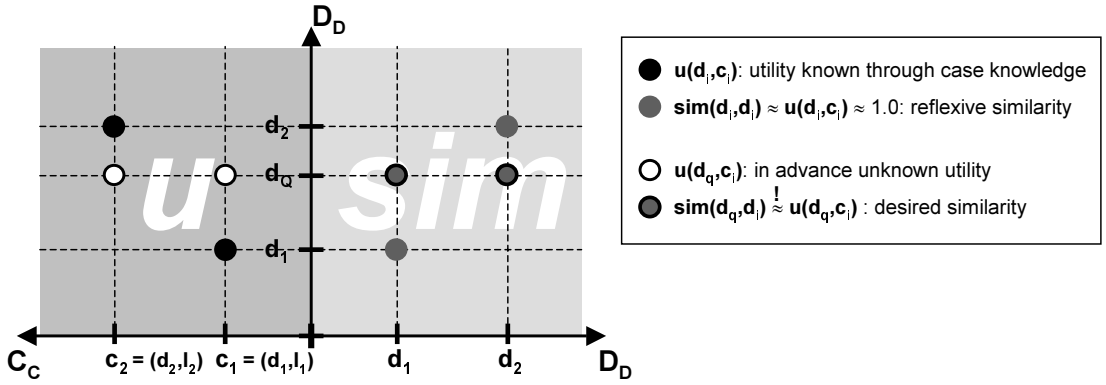


Figure 4.2.: Relation between Utility Functions and Similarity Measures

measures are commonly reflexive (see Section 3.2.2), for these points it should hold: $sim(d_1, d_1) = sim(d_2, d_2) = 1.0$.

In a new problem situation to be solved by reusing existing case knowledge, one is confronted with a query, here denoted as d_q. The crucial task of the retrieval phase is then the determination of the most useful case. This means, in the shown example the similarity between d_q and the two case characterisations d_1 and d_2 has to be determined. If no additional knowledge about the utility function is available the determination of the geometric distances $d(d_1, d_1)$ and $d(d_2, d_2)$ is the only possibility to obtain values for $sim(d_q, d_1)$ and $sim(d_q, d_2)$ (black bounded dark grey dots).

However, the actual utility of the cases ($u(d_q, c_1)$ and $u(d_q, c_2)$, here illustrated as black bounded white dots) might be very different. If it would be possible to determine these utility values, at least after solving the current problem, one could store this information to obtain a more detailed idea of the utility function's structure. Hence, by repeating this procedure for a set of given queries, one could acquire knowledge about numerous additional points of the utility function (see Figure 4.3). By mapping these points to the corresponding similarity measure, one would also obtain more detailed information about its structure required for representing a good approximation of the utility function.

From the abstract point of view, a similarity measure is a 3-dimensional plane, where the x- and y-coordinates represent the case characterisations to be compared and the z-coordinate represents the corresponding similarity value. When acquiring knowledge about a sufficient number of certain points of this plane, this information can be used to construct an accurate similarity measure by applying an *interpolation* process. In Figure 4.3 this idea is illustrated for a 2-dimensional intersection through the entire similarity measure representing all possible similarity values for case c_2. Here, the desired similarity measure is constructed by performing a non-linear interpolation between the known points of the measure. The foundation for the known points is the acquired knowledge about the cases' utility for a set of given

queries.

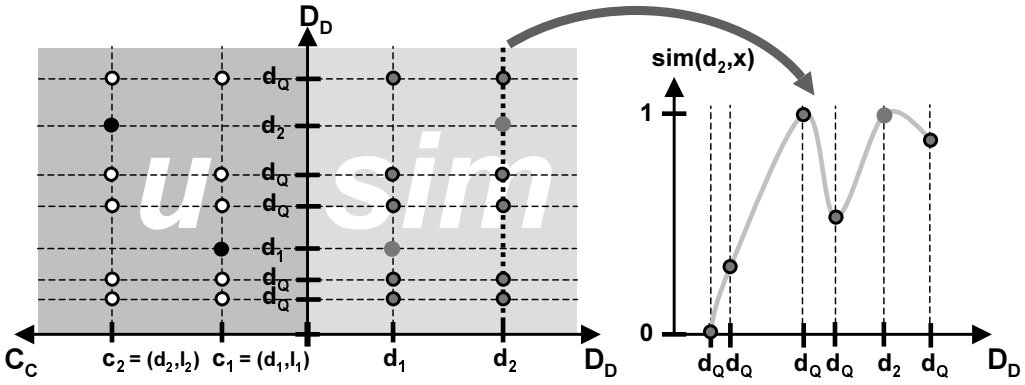


Figure 4.3.: Constructing Similarity Measures through Interpolation

4.2.2. Utility Feedback

In the previous section a very abstract idea about an alternative method to construct similarity measures has been presented. The crucial precondition of this method is the assumption that it is possible to estimate the actual utility of cases for given problems. However, as discussed in Section 3.1.2, the actual utility of a case can mostly only be determined after the current problem has been solved. Nevertheless, in many application scenarios it is possible to acquire certain knowledge about the cases' utility for particular problem situations. On the one hand, such knowledge might be provided by a domain expert during a knowledge acquisition phase. On the other hand, it might also be acquired during the daily application of a CBR system.

In both situations we presume a special *similarity teacher* (Stahl, 2001) who is able to give feedback about the utility of cases—in the following called *utility feedback*—during the refined revise phase of the CBR cycle introduced in Section 4.1.3. The similarity teacher must possess some implicit knowledge about the underlying utility function. For our framework we do not presume a special implementation of this concept. Depending on the particular application scenario, the similarity teacher can be represented very differently. This issue will be discussed in more detail in Chapter 6. Basically, it is possible to distinguish the following kinds of similarity teachers:

- A *human domain expert* who is able to estimate the utility of cases due to her/his background knowledge about the domain.
- The *users of the system* (e.g. customers in eCommerce) who determine the cases' utility due to their own demands and preferences.

- Some *performance measure* of the system's environment. This performance measure might be, for example, the real world or a simulation environment where a suggested solution can be evaluated.
- Some additional *general knowledge* (e.g. a domain model) provided to the CBR system.

In the previous section we assumed that utility feedback is expressed through points of the utility function, i.e. through concrete utility values. However, depending on the kind of similarity teacher, the acquisition of such exact feedback might not be realistic. For example, a customer in eCommerce is usually not able or willing to express the utility of a product through a real number. But a customer is usually able to compare the utility of different products and to express preferences. In this case, knowledge about the utility of a case is not available absolutely, but relatively to the utility of other cases. Basically, three different types of utility feedback can be distinguished. The major difference between these types is the amount of knowledge provided about the utility function.

Ordinal Utility Feedback

Ordinal utility feedback requires the least knowledge about the utility function. Here, the similarity teacher must only be able to compare the utility of two cases c_1 and c_2 with respect to a given query. This means, the similarity teacher must be able to convey the outcome of the preference relation introduced in Definition 3.2. Three statements of the similarity teacher are possible:

- case c_1 is more useful than case c_2
- case c_2 is more useful than case c_1
- both cases have the same utility or the utility difference is too small to be recognised

An illustration of a scenario where ordinal utility feedback can be acquired is given in the example in Figure 4.4. Here a customer wants to buy a car. When being confronted with the two alternatives of a Smart and a caravan, the customer would prefer the caravan. In the example, the bigger car is more useful for the customer due to her/his desire to buy a family car. Here utility feedback represents the individual preferences of this particular customer. Richter (2003) characterised such a comparison used to acquire similarity knowledge as a modification of the Turing Test. To build a personalised product recommendation system (see Section 6.4) one would need a similarity measure that reflects the individual preferences of the customer or the corresponding customer class (here, e.g. family fathers), respectively.

By pairwise comparing a set of cases, ordinal utility feedback allows to arrange a complete set of cases with respect to their utility. The result is a partial order.

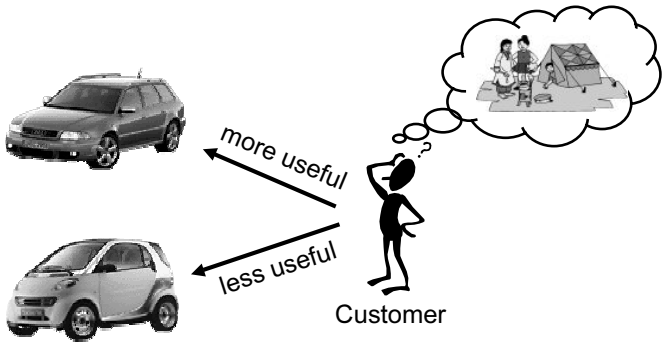


Figure 4.4.: Ordinal Utility Feedback

Quantitative Utility Feedback

While ordinal feedback only provides qualitative information about the relative utility of cases, quantitative feedback also allows an estimation of the degree of the cases' utility relatively to other cases. Basically, the following two possibilities can be distinguished:

Relative Utility: Consider the same situation as described previously, i.e. the similarity teacher is confronted with a query q and two cases c_1 and c_2. Instead of only selecting the more useful case, the similarity teacher could also estimate the utility of case c_1 in relation to the utility of case c_2. This could be realised, for example, through a statement like "c_1 is about twice as useful as c_2". Formally spoken this would lead to the following equation, where x represents an accurate factor: $u(q, c_1) \approx x \cdot u(q, c_2)$.

Relative Utility Differences: A weaker form of such relative quantitative feedback is the possibility to compare utility differences instead of comparing utilities directly. Consider a query q and three cases c_1, c_2, and c_3. Then a similarity teacher could also state: "c_1 is the most useful case and the utility difference between c_1 and c_3 is about three times bigger than the utility difference between c_1 and c_2". This can be expressed with the following equations:

- $u(q, c_1) > u(q, c_2) > u(q, c_3)$
- $u(q, c_1) - u(q, c_3) \approx x \cdot (u(q, c_1) - u(q, c_2))$

Both approaches to express quantitative utility feedback can be illustrated as an arrangement of cases on a line representing the continuum of utility values like shown in Figure 4.5. The only difference between both approaches lays in the interpretation of the cases' positions on the line. Of course, the illustrated procedure can also be performed with a larger set of cases. The outcome is again a partial order, however, with some additional information, for example, represented through the described equations.

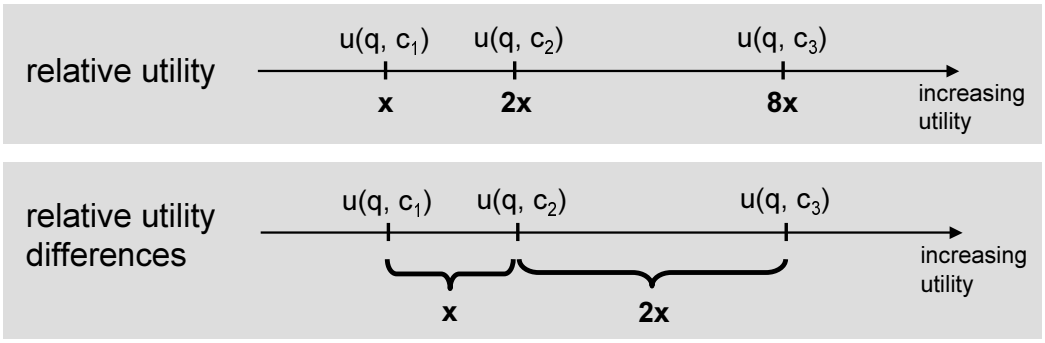


Figure 4.5.: Quantitative Utility Feedback

Exact Utility Feedback

Instead of estimating case utility only in an ordinal manner by comparing the utility of different cases relatively, utility feedback can also be expressed in a cardinal way. This leads to exact utility feedback where the utility of a particular case is represented absolutely by a real number (e.g. out of the interval [0,1]).

Although exact utility feedback provides definitely more knowledge about the underlying utility function than the relative types of feedback described previously, its acquisition is often not feasible in practice. For example, the customer shown in Figure 4.4 would usually have problems to express the utility of the Smart or the caravan through real numbers. Nevertheless, in other application scenarios where the similarity teacher is not represented by a human being, acquisition of exact utility feedback may be possible.

4.2.3. Acquisition of Utility Feedback

In the previous section we have discussed how feedback about the utility of cases might look like. Although we also mentioned the possibility to acquire quantitative feedback by estimating the degree of a case's utility depending on the utility of other cases, in following we focus on ordinal feedback and on exact feedback. On the one hand, quantitative feedback can be transformed into exact feedback by exploiting the respective equations introduced in the previous section, if it is possible to determine exact utility values at least for some cases. On the other hand, handling this kind of feedback is significantly more complicated than handling the other forms.

As already described in Section 4.1.3, utility feedback is acquired when evaluating a presented retrieval ranking. The first question to be answered when considering the realisation of this evaluation process, is the choice of an accurate point of time. Two basic possibilities can be distinguished:

1. The evaluation of retrieval rankings is only performed *offline* in separate training phases. In such phases special training queries are used to start the CBR

cycle in order to acquire utility feedback.

2. Another possibility is the integration of the evaluation process into the daily use of the CBR system. This means, retrieval rankings for real-world queries are evaluated *online* during actual problem-solving.

The offline approach is in particular suited for an initial knowledge acquisition phase or maintenance operations to be performed from time to time. The online approach is more suitable to improve the problem-solving competence of the CBR system continuously or to acquire the required utility knowledge from the system's users.

Depending on the implementation of the similarity teacher, utility feedback might be acquired manually from a human being or automatically through an accurate evaluation procedure employing general knowledge or an external performance measure.

Manual Acquisition

If the utility of cases has to be estimated manually by human beings, for example, domain experts or users, this leads to some crucial questions to be considered when realising the knowledge acquisition process.

Basically, this process has to fulfill two important but contrary requirements. Firstly, the effort required to evaluate a sufficient number of retrieval rankings should be minimised. If it is possible to define an accurate similarity measure by applying the bottom-up procedure (see Section 3.5) with less effort, the entire learning approach would be pointless. Secondly, the acquisition process must provide enough information about the unknown utility function to enable a learning algorithm to construct an accurate similarity measure. This means, during the manual acquisition of utility feedback we have to deal with the tradeoff between minimal effort for the similarity teacher and the assessment of a sufficient amount of feedback. Regarding this aspect two basic approaches are possible:

Invasive Acquisition: Here, the human similarity teacher is explicitly asked to give utility feedback for presented retrieval rankings. So, s/he plays an active role and should know that the corresponding effort is required for improving the entire system and not only for solving the current problem.

Non-Invasive Acquisition: When acquiring utility feedback online, some application scenarios also allow a more passive role of the similarity teacher. A user of the system might provide some feedback about the cases' utility during normal interaction with the system without recognising it. For example, a customer in an eCommerce scenario may provide ordinal utility feedback implicitly through his buying patterns (see Section 6.4).

The invasive approach is particularly suited if utility feedback is acquired offline, for example, during an initial training phase. Here, the similarity teacher is typically an expert with reasonable expertise in the underlying domain. The non-invasive approach can only be applied, if utility feedback is acquired online during the actual usage of the system. Here, feedback is usually provided by normal users that are not necessarily familiar with the domain and the underlying utility function.

When acquiring utility feedback manually with an invasive method, one has to take care about the already mentioned tradeoff between effort and benefit. If the estimation of the cases' utility is a very complex task, the effort to acquire even ordinal feedback might be very high. This situation will typically appear in domains where a comparison of the utility of two given cases requires a deeper analysis of the cases' properties. Then one has to deliberate whether the acquisition of enough utility feedback is more expensive than defining the similarity measure bottom-up. Unfortunately, this deliberation process is not easy because generally it is difficult to foresee how much utility feedback will be required to obtain reasonable learning results.

However, in some application domains domain experts are able to estimate the utility of cases without analysing them in detail. A domain expert might be familiar with the cases stored in the case base because they represent typical situations appearing in the domain. For example, in a help desk system (see Section 2.3.2) an expert might easily be able to identify useful or exclude absurd solution suggestions due to her/his own experiences.

A crucial problem that might occur—at least if utility feedback is acquired manually—are inconsistencies. When asking a domain expert to rank some cases with respect to their utility for a given problem, the resulting ordinal utility feedback is often based on some vague and possibly also subjective decisions of the expert. This can lead to the undesired effect that human similarity teachers tend to determine different case rankings when asking them to estimate the utility of the same cases after some time (e.g. the next day). This problem will become even more obvious if utility feedback is delivered from several domain experts or several users. Then the collected utility feedback will inevitably contain inconsistencies making it harder to learn an accurate similarity measure.

Automatic Acquisition

Sometimes it is also possible to automate the acquisition of utility feedback. Then the utility of given cases is not determined by a human being, but by some procedure or performance measure. Generally, we distinguish two kinds of automatic acquisition:

Semi-Automatic: Here, a human user still has to select cases to be evaluated. The utility of the selected case then has to be measured by some performance

measure in a more or less automated way. For example, it might be used as input for a simulation procedure or it might be applied in the real world. Moreover, the human user might select appropriate training queries.

Fully-Automatic: Here, the evaluation of the entire retrieval ranking is automated. When acquiring utility feedback offline, this also means that appropriate training queries have to be selected or generated automatically.

When recalling the refined CBR cycle introduced in Section 4.1.3, the major difference between the two approaches becomes clear. While the full-automatic approach automates the entire refined revise phase, the semi-automatic approach still requires a human user performing the "evaluate retrieval ranking" task. So, s/he controls the entire evaluation process while the evaluation of single cases is performed automatically.

Of course, acquiring utility feedback automatically leads to some important advantages compared to manual acquisition:

- *Availability of exact utility feedback:* A human similarity teacher has often difficulties to estimate the cases' utility absolutely. Instead s/he is often only able to compare the utility of two cases relatively to each other leading to ordinal utility feedback only. However, when measuring case utility in an automated way, the employed performance measure might allow an absolute estimation leading to exact utility feedback.

- *Less inconsistencies:* Another important aspect is the occurrence of inconsistencies. While decisions of human similarity teachers might be very subjective, the feedback provided by a performance measure should be based on objective criteria. So, the quality of feedback generated automatically should usually be higher than the quality of feedback provided by human similarity teachers.

- *Amount of available feedback:* The most important point probably is the effort required to obtain the feedback. With an automated procedure it might be possible to acquire much more training data with little effort. While the procedure might still be expensive with respect to computation time, it will at least safe costs arising through expensive domain experts. However, an automatic acquisition is, of course, only feasible, if an appropriate performance measure is already available or can easily be implemented (see also Section 6.5 and 6.6).

Number of Training Queries vs. Number of Evaluated Cases

When considering a way to acquire enough utility feedback, one must be aware of the fact that the emphasis of a particular set of utility feedback can be set differently.

One may base utility feedback on a large set of training queries while the number of evaluated cases is small for each query. Reasonable amount of utility feedback might also be obtained with few training queries, provided that many cases are evaluated for each query. How many queries and cases actually have to be evaluated, of course, also strongly depends on the complexity of the domain.

Which approach is more suitable, mainly depends on the realisation of the similarity teacher. When acquiring the feedback automatically, experiments indicate that it seems not be crucial whether the emphasis lays on the number of training queries or the number of cases to be evaluated (Stahl, 2001). Since an automated similarity teacher uses objective criteria to estimate the utility of cases, the evaluation of a huge amount of cases regarding one query is possible. However, when working with a human similarity teacher, who is mostly only able to provide ordinal utility feedback, the evaluation of many cases can lead to problems. Then the difference of the evaluated cases regarding their utility might be very small, so that the teacher is not able to recognise them. So, human similarity teachers will mostly provide more suitable feedback if they are confronted with few cases only. Of course, to obtain enough feedback then a human similarity teacher has to evaluate more queries.

4.3. Learning Similarity Measures from Utility Feedback

After the discussion of conceivable forms of utility feedback and how to acquire it, in this section it is described how the described feedback can be used to evaluate and learn similarity measures. First, the aim of the learning algorithm is motivated and some important definitions are introduced.

4.3.1. Top-Down Definition of Similarity Measures

First, we consider the situation that we are dealing with cardinal utility feedback (cf. Section 4.2.2). In Section 4.2.1 we have seen that from an abstract point of view, utility values can be interpreted as known points of an unknown function. The basic task of a learning algorithm can be described as a search for a similarity measure matching these given points as good as possible. Mathematically, this task can be seen as an interpolation process.

From a procedural point of view it can be interpreted as a *top-down definition* of similarity measures compared to the bottom-up definition described in Section 3.5 (Stahl, 2002a). Figure 4.6 illustrates the difference between these two contrary procedures. When defining similarity measures bottom-up, a domain expert encodes general knowledge about the utility of cases directly into the similarity representation formalism. We have characterised this knowledge as low-level knowledge.

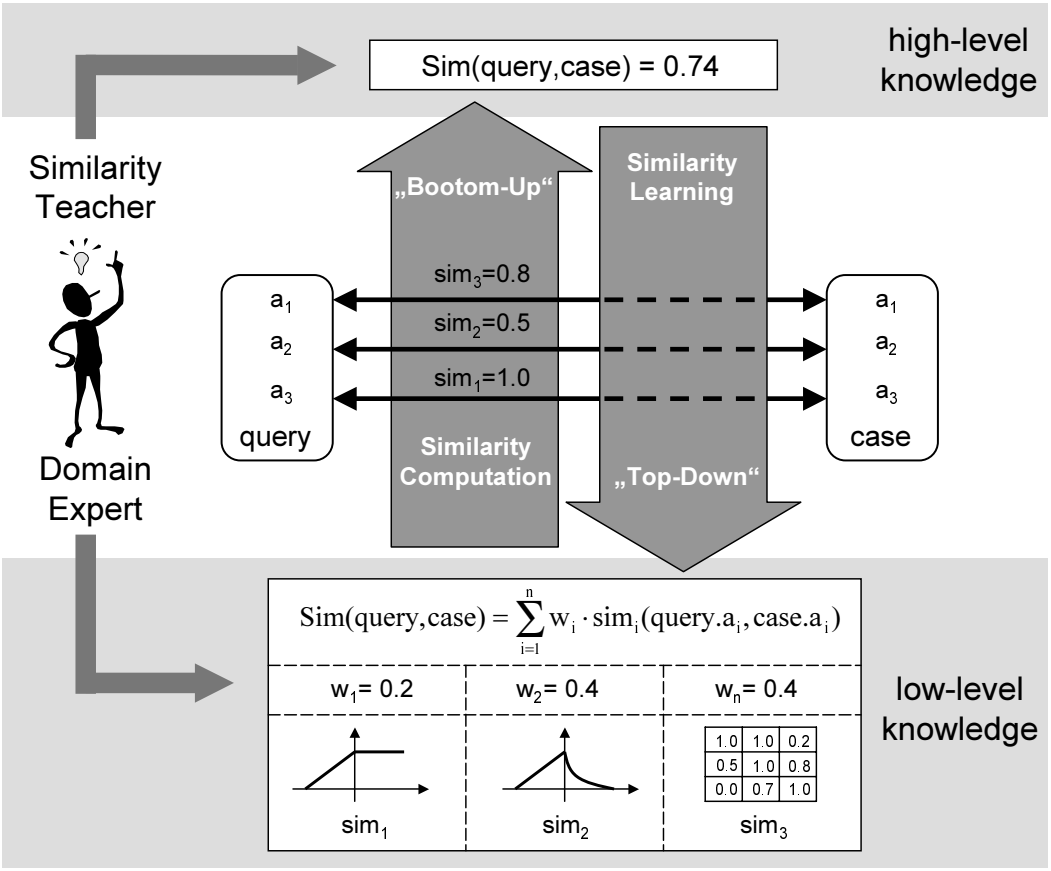


Figure 4.6.: Defining Similarity Measures: Bottom-Up vs. Top-Down

During case retrieval this knowledge is used to compute the actual utility of particular cases which we have characterised as high-level knowledge.

When defining similarity measures in a top-down manner, this process runs so to speak inversely. Now, the similarity teacher provides only high-level knowledge and it is the task of the learning procedure to construct an appropriate similarity measure by using the given representation formalism. Such a process might also be described as *goal-driven* and can be compared to a declarative programming language. Instead of defining how to reach a goal (like in procedural programming languages), here only the goal itself (i.e. the actual utility of cases) is described. The selection of a way to reach that goal (if possible) is left to the system, here the learning algorithm.

Dealing with Ordinal Feedback

If the learning algorithm has to deal with ordinal utility feedback (see Section 4.2.2), then we will be confronted with a slightly different situation. Because ordinal feed-

back does not provide absolute utility values, this feedback does not represent concrete points of the desired similarity measure. However, it provides *constraints* to be fulfilled by an adequate similarity measure.

For example, consider again an intersection through the unknown similarity measure like already illustrated in Figure 4.3. Now, we assume the existence of ordinal utility feedback leading to constraints like shown in Figure 4.7. Here, the learning algorithm is not confronted with an interpolation task but more with a *constraint satisfaction problem*. From an abstract point of view, it has to find a plane or a 2-dimensional function, respectively, (in the example a line or a linear function, respectively) that fulfills the constraints defined by the given ordinal utility feedback. Of course, this leads to much more possible similarity measures that are valid with respect to the feedback. In the example, not only the black line represents a sound function but also the shadowed lines are alternative functions.

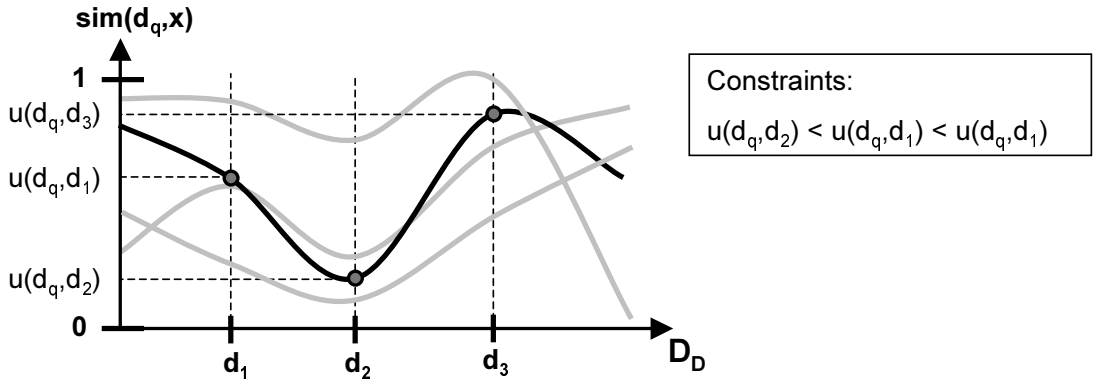


Figure 4.7.: Learning as a Constraint Satisfaction Problem

However, regarding the soundness criteria for similarity measures discussed in Section 3.2.2 we are usually only interested in constructing measures matching the utility preference relation. The construction of a measure that computes exact utility values usually is not required. This means many different similarity measures might be sound even if they compute quite different similarity values.

4.3.2. Training Data

In Section 4.2.2 the concept of utility feedback has already been described informally. In order to employ this concept for the development of an approach to learn similarity measures some basic definitions have to be introduced.

From the machine learning point of view, utility feedback can be characterised as *training data* to be used as input for a learning algorithm. Such training data implicitly contains knowledge about the utility function. The task of the learning algorithm is to make this knowledge explicit to be used more easily.

We now define how utility feedback can be represented formally to be processed by the learning algorithm. Basically, utility feedback is determined by an unknown utility function u. The only information available about u are the statements of a particular similarity teacher who possesses some implicit knowledge about that function u. Moreover, utility feedback always relies on a particular query because the utility of cases strongly depends on the current problem situation. So, feedback with respect to one query q can be seen as a single training example to be represented as follows:

Definition 4.2 (Training Example) Given a query q, a case base CB, and a utility function u that is implicitly given through some similarity teacher, a *training example*

$$TE_u(q) = (\ (c_1, u(q, c_1)), (c_2, u(q, c_2)), \ldots, (c_n, u(q, c_n))\ )$$

is an ordered set of pairs $(c_i, u(q, c_i))$ that fulfills the following conditions:

- $c_i \in CB$
- $u(q, c_i) \geq 0$
- $\forall\ 1 \leq i < j \leq n$ holds $u(q, c_i) \geq u(q, c_j)$

If $u(q, c_i) \in [0, 1]$ for all $1 \leq i \leq n$ represents the exact utility of c_i w.r.t. q, TE is called *cardinal training example*. If $u(q, c_i) \in \mathbb{N}^+$ for all $1 \leq i \leq n$, the utility values are only used as an index to express ordinal utility feedback, TE is called *ordinal training example*.

For ordinal training examples the values $u(q, c_i)$ do not represent actual utility values, but are only used to represent the corresponding partial order. For example, if it holds $u(q, c_1) = 2$ and $u(q, c_2) = u(q, c_3) = 1$, this means that case c_1 is more useful than c_2 and c_3. Moreover, the cases c_2 and c_3 are considered to be equally useful.

Of course, a single training example is insufficient to obtain useful learning results. This leads to the following definition:

Definition 4.3 (Training Data) Given a set of training queries $Q = \{q_1, q_2, \ldots, q_m\}$, a case base CB, and a utility function u, a set

$$TD_u = \{TE_u(q_1), TE_u(q_2), \ldots, TE_u(q_m)\}$$

is called *training data* where $TE_u(q_i)$ is the corresponding training example for query q_i. We distinguish between *ordinal training data* consisting of ordinal training examples and *cardinal training data* consisting of cardinal training examples.

Note, that individual training examples of a particular training data set do not necessarily have to contain the same cases. Further, training data has not necessarily to be consistent, i.e. one training example might rank cases differently than another one in the presence of noise.

4.3.3. Retrieval Error

When comparing Definitions 3.4 and 4.2 one can notice a common property, namely the determination of a partial order for a set of given cases. Hence, a single training example can be characterised as the partial definition of an *optimal retrieval result*, i.e. cases in the retrieval result should be ordered in the same manner as determined through the training example. However, a similarity-based retrieval result[1] is determined by a particular similarity measure that not necessarily represents a sufficient approximation of the underlying utility function. The similarity measure might determine case orders that differ from the optimal orders defined by the training data.

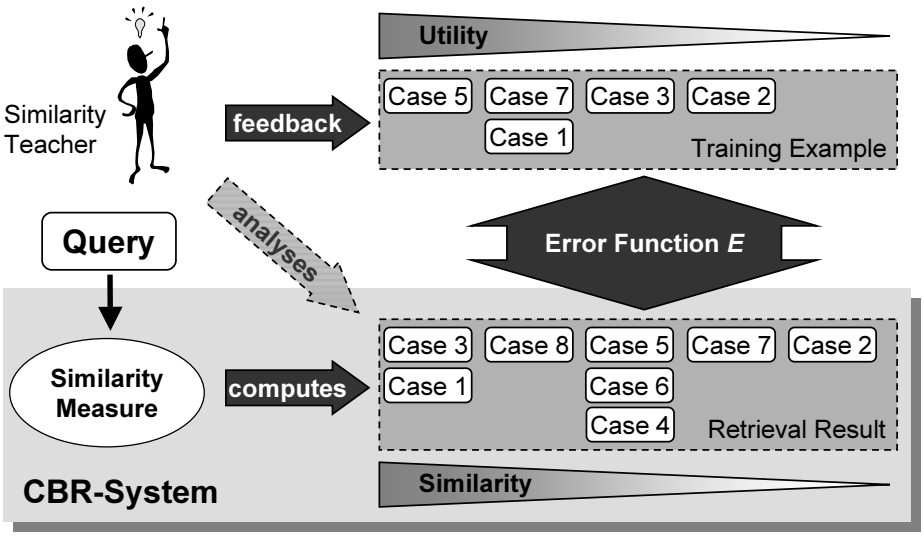


Figure 4.8.: Retrieval Error

To be able to judge the quality of a given similarity measure one is interested in the degree of the deviation between the computed retrieval results and corresponding training examples. To measure this deviation we introduce the concept of the *retrieval error* computed by an *error function* that compares two case orders like illustrated in Figure 4.8. The objective of this function is the computation of a value measuring the differences between a retrieval result determined through a given sim-

[1]Here, we assume unlimited retrieval results according to Definition 3.4.

ilarity measure and a training example based on the feedback of a similarity teacher. Basically, this error function should fulfill the following requirements:

- It should consider cases only that appear in the training example. Cases exclusively contained in the retrieval result are not relevant because no information about their utility is available.
- If and only if the case order of the retrieval result matches the order of the cases contained in the training example, the resulting error value should be zero.
- An error value greater zero should estimate the differences between the two case orders. This means, more differences should lead to a greater error value.

When dealing with cardinal utility feedback, we do not expect that the computed similarity values match the given utility values, i.e. we are only interested in correct case rankings. The idea of exact utility values provided by some similarity teacher is to facilitate the learning process only by providing more information.

When considering training examples containing ordinal information only, the following definitions can be seen as a first attempt to obtain an error function fulfilling the requirements enumerated previously:

Definition 4.4 (Ordinal Evaluation Function) Let q be a query, $TE_u(q)$ be a corresponding training example with $(c_i, u(q,c_i)), (c_j, u(q,c_j)) \in TE_u(q)$ and $u(q,c_i) \geq u(q,c_j)$. Further, let Sim be a similarity measure. The function ef_o defined as

$$ef_o(TE_u(q), Sim, (c_i,c_j)) := \begin{cases} \alpha & if \begin{array}{l} u(q,c_i) > u(q,c_j) \wedge \\ \mid Sim(q,c_i) - Sim(q,c_j) \mid \leq \epsilon \end{array} \\ \beta & if \begin{array}{l} u(q,c_i) = u(q,c_j) \wedge \\ \mid Sim(q,c_i) - Sim(q,c_j) \mid > \epsilon \end{array} \\ \gamma & if \begin{array}{l} u(q,c_i) > u(q,c_j) \wedge \\ Sim(q,c_i) - Sim(q,c_j) < -\epsilon \end{array} \\ 0 & otherwise \end{cases}$$

is called *ordinal evaluation function* where $\epsilon \in [0,1]$ is called *similarity indistinguishability threshold*. The values $\alpha, \beta, \gamma \geq 0$ are called *error qualifiers*.

The objective of this function is the evaluation of a given similarity measure sim regarding the correct ranking for a particular case pair (c_i, c_j). The similarity indistinguishability threshold ϵ can be used to neglect influences on the ranking induced by very small similarity differences. The error qualifiers α, β, γ can be used to define the influence of the three distinguished types of errors, namely the

α-Error: Here, case c_1 is considered to be more useful than case c_2, but the similarity measure assigns the same similarity to both cases.

β-Error: Here, case c_1 is considered to be equally useful as case c_2, but the similarity measure computes significantly different similarities.

γ-Error: While case c_1 is considered to be more useful than case c_2, the similarity measure computes a greater similarity for c_2 than for c_1.

Typically, γ should be larger than α and β because it corresponds to the worst error where the similarity measure ranks the two cases contrary to the training data. In both remaining situations, either the similarity measure or the training data does not differentiate between the two presented cases. The introduced evaluation function can now be used to define a first version of the desired error function:

Definition 4.5 (Index Error) Consider a similarity measure Sim, a query q and a respective training example $TE_u(q) = ((c_1, u(q, c_1)), (c_2, u(q, c_2)), \ldots, (c_n, u(q, c_n))$. We define the *index error* induced by Sim w.r.t. to $TE_u(q)$ as

$$E_I(TE_u(q), Sim) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} ef_o(TE_u(q), Sim, (c_i, c_j)) \cdot \frac{i+\theta}{i}$$

where $\theta \geq 0$ is called the *error-position weight*.

The error-position weight allows to increase the influence of wrong case rankings occurring among more useful cases compared with failures occurring among less useful cases. The greater the parameter θ, the greater is the impact of wrong similarities computed for more useful cases. If $\theta = 0$, the position within the training example of a case that has been ranked wrongly by the similarity measure is irrelevant for the computation of the index error.

The index error can be seen as a measure for the quality of a given similarity measure regarding a particular query and a particular utility function. However, one is interested in similarity measures that supply reasonable case rankings for arbitrary queries or at least for some set of queries. This leads to the following extension of the index error allowing the evaluation of an entire training data set:

Definition 4.6 (Average Index Error) Consider a set of training queries $Q = \{q_1, q_2, \ldots, q_m\}$, corresponding training data $TD_u = \{TE_u(q_1), TE_u(q_2), \ldots, TE_u(q_m)\}$ and a similarity measure Sim. We define the *average index error* induced by Sim w.r.t. to Q as

$$\hat{E}_I(TD_u(Q), Sim) = \frac{1}{m} \cdot \sum_{i=1}^{m} E_I(TE_u(q_i), Sim)$$

In Section 4.1.3 we have introduced a new task called "evaluate similarity measure" to be carried out during the refined retain phase of the CBR cycle. The average

index error now provides an instrument to implement this task. It represents a measure reflecting the quality of a given similarity measure according to a training data set acquired during the refined revise phase. The higher the average index error, the lower is the ability of the similarity measure to approximate the unknown utility function. By introducing some threshold, the average index error might be used to trigger a maintenance operation with the aim to improve the similarity measure currently used by the CBR system.

4.3.4. Optimising Similarity Measures

This section presents an approach to improve similarity measures by employing the kind of training data introduced in the previous section. This approach applies machine learning strategies to construct a similarity measure that corresponds to knowledge about the utility function implicitly contained in the training data. Hence, the learning procedure can also be seen as a compilation process that transfers knowledge from the training data into the representation of the similarity measure.

The Learning Scenario

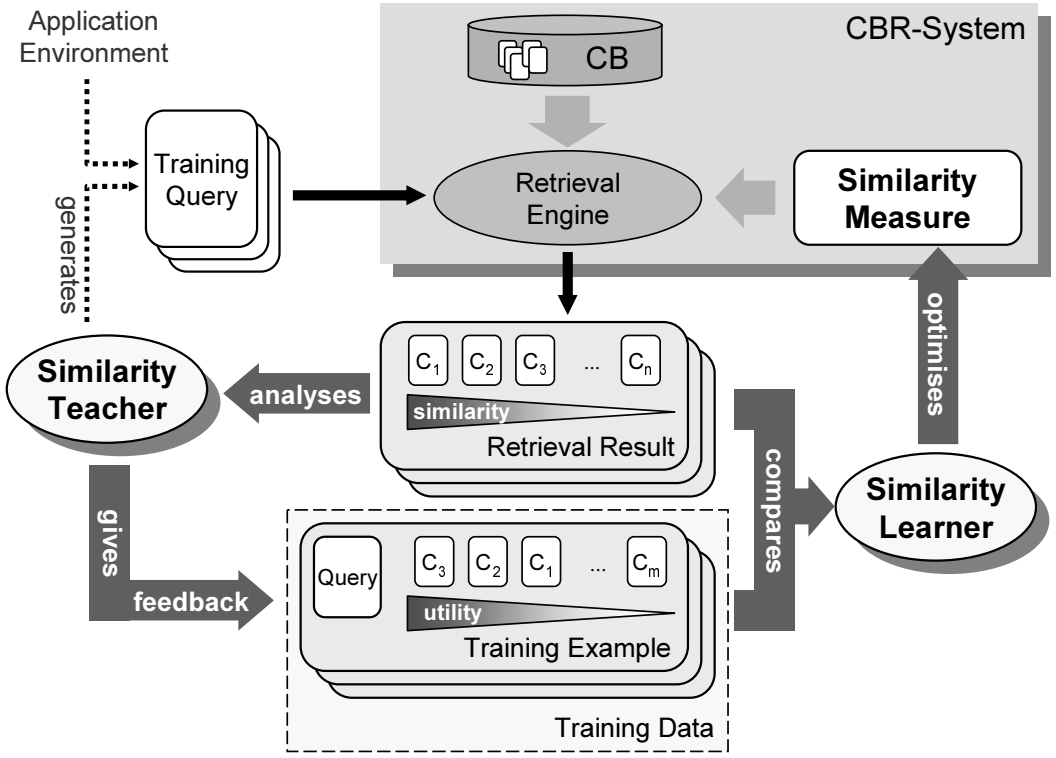


Figure 4.9.: Scenario for Learning Similarity Measures

The general scenario of this learning approach is illustrated in Figure 4.9. On the one hand, we have a CBR system consisting of a case base, a similarity measure, and a retrieval engine. Of course, it might also contain additional elements, for example, adaptation functionality. However, here these elements can be neglected because they are not important for our learning framework. The first crucial element of the learning scenario is the already introduced similarity teacher. After the execution of a similarity-based retrieval the similarity teacher analyses the retrieval results and gives feedback about the retrieved cases' utility. The queries initiating the retrieval might be provided by the application environment or by the similarity teacher. The result of the similarity teacher's feedback is a training data set consisting of one training example for each query.

The second crucial element of the learning scenario is a concept called *similarity learner*. The task of the similarity learner is the modification of the system's similarity measure in order to increase the retrieval quality measured by an error function that compares achieved retrieval results with available training data. A possible implementation of such an error function has been introduced in Definition 4.6. In the following it is shown that an appropriate error function also provides a powerful instrument to carry out a successful modification of the similarity measure.

Basically, the modification of the similarity measure can be seen as an optimisation procedure to be realised by the similarity teacher implemented in form of a software agent. Therefore, this software agent must be able to initiate and evaluate retrieval results. Further, the agent must have access to the system's internal similarity measure and must be allowed to modify it.

Optimisation Loop

The realisation of the optimisation procedure is illustrated in Figure 4.10. In principle, this procedure can be characterised as a loop consisting of the following processes executed sequentially:

1. **Initiate Retrievals:** This first process initiates a similarity-based retrieval for all queries stored in the training examples. The result is a set of retrieval results.

2. **Evaluate Measure:** The task of this process is the evaluation of the similarity measure currently used. Therefore, the retrieval results previously achieved have to be compared with the training data by applying the presumed error function.

3. **Check Stop Criterion:** The similarity measure's quality estimation then determines whether further modification of the measure seems to be promising or not. If not, the entire optimisation loop is stopped. This decision might also

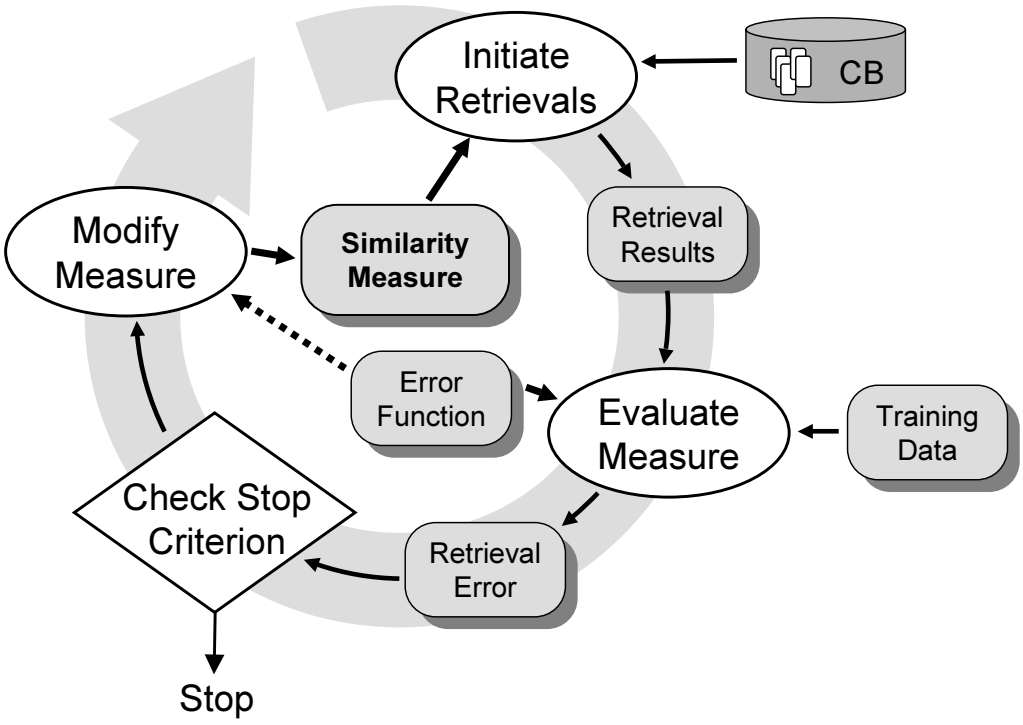


Figure 4.10.: Optimisation Loop

be influenced by other criteria, for example, by the number of modifications already performed.

4. **Modify Measure:** If the stop criterion is not reached, the last process of the optimisation loop has to modify the similarity measure in a way that its quality will hopefully be increased. Depending on the particular implementation of this modification process, it might also use the error function to control the process. After modifying the similarity measure a new iteration of the optimisation loop is started.

Limitations

If the optimisation procedure has been stopped, the quality of the system's similarity measure should be higher or at least not lower than before optimisation. Unfortunately, this can only be guaranteed with respect to the queries contained in the training data. And even with respect to these queries it cannot be ensured that more accurate case rankings will now be achieved *for all* of these queries. The optimised similarity might even lead to more inaccurate rankings for some of these queries. The reason for this is the nature of the supposed error function that determines the retrieval quality in average for all available training examples. Therefore,

it can only be guaranteed that the retrieval quality is not reduced in average regarding all considered queries.

Optimisation Triggers

If optimisation of the similarity measure is part of the initial knowledge acquisition phase when developing a CBR application, a single optimisation process might be sufficient. However, the objective of the learning approach might also be to improve or maintain the quality of the similarity measure continuously during the entire lifetime of the application. Then one has to consider accurate triggers that initiate the described optimisation loop. After the optimisation loop has been executed once two possible reasons can require anew optimisation procedure:

New Training Data: Since the optimisation process is restricted to the available training data, the availability of new training examples may be a reason for anew optimisation attempt. On the one hand, additional training examples increase the foundation of the optimisation. Hence, more training data should enable the similarity learner to generate more generalised similarity measures leading to more accurate retrieval results even for queries not contained in the training data. On the other hand, new training examples might also reflect changes in the underlying utility function. For example, if a customer in an eCommerce scenario changes her/his preferences, new utility feedback can be used to adapt the similarity measure regarding the changed preferences. In this situation it might also be useful to remove old training examples from the training data because they probably contain obsolete knowledge about the utility function.

Changes in the Case Base: Another event that might require anew optimisation process, is a modification of the case base. On the one hand, adding new cases is not crucial because the training data only considers a subset of the cases in the case base anyway. However, removing cases can have significant effects on the optimisation results. Since the training data represents not only knowledge about a particular set of queries but also about a particular set of cases, deleting some of these cases might change the outcome of the optimisation. In such a case, the training data has to be updated by removing the obsolete cases. After this update, anew optimisation process might lead to another, hopefully more accurate similarity measure as the one generated with the old training data.

Modifying the Similarity Measure

Obviously, the most crucial sub-process of the optimisation loop is the one that modifies the similarity measure. So far, we have said nothing about the concrete

implementation of this important process. Basically, a modification of the current similarity measure can be realised in two different ways:

By Chance: Here, the modification is performed randomly without using knowledge about the training data. The effect of the modification is then only estimated in the subsequent evaluation process.

By Knowledge: In order to increase the probability that the modification leads to an improved similarity measure, knowledge about the training data can be used to guide the modification. This knowledge might be extracted directly from the training data or it might be provided by the error function actually used to evaluate the similarity measure.

While a modification by chance can be realised quite simple, a knowledge driven modification requires more sophisticated learning algorithms. However, the exploitation of knowledge, of course, increases the performance of the optimisation process, i.e. probably less iterations of the optimisation loop are required.

Besides the exclusive application of only one of these methods, both approaches can also be combined. The next chapter deals with concrete possibilities to implement the modification process. Two particular learning algorithms based on two common machine learning approaches are introduced in detail.

5. Learning Algorithms

In Chapter 4 a general framework for learning similarity measures has been introduced. The basic idea of this framework is to exploit feedback about the utility of cases for particular problem situations provided by some similarity teacher in order to learn accurate similarity measures. It was shown that the central learning task can be interpreted as an optimisation procedure. During this procedure an initial similarity measure has to be modified in order to obtain a more suitable one leading to better retrieval results. However, so far no concrete approaches to implement accurate modification operations have been discussed. In this section two different possibilities for controlling the modification process are presented in detail. Both approaches are coupled with concrete algorithms realising the central functionality of the similarity learner introduced in Section 4.3.4. First, we shortly discuss the formal task to be solved by these learning algorithms.

5.1. The Formal Learning Task

Basically, the introduced optimisation loop (cf. Figure 4.10) can be described as a search process within the space of representable similarity measures. This means, it is the goal of the similarity learner to find a similarity measure leading to desired retrieval results and that can be represented with the assumed representation formalism (see Section 3.3). In order to implement this search we have already introduced a powerful instrument, namely the retrieval error computed by some error function (cf. Section 4.3.3). From an abstract point of view, this error function can be defined as follows:

Definition 5.1 (Error Function Induced by Training Data) A given training data set TD induces an error function $E_{TD} : SIM \longrightarrow \mathbb{R}$ that measures the retrieval error produced by a similarity measure $Sim \in SIM$ w.r.t. TD where SIM represents the set of all representable similarity measures.

So, the particular error function E_{TD} assigns an error value to each representable similarity measure with respect to a training data set TD like illustrated in Figure 5.1. From this abstract point of view, the issue of the optimisation process can now be redefined as a minimisation of the error function E_{TD}. In other words, the *optimal similarity measure* Sim_{opt} leading to the minimum value $e_{min} = E_{TD}(Sim_{opt})$

of the error function E_{TD} has to be found. During this minimisation process two crucial problems have to be considered.

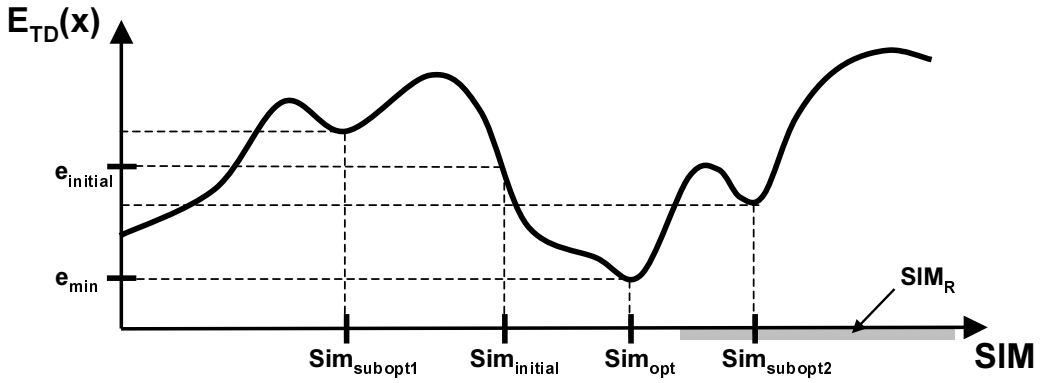


Figure 5.1.: Minimising the Error Function

On the one hand, the minimal error value e_{min} has not necessarily to be equal zero—in practice e_{min} is mostly significantly greater zero. This problem is caused by the *hypothesis space* SIM. Because this space of representable similarity measures does only contain a certain subset of all possible measures (regarding Definition 3.3), the *ideal similarity measure* that fulfills all constraints defined by the training data is mostly not included. Hence, in the best case the similarity learner is only able to find the optimal similarity measure $Sim_{opt} \in SIM$ minimising E_{TD}.

On the other hand, the similarity learner has usually to deal with several minima of the error function E_{TD}. Basically, we distinguish between two kinds of minima:

- A *local minima* is given through a similarity measure $Sim_{subopt} \in SIM_R \subset SIM$ coupled with a minimal error value for some region SIM_R of the entire search space, i.e. $\forall x \in SIM_R \; E_{TD}(Sim_{subopt}) \leq E_{TD}(x)$. Such a similarity measure is called *suboptimal*.

- A *global minima* is given through a similarity measure Sim_{opt} so that it holds $\forall x \in SIM \; E_{TD}(Sim_{opt}) \leq E_{TD}(x)$. Such a similarity measure is called *optimal*.

While two different local minima $Sim_{subopt1}$ and $Sim_{subopt2}$ might have different error values, i.e. $E_{TD}(Sim_{subopt1}) \neq E_{TD}(Sim_{subopt2})$, two possible global minima Sim_{opt1} and Sim_{opt2} are always characterised by identical error values, i.e. $E_{TD}(Sim_{opt1}) = E_{TD}(Sim_{opt2})$. Hence, given several local minima one would prefer the one with the lowest error value, but, in principle, it makes no sense to prefer a particular global minima provided that the error function contains more than one.

Unfortunately, in general it is not possible to guarantee the determination of an optimal similarity measure. This would require the total enumeration and evaluation

of all representable similarity measures. This is usually an infeasible approach due to its computational complexity. However, depending on the applied learning algorithm at least finding a suboptimal similarity measure can be ensured. Nevertheless, the return of a suboptimal similarity measure can only be seen as a success, if it also leads to a decrease of the error value compared to the error value of the initial similarity measure $Sim_{initial}$. For example, in Figure 5.1 the local minimum determined through $Sim_{subopt1}$ corresponds to a greater error value than the error value $e_{initial}$ determined through $Sim_{initial}$. Therefore, the similarity teacher must at least be able to find the similarity measure $Sim_{subopt2}$ to perform successful optimisation of $Sim_{initial}$.

In Section 4.3.4 we have already mentioned two principle approaches to implement the described minimisation task. For example, one might repeatedly select similarity measures randomly in the hope to find one that is coupled with a low error value. Although such a minimisation by chance sounds infeasible due to its presumably poor efficiency, a cunning realisation allows the development of a powerful similarity teacher. In Section 5.3 we describe how this approach can be implemented. Before, we introduce an alternative approach that exploits knowledge about the shape of the error function in order to find an accurate similarity measure efficiently. However, this approach can only be applied to learn one part of the entire similarity measure, namely attribute weights. In the following we assume a starting situation given through

- an attribute-value based case model $C = (D, L)$ where the case characterisation model D consists of $n > 0$ attributes $A_1, A_2, \ldots, A_n$,
- a global similarity measure Sim according to Definition 3.17 consisting of local similarity measures $sim_1, \ldots, sim_n$ represented as difference-based similarity functions (see Definition 3.15) or similarity tables (see Definition 3.14), a global attribute weight-vector $\vec{w} = (w_1, \ldots, w_n)$, and a weighted average aggregation function, i.e $Sim(q, c) = \sum_{i=1}^{n} w_i \cdot sim_i$,
- and some training data set $TD_u(Q)$ where u is the utility function implicitly given through some similarity teacher and $Q = \{q_1, q_2, \ldots, q_m\}$ is a set of training queries.

5.2. Gradient Descent Approach

When having certain information about the shape of the error function, this knowledge can be used to guide the described search for an optimal similarity measure. An approach that utilises knowledge about error functions in order to implement efficient search strategies is the *gradient descent approach*. This approach is commonly used by several machine learning strategies, for example, by the backpropagation

method used in neural networks (Mitchell, 1997). It is also employed by some existing approaches to learning attribute weights (Lowe, 1993; Wettschereck and Aha, 1995) (see also Section 9). The basic presumption of this approach is the possibility to compute the gradient of the error function for a particular point of the search space. Having this information, it is possible to direct the search into regions of the search space that seem to be promising regarding the search goal, i.e. finding minima of the error function.

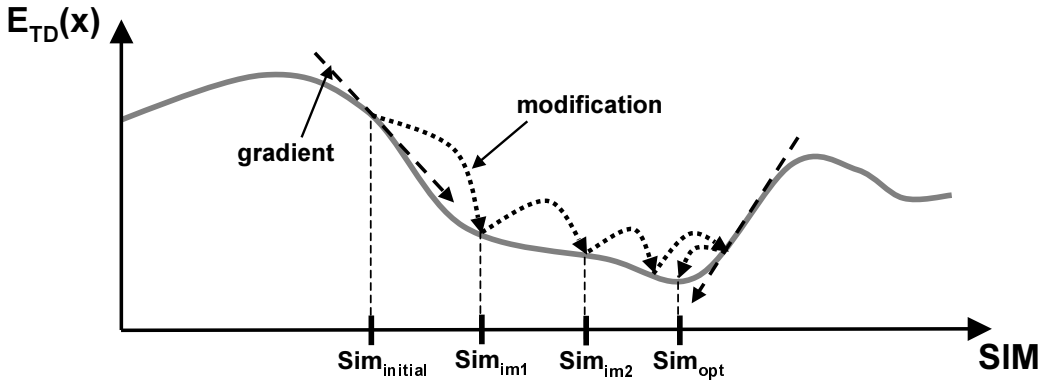


Figure 5.2.: Gradient Descent Approach

The basic idea of this approach is illustrated in Figure 5.2. The start point of the search is again an initial similarity measure $Sim_{initial}$. If it would be possible to determine the gradient of the error function in this point, here illustrated as a tangent, this information could be used to modify $Sim_{initial}$ in a controlled way. The resulting similarity measure Sim_{im1} can be interpreted as the outcome of one single iteration of the optimisation loop described in Section 4.3.4, and thus represents an intermediate result of the entire optimisation process. Because the actual modification exploits the gradient information, i.e. it "follows" the direction of the gradient, the intermediate similarity measure Sim_{im} hopefully represents a better solution characterised by a lower error value. By repeating this procedure in a controlled way, it can be ensured that the approach converges to a minimum of the error function. Unfortunately, in general it cannot be guaranteed that this minimum is indeed a global minimum. Nevertheless, the gradient approach is a very efficient approach to find at least suboptimal solutions.

Although the approach seems to be quite simple, in practice some crucial requirements must be fulfilled. To be able to compute the gradient of the error function for a particular point of the search space, the error function must be partially differentiable with respect to the parameters to be optimised. In the following, we introduce a special error function building the foundation for the development of a gradient descent algorithm for learning global attribute weights.

5.2.1. Choosing the Error Function

When recalling the average index error introduced in Definition 4.6, it becomes clear that this particular error function is not suitable to realise a gradient descent approach. The problem with this function is that it is not partially differentiable with respect to some parameter of the similarity measure representation. In order to overcome this problem, an alternative error function has to be developed.

We consider again the situation that only ordinal information contained in the training examples is exploited during the learning process. Thus, we choose again the ordinal evaluation function already introduced in Definition 4.4 to be used as the foundation of the desired error function:

Definition 5.2 (Similarity Error for a Case Pair) Let q be a query, Sim be a similarity measure, and (c_1, c_2) be a case pair with $u(q, c_1) \geq u(q, c_2)$. Further, we assume that the underlying case characterisation model consists of r attributes with corresponding local similarity measures $sim_1, \ldots, sim_r$. We define the *similarity error for a case pair* (c_1, c_2) as

$$\begin{aligned} & E_S(TE_u(q), Sim, (c_1, c_2)) = \\ & \quad = \; \mid (Sim(q, c_1) - Sim(q, c_2)) \mid \cdot \, ef_o(TE_u(q), Sim, (c_1, c_2)) \\ & \quad = \; \mid (\sum_{i=1}^{r} w_i \cdot sim_i(q, c_1) - \sum_{i=1}^{r} w_i \cdot sim_i(q, c_2)) \mid \cdot \, ef_o(TE_u(q), Sim, (c_1, c_2)) \\ & \quad = \; \mid (\sum_{i=1}^{r} w_i \cdot (sim_i(q, c_1) - sim_i(q, c_2))) \cdot ef_o(TE_u(q), Sim, (c_1, c_2)) \mid \end{aligned}$$

This error definition can be seen as an extension of the ordinal evaluation function (cf. Definition 4.4). Now the error concerning a case pair is not only influenced by the error qualifiers of the ordinal evaluation function, but it directly depends on the amalgamation function of the similarity measure. This error for a case pair can simply be extended to an error computation for an entire training example:

Definition 5.3 (Similarity Error) Let q be a query and $TE_u(q) = ((c_1, u(q, c_1)), \ldots, (c_n, u(q, c_n)))$ the corresponding training example. The *similarity error* for TE induced by a similarity measure Sim is defined as

$$E_S(TE_u(q), Sim) = \sum_{k=1}^{n-1} \sum_{l=k+1}^{n} E_S(TE_u(q), Sim, (c_k, c_l)) \cdot \frac{k + \theta}{k}$$

where $E_S(TE_u(q), Sim, (c_k, c_l))$ is the prior defined similarity error for the case pair (c_k, c_l) and θ is the error-position weight.

Here, the error-position weight θ plays the same role as already described for the index error (see Definition 4.5). As for the index error, the similarity error can be extended to consider the entire training data set and not only one single training example:

Definition 5.4 (Average Similarity Error) Consider a set of training queries $Q = \{q_1, q_2, \ldots, q_m\}$, respective training data $TD_u = \{TE_u(q_1), TE_u(q_2), \ldots, TE_u(q_m)\}$, and a similarity measure Sim. We define the *average similarity error* induced by Sim as

$$\begin{aligned}
\hat{E}_S(TD_u(Q), Sim) &= \frac{1}{m} \cdot \sum_{j=1}^{m} E_S(TE_u(q_j), Sim) \\
&= \frac{1}{m} \cdot \sum_{j=1}^{m} \sum_{k=1}^{n_j-1} \sum_{l=k+1}^{n_j} E_S(TE_u(q_j), Sim, (c_k, c_l)) \cdot \frac{k+\theta}{k} \\
&= \frac{1}{m} \cdot \sum_{j=1}^{m} \sum_{k=1}^{n_j-1} \sum_{l=k+1}^{n_j} \mid (\sum_{i=1}^{r} w_i \cdot (sim_i(q_j, c_k) - sim_i(q_j, c_l))) \cdot \\
&\quad ef_o(TE_u(q_j), Sim, (c_k, c_l)) \mid \cdot \frac{k+\theta}{k} \qquad (5.1)
\end{aligned}$$

where n_j is the number of cases contained in training example $TE_u(q_j)$.

As required by the gradient descent approach, this special error function can be partially differentiated with respect to some important part of the similarity measure, namely the attribute weights. So, the partial derivation of Equation 5.1 arises as follows:

$$\begin{aligned}
\frac{\partial \hat{E}_S(TD_u(Q), Sim)}{\partial w_i} &= \frac{1}{m} \cdot \sum_{j=1}^{m} \sum_{k=1}^{n_j-1} \sum_{l=k+1}^{n_j} (sim_i(q_j, c_k) - sim_i(q_j, c_l)) \cdot \\
&\quad sgn(Sim(q, c_k) - Sim(q, c_l)) \cdot \\
&\quad ef_o(TE_u(q_j), Sim, (c_k, c_l)) \cdot \frac{k+\theta}{k}
\end{aligned}$$

5.2.2. The Gradient Descent Algorithm

After introducing the average similarity error $\hat{E}_S$ we now describe an algorithm that minimises this error function by adjusting the attribute weights w_i as part of the similarity measure Sim. This gradient descent algorithm performs an iterative search for a local minimum of the error function $\hat{E}_S$ according to the following pseudo code:

Input: training data $TD_u(Q)$, initial similarity measure Sim
Output: optimised similarity measure

procedure gradient_descent_algorithm($TD_u(Q)$, Sim) {

1. *stop-predicate* := false;
2. compute average similarity error $\hat{E}_S(TD_u(Q), Sim)$;
3. initialise learning rate λ;
4. **while** *stop-predicate* = false **do**:
   a) generate new Sim' by updating weights $\vec{w}$ contained in Sim: $\forall i\ w_i' := w_i - \frac{\partial \hat{E}_S(TD_u(Q),Sim)}{\partial w_i} \cdot \lambda$;
   b) normalise $\vec{w}'$: $\forall i\ w_i' := \frac{w_i}{\sum_{j=1}^{n} w_j}$;
   c) compute average similarity error $\hat{E}_S(TD_u(Q), Sim')$;
   d) **if** $\hat{E}_S(TD_u(Q), Sim') < \hat{E}_S(TD_u(Q), Sim)$ **then** $Sim := Sim'$ **else** $\lambda := \frac{\lambda}{\lambda - reduction - rate}$
   e) *stop-predicate* := evaluate_stop-predicate();
5. **return** optimised similarity measure Sim;

}

Basically, the algorithm can be separated into two phases which are repeated iteratively until the applied stop-predicate becomes true. These phases correspond to the evaluation and the modification processes of the optimisation loop introduced in Section 4.3.4. In the first phase, the algorithm computes the average similarity error E_S for the current similarity measure Sim including the actual weight-vector w regarding the provided training data set $TD_u(Q)$. In the second phase, the derivation of the error function is used to modify the current weight-vector $\vec{w}$ resulting in an updated weight-vector $\vec{w}'$. If the next iteration shows an improvement in the error, $\vec{w}'$ and so Sim' is accepted, i.e. $Sim := Sim'$. Otherwise the learning rate is decreased by dividing it through the $\lambda - reduction - rate$ and the process is continued with the previous similarity measure Sim. In the following section four important parameters that influence the execution of the algorithm are discussed in more detail.

5.2.3. Control Parameters

Of course, the outcome of the presented algorithm, i.e. the returned similarity measure containing the updated weights, strongly relies on the provided training data set $TD_u(Q)$. However, it is also influenced by four additional parameters, namely

- the *initial weight-vector* $\vec{w}$,
- the used *stop-predicate*,
- the initial *learning rate* $\lambda > 0$,
- and the λ-*reduction-rate*> 1.

Initialisation of Feature Weights

Basically, the determination of the starting point, here given by the initial weights, is crucial for the success of any gradient descent algorithm. As already discussed in Section 5.1, in general, it cannot be guaranteed that a global minimum of the error function will be found. Whether the algorithm actually converges towards a global minimum or only to a local one strongly depends on the initial similarity measure chosen. For example, in Figure 5.3 the two initial similarity measures shown, $Sim_{initial1}$ and $Sim_{initial2}$, would lead to different outcomes of the gradient descent search. While selecting $Sim_{initial1}$ would enable the algorithm to find the global minimum Sim_{opt}, starting the search at $Sim_{initial2}$ would cause the algorithm to be "caught" in the local minimum Sim_{subopt}.

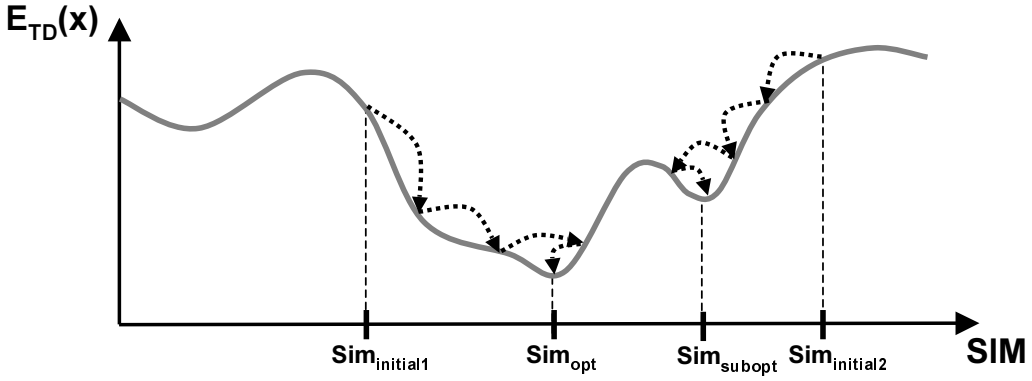


Figure 5.3.: Impact of the Initial Similarity Measure

The problem with the selection of an accurate starting point is the missing knowledge about the shape of the error function. Although the derivation provides some knowledge about the shape in a certain small area, the entire shape is not known,

otherwise the whole search process would be, of course, superfluous. Basically, three possible approaches to initialise the local similarity measure or the weights, respectively, can be distinguished:

1. the use of a uniform weight-vector, i.e., $\forall i\ w_i = \frac{1}{n}$
2. a random initialisation of the weights
3. a domain expert defines the initial weights

The first two approaches obviously do not exploit any knowledge about the domain or the shape of the error function. Thus, here one can only hope that the initially chosen weights lead to a satisfying outcome of the gradient descent algorithm. The third approach is usually the best choice because a domain expert should be able to initialise the weights in a more "intelligent" way due to her/his domain knowledge. If the domain expert would be able to determine an initial similarity measure close to the optimal similarity measure Sim_{opt}, this would increase the probability that the algorithm converges towards Sim_{opt}.

The Learning Rate

As in other machine learning approaches, the choice of the initial learning rate λ is also crucial for the outcome of the algorithm. Generally, a tradeoff between a very small and a too large λ can be noticed. On the one hand, a quite small λ leads to a poor convergence speed, however, the advantage of a small λ is a higher probability of finding a local minimum close to the starting point. This might be of interest if the weights have been initialised by a domain expert because this local minimum often corresponds to a very accurate similarity measure like described in the previous section. On the other hand, if λ is too large, the algorithm might "jump over" this nearby minimum. This leads to the risk that the algorithm will "get caught" in another local minimum corresponding to a much higher error value.

The relation between the initial similarity measure and the initial learning rate is illustrated in Figure 5.4. Here, the initial similarity measure $Sim_{initial1}$ is closer to the global minimum than $Sim_{initial2}$. When starting at $Sim_{initial1}$, the smaller learning rate λ_2 leads to the optimum while the greater learning rate λ_1 causes the algorithm to "jump over" the optimum. However, if the search is started at $Sim_{initial2}$, the greater learning rate should be preferred. Here, it enables the algorithm to "jump over" the local minimum representing an "obstacle" on the way towards the global minimum.

Of course, similar to the selection of the initial similarity measure, the selection of an accurate learning rate is difficult due to missing knowledge about the location of the error function's local and global minima. The only possibility is the choice of a learning rate that has led to good results in some evaluation experiments. To

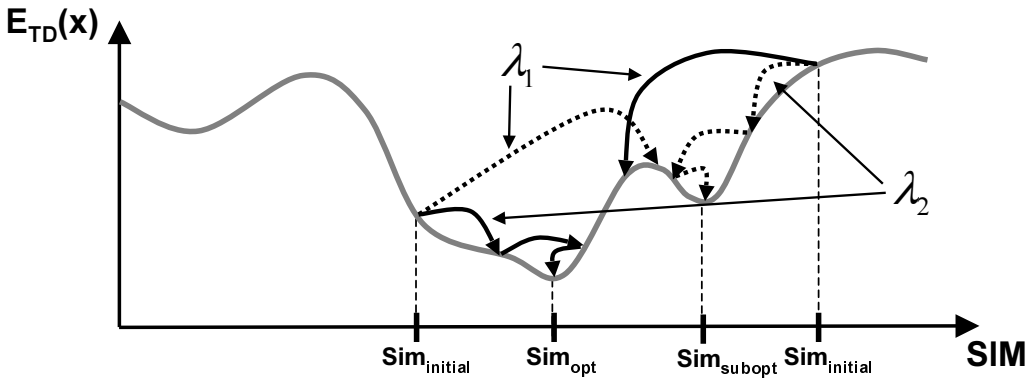


Figure 5.4.: Impact of Learning Rate

facilitate the specification of the initial learning rate we propose to define it with the help of the desired changes in the initial weights during the first iteration of the optimisation loop. Therefore, one has to define the maximal change in one of the weights w_i represented by the value $\Delta_{max}w \in [0, 1]$.

Consider the example of a domain with four attributes and an initial weight-vector $w = (0.25, 0.25, 0.25, 0.25)$. If the user determines $\Delta_{max}w = 0.05$, the new weight-vector computed in the first learning step will at least contain one weight w_i' with $w_i' = 0.2$ or $w_i' = 0.3$. For all other weights w_j' it then holds $w_j' \in [0.2, 0.3]$.

Unfortunately, $\Delta_{max}w$ cannot directly be employed as learning rate. To achieve the described behaviour of the learning algorithm the initial learning rate has to be computed dynamically prior to the actual search process. Here, it has to be considered that the required learning rate also depends on the derivation of the error function. The value of the error function's derivation for the initial similarity measure is used to modify the initial weights. Therefore, a "flat" error function in the area of the starting point requires a greater learning rate compared to a very "steep" error function. To obtain a learning rate leading to the desired changes in the weights, it must be computed as follows:

$$\lambda = \frac{\Delta_{max}w}{max_{i=1}^{r}\{\Delta w_i\}} \quad \text{where} \quad \Delta w_i = \frac{\partial \hat{E}_{Sim}}{\partial w_i}$$

The Stop-Predicate

To guarantee the termination of a gradient descent algorithm a stop-predicate has to be introduced. Generally, various possibilities to realise the stop-predicate exist (Wilke and Bergmann, 1996):

Minimal Improvement of Similarity Error: When converging to a minimum of the error function, the achieved improvements concerning the quality of the optimised similarity measure will decrease. So, a threshold τ for the difference

between the computed error values of two subsequent intermediate similarity measures can be used to break off the learning process, i.e. the algorithm stops if $\mid E_{Sim} - E_{Sim'} \mid < \theta$.

Number of failed Optimisation Iterations: Another criterion that can be used to break off the optimisation process is a repeated failure of optimisation steps. If the algorithm runs repeatedly into the else-branch of the if-statement in step 5.d, this might be a hint that no further improvements can be achieved. To realise this kind of stop-criterion the number of subsequently failed optimisation steps has to be recorded. If this number exceeds a previously defined threshold, the algorithm will return the best similarity measure found so far.

Maximal Number of Optimisation Iterations: A quite simple criterion to guarantee the termination of the algorithm is to restrict the number of iterations. This might simply be realised by exchanging the while-loop in step 5 against a for-loop that specifies a concrete number of iterations to be executed.

Although the minimal improvement of the similarity error seems to be a very accurate criterion, it might be difficult to define an appropriate threshold τ. On the one hand, when choosing a very small value this might lead to a poor efficiency of the algorithm. On the other hand, a too great value might cause the break off of the optimisation process although further significant improvements are possible.

Basically, one of the presented stop-criteria might not only be applied exclusively, but it is also possible to combine them. To avoid the problem of undesired runtimes, an appropriate stop-criterion should always restrict the maximal number of optimisation iterations. By combining this static criterion with one of the two more dynamical ones, it is possible to profit of the advantages of both.

5.2.4. Monte Carlo Strategy

As discussed in the previous section, the execution of a gradient descent algorithm is influenced by several parameters. Unfortunately, the determination of optimal values for these parameters prior to the execution of the algorithm is mostly impossible. This holds particularly for the selection of an appropriate initial similarity measure.

To avoid unsatisfactory learning results due to a badly chosen initial similarity measure, one may employ an approach commonly called *Monte Carlo strategy*. The objective of this approach is to ensure stable learning results. Basically, one is not interested in learning algorithms leading to quite good results in one but bad results in another execution due to random influences. Instead, learning should always lead to similar results for the same training data.

Concerning the described gradient descent algorithm this can be achieved by a repeated execution of the algorithm with differently chosen initial similarity measures. Typically, these similarity measures are generated randomly. This leads to the following algorithm:

Input: training data $TD_u(Q)$
Output: optimised similarity measure

procedure Monte_Carlo_gradient_descent_algorithm($TD_u(Q)$) {

1. Sim_{best} := nil;
2. **for** $i := 1$ to n **do**:
   a) initialise similarity measure Sim with random weight vector $\vec{w}$;
   b) Sim_{new} := gradient_descent_algorithm($TD_u(Q)$, Sim)
   c) **if** Sim_{best} = nil **or**
   $\hat{E}_S(TD_u(Q), sim_{new}) < \hat{E}_S(TD_u(Q), sim_{best})$
   then $Sim_{best} := Sim_{new}$
3. **return** optimised similarity measure Sim_{best};

}

For each similarity measure generated, the basic gradient descent algorithm has to be called, which then returns a new optimised similarity measure Sim_new. If Sim_{new} leads to a lower average similarity error $\hat{E}_S$ than the best already generated measure Sim_{best}, Sim_{new} is memorised. After repeating the gradient descent optimisation for n different initial similarity measures, finally Sim_{best} is returned as the final outcome of the optimisation process.

By increasing the parameter n, the probability to find a global minimum or at least a "good" local minimum of the error function can be increased, too. Moreover, the variance of achieved learning results should become smaller. Of course, the drawback of the Monte Carlo approach is the clearly higher computational effort which increases linearly with n.

5.2.5. Advantages and Drawbacks

After having described how the common gradient descent approach can be employed to implement the similarity learner of our learning framework, finally a discussion of this approach regarding its major advantages and drawbacks is given.

Advantages

The major advantage of the gradient descent approach is the fact that it exploits knowledge about the error function in order to guide the search process described formally in Section 5.1. This knowledge, provided in form of the derivation of the error function, allows to implement a very efficient search process. The computation time usually required to perform a gradient descent search is relatively low compared to other search strategies.

Another important property of this approach is the guaranteed convergence of the gradient descent algorithm. Although it cannot be ensured that it converges towards a global minimum, the convergence towards a local minimum is for sure. Presuming the usage of an accurate stop-criterion, this property enables the algorithm to perform a successful optimisation procedure in the majority of its applications. Of course, a successful optimisation requires accurate training data that induces an error function allowing minimisation.

Last but not least, the presented algorithm can be implemented very easily. The major task is the computation of the error function and its derivation. All control structures (e.g. the stop-criterion) required additionally are quite simple and can be implemented with few lines of code.

Drawbacks

Although the exploitation of knowledge about the error function is the major advantage of the algorithm, at the same time it is also connected with the major drawback of the gradient descent approach. Because the approach absolutely relies on the derivation of the error function, only functions that are partially differentiable with respect to some parameter of the similarity measure can be used. In Definition 5.4 we therefore proposed a special error function allowing to apply the approach for learning attribute weights. An extension to local similarity measures seems to be infeasible due to their complex and heterogeneous representation (see Section 3.3) making it much harder to develop an accurate error function.

The introduced error function is also responsible for another, less obvious drawback of the presented approach. Unfortunately, this drawback prevents the application of the approach if the application domain has some particular characteristics. In Section 4.3.3 we have described three basic requirements the error function should fulfill. One of these requirements was that the error value should equal zero if *and only if* the case order of the retrieval result matches the order determined by the training data. Unfortunately, the introduced similarity error violates this requirement. When reviewing the definition of the similarity error (cf. Definition 5.4), it becomes clear that this function results in an error equal zero if a similarity measure assigns all cases a uniform similarity for each query contained in the training data. Consequently, such a similarity measure would inevitably correspond to an

undesired global minimum of the error function. This means, according to the error function, particular similarity measures seem to be optimal measures, even if this is not true when evaluating their outcomes regarding the training data.

Fortunately, such critical similarity measures are often not included in the hypotheses space because our approach is restricted to learning attribute weights. However, one crucial property of the application domain allows the definition of a particular weight-vector leading to a critical similarity measure. If all cases contained in the training data have at least one identical attribute value in common, a weight-vector leading to uniform similarities can be constructed. Here, the particular value of the critical attribute is arbitrary—also the special value "unknown" causes the problem.

Consider the set of cases $c_1, c_2, \ldots, c_n$ contained in the training data and suppose an attribute a_i with $c_1.a_i = c_2.a_i = \cdots = c_n.a_i$. Then, the weight-vector $\vec{w}$ with $w_i = 1$ and $\forall j \neq i \; w_j = 0$ would lead to uniform similarities $Sim(q, c_1) = Sim(q, c_2) = \cdots = Sim(q, c_n)$ for every arbitrary query q, and hence, to a similarity error $E_{Sim} = 0$.

A final drawback of the gradient descent approach is its strong dependency on the chosen initial similarity measure. However, this undesired effect can be weakened by employing the Monte Carlo approach described in Section 5.2.4.

5.3. Genetic Algorithm

In this section, an alternative method to implement the similarity learner is presented. While the gradient descent approach exploits particular knowledge about the error function in order to find a minimum, *genetic algorithms* realise a search strategy more driven by chance. Genetic algorithms are a commonly applied technique for solving optimisation and machine learning tasks in numerous application fields. Basically, the foundation of genetic algorithms is the idea to apply the principles of natural evolution to search strategies in computer science. Therefore, the mechanisms of natural selection, natural genetics, and the principle of the "survival of the fittest" have to be modelled through accurate data structures and algorithms.

The actual search process of a genetic algorithm is again an iterative procedure. In each iteration a new set of artificial creatures—called *individuals*—is created using pieces—called *genes*—of the description of individuals—called *genome*—generated in the previous iteration. Each iteration corresponds to a particular *generation* of individuals. The actual creation of new individuals is based on two major principles, called *crossover* and *mutation*. Crossover means that genes of different individuals of the previous generation are combined to create new individuals of the current generation. During this process, occasionally small modifications of the genes are introduced, leading to individuals with new genes so far not included in the genomes of the previous generation. Crossover, as well as mutation have to be implemented

in form of accurate operators, summarised as *genetic operators*.

Generally, each individual represents one point of the search space, i.e. in our search scenario a particular similarity measure. Of course, one is not interested in creating arbitrary individuals, but individuals corresponding to accurate similarity measures characterised through low error values (see Section 5.1). To increase the probability that new individuals are also "better" individuals, genetic algorithms apply the principle of the "survival of the fittest". This means, only genes of the "fittest" individuals of the previous generation are reused to create new individuals. Here, the "fitness" of individuals corresponds to the particular search criterion and has to be measured by some *fitness function*.

An illustration of the described procedure is shown in Figure 5.5. For the foundations of and more details on genetic algorithms the reader is referred to Michalewicz (1996); Holland (1975).

In the following we present an approach employing the idea of genetic algorithms to implement the similarity learner required by our learning framework (see Chapter 4). While the gradient descent approach was only suitable for learning attribute weights, this approach allows both, learning weights as well as learning local similarity measures (Stahl and Gabel, 2003). Further, it can easily be extended for optimising the third part of the presumed similarity measure representation, namely the aggregation function. However, in the scope of this thesis this will not be discussed in detail.

For the following reasons genetic algorithms are a powerful instrument for optimising similarity measures, and especially for optimising local similarity measures:

- Genetic algorithms have proved to provide flexible, powerful and robust[1] mechanisms for searching optima in complex search spaces.

- The presumed fitness function, i.e., the actual optimisation criteria has not necessarily to fulfill some crucial properties. While the gradient descent approach requires a partially differentiable error function, genetic algorithms allow optimisation of arbitrary target functions. Since local similarity measures depict complex entities that are characterised by several quite different parameters, it is not feasible to construct an adequate error function that can be derived with respect to these parameters.

- Attribute weights as well as local similarity measures can adequately be represented as individuals described by genomes.

[1] The success of *robust* learning algorithms does not rely on special circumstances of the particular learning situation, e.g. the starting point of the search process.

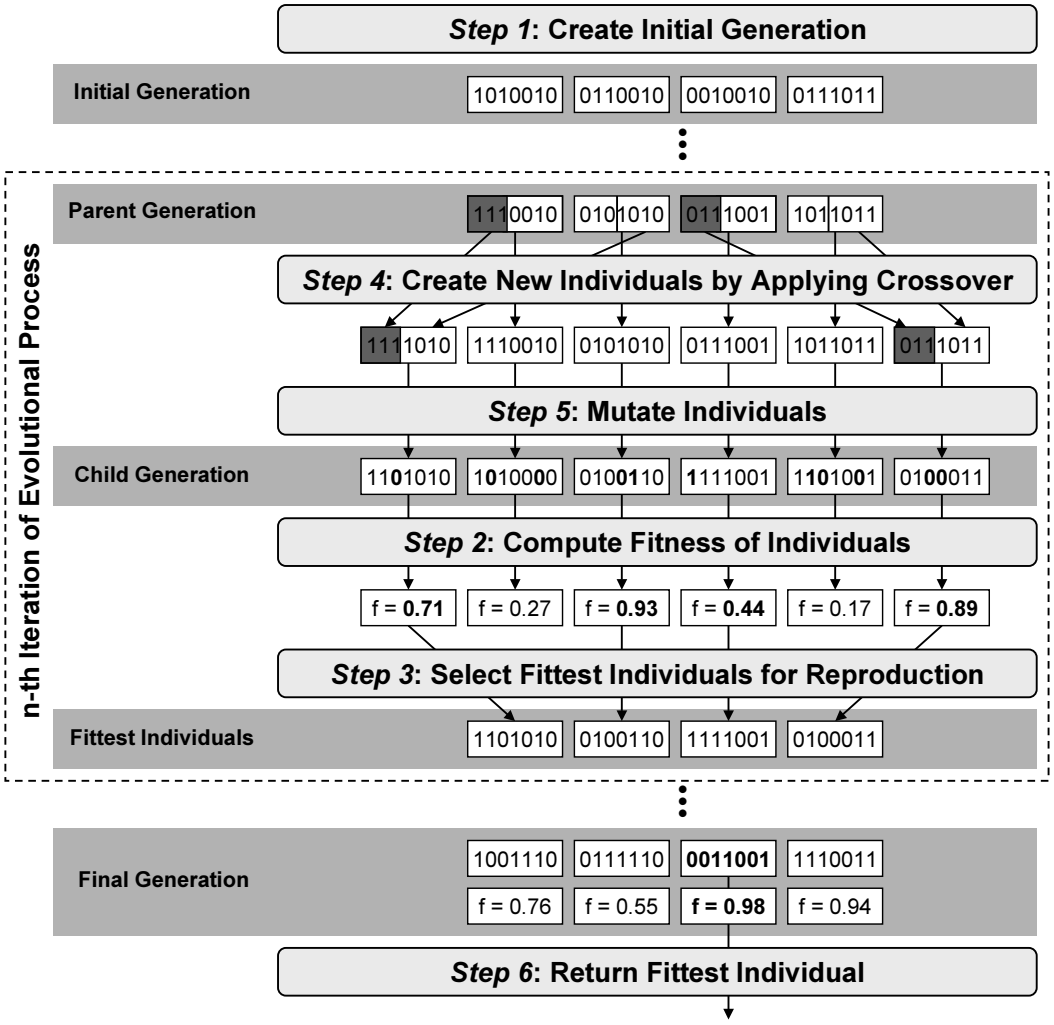


Figure 5.5.: The Idea of Genetic Algorithms

5.3.1. Genetic Algorithms vs. Evolution Programs

In traditional genetic algorithms individuals are commonly represented as bit strings—lists of 0s and 1s—to which crossover or mutation operators are applied (Holland, 1975). However, depending on the particular structure of the underlying search space, this representation sometimes leads to certain problems. One drawback of bit strings is the difficulty to incorporate constraints on the description of individuals. When dealing with the bit string representation, every possible bit string ideally should represent a valid individual, i.e. a point of the addressed search space. However, the required mapping between a possibly very complex search space and the set of all bit strings with a particular length can often not easily be defined. In our scenario constraints are in particular a problem when dealing with local similarity

measures. Here, restricting the overall search space by introducing constraints might avoid to generate "strange" similarity measures not expected to appear in real world applications. For example, we usually suppose to have reflexive similarity measures (see Section 3.2.2). Another problem occurs, if the description of individuals includes real-valued numbers, which cannot accurately be described with bit strings. Because we are dealing with similarity values, i.e. values of the interval [0,1], the description of individuals, of course, also contains numerous real-valued numbers.

An alternative approach avoiding the described problems of traditional genetic algorithms, are so-called *evolution programs*[2] (Michalewicz, 1996). The idea of this approach is the usage of more sophisticated data structures and the application of appropriate genetic operators, while still exploiting the evolutionary principle of the traditional genetic algorithm approach. Because evolution programs allow to optimise the original data structures of the application domain directly, a complex mapping between these data structures and bit strings is needless. The learning algorithm presented in the following employs both, a traditional genetic algorithm and an evolution program.

5.3.2. Choosing the Fitness Function

To guide the optimisation process any optimisation technique, of course, requires a formalisation of the optimisation goal. As well as other optimisation techniques, genetic algorithms allow to describe the optimisation goal in form of a mathematical function. The so-called fitness function enables the genetic algorithm to estimate the fitness or quality, respectively, of generated individuals. This information is then exploited by the crossover operator in order to control the evolutionary process towards the generation of new, hopefully promising individuals.

For our learning task the presumed error function (cf. Section 5.1) already represents an accurate fitness function to be used by a genetic algorithm. The lower the retrieval error the fitter is the respective individual, here a particular similarity measure. Because genetic algorithms do not require a fitness function with certain properties, this approach allows the usage of our originally introduced error function, namely the average index error (cf. Definition 4.6). The usage of this error function avoids undesired effects that may occur when using more artificial error functions. For example, we saw that the average similarity error (cf. Definition 5.4) required by the gradient descent algorithm may lead to a failure of the optimisation process due to the existence of undesired minima of the error function.

[2]The used terminology is ambiguous. Some researches also speak about *evolution algorithms* or *evolution strategies*.

5.3.3. Representation of Individuals

Besides an accurate error function, also the representation of individuals by an accurate genome is a very crucial aspect when employing genetic algorithms or evolution programs, respectively. To be able to learn attribute weights and local similarity measures we have to settle how to represent the respective individuals. While attribute weights can easily be represented by using bit strings, the representation of local similarity measures is more difficult to handle. So, we decided to employ the traditional bit string representation for attribute weights, while employing a more sophisticated representation for representing local similarity measures.

Representing Weights

To apply a standard genetic algorithm for learning global attribute weights as introduced in Definition 3.16, the following bit string representation can be used:

Definition 5.5 (Attribute Weights Individual) Let C be a case characterisation model consisting of n attributes $A_1, \ldots, A_n$. An individual I representing a global weight vector $\vec{w}$ for C is coded as a bit string $BS^I_{\vec{w}} = b_{s-1}b_{s-2}\ldots b_2b_1b_0$ with $b_i \in \{0,1\}$ of fixed size $s = n \cdot s_w$. The entire bit string consists of n sub-strings $bs_{w_1}, \cdots, bs_{w_n}$ where each $bs_{w_i} = b_{i\cdot s_w-1}b_{i\cdot s_w-2}\ldots b_{i\cdot s_w-(s_w-1)}b_{i\cdot s_w-s_w}$ represents the bit string representation for weight w_i, i.e. $w_i = \sum_{j=0}^{s_w-1} 2^j \cdot b_{(i-1)\cdot s_w+j}$.

The parameter s_w determines how exact the real valued attribute weights are represented by the bit string representation, i.e. the greater s_w the more exactly the weights can be expressed.

With this representation it cannot be guaranteed that each individual corresponds to a normalised weight vector, i.e. $\sum_{i=1}^{n} w_i = 1$, like introduced in Definition 3.16. However, each non-normalised weight vector can easily be transformed to an unambiguous normalised one. So, the only disadvantage of the described representation is the existence of different individuals corresponding to the same normalised weight vector. However, this actually unnecessary extension of the search space can be tolerated. The construction of a more accurate representation only allowing normalised weight vectors would require the definition of additional constraints that are difficult to handle with a genetic algorithm.

Representing Difference-Based Similarity Functions

Concerning local similarity measures we presume the representation formalism introduced in Definition 3.15 and 3.14, i.e. difference-based similarity functions and similarity tables. Moreover, we assume difference-based similarity functions based on linear difference because this kind of similarity function is sufficient in most domains (cf. Section 3.3.3). Nevertheless, the algorithm presented here can easily be

extended to consider also other difference functions by introducing an additional parameter for selecting the optimal difference function to be determined by the evolution algorithm.

Consider some difference-based similarity function Sim_A used as local similarity measure for a numeric attribute A. Since Sim_A may be continuous in its value range $[min(A_{range}) - max(A_{range}), max(A_{range}) - min(A_{range})]$ it is generally not possible to describe it with a fixed set of parameters. In certain cases this may be possible (e.g. when using a limited set of base functions), but in general it is not. Thus, we employ an approximation based on a number of sampling points to describe arbitrary functions:

Definition 5.6 (Similarity Function Individual, Similarity Vector) An individual I representing a similarity function Sim_A for the numeric attribute A is coded as a vector V_A^I of fixed size s. The elements of that *similarity vector* are interpreted as *sampling points* of Sim_A, between which the similarity function is linearly interpolated. Accordingly, it holds for all $i \in \{1, \ldots, s\}$: $v_i^I = (V_A^I)_i \in [0, 1]$.

The number of sampling points s may be chosen due to the demands of the application domain: The more elements V_A^I contains, the more accurate the approximation of the corresponding similarity function, but on the other hand, the higher the computational effort required for optimisation. Depending on the characteristics of the application domain and the particular attribute, different strategies for distributing sampling points over the value range of the similarity function might be promising:

Uniform Sampling: The simplest strategy is to distribute sampling points equidistantly over the entire value range (see Figure 5.6a). When applying this strategy, all areas of the underlying similarity function are considered to be equally important.

Center-Focused Sampling: However, when analysing the structure of difference-based similarity functions in more detail, it becomes clear that different inputs of the functions will usually occur with different probabilities. While the maximal and minimal inputs, i.e. the values $d_{min} = (min(A_{range}) - max(A_{range}))$ and $d_{max} = (max(A_{range}) - min(A_{range}))$, can only occur for one combination of query and case values, inputs corresponding to small differences can be generated by various of such combinations. Thus, case data and corresponding training data usually provides much more information about the influences of small differences, compared to the information available about extreme differences. Another aspect is that changes in similarity are usually more important for small value differences, since greater differences usually correspond to very small similarity values. In order to consider these facts during learning,

it might be useful to use more sampling points around the "center" of the difference-based similarity function like illustrated in Figure 5.6b.

Dynamic Sampling: While the center-focused approach is more a heuristics, it is also possible to perform a statistical analysis of the training data in order to determine an optimal distribution for the sampling points. Then, areas where a lot of training information is available might be covered with more sampling points compared to areas for which no respective information is contained in the training data.

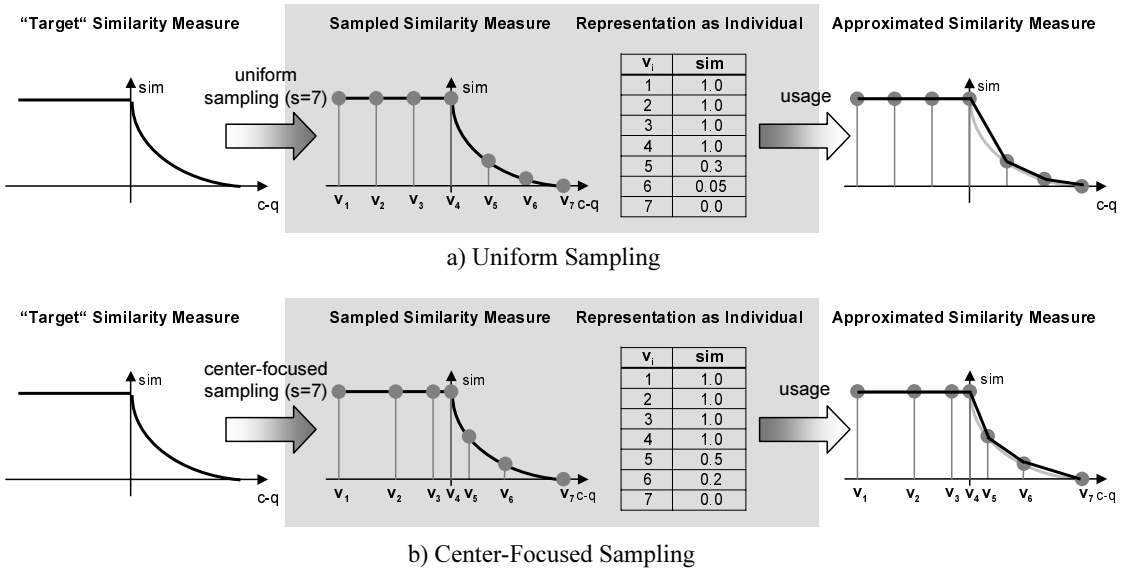


Figure 5.6.: Representing Similarity Functions as Individuals

Representing Similarity Tables

Similarity tables, as the second type of local similarity measures of concern, are represented as matrices of floating point numbers within the interval $[0, 1]$:

Definition 5.7 (Similarity Table Individual, Similarity Matrix) An individual I representing a similarity table for a symbolic attribute A with a list of allowed values $A_{range} = (d_1, d_2, \ldots, d_n)$ is a $n \times n$-matrix M_A^I with entries $m_{ij}^I = (M_A^I)_{ij} \in [0, 1]$ for all $i, j \in \{1, \ldots, n\}$.

This definition corresponds to the representation of similarity tables (cf. Definition 3.14), i.e. the original representation of this type of local similarity measures is directly used by the evolution program. So, the definition presented here is only required for introducing the necessary notation.

5.3.4. Genetic Operators

Another important issue is the definition of accurate genetic operators used to perform crossover and mutation operations. When deciding not to use bit strings, but other data structures for representing individuals, the genetic operators have to consider the particularly used genome representation. Therefore, in this section specialised genetic operators for the previously introduced genome representation are presented.

Genetic operators are responsible for the creation of new individuals and thus have a significant influence on the way a population develops. By using parts of the genome of two or more parent individuals, new ones are composed. Further, random mutations can be applied to recently generated individuals—called *adapting mutation*—or to existing individuals in order to form a new one—called *reproducing mutation*.

The operators we use for learning local similarity measures are quite different from classical ones since they operate on a different genome representation. However, because of underlying similarities, we divide them also into the two standard groups: mutation and crossover operators. But first, classic genetic operators used for creating new individuals for representing attribute weights like introduced in Definition 5.5 are described.

Crossover Operator for Weights

Because we apply the traditional bit string approach for representing attribute weights, also the traditional genetic operators for implementing crossover and mutation can be employed.

To create a new attribute weights individual I_{new} for the child generation, two of the fittest individuals I_1 and I_2 of the parent generation have to be selected. Then, a *split point* $sp \in \{0, 1, \ldots, n \cdot s\}$ has to be determined randomly. Finally, the genome of the new individual I_{new} is assembled by using the bits $b_{s-1}b_{s-2}\ldots b_{sp}$ from I_1 and the bits $b_{sp-1}\ldots b_1b_0$ from I_2. The remaining bits $b_{sp-1}\ldots b_1b_0$ of I_1 and $b_{s-1}b_{s-2}\ldots b_{sp}$ of I_2 might be used directly to create a second new individual or they might be combined with genome parts of other parent individuals to obtain further new individuals. In the second case, the parent individuals selected additionally have to be split at the same split point sp, of course.

Mutation Operator for Weights

For the mutation of attribute weights individuals holds the same as for crossover. Due to the bit string representation the mutation operator is also realised as in common genetic algorithms. According to some probability ρ, every bit of the representation is changed from 0 to 1 or 1 to 0, respectively. So, the number of single mutations, i.e. changed bits, is not determined a priori but depends in a more

flexible way on ρ. The larger ρ the more mutations will occur in average. Therefore, ρ might also be determined with respect to the length of the particular bit string used. For relative small bit strings it might be more accurate to chose a greater ρ to increase the probability of mutations. When dealing with larger bit strings, ρ might be chosen smaller to avoid extensive changes in the genome due to a large number of mutations.

Crossover Operators for Similarity Vectors and Matrices

By applying crossover operators on the more complex data structures used for representing local similarity measures, a new individual in form of a similarity vector or matrix is created using elements of its parents. Though there are variations of crossover operators described that exploit arbitrary number of parents (Michalewicz, 1996), we rely on the traditional approach using exactly two parental individuals, I_1 and I_2.

- *Simple crossover* is defined in the traditional way as used for bit string representations: First a split point for the particular similarity vector or matrix is chosen. The new individual is then assembled by using the first part of parent I_1's similarity vector or matrix and the second part of parent I_2's vector/matrix.

- *Arbitrary crossover* represents a kind of multi-split-point crossover with a random number of split points. Here, for each component of the offspring individual it is decided randomly whether to use the corresponding vector or matrix element from parent I_1 or I_2.

- *Arithmetical crossover* is defined as the linear combination of both parent similarity vectors or matrices. In the case of similarity matrices the offspring is generated according to: $(M_A^{I_{new}})_{ij} = m_{ij}^{I_{new}}$ with $m_{ij}^{I_{new}} = \frac{1}{2}m_{ij}^{I_1} + \frac{1}{2}m_{ij}^{I_2}$ for all $i, j \in \{1, \ldots, d\}$.

- *Line/column crossover* is employed for similarity tables, i.e. for symbolic attributes, only. Lines and columns in a similarity matrix contain coherent information, because their similarity entries refer to the same query or case value, respectively. Therefore, cutting a line/column by applying simple or arbitrary crossover may lead to less valuable lines/columns for the offspring individual. We define line crossover as follows: For each line $i \in \{1, \ldots, n\}$ we randomly determine individual I_1 or I_2 to be the parent individual I_P for that line. Then it holds $m_{ij}^{I_{new}} = m_{ij}^{I_P}$ for all $j \in \{1, \ldots, n\}$. Column crossover is defined accordingly.

For each of the described operators a particular probability value has to be specified. When performing crossover, one of the described operators is then selected according to this probability.

Mutation Operators for Similarity Vectors and Matrices

Operators of this class are the same for both kinds of local similarity measures we are dealing with. They change one or more values of a similarity vector V_A^I or matrix M_A^I according to the respective mutation rule. Doing so, the constraint that every new value has to lie within the interval $[0, 1]$ is met. The second constraint that needs to be considered concerns the reflexivity of local similarity measures (cf. Section 3.2.2). As a consequence, the medial sampling point of a similarity vector should be 1.0 as well as the elements m_{ii}^I of a similarity matrix for all $i \in \{1, \ldots, n\}$. Since any matrix can be understood as a vector, we describe the functionality of our mutation operators for similarity vectors only:

- *Simple mutation*: If $V_A^I = (v_1^I, \ldots, v_s^I)$ is a similarity vector individual, then each element v_i^I has the same probability of undergoing a mutation. The result of a single application of this operator is a changed similarity vector $(v_1^I, \ldots, \hat{v}_j^I, \ldots, v_s)$, with $1 \leq j \leq s$ and $\hat{v}_j^I$ chosen randomly from $[0, 1]$.

- *Multivariate non-uniform mutation* applies the simple mutation to several elements of V_a^I. Moreover, the alterations introduced to an element of that vector become smaller as the age of the population is increasing. The new value for v_j^I is computed according to $\hat{v}_j^I = v_j^I \pm (1 - r^{(1-\frac{t}{T})^2})$, where t is the current age of the population at hand, T its maximal age, and r a random number from $[0, 1]$. Hence, this property makes the operator searching the space more uniformly at early stages of the evolutional process (when t is small) and rather locally at later times (when t becomes greater). The sign $\pm$ indicates, that the alteration is either additive or subtractive. The decision about that is made randomly as well.

- *In-/decreasing mutation* represents a specialisation of the previous operator. Sometimes it is helpful to modify a number of neighbouring sampling points uniformly. The operator for in-/decreasing mutation randomly picks two sampling points v_j^I and v_k^I and increases or decreases the values for all v_i^I with $j \leq i \leq k$ by a fixed increment.

As for crossover operators, mutation operators are applied according to some probability to be specified a priori.

5.3.5. Controlling the Genetic Algorithm

In contrast to the gradient descent algorithm described in Section 5.2.2, the control structure for the genetic algorithm is more sophisticated. While the gradient descent approach considers attribute weights only, the genetic algorithm is able to optimise all major elements of the entire similarity measure, i.e. besides the attribute weights also all specific local similarity measures required for each attribute. In this work we have neglected optimisation of the aggregation function. Instead, always a weighted sum is employed because this function is accurate in most domains. However, this element of the similarity measure might be optimised easily with a genetic algorithm, too.

Weights and local similarity measures are quite different parts of the similarity measure. Because the effects of these two parts on the entire similarity computation interact with each other, one must be aware how to integrate the respective optimisation processes. Basically, the following two approaches for an integration can be distinguished.

Sequential Processing

One possibility is to separate the actual optimisation of attribute weights and local similarity measures. One might first optimise the weights while holding the local similarity measures fixed. The optimisation procedure for the local similarity measures would be started after having obtained an optimised weight vector. Of course, the order of processing could also be inverted, i.e. one might optimise attribute weights after the optimisation of local similarity measures.

The sequential approach can also be applied for processing all involved local similarity measures. Because local similarity measures are defined independently from each other, the optimisation of them can be separated and this separation should not significantly influence the outcome of the optimisation process. This means, one could optimise each local similarity measure until some stop criterion (e.g. a reached maximal number of generations) stops the current optimisation loop and start the optimisation of the next local similarity measure.

So, the entire optimisation process can be decomposed into several optimisation loops that are responsible for optimising attribute weights and single local similarity measures, respectively. The control structure for implementing such a single optimisation loop is similar to a standard genetic algorithm (Holland, 1975) and proceeds at a high level as follows:

Input: training data $TD_u(Q)$, initial similarity measures Sim,
optimisation entity o_e
Output: optimised o_e

procedure base_genetic_algorithm($TD_u(Q)$, Sim, o_e) {

1. generate initial population $P_0 = \{I_1, I_2, \ldots, I_{p_{size}}\}$ for o_e;
2. **for** $i = 0$ to *number_of_generations* **do:**
   a) create a set P_{child} of $p_{size} \cdot$ *reproduction_rate* new individuals by applying crossover operators and reproducing mutation to mating partners randomly selected from P_i;
   b) $P_{i+1} := P_i + P_{child}$;
   c) mutate the individuals in P_{i+1} (adaptive mutation);
   d) evaluate fitness for each $I_r \in P_{i+1}$, i.e. compute $E_I(TD_u(Q), Sim_{I_r})$ where Sim_{I_r} is the similarity measure assembled from Sim and I_r;
   e) remove the $p_{size} \cdot$ *reproduction_rate* most unfit individuals from P_{i+1};
3. return optimised o_e according to the fittest individual $I_{fittest}$ of population $P_{number\_of\_generations}$;

}

As input the algorithm is provided with training data, an initial similarity measure, and an identifier—here, denoted as *optimisation entity*—that determines which part of the entire similarity measure has to be optimised. For example, this identifier might be an attribute name for identifying the corresponding local similarity measure. According to this identifier the algorithm then selects the appropriate representation for describing individuals and applies accurate genetic operators.

The major drawback of the sequential approach is the necessity of some initial similarity measure. When optimising only one part of the entire measure, one already needs the remaining parts when evaluating the fitness of individuals, i.e. when computing the retrieval error. So, the remaining parts have to be initialised in some way, for example, randomly or by a domain expert. However, the chosen attribute weights or local similarity measures, respectively, have a crucial impact on the optimisation process.

First, consider the situation that we start with the optimisation of a particular local similarity measure sim_i. Further, we assume that the weight vector chosen

initially contains a weight w_i with a high value, although the actual (but unknown) utility function requires a significantly lower value for w_i. In this case, the optimisation might yield an inaccurate corresponding measure sim_i assigning nearly identical similarity values for all attribute value combinations. Such a measure might compensate the retrieval error caused by the wrong weight w_i and so the learning algorithm is not able to find an accurate version of sim_i as required by the utility function.

Also the opposite processing order might lead to crucial problems. Assume that one of the local similarity measures sim_i chosen initially computes very wrong similarities for many or all possible combinations of input values. This might lead to the determination of a small value for the corresponding weight w_i when optimising attribute weights. The reason is the fact that a small w_i might compensate the retrieval error caused by sim_i. However, the utility function might require a much greater value for w_i.

Concerning the processing order of the individual local similarity measures such problems should not occur, because local similarity measures do not depend on each other due to the local-global principle (cf. Section 3.3.2).

Parallel Processing

To avoid the problems of sequential processing described previously, one could also optimise weights and all local measures simultaneously. This means, the applied genetic algorithm has to manage *composed individuals* consisting of one *atomic individual* for the weight vector (according to Definition 5.5) and one atomic individual for each local similarity measure (according to Definitions 5.6 and 5.7). The crossover and mutation operators required for composed individuals then have to apply accurate atomic operators (as described in Section 5.3.4) for the involved atomic individuals.

The control structure of the parallel processing approach is nearly identical to the base algorithm presented previously. The only differences are:

- an initial similarity measure is not required because the complete similarity measure is optimised,
- so, also the identifier for the optimisation entity can be omitted,
- crossover and mutation is not applied directly to the entire composed individuals, but only to the single atomic individuals contained in the composed individuals

Unfortunately, the described composed individuals lead to a very huge search space to be considered by the genetic algorithm. Because parallel processing of

weights and local measures does not allow to separate the optimisation of the individual elements of the entire similarity measure, the complete search space has to be processed at once.

Pseudo Parallel Processing

Another approach that avoids very large individuals like required by the previous strategy, is an alternated processing of two independent optimisation loops. The first optimisation loop is responsible for learning attribute weights, and the second loop optimises local similarity measures only. Each loop has to be realised by a corresponding genetic algorithm which can be processed "pseudo parallel". This means, algorithm one processes a fixed number of generations for learning weights, stops, and then the second algorithm processes also a fixed number of generations for learning local similarity measures. This procedure has to be repeated until the stop criterion stops both algorithms.

The main idea of the pseudo parallel processing is that the two algorithms profit from the improvements of each other when computing the fitness of individuals. The fitness - e.g. the index error - relies on retrieval results which can only be determined by using weights and local similarity measures. Thus, both algorithms have to operate on one similarity measure, but each algorithm optimises another part of this measure.

The main difference to the real parallel processing by using composed individuals is, that it is possible to assign the learning procedures for both parts of the similarity measure individual processing times. Since learning of local similarity measures is connected with larger search spaces, it might useful to allow this algorithm to process more generations in each iteration. For example, if the weight learning algorithm is allowed to process 100 generations in total while the local similarity measure learning algorithm is allowed to process 200 generations, it might be useful to retain this ration in every iteration, too. Thus, the weight learner might process 5 generations next the local measure learner 10 generations, and so on.

5.3.6. Control Parameters

Similar to the gradient descent algorithm (see Section 5.2.3), also the behavior of the genetic algorithm presented can be influenced by some control parameters. However, because genetic algorithms are quite robust, these parameters do not have such a strong impact on the learning results compared to the parameters of the gradient descent algorithm. The following parameters have to be specified when applying the algorithm previously described:

Population Size: This integer number determines the number of individuals contained in one generation of the evolutional process.

Reproduction Rate: The reproduction rate is a real valued parameter from the interval $[0, 1]$ and determines the number of generated child individuals against the population size. For example, a reproduction rate of 0.5 will lead to 10 child individuals when using a population size of 20.

Number of generations: This important parameter specifies a fixed stop criterion to ensure the termination of the algorithm. This means, the evolution process is stopped after having proceeded for the specified number of generations. The more generations have to be generated the longer is the runtime of the algorithm, of course. However, more generations usually lead to fitter individuals and so to more accurate learning results.

5.3.7. Advantages and Drawbacks

As for the learning algorithm based on the gradient descent approach, also the genetic algorithm has certain advantages but also a major drawback.

Advantages

The most obvious advantage of the introduced genetic algorithm is the possibility to learn also local similarity measures and not only attribute weights. Local similarity measures are a very important aspect of the similarity representation assumed in this work. So, an implementation of our learning framework has to provide also learning capability for this part of similarity measures to be relevant in practice.

Another advantage of genetic algorithms is that it does not require crucial properties on the fitness function to be used to guide the optimisation process. In contrast to the gradient descent approach genetic algorithms do not require the definition of an unnecessary sophisticated error function in order to ensure its differentiability. This allows us to use the error function originally introduced to compare retrieval results with corresponding training data, namely the index error (cf. Definition 4.6). This also avoids problems due to undesired minima of the error function caused by specific properties of the considered case data (cf. Section 5.2.5).

The quality of the achieved learning results mainly depends on the quality of the provided training data. The outcome of a single optimisation process does not rely on crucial parameters compared to the gradient descent approach where the initial similarity measure and the employed learning rate have a major impact. This means, the presented genetic algorithm represents a very robust learning method mostly converging to a global minimum of the error function and so to an accurate similarity measure regarding the representation formalism employed and vocabulary chosen.

Drawbacks

Unfortunately, it is not sufficient to know that a learning algorithm, in principle, converges to an optimum of the considered search space. Another very important aspect is also the *convergence speed*. The required computation time is the only crucial drawback of the presented genetic algorithm for learning similarity measures. Because the algorithm does not exploit particular knowledge about the shape of the error function to guide the search process, experiments have shown that the convergence speed is dramatically lower compared to the gradient descent algorithm (see Section 8.4.2). Nevertheless, in our experiments carried out in the scope of this work, the genetic algorithm was able to find accurate similarity measures in tolerable computation times. The required time for learning, of course, strongly depends also on the case model, the size of the case base, the amount of available training data and, last but not least, on the employed computer hardware.

Because the described application scenarios (see Chapter 6) do not necessarily require fast generation of results, the genetic algorithm seems to be the most promising approach for realising the optimisation loop. For example, if it is employed to generate an initial similarity measure during the development phase of a CBR-System, one might even tolerate computation times of several days. If it is used for maintenance operations, it could also be run over night or parallel to the daily use of the system. Moreover, one might apply techniques to distribute the processing of the algorithm on several machines in order to speed up the entire learning process (Moser, 1999).

6. Application Scenarios

After having introduced our approach to learning similarity measures in Chapter 4 and Chapter 5, this chapter now deals with its application in practice. So far it was assumed that the mandatory training data is provided by some arbitrary similarity teacher. Although in Section 4.2.2 it was already mentioned that this similarity teacher might be realised in various ways, particular examples how this might look like in practice have not been discussed so far. In this chapter several application scenarios, where the introduced learning approach might facilitate the acquisition of similarity knowledge, are presented. Before, we briefly recall some general aspects concerning the application of our learning framework.

6.1. Applying the Framework

When taking into consideration to apply the presented framework, one first should become aware of the capabilities and the requirements of the framework. In general, machine learning approaches should only be applied if the knowledge to be learned cannot be acquired more efficiently through an alternative way. If the required knowledge can be provided by an experienced domain expert and if it can be formalised accurately with reasonable effort, the application of machine learning approaches is mostly not a good choice. The general problem of any learning approach is the extraction of knowledge from a finite training data set. So, every particular aspect not contained in the training data, for example, an exception only occurring rarely in the underlying domain, will presumably not be considered by the knowledge learned. Although a human expert might also forget to mention some rare exceptions, under the mentioned conditions the quality of manually acquired knowledge is mostly higher than the quality of learned knowledge. Moreover, machine learning approaches are usually only able to learn meaningful knowledge, if a rough idea about the content of the expected knowledge is already known. Formally spoken this means, learning is only feasible, if the hypothesis space can be restricted appropriately.

However, in real-world applications manual knowledge acquisition is often complicated due to certain problems. Concerning the acquisition of similarity knowledge, the following principle reasons might complicate or even prevent a manual acquisition:

- *Accessibility of Explicit Utility Knowledge:* Sometimes, no expert who is able

to provide knowledge about the actual utility of cases for particular problem situations in an explicit form is available. This problem might be caused by a poorly understood application domain or by economic considerations due to the high personal costs when employing experienced experts. In this situation only knowledge-poor similarity measures (see Section 3.4.1) can be defined. However, this might lead to an insufficient approximation of the underlying utility function and so to inaccurate retrieval results.

- *Difficulties of Knowledge Compilation:* If knowledge about the cases' utility is only available in an informal form (e.g. natural language), the transformation of this knowledge into the representation formalisms used by the CBR system might be very difficult. This holds in particular if no experienced CBR expert is accessible to do this job.

- *Changes of Utility Requirements:* Some domains are characterised by rapid changes that might require regular maintenance operations to ensure the validity of the used knowledge. If these changes also concern the underlying utility function to be approximated by the similarity measure (e.g. due to changed user preferences), repeated manual updates might be infeasible or to expensive.

Each of the described aspects might be a reason to apply our learning framework. Nevertheless, also if a certain CBR application might potentially benefit from our learning approach, the employment of the approach makes only sense, if the following questions can be confirmed:

- Is the presumed similarity measure representation sufficient to obtain a reasonable approximation of the utility function or is a more sophisticated representation required due to the complexity of the domain?

- Is it clear how the mandatory training data can be obtained (i.e., how to realise the similarity teacher)?

- Is the intended similarity teacher presumably able to provide *enough* training data required to achieve reasonable learning results?

- Is the intended similarity teacher presumably able to provide training data with a tolerable amount of noise, i.e. is the number of included failures and contradictions low compared to the entire amount of available training data?

- Is the effort for acquiring the training data lower than the effort required when modelling the similarity measure manually?

In the following some typical application scenarios where the enumerated conditions are often fulfilled are discussed. As described in Chapter 4, the aim of our learning approach is to construct a similarity measure that approximates a certain utility function as good as possible. The application scenarios presented in this chapter differ, in principle, how this utility function is determined. Basically, the utility function might be determined by

- the application domain,
- the users (perhaps also by the providers) of the CBR system,
- or some functionality of the CBR system used.

Depending on these criteria, the acquisition of training data, and hence the realisation of the similarity teacher might be very different. While the first three scenarios presented in the following are characterised by human similarity teachers, in the last two scenarios we present approaches for acquiring training data automatically.

6.2. Supporting the Domain Expert

Consider a traditional CBR application, for example, a classification (cf. Section 2.3.1) or help desk (cf. Section 2.3.2) scenario. Here, the utility function is exclusively determined by the underlying application domain. Cases are useful if and only if they contribute to solving a given problem described by a query. Whether a case is useful or not, here typically depends on the case's lesson part representing a solution, for example, a class identifier or a diagnosis with a corresponding therapy.

When developing such a traditional CBR system, an accurate similarity measure usually can only be defined by a domain expert[1] who possesses some knowledge about the underlying utility function of the domain. Here, two different types of knowledge can be distinguished:

Explicit Knowledge: In the ideal case, the expert possesses a deeper understanding of the application domain and is able to explain relations between the problem space and the solution space. This means, the expert is able to identify important problem characteristics and can explain their influence on the corresponding solutions. For example, in a medical diagnosis scenario a doctor might state: "If the temperature exceeds 40°C, then it is more likely an influenca than a usual cold." So, explicit knowledge can also be interpreted as general domain knowledge.

[1] An alternative approach that allows to extract similarity knowledge from case knowledge is presented in Section 6.6.

Implicit Knowledge: When dealing with poorly understood domains, unfortunately even domain experts are often not able to explain the relation between problems and solutions detailed enough to construct an accurate similarity measure. Nevertheless, due to their experiences experts are often able to estimate the quality of presented solutions alternatives without being able to justify their decisions in detail. In a medical diagnosis scenario, for example, a doctor might state: "I suppose this is not an influenca, but probably only a hard cold." So, implicit knowledge is often expressed in form of intuition.

While explicit knowledge can be employed directly to construct similarity measures manually in the bottom-up manner (see Section 3.5), implicit knowledge cannot directly be encoded into the similarity measure representation because it only represents high-level knowledge about the utility of cases not considering the influence of individual attributes.

Implicit knowledge typically also occurs in "visual domains" where the similarity of cases is based on visual criteria. When dealing with cases that can be represented visually, the strength of human beings to recognise and judge visual objects and patterns might facilitate the definition of similarity measures significantly. For example, one aim of the FIORES-II[2] project was the case-based classification of design objects (e.g. cars, bottles, vases) with respect to their aesthetic character (e.g. sporty, elegant). Here, the similarity measure has to deal with the mathematical descriptions (e.g. CAD models) of the design objects. Due to the complex formalisation, designers usually have problems to describe the mathematical parameters determining a particular aesthetic character explicitly. However, they have no problems to compare and judge two objects with respect to their aesthetic character when they are presented visually, for example, in form of a picture.

Such high-level knowledge represents the kind of utility feedback required to obtain training data to be used for learning the similarity measure. So, our learning framework can support a domain expert during the definition of domain specific similarity measures. Here, the similarity teacher is realised by domain experts due to their implicit domain knowledge. In particular, in domains where experts possess a reasonable amount of implicit knowledge, but only few explicit knowledge, the learning framework might facilitate the definition of accurate similarity measures significantly.

Another characteristic of this application scenario is the execution of the learning procedure only at certain points of time, typically once during the development phase of the CBR system. Nevertheless, the learning procedure might be repeated during the lifetime of the system if additional knowledge becomes available.

Because often both, some explicit knowledge and additional implicit knowledge can be provided by domain experts, the consideration of available background knowl-

[2]Character Preservation and Modelling in Aesthetic and Engineering Design. Founded by the European Commission under the GROWTH programme (contract number GRD1-1999-10385).

edge during the learning procedure might be particularly promising in this application scenario.

6.3. Distributed Utility Knowledge

In many application domains the users of knowledge-based systems are often more or less experienced experts in the domain themselves. Thus, they are principally able to recognise faulty or suboptimal solutions proposed by the systems in certain situations. When developing expert systems that give expert users the opportunity to communicate these deficiencies back to the system, this feedback can obviously be used to avoid similar errors in future use of the system. If a user disagrees with the presented case ranking due to his well-founded domain knowledge, one should give her/him the opportunity to comment the unsatisfactory retrieval result. This again may lead to utility feedback like required by our learning approach and can be used to improve the faulty similarity measure.

How to apply our framework in the described situation is illustrated in Figure 6.1. On the one hand, we have a typical CBR system consisting of a similarity-based retrieval engine and perhaps also some adaptation functionality. On the other hand, there are the users of the system who possess, in principle, the knowledge to recognise suboptimal retrieval results. By giving feedback all users together play the role of one experienced similarity teacher. The provided feedback is then used by the system's similarity learner to optimise the similarity measure that is responsible for computing case rankings.

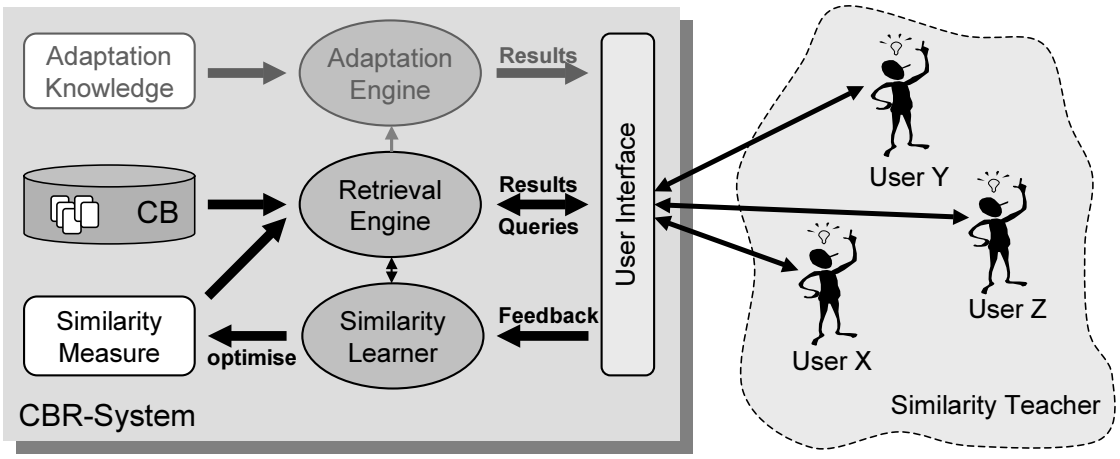


Figure 6.1.: Distributed Utility Knowledge

Consider a CBR application that provides the user with a set of alternative solutions or cases, respectively, ranked by their utility estimated by the system's internal

similarity measure. Whether the system performs case adaptation or not is not crucial, however, in the following we suppose that the system does not adapt cases after retrieval. A typical application scenario where this situation occurs are knowledge management systems (cf. Section 2.3.3). The idea of these systems is to collect and store knowledge required to perform the business processes of organisations or companies. This knowledge is typically distributed over the different departments and employees of organisations and can be collected in form of case knowledge. To profit from the collected knowledge its availability at the right time must be ensured. Therefore, case-based knowledge management systems allow to retrieve knowledge judged to be useful regarding a given query.

However, during the development of a knowledge management system it is usually impossible to decide which cases might be useful in which situation. So, a similarity measure defined initially might represent a poor approximation of the utility function. In contrast to the scenario discussed in the previous section, here, the utility function is not strictly determined by the domain. It rather depends on specific information needs of the knowledge management system's users. For example, the employees of a software development company probably will have other information needs than the employees of a mechanical engineering company. This means, not only the case knowledge to be managed is distributed, but also the general knowledge required to retrieve useful cases is distributed within the organisation or company. To acquire this distributed knowledge all the system's users together have to play the role of the similarity teacher.

Distributed knowledge about the utility of cases might also occur in classic application scenarios. Consider a CBR system that provides decision support for the diagnosis of technical systems, for example, aircraft engines[3]. Suppose that several experienced technical specialists are using the system during their daily maintenance operations. Due to their individual experiences, the specialists might occasionally recognise suboptimal retrieval results of the CBR system. By exploiting the mentioned feedback possibility, the system will be enabled to acquire general domain knowledge from all its experienced users. This means, with gradually usage the system will encode particular domain knowledge of several domain experts into its similarity measure. So, the system might obtain respective high expertise leading to powerful problem solving capabilities. This approach would also allow collaborative maintenance of the similarity knowledge. So far, approaches to collaborative maintenance have only been proposed concerning case knowledge, for example, by Ferrario and Smyth (2000).

Nevertheless, it must be considered that feedback from different users may contain an increased amount of contradictions leading to noisier training data. This problem might partially be compensated by the opportunity to acquire more training data depending on the number of the system's users. The actual impact of noisy training

[3]In this application domain CBR has already been applied successfully.

data on our learning algorithms is discussed in Section 8.4.

6.4. Personalised Utility Requirements

Another situation occurs if different users of a CBR system make different demands on the outcome of the case retrieval. This means, for identical queries, cases may have different utilities depending on the context of the particular user. That knowledge about the preferences of individual users is essential for intelligent systems, for example, has already been pointed out by Branting (1999); Göker and Thompson (2000). In order to enable a CBR system to consider such personalised utility requirements, the architecture presented in the previous section has to be enhanced like shown in Figure 6.2. The major difference is the availability of several similarity measures used to perform retrieval regarding queries of different users.

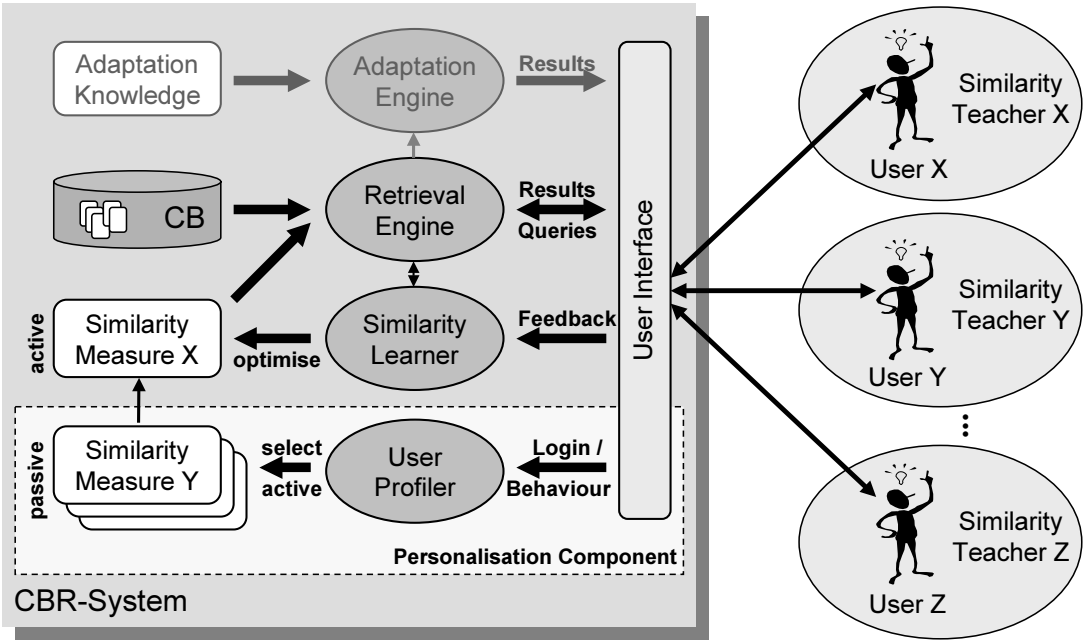


Figure 6.2.: Personalised Utility Knowledge

6.4.1. Product Recommendation Systems

In the last years an increased desire for personalised systems, particular in the field of eCommerce, could be noticed. A very successful technique to realise personalised product recommendation systems is collaborative filtering (see Section 2.3.4). Although CBR is already used to build commercial product recommendation systems,

here, the issue of personalisation is still neglected. However, CBR appears to be a powerful approach to realise personalised recommendation functionality, too.

Consider a product recommendation system in eCommerce. Here, the users are customers having individual preferences with respect to the offered products or services (Branting, 1999; Göker and Thompson, 2000; Smyth and Cotter, 1999). For example, some customers might focus more on the products' prices while others are mainly interested in the technical properties. Such preferences can be represented in form of specific attributes weights and specific local similarity measures, leading to *personalised similarity measures*. While attribute weights can be used to express individual preferences regarding the general importance of product features (e.g. "price is more important than colour"), local similarity measures allow the modelling of preferences w.r.t. to particular attribute values (e.g. "I prefer the colour blue").

Figure 6.3 illustrates examples of personalised local similarity measures that might be used in a product recommendation system. The figure shows two similarity functions for a numeric attribute price and two similarity tables for a symbolic attribute colour used to represent similarity measures that consider the preferences of two different customers. Concerning the price of the product, customer1 might be a business customer that is provided with a fixed budget and therefore every product with a higher price than demanded will lead to a similarity of zero. Products with significantly lower prices will also lead to a high decrease of similarity because the quality of the product is very important, too, and the respective money is available. Suppose customer2 has a more flexible budget. Here, exceeding the price demand will still decrease the similarity, but not as strictly as in the previous case, i.e. the customer will probably also buy a more expensive product if it provides very good functionality. Further, cheaper products will be preferred as long as the required functionality is fulfilled, i.e. the local similarity with respect to the price of cheaper products is always maximal. Further, the two similarity tables might express individual preferences of the two customers regarding the colour of the offered product.

6.4.2. Assessing Customer Preferences

Basically, one must distinguish between two kinds of preferences to be expressed differently:

Explicit Preferences: Explicit preferences are commonly expressed in form of concrete demands and wishes concerning the desired product. They are the foundation of the query to be submitted to the recommendation system. For example, a customer might state: "I want a red car".

Implicit Preferences: Implicit preferences are of a more subtle nature and concern mostly the properties of alternative products, if the optimal product is not

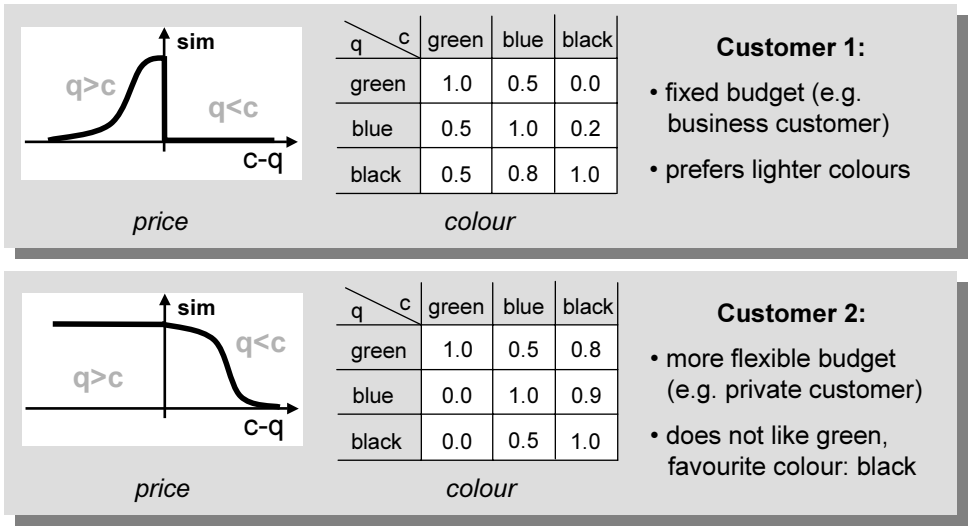


Figure 6.3.: Examples of Personalised Local Similarity Measures

available. They often correspond to individual but general valid preferences (e.g. favourite colours) or they are influenced by individual experiences of the customers. For example, a customer might describe implicit preferences as follows: "If no red car is available, I would prefer a blue one" or "I definitely would not buy a Ford because I had many problems with a car of this company in the past".

Of course, often it is not possible to draw a clear line between explicit and implicit preferences. By asking the customer, implicit preferences can be become obvious and so can be expressed as explicit preferences to be included in the query. Nevertheless, in practice it is not feasible to ask the customer for all her/his preferences when acquiring her/his demands on the desired product. The strength of experienced human shop assistants is their capability to guess many implicit preferences of a particular customer due to subtle hints (clothes or language of the customer, etc.) and their knowledge of human nature. Of course, this capability cannot directly be modelled when implementing an artificial shop assistant to be employed in an electronic shop.

6.4.3. Learning Personalised Similarity Measures

Employing personalised similarity measures, of course, increases the knowledge engineering effort when developing a recommendation system based on CBR. Instead of defining one domain specific similarity measure, one has to define several measures considering the specific preferences of individual users or user classes, respectively. When dealing with user classes, a particular user has to be classified before the

actual recommendation process starts. This can be done through self-classification, user-profiling or similar techniques. For the realisation of personalised similarity measures it makes no crucial difference whether the personalisation level considers individual customers or only customer classes.

Depending on the number of users or user classes, respectively, the system has to deal with several utility functions and corresponding similarity measures in order to adapt the retrieval functionality to the individual needs of the current user. But even if one is willed to put up with this additional effort, it is still an open question how to acquire the required knowledge. A customer is mostly not willed to describe all his implicit preferences before obtaining a product recommendation.

We argue that the presented learning approach is able to facilitate both issues. First, it may reduce the effort to define several similarity measures. Second, it is probably even the only feasible way to obtain the required knowledge. In order to apply the approach to the described scenario, one has to define one or several initial similarity measures that approximate the users' specific utility functions as good as possible. During the use of the system one has to acquire utility feedback from the users to learn more specific measures for each user or user class. Generally, one can distinguish between two basic possibilities to obtain utility feedback from customers (Hayes et al., 2001):

Non-Invasive Approach: Here, the feedback is only collected implicitly, e.g. by observing the buying patterns of customers. If a customer finally does not buy the product assumed to be the most useful one according to the defined similarity measure but another product presented alternatively, this might be a hint that the similarity measure used currently is suboptimal.

Invasive Approach: To obtain more training data with higher quality one can alternatively ask the users for explicit feedback about the presented case or product rankings, respectively. However, this approach is coupled with the risk to annoy the user because s/he is mostly not willed to provide information and to waste time when not directly profiting from this interaction with the system.

While the non-invasive approach is more suitable for B2C[4] scenarios, it is likely that the invasive approach might be accepted by users in B2B[5] scenarios. In B2B scenarios the relationship between the shop provider and the customer is often closer and one strives for a longer business cooperation. So, the customer might realise that his active feedback and cooperation enables the system to improve the recommendation quality in future sales situations. This important trade-off between customer feedback and *cognitive load* on the customer is also discussed by Branting (1999).

[4]Business-to-Customer
[5]Business-to-Business

When applying the non-invasive approach, this is a typical example where the acquired utility feedback is mainly based on the number of queries (cf. Section 4.2.3). If feedback is only based on the buying decision of the customer, we only obtain the information about the most useful product and some information about less useful products. However, information about utility differences among the less useful products will not be available. Further, one or several of the products not bought, i.e. the products assumed to be less useful, might also be equally useful compared to the actually bought product. Due to this few information, a lot of sales processes based on certain customer demands (i.e. queries) have to be recorded to obtain enough training data required to achieve accurate learning results. However, this should not be a crucial problem as long as the recommendation system is used frequently by many users.

6.5. Considering Adaptability of Cases

Although most commercial CBR systems still neglect the adaptation phase of the CBR cycle, many researchers have already remarked that performing adaptation makes a special demand on the similarity measure to be used in the precedent retrieval step (Leake et al., 1996a; Smyth and Keane, 1993, 1995a). Instead of guaranteeing the retrieval of cases that contain a solution that is maximally useful for solving the current problem directly, the similarity measure rather has to consider the *adaptability* of cases. This means, under certain circumstances it might be better to select a particular case with a solution less useful than solutions contained in other cases. This situation occurs if this particular solution can be adapted more easily to solve the given problem compared to alternative solutions contained in other cases.

This relation between adaptability and utility of cases is illustrated in Figure 6.4. For simplicity, here we assume that cases are not strictly divided into a problem and a solution part. Black dots represent particular cases available in the case base and arrows represent valid modifications to be performed during adaptation, for example, defined by adaptation rules. The result of a particular adaptation process is then an adapted case, here, represented as grey dots. One expects that this adapted case is closer to the query than the original cases. However, it must also be considered, that an adaptation process might consist of several single adaptation steps realised, for example, by adaptation rules or adaptation operators. Temporarily, these elementary adaptation steps might also increase the distance to the query. For example, in the shown figure the adaptation process (a) is realised by three elementary steps that lead to intermediate cases. Although the final outcome of this particular adaptation process represents the best case available, the intermediate cases are far away from the query.

Now consider a query q and two cases c_1 and c_2. Although c_1 is the nearest neighbour to q, in the shown example the actual utility of c_1 is lower than the utility

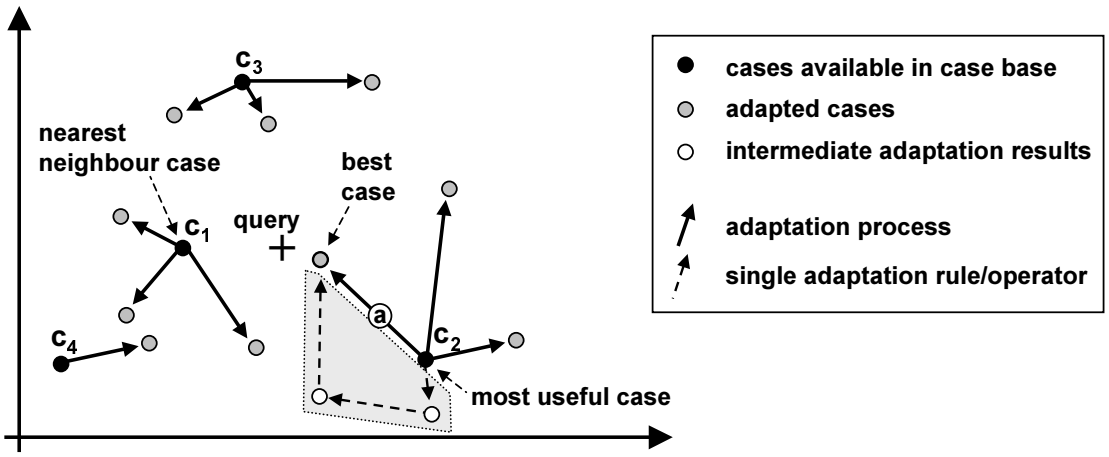


Figure 6.4.: Relation between Adaptability and Utility of Cases

of case c_2. The reason for this little strange fact is the different adaptability of the two cases. While the defined adaptation rules do not allow to improve case c_1 with respect to the q, adaptation of c_2 results in a case that is quite close to the query although the original case c_2 is far away. So, an accurate similarity measure that takes the adaptation possibilities into consideration should assign c_1 a higher similarity w.r.t. to q than c_2. Obviously, a simple distance metric is not sufficient to ensure this.

6.5.1. Learning to Estimate Case Adaptability

However, the definition of similarity measures that approximate this special kind of utility function is a really difficult task due to the complex dependencies between the particular adaptation procedure, the available cases, and the causal relationships in the application domain. The consultation of a domain expert will often not simplify this task significantly because the expert is also not familiar with the relationship between adaptation procedure and solution quality. In the following we show how our learning framework can be applied to facilitate the definition of similarity measures being aware of provided adaptation functionality. In principal, the task of the learning algorithm is to transfer knowledge from the adaptation knowledge container to the similarity knowledge container (cf. Section 2.2.3). This means, the required knowledge is already available in form of adaptation knowledge (e.g. rules), but has to be transformed and to be encoded into the similarity measure, too. Of course, because the corresponding adaptation knowledge is still required, the described process does not modify the adaptation knowledge container. Hence, the same knowledge will be available in form of two different representation formalisms after the knowledge transfer.

The basic precondition to be fulfilled to apply the learning framework is, of course,

the availability of the presumed training data (cf. Definition 4.3) consisting of training examples (cf. Definition 4.2). To obtain a training example representing utility feedback according to the described semantics, the following steps have to be executed:

1. **Generate Query:** First a training query has to be determined. This query might be generated in some way, for example, randomly, or it might be selected from a given set of queries. The advantage of the second method is the possibility to consider real-world queries occurring typically instead of artificial ones resulting in probably more accurate learning results. Therefore, one might record queries that occur during the application of the CBR system, for example, already submitted customer demands in a product recommendation system.

2. **Start Retrieval:** The training query is then used to start a case retrieval based on some initial similarity measure. When dealing with large case bases, the according retrieval result should typically be size-limited (cf. Definition 3.4).

3. **Perform Adaptation:** After retrieving the n most similar cases, all these cases have to be adapted by applying the provided adaptation mechanism. The parameter n has to be determined accurately depending on the complexity of the adaptation procedure. If adaptation is computational intensive, of course, a smaller n must be chosen to guarantee the feasibility of adaptation.

4. **Evaluate Adaptation Results:** To obtain the required utility feedback, finally all adapted cases (or solutions, respectively) have to be evaluated. Therefore, their utility regarding the training query has to be determined. As described in Section 4.2.3 this evaluation might be performed manually or automatically. Depending on the applied evaluation method, ordinal or cardinal feedback (cf. Section 4.2.2) about the adapted cases' utility has to be acquired.

5. **Construct Training Example**: Because our aim is to optimise the similarity measure used to retrieve the original cases and not the adapted cases, the finally constructed training example obviously has to contain the original cases. However, the utility feedback (ordinal or cardinal) is based on the feedback about the corresponding adapted cases acquired during the previously performed evaluation.

By repeating the described procedure with different training queries, it is possible to obtain the required training data. Because the amount of available training data strongly influences the quality of learning results, as many training examples

as possible have to be constructed by executing this procedure. Generally, two different aspects might restrict the amount of available training data:

- If the evaluation of adapted cases has to be performed manually, this may lead again to *high knowledge acquisition effort*. Then, as already described in Section 6.1, the general usefulness of the learning approach has to be estimated carefully prior to the application of our framework.
- Depending on the computational complexity of the adaptation procedure, adaptation of many cases might lead to *infeasible computation times*. By providing more computational resources in form of additional or more powerful machines, this effect can be weakened. Because different training queries can be processed independently from each other on different machines, additional resources allow a linearly increase of the number of available training examples. Nevertheless, one must always be aware that enough training data can be generated. If the evaluation procedure is performed automatically, the described problem holds as well for the complexity of the evaluation procedure.

6.5.2. Automated Evaluation of Adapted Cases

High knowledge acquisition effort can obviously be avoided when automating the evaluation procedure. At least for one typical application domain such an automation can be realised easily. Consider a product recommendation system used for selling products that can be customised in some way, for example, personal computers, cars, insurance products, vacation trips, etc. In the scope of CBR, adaptation functionality can be used to parameterise or configure products with respect to the individual demands of the customers (Stahl and Bergmann, 2000; Stahl et al., 2000).

In this domain, the traditional problem-solution distinction is usually missing. Instead, queries can be characterised as incomplete or inconsistent solutions, while cases represent complete and consistent solutions. However, both entities are described by the case characterisation part of the case representation and the lesson part is empty or not really important, for example, if it only contains a product-number.

In this situation the utility of an adapted case or product, respectively, can be estimated by employing an additional similarity measure. On the one hand, we have a query describing the demands of a customer given in form of an incomplete and maybe also inconsistent product description. On the other hand, we have a product represented by a case contained in the case base to be customised to fit the customer's demands as good as possible. Then, it is possible to define two similarity measures with the following semantics:

- The similarity measure Sim_A compares the query with the cases contained in the case base. The aim of this measure is the selection of products hopefully

representing suitable products w.r.t. to the particular customer demands after being customised. This means, Sim_A considers the adaptability of cases. It is important to note that the described scenario does not rely on a particular realisation of the adaptation procedure, it rather works with any adaptation functionality.

- The similarity measure Sim_U compares the query with already customised products. Its aim is to estimate how far the adapted products fulfill the given demands, i.e. it evaluates the actual utility of the final product. When not providing adaptation possibilities, this would be the measure to be used to retrieve the most useful products.

Obviously, the similarity measure Sim_U can be employed to realise the described evaluation of adapted cases automatically. Therefore, Sim_U has to be defined manually to allow learning of the actually required measure Sim_A. Mostly, the definition of Sim_U should be easier than the manual definition of Sim_A because complex consequences of available adaptation opportunities have not to be considered.

In Section 8.2 an experimental evaluation of our learning framework regarding the described application scenario will be presented.

Meta-Learning

Nevertheless, it is possible to combine the described learning scenario with the ones that require a human similarity teacher, for example, the one already described in Section 6.4, in order to learn also Sim_U. This means, when implementing adaptation functionality in domains without the traditional problem-solution distinction (e.g. the described eCommerce scenario) our framework might be employed in the two described ways simultaneously (see Figure 6.5) to ensure retrieval of adaptable cases. The basic idea is to learn the required evaluation measure Sim_U through a similarity learner U that exploits feedback from some human similarity teacher(s). After having learned an accurate Sim_U, this measure can be used to evaluate obtained adaptation results automatically to generate a sufficient number of training example to be able to learn Sim_A. This approach might be useful, if the amount of manual feedback is sufficient to learn the similarity measure Sim_U, but is not sufficient to learn the more complex measure Sim_A. Otherwise, the available amount of manual feedback could be used to learn Sim_A directly, of course. So, the first learning step can be characterised as a kind of *meta learning* only required to simplify the learning step actually of interest.

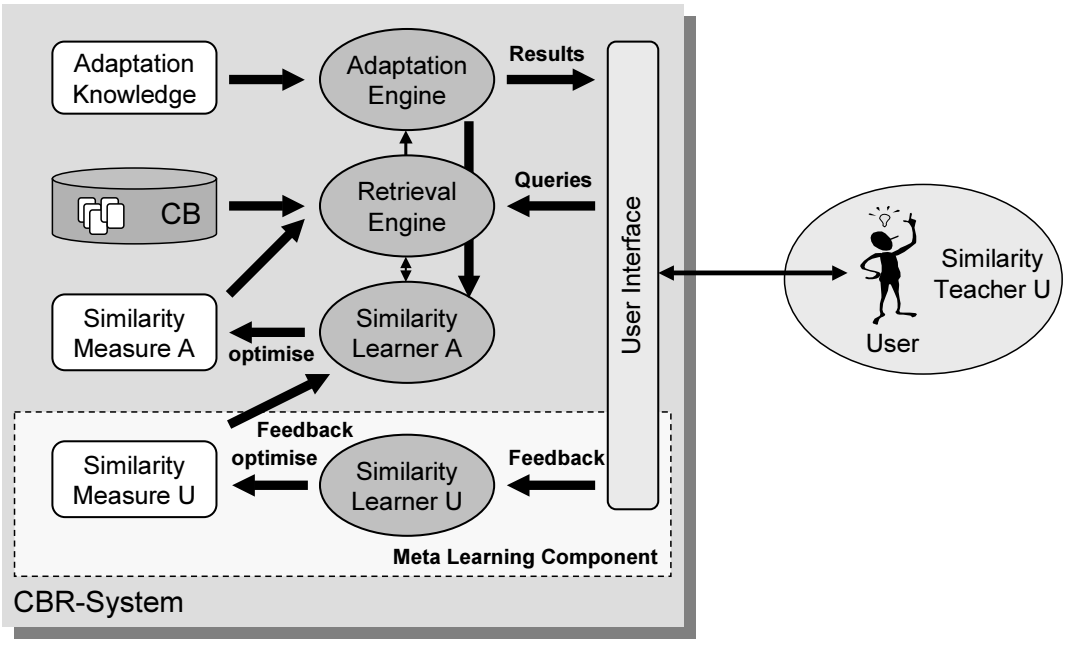


Figure 6.5.: Meta Learning

6.6. Introducing Solution Similarity

In this Section, we present another approach that allows the acquisition of the presumed training data in an automated manner (Stahl and Schmitt, 2002). While the approach described previously is only feasible in domains without the traditional problem-solution distinction, this approach addresses particularly these more classic application domains of CBR (e.g. classification, diagnosis, planning).

In Section 4.2.1 we have already discussed that each case can be seen as a known point of the actually unknown utility function (see also Figure 4.2). Here, we assume that the case base contains only optimal cases, i.e. cases where the corresponding solution is maximally useful with respect to the described problem. When acquiring training data to apply our learning framework, abstractly spoken we are interested in getting additional known points of the utility function. These points then are used for interpolating the similarity measure (see Figure 4.3).

One central element of each training example is obviously the included query (cf. Definition 4.2). So far, we have assumed that training examples might base on arbitrary queries. However, in principle, it is also possible to use problem descriptions of cases contained in a given case base as training queries. Training data will then capture information about the cases' utility for problems represented by other available cases. The major advantage when using given cases as training queries, is the availability of a corresponding solution which might facilitate the acquisition of utility feedback significantly.

Therefore, we introduce a concept called *solution similarity* like illustrated in Figure 6.6. Basically, it should be possible to introduce an additional similarity measure which compares the lesson parts of cases instead of comparing case characterisation parts when estimating the *problem similarity*. Computing similarities between lessons or solutions, respectively, enables us to estimate the utility of a given solution, and so the utility of a particular case, regarding the problem represented by another case. The more similar the solutions, the more useful should be the case because the contained solution is quite close to the optimal solution of the considered query case. Of course, this is again only a heuristics and does not hold in general. Sometimes quite small differences between two solutions might avoid the applicability of both solutions to the same problem.

If a computable solution similarity measure Sim_S is available, the following algorithm allows the automated acquisition of training data based on available cases:

Input: solution similarity Sim_S, case base CB
Output: training data TD_u

procedure solution_similarity(Sim_S, CB) {

1. **for all** $c_i = (d_i, l_i) \in CB = \{c_1, c_2, \ldots, c_n\}$ **do**:
   a) **for all** $c_j = (d_j, l_j) \in CB$ with $i \neq j$ **do**:
      i. compute $Sim_S(l_i, l_j)$;
      ii. set $u(d_i, c_j) = Sim_S(l_i, l_j)$;
   b) construct training example
   $TE_u(d_i) = ((c_1, u(d_i, c_1)), \ldots, (c_{i-1}, u(d_i, c_{i-1})), (c_{i+1}, u(d_i, c_{i+1})),$
   $\ldots, (c_n, u(d_i, c_n))$;
2. **return** training data $TD_u = \{TE_u(d_1), TE_u(d_2), \ldots, TE_u(d_n)\}$

}

This procedure leads to $n^2 - n$ additional known points of the utility function, and so supplies a notable amount of training data provided that a reasonable number of cases is available. Of course, the computed data usually only represents an approximation because the equation in step 1.a.ii of the presented algorithm is only a heuristics. However, also manually acquired training data does mostly not provide absolutely correct utility values.

The basic assumption of the presented idea is the hope that the solution similarity measure Sim_S can be defined easily. Of course, the manual definition of Sim_S must at least be easier than the manual definition of the problem similarity measure to

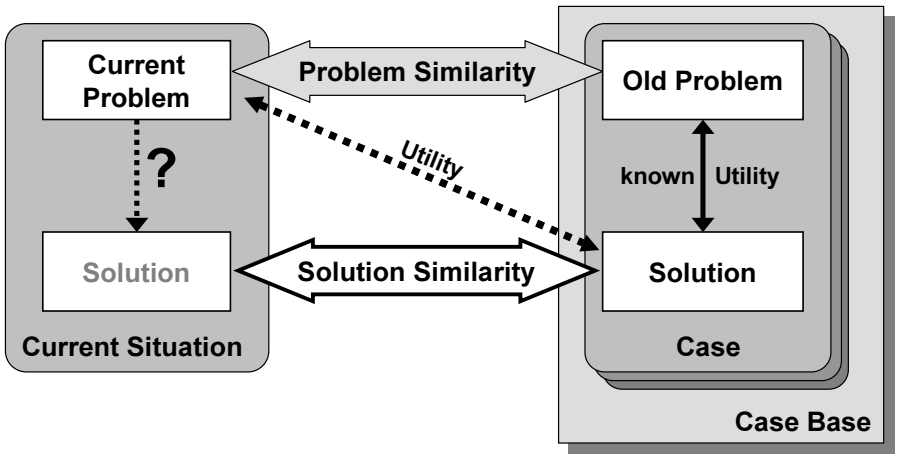


Figure 6.6.: Introducing Solution Similarity

be learned. However, in some application domains this condition is often fulfilled. Consider a classification scenario where the set of possible classes contains more than two class identifiers. Here, misclassifications might lead to different consequences, for example, some misclassifications might be serious, while other might be tolerable. The consequences of misclassifications can be formalised by a solution similarity measure. Because the solutions of classification problems are mostly represented in form of single symbols representing the corresponding class identifiers, a similarity table (cf. Definition 3.14) can be used.

Figure 6.7 shows such a solution similarity measure for a theoretical scenario where some objects shall be classified with respect to their colour. The rows represent the correct classification, i.e. the query value, while the columns represent the classification suggested by a retrieved case. Here, similarities between colours that can easily be determined represent the degree of a misclassification. If an object is indeed "yellow", we would usually prefer a case suggesting to classify it "orange" compared to a case suggesting the colour "blue". This means, the degree of the misclassification might be used as an approximation of a case's utility regarding the particular classification task.

A similar situation might occur in medical diagnosis tasks. Here, on the one hand, we would tolerate a wrong diagnosis, if the corresponding therapy also works (at least a little bit) for the actual disease. On the other hand, a wrong diagnosis, that would lead to a contra-productive therapy perhaps resulting in serious medical complications, must be avoided and is therefore not tolerable.

Another real-world classification domain where similarities between classes might be used to express the consequences of classification failures is described by Schmitt and Stahl (2002). An approach that might be compared to our idea of solution similarity is described by Wilke and Bergmann (1996). Here, different types of clas-

q \ c	yellow	orange	red	blue
yellow	1.0	0.8	0.3	0.0
orange	0.8	1.0	0.8	0.0
red	0.3	0.8	1.0	0.0
blue	0.0	0.0	0.0	1.0

Figure 6.7.: Expressing Consequences of Misclassifications through Solution Similarity

sification failures are considered to cause different costs. A more detailed description of this approach is given in Section 9.2.

However, the approach of solution similarity might be used beyond classification and diagnosis tasks. It can even be applied in domains where adaptation is mostly required (e.g. configuration, planning). Then the concept of solution similarity has to be combined with the approach introduced in the previous section in order to obtain the described training data.

6.7. Maintaining Utility Knowledge

Another important issue is that the utility function to be approximated by the similarity measure may change over time. The change of domain knowledge and the resulting necessity for maintaining this knowledge is a well-known problem in AI. Especially rule-based systems are strongly affected by this maintenance problem. One strength of traditional CBR systems is that a major part of the required knowledge is represented in form of cases. The maintenance of case knowledge is much easier and a lot of respective strategies have already been developed (Leake and Wilson, 2000; Reinartz et al., 2000; Smyth and McKenna, 1998).

However, if we employ utility-based similarity measures instead of pure distance-based measures, we are confronted with an additional maintenance problem. How to ensure the retrieval of the most useful cases, even if the underlying utility function is changing with time?

One idea is to check the retrieval quality in fixed periods by evaluating some example queries. In order to get a measure for the retrieval quality one may determine the average index error introduced in Definition 4.6. If the value exceeds a critical threshold, it is possible to start the optimisation loop (see Section 4.3.4) by using the retrieval results evaluated previously.

Another idea corresponds to the strategies discussed in the scenarios described in Sections 6.3 and 6.4. If we enable the user to give feedback on returned retrieval results computed during the daily use of the system, and if we assume users possess

implicit knowledge about the changing utility function, it should be possible to adapt the similarity measure continuously. For example, collecting utility feedback from customers and optimising similarity measures continuously enables the system to react to changing customer preferences.

Part III.

Implementation and Evaluation

7. Implementing the Framework

In Chapters 4 and 5 concepts and algorithms for learning similarity measures have been introduced. This chapter describes a prototypical implementation of these concepts and algorithms. The objective of the implemented prototype is to show the general feasibility of the learning framework developed in the scope of this thesis.

First, important requirements to such a prototype are discussed. After that, the basic system architecture of the implemented prototype is presented, followed by a detailed description of the new software component realising the actual learning functionality.

7.1. Requirements

Before implementing a new software component, one must be aware of the desired functionality. Basically, it is important to specify clear requirements to be used as the foundation of the implementation process. After discussing basic requirements concerning the implementation of our software component, a short overview of the desired functionality to be realised is given.

7.1.1. Basic Requirements

The implementation of our prototypical system was driven by two major objectives. On the one hand, the prototype can be seen as a proof of concept. It shows that the presented concepts can be realised and represents a kind of *design study*, in particular regarding the graphical user interfaces (GUIs) developed. However, to show that the developed learning framework can be applied in real-world CBR systems, it is not enough to implement it. So, on the other hand, the second important objective of our prototype was the *availability of a test environment* to be used for an experimental evaluation of the learning algorithms developed. The experimental evaluation actually performed and the corresponding results will be described in Chapter 8.

It must be emphasised that it was not our aim to implement a final system for learning similarity measures in daily usage. Therefore, in particular the GUIs are not supposed to be used by novices who are not familiar with the learning framework. Instead, the GUIs provide numerous possibilities to influence the learning algorithms in order to experiment with them.

Because the learning functionality strongly relies on basic CBR functionalities (e.g. similarity-based retrieval) and because it would be infeasible to implement all this functionality from scratch in the scope of this work, an important requirement to the prototype was the realisation in form of an additional component for an existing CBR tool. We selected the commercial CBR shell CBR-Works (Schulz, 1999) already described in Section 3.5.1 to be used as the foundation of our implementation. On the one hand, CBR-Works employs the presumed representation of similarity measures and provides a lot of additional functionality considered to be useful for our prototype (e.g. adaptation rules). On the other hand, it is written in the programming language Smalltalk allowing rapid prototyping.

It was our aim to realise the complete learning framework as an additional module for the CBR-Works system. The module should consist of all data-structures (e.g. for representing training data), learning algorithms (e.g. the presented gradient descent algorithm), and GUIs required by the learning framework. To avoid unnecessary implementation effort, all functionality required additionally should be used from CBR-Works. To ensure a clear separation between existing and new software, a clear interface between CBR-Works and our learning module had to be defined.

Finally, to simplify changes or extensions of the prototype in the future, it was important to pay attention to a well-structured object-oriented design. Such a software design should, for example, allow to add new learning algorithms without major changes in existing software code.

7.1.2. Required Functionality

Now, basic functional requirements expected to be fulfilled by the prototype to be developed are described. These requirements are mainly driven by the concepts presented in Chapters 4 and 5.

Support for Acquiring Utility Feedback

The foundation of the entire learning framework presented in this work is so-called utility feedback to be used as training data for the actual learning algorithms. Hence, the prototype should also support the acquisition of utility feedback in a comfortable way. Here, one must distinguish between the two basic procedures for acquiring utility feedback discussed in Section 4.2.3. On the one hand, manual acquisition with help of a human similarity teacher, and on the other hand, automated acquisition by using some software agents to evaluate the utility of retrieved cases.

Concerning manual acquisition of utility feedback we expected from the prototype that it provides comfortable GUIs. They should enable the user, i.e. in this case some human similarity teacher, to evaluate cases with respect to their utility for a given query. To facilitate the acquisition of ordinal utility feedback it seemed to be advantageous to provide some functionality that allows to analyse and compare two

cases simultaneously. Moreover, the user must be able to arrange cases in form of a partial order to express ordinal utility feedback for a set of cases. Because we expect that human similarity teachers will mostly have problems to express exact utility feedback in form of real numbers, it was also our goal to support the acquisition of that particular feedback type in a more intuitive manner.

Besides supporting manual acquisition of utility feedback, the prototype should also provide functionality to acquire training data in an automated fashion. At least the application scenario described in Section 6.5 should be considered by the prototype to allow an experimental evaluation regarding this scenario (see Section 8.2). Here, the focus did not lie on the development of GUIs, but more on the functionality required to generate training data by employing existing adaptation procedures.

Finally, the prototype should comprehend functionality to manage several training data sets, for example, acquired by different similarity teachers.

Comfortable Learning Environment

After the acquisition of training data, the prototype must, of course, also be able to employ this data to learn accurate similarity measures. In the scope of this work, we expected from the prototype that it implements at least the learning algorithms presented in Chapter 5, i.e., a gradient descent algorithm to learn global attribute weights and a genetic algorithm that allows learning of global attribute weights as well as learning of local similarity measures. However, to simplify testing of alternative algorithms, a further requirement was a modular design of the learning functionality. This means, single learning algorithms should be realised as modules to be integrated into the prototype easily.

In Chapter 4 and 5 we have discussed several important parameters influencing the learning process. Some parameters allow modifications of the error function representing the major foundation of the learning procedure (e.g. see Definition 4.4 or Definition 4.5). Other parameters influence the particular behaviour of the employed learning algorithms (cf. Section 5.2.3 and 5.3.6). To allow an exhaustive and flexible experimental evaluation of the implemented learning algorithms, it should be possible to change these parameters easily in order to measure their impact on the corresponding learning results.

In order to be able to measure not only the final learning results but also the progress of the learning procedure, the test environment should support logging of important parameters (e.g. retrieval errors of intermediate similarity measures) in order to allow a detailed comparison of different settings of learning algorithms.

Similarity Measure Management

The last requirement to the prototype was the possibility to manage several similarity measures learned, for example, obtained by employing different training data

sets or by using different settings of learning algorithms. Another advantage of such a similarity measure management is the possibility to select particular measures as initial measures for learning. Such initial measures might also be defined manually within CBR-Works and it should be possible to import them into the learning module. Then, it would also be possible to evaluate measures defined manually by employing the implemented error functions. Further, measures defined manually might be compared with alternative measures learned from some training data.

7.2. System Architecture

The requirements described previously led to the basic system architecture shown in Figure 7.1. The entire functionality for learning similarity measures is captured in a new software module—called Similarity-Optimiser—to be integrated into CBR-Works. Basically, the Similarity-Optimiser consists of three major components:

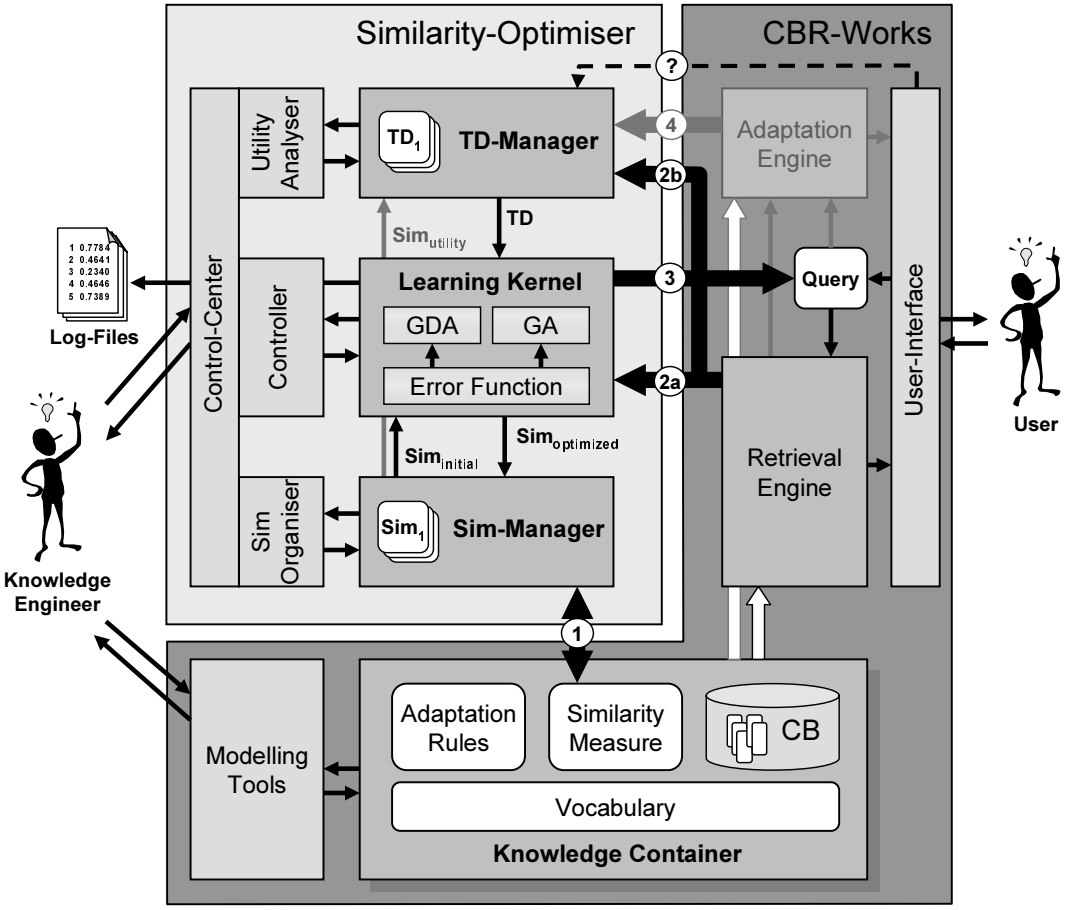


Figure 7.1.: System Architecture

Training Data Manager: This component is responsible for the acquisition of the mandatory utility feedback. It supports manual acquisition of ordinal and exact feedback as well as automated acquisition of feedback based on some evaluation measure. Further, it allows to manage several acquired training data sets.

Learning Kernel: The Learning Kernel realises the actual learning functionality. It includes single modules capturing the functionality of particular learning algorithms and a module for computing different forms of retrieval errors required by the learning algorithms. In the current version of the Similarity-Optimiser the algorithms described in Chapter 5 have been implemented.

Similarity Measure Manager: This component allows to manage several similarity measures. On the one hand, similarity measures defined manually with help of the modelling tools of CBR-Works can be imported. On the other hand, learned similarity measures can be exported to CBR-Works to be used by the retrieval engine. Further, this component allows to select similarity measures to be used as initial measures by the learning algorithms.

To be able to operate these components of the Similarity-Optimiser, for each component a particular GUI is available:

- The Utility Analyser is a sophisticated GUI to support a human similarity teacher during the acquisition of utility knowledge.
- The Controller allows to configure and parameterise the *Learning Kernel* in order to test and evaluate implemented learning algorithms.
- The Similarity Measure Organiser is used to operate with the Similarity Measure Manager.

These specific GUIs can be accessed via one central graphical user interface of the Similarity-Optimiser, the Control-Center. To integrate the new functionality for modelling similarity measures into CBR-Works, four major interfaces are required:

1. The *similarity measure import/export interface (1)* used to transfer similarity measures between the Similarity-Optimiser and CBR-Works.
2. The Learning Kernel as well as the Training Data Manager are reliant on retrieval results generated by the retrieval engine of CBR-Works. Thus, via the *retrieval-result interface (2a,2b)* the Similarity-Optimiser receives retrieval results.
3. To perform a retrieval, the retrieval engine of CBR-Works relies on a query. The *query interface (3)* is used to transfer a query of a particular training example to CBR-Works and to start the retrieval engine.

4. When acquiring utility feedback automatically according to the application scenario described in Section 6.5, the Training Data Manager must obtain adapted cases. Hence, the results of the adaptation engine of CBR-Works are transferred to the Training Data Manager via the *adaptation interface (4)*.

In Sections 6.4 and 6.3 we have discussed the possibility to acquire utility feedback directly from the system's users. When applying the invasive acquisition approach (cf. Section 4.2.3), this would also require the implementation of a specific GUI enabling users to comment retrieval results. Because for the experimental evaluation carried out in the scope of this thesis such a GUI was not required, the current version of the implemented prototype does not comprehend a corresponding interface between the Similarity-Optimiser and the GUI of CBR-Works (marked as *(?)* in Figure 7.1).

7.3. The Similarity-Optimiser

In this section, the three central components of the Similarity-Optimiser together with their GUIs are described in more detail.

7.3.1. The Training Data Manager

The main GUI of the Training Data Manager is shown in Figure 7.2. It is subdivided into 3 parts and provides functionality to manage several training data sets and corresponding training examples.

On the left side the user may create new, duplicate (clone), or remove old training data sets. Each training data set is identified by a unique name. For example, a useful name might correspond to the name of a human similarity teacher or the date of the acquisition of this particular data set.

The middle part of the GUI shows the training examples contained in the training data set currently selected. The individual training examples are again identified by unique names that typically provide a hint on the underlying query.

In order to obtain queries to be used as foundation for training examples, the right part of the GUI supports manual definition by the user or random generation. In the first case, the current query of the CBR-Works retrieval GUI is taken, in the second case the user is able to enter the number of required queries to be generated.

Further, the Training Data Manager supports automated acquisition of training data according to the scenario described in Section 6.5. Therefore, queries are generated randomly and retrieved cases are adapted by applying adaptation rules of CBR-Works. Then, adapted cases are evaluated with respect to their utility by using a special utility measure to be selected by the user. This utility measure is represented in form of a arbitrary similarity measure of CBR-Works.

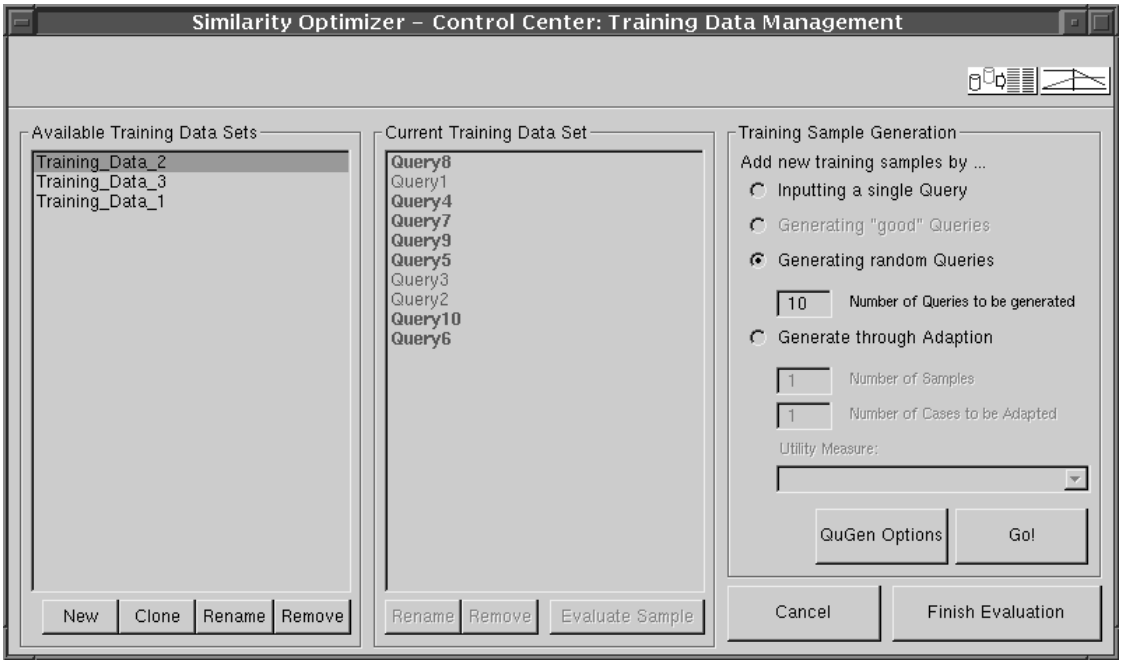


Figure 7.2.: Training Data Manager

The Utility Analyser

In order to acquire utility feedback manually with help of a human similarity teacher, one has to select a query or training example, respectively. After clicking the "Evaluate Sample" button a new window appears , the so-called Utility Analyser (see Figure 7.3) which is subdivided into two major parts.

The upper part is used to rank retrieved cases with respect to their utility for the given query. The list in the left upper corner contains all retrieved cases ranked by their similarity to the query according to the similarity measure currently used by CBR-Works. The user is then able to select cases and to move them via drag and drop into a second list, called "Reordered Retrieval Result". Moreover, the selected cases can be brought into a different order representing the actual utility of the cases. Cases at the top of this list are judged to be more useful by the user than cases below. Within the list the user is also able to create so-called "clusters" containing several cases. The semantics of these clusters is that all contained cases are (nearly) equally useful. In order to visualise clusters, the names of cases are shown in two different colours (blue and grey). Neighbouring cases with the same colour are considered to build a cluster. If the colour between neighbouring cases changes (from blue to grey, or grey to blue), a new cluster starts.

With the described method the user is enabled to express ordinal utility feedback (cf. Section 4.2.2). Nevertheless, the Utility Analyser also supports the definition of exact utility feedback. Because most users might have difficulties to define real

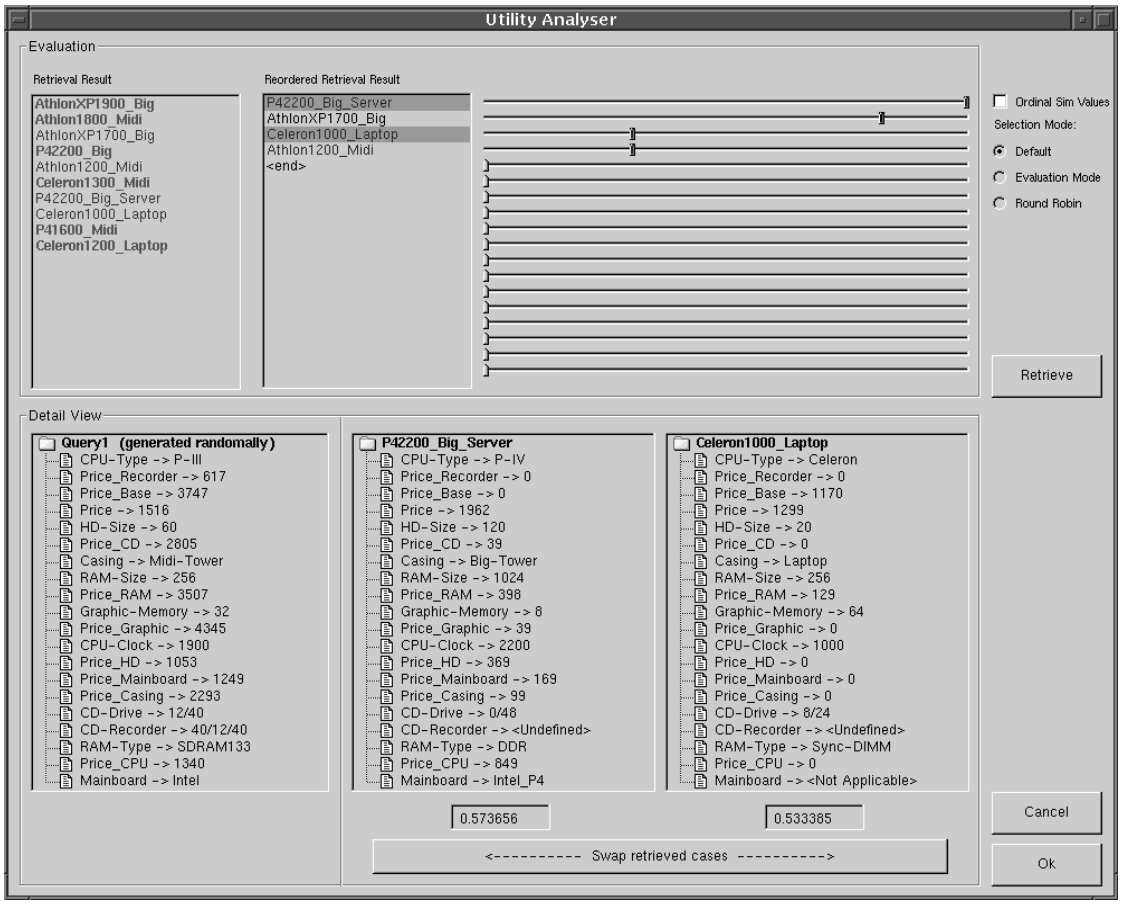


Figure 7.3.: Utility Analyser

valued numbers to express the utility of cases, the GUI provides a slider for each of the selected cases. The meaning of these sliders is that the very right position represents a utility of 1.0 and the very left position a utility of 0.0. By positioning a slider relatively to sliders of other cases, the users may estimate the utility of a case in a more fine-grained way. However, the system internally interprets the position of sliders as exact utility values that might be used by learning algorithms. In spite of that, if the user wants to input exact values, an ad-hoc menu within the utility list supports also this opportunity.

The lower part of the Utility Analyser is used to analyse the utility of a selected case in more detail. Therefore, it consists of three tree views, where the tree view on the very left side shows the attribute values of the query, while the two other tree views are used to show the attributes of two arbitrary cases. These cases can be selected by the user from the retrieved cases list or from the reordered list. Further, the computed similarity values for the two selected cases are shown below the respective tree views. By using these elements of the utility analyser a human similarity teacher

is able to analyse and compare cases in detail in order to decide whether some case is more useful than another one.

After having analysed and ranked an arbitrary number of cases, the utility analyser can be closed with the "Ok" button. The acquired utility feedback is then stored and saved in the corresponding training example listed in the Training Data Manager (cf. Figure 7.2). The user then might select another training example to give further utility feedback regarding another query. Of course, it is also possible to select an already processed training example in order to change or extend the corresponding feedback. To clarify which training examples have already been processed, the names of training examples are shown in bold (not processed yet) or normal (already processed) font.

7.3.2. The Learning Kernel

Basically, the implementation of the Learning Kernel consists of a class hierarchy, where a particular learning algorithm is represented by one class within this hierarchy. To facilitate the implementation of new learning algorithm, the class hierarchy also includes several abstract classes that provide basic functionality required by all or a subset of learning algorithms. Further, extensions or modifications of existing learning algorithms may simply be realised by creating a new subclass that inherits the major learning functionality from an algorithm already implemented. So, for example, the Monte Carlo strategy for the gradient descent approach (cf. Section 5.2.4) is implemented in form of a class MonteCarloGradientWeightLearner that inherits the basic learning functionality from its superclass GradientWeightLearner.

To determine important parameters of learning algorithms (e.g. the learning rate of the gradient descent algorithm, or population size of the genetic algorithm) each learning algorithm or its corresponding class, respectively, includes a dictionary[1], containing default values for the parameters to be used during learning. However, by changing the contents of a dictionary also specific parameter values might be chosen.

Learning

To operate the *Learning Kernel*, the graphical user interface shown in Figure 7.4 can be used. The very right part of this GUI is used to select all important parameters required for learning:

- It provides a menu button[2] where the user must select a training data set to be used as input for the learning algorithm. Therefore, the menu button shows

[1] a particular data structure provided by Smalltalk

[2] a menu button provides a list of alternatives where the user is able to select one of these alternatives

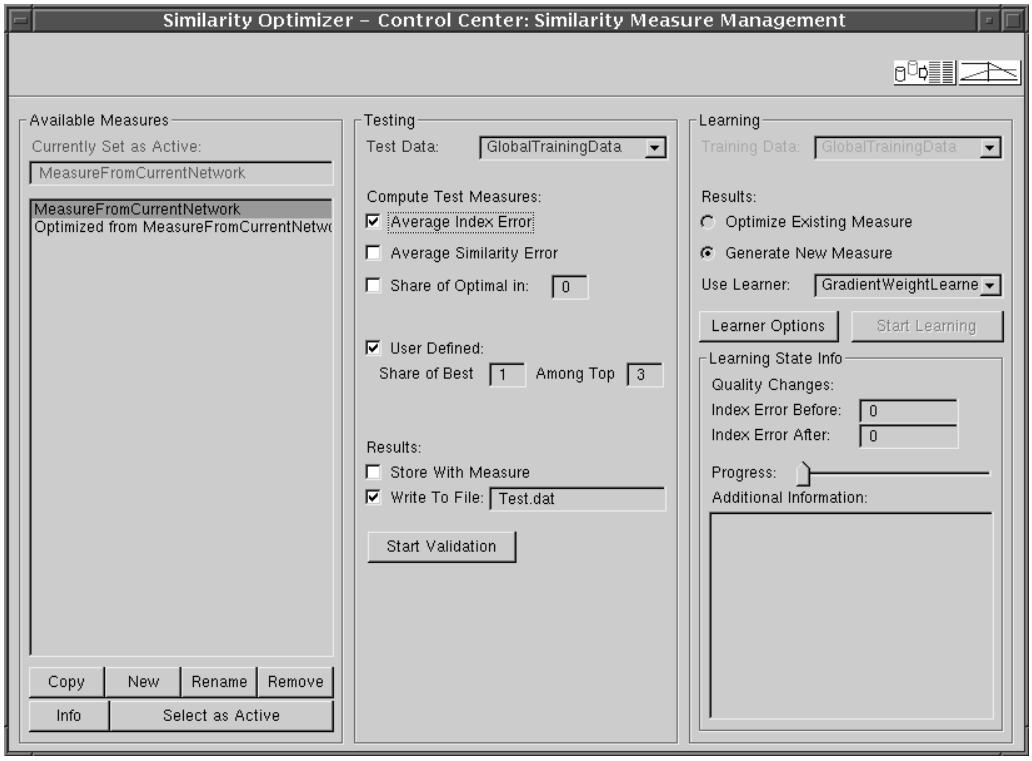


Figure 7.4.: The Learning Control Center

all training data sets currently available within the Training Data Manager (cf. Section 7.2).

- The user can chose between the possibility to optimise the similarity measure currently used by CBR-Works or the generation of a new measure. In the first case, the current similarity measure is used as initial measure for the learning algorithm if some is required (e.g. by the gradient descent algorithm). In the second case, a typical knowledge-poor measure (cf. Figure 3.7) is used as initial similarity measure (if required).

- The user has to select a particular learning algorithm. The selection of available algorithms is provided by a respective menu button.

- Finally, the user might chose particular parameters for controlling the selected learning algorithm. When clicking the "Learner Options" button, a small editor is opened which allows to edit the content of the parameter dictionary of the class implementing the learning algorithm.

When clicking the "Start Learning" button, the learning process is started with respect to the selected parameters. The progress of the learning process is illustrated

below. Here, the average index error (cf. Definition 4.6) caused by the currently learned similarity measure with respect to the used training data is shown.

Evaluation of Similarity Measures

Besides learning, the Learning Control Center also includes functionality to evaluate an existing similarity measure regarding its quality. The parameters required for performing this evaluation procedure can be determined in the middle part of the Control Center. Again the user has to select a particular training data set. This training data set is then used as *test set* in order to compute the respective retrieval error caused by the similarity measure to be evaluated. The following quality measures used to compute the retrieval error are provided:

- average index error according to Definition 4.6
- average similarity error according to Definition 5.4
- 1-in-m and n-in-m soundness according to Definition 3.12

The results of the evaluation process might be stored internally with the corresponding similarity measure or they might be written to a file. When starting the evaluation procedure with the "Start Validation" button, the similarity measure currently used by CBR-Works is evaluated.

7.3.3. The Similarity Measure Manager

The Learning Control Center (see Figure 7.4) also includes the GUI of the Similarity Measure Manager. This component allows to manage several similarity measures where each available similarity measure is identified by a unique name. It shows which measure is currently used by CBR-Works. This measure is also called *active*. Further, it shows alternative measure that might, for example, have been obtained by learning from different training data sets. The user is able to copy existing measures (e.g. to be used as initial similarity measure for a following optimisation process), to create new measures (new measures are initially always of a knowledge-poor manner), or to remove obsolete measures. Further, the names of similarity measures can be modified and corresponding information like data of creation or results of a evaluation process, can be shown. Last but not least, the user is able to export a particular similarity measure to CBR-Works to be used for following similarity-based retrievals. For example, this is required to use a similarity measure as initial measure for learning or to be evaluated.

8. Evaluation

In this chapter, a first experimental evaluation of the presented framework and learning algorithms is presented. This evaluation comprehends two experimental domains that correspond to two application scenarios already introduced in Chapter 6.

First, the basic goals of the presented experimental evaluation are discussed. After that, the design of applications domains and particular settings of the learning algorithms employed are described in detail. Finally, this chapter is concluded with a summary and discussion of the achieved evaluation results.

8.1. Evaluation Goals

In general, before performing an experimental evaluation it is important that one is aware about the actual objectives of this evaluation. Only if the evaluation goals are clear, it is possible to design accurate experiments in order to obtain the desired data.

Concerning the evaluation of our framework presented in this chapter, we have to emphasise that the described experiments are not sufficient to investigate all aspects of our work. Since one of the objectives of this thesis was the development of an approach that can be applied in various domains and application scenarios, such an exhaustive evaluation was infeasible in the scope of this thesis. On the one hand, an detailed investigation of the applicability of our framework regarding all application scenarios described in Chapter 6 would require extensive and detailed data about several application domains. Moreover, this data would have to be modelled by using the employed CBR system. On the other hand, several of the application scenarios discussed presume human similarity teachers who provide the required training data. Since accurate human test persons are difficult to acquire in pure research environments, a realistic evaluation of these scenarios is usually feasible in real-world applications only. Nevertheless, in order to be able to evaluate also one of these application scenarios we have simulated the feedback of human similarity teachers by applying an artificial software agent.

Since we were not able to evaluate all application scenarios due to the mentioned reasons, we have selected the following two scenarios:

1. learning a similarity measure that considers the adaptability of cases to ensure the retrieval of adaptable cases (cf. Section 6.5)

2. learning a personalised similarity measure (cf. Section 6.4)

From our point of view these scenarios represent good examples for demonstrating the basic capabilities of our framework. On the one hand, they are very relevant from a practical point of view. The first scenario is very interesting because it allows to improve the competence and/or performance of a CBR system without much effort. Concerning the second scenario, one can notice that the desire for user adaptive systems increases more and more. Personalised similarity measures are a powerful approach towards the realisation of user adaptive CBR systems. On the other hand, the two selected scenarios enabled us to generate the mandatory training data without the help of human similarity teachers.

Besides an investigation of the remaining application scenarios, a complete evaluation of our framework would also require an investigation of various conceptual aspects and parameters discussed in this work. For example, in Chapter 4 we have introduced different types of utility feedback, and in Chapter 5 we have described several parameters that influence the behaviour of the presented learning algorithms. Because an investigation of all these aspects would require an enormous number of experiments requiring respective hardware ressources, we have restricted our evaluation on some of the most interesting aspects only. In the experiments described in the following we have focused on the following evaluation goals:

- First of all, one objective of our evaluation is to get a *rough impression of the possible benefit* when applying our framework. We want to show which improvements of retrieval quality can be achieved when employing a knowledge-intensive similarity measure that was learned through the application of our framework instead of using a standard knowledge-poor measure (see Section 8.3) or some other initial similarity measure (see Section 8.2).

- Since the amount of available training data is one of the most crucial issues in machine learning, we have paid much attention to the question of *how much training data is required* in order to obtain reasonable learning results. Of course, since the required amount of training data strongly depends on the underlying domain, our experiments can only give an impression for domains with similar characteristics as the domains used here.

- Not only the impact of the amount, but also the quality of training data is an important question. Therefore, one of the described experiments in particular aims on the *investigation of the impact of noisy training data* on our learning algorithms.

- Since we have presented two alternative algorithms for learning attribute weights in Chapter 5, another goal of the evaluation was a *comparison of the performance of the gradient descent and the genetic algorithm.*

- The introduced genetic algorithm is certainly the more interesting learning algorithm due to its capability to learn local similarity measures, too. This aspect is one of the major novelties of our work (see also Section 10.1.2). However, it is still not clear how learning of attribute weights has to be combined with learning of local similarity measures in order to achieve optimal results. Therefore, it was also our goal to *experiment with the different strategies for integrating learning of attribute weights and local similarity measures* discussed in Section 5.3.5.

In the following, two experimental scenarios for evaluating the applicability of our framework are described. Both scenarios have been executed by using the CBR tool CBR-Works (cf. Section 3.5.1) in connection with our implemented prototype for learning similarity measures described in Chapter 7.

The first scenario focuses on the basic functionality of our learning algorithms and aims to demonstrate how the quality of a given similarity measure might be improved by learning from well-founded training data. The second scenario then investigates the impact of noisy training data on the achieved learning results.

8.2. Scenario 1: Learning to Retrieve Adaptable Cases

In Section 6.5 we have described that the utilisation of adaptation functionality makes special demands on the employed similarity measure. In this situation it is not sufficient that a similarity measure approximates the utility of cases with respect to a given query directly. Instead, the similarity measure must also take the provided adaptation functionality into account. It might happen that an apparently less useful case is the most useful one, because it is better adaptable than other cases. In order to consider the adaptability of cases during retrieval, adaptation knowledge has to be encoded into the similarity measure. When not applying learning procedures, therefore, a human knowledge engineer must analyse the available adaptation knowledge in order to extract knowledge that is relevant for the similarity assessment. Moreover, this knowledge then has to be translated into the formalisms used for representing similarity measures. Unfortunately, this procedure is very complicated and requires very experienced knowledge engineers.

With the following experiment, we demonstrate that such a knowledge transfer between adaptation knowledge and similarity knowledge container (cf. Section 2.2.3) can be simplified significantly by applying our learning framework.

8.2.1. The Domain: Recommendation of Personal Computers

We have chosen a typical e-Commerce domain where adaptation increases the performance of a CBR system significantly, namely recommendation of personal computers (PCs). The basic task of our test CBR system is to select and recommend the most useful PC available with respect to the individual demands and wishes of a customer.

In principle, it would be possible to store all descriptions of available and technically valid PC configurations in the case base. Then, the similarity measure would only have to judge how well a PC fulfills the particular demands given by the query. The basic assumption of the application scenario evaluated here is the existence of such a similarity measure.

However, due to numerous customisation possibilities within the PC domain, storing all possible PCs would lead to an infeasible case base size. Thus, adaptation functionality is required to customise a limited set of exemplary PC configurations to the individual customer demands. For example, if a customer wishes a hard disk (HD) with more capacity, the HD installed in a given example PC can easily be exchanged against another one. But to profit from such adaptation functionality as much as possible, the similarity measure used to retrieve suitable example PCs has to be more sophisticated in order to consider provided adaptation possibilities. For example, specific types of memory modules can only be installed if the motherboard supports them.

The case representation used for our experiments consists of 11 attributes (6 symbolic and 5 numeric ones) describing the basic properties of a PC as summarised in Table 8.1. Each attribute possesses its own specialised value range which defines meaningful values used to describe currently valid computer configurations[1]. The corresponding case base used during our experiments contained 15 exemplary descriptions of personal computers. Here, we have paid attention to diversity, this means, the cases represent very different configurations covering the search space sufficiently.

In order to enable the system to perform product customisation, we have defined 15 more and less complex adaptation rules, for example, to adapt the size of the RAM or to add additional components (CD-Recorder, DVD-Drive, etc.). These rules consist of *preconditions* and *actions* (Wilke and Bergmann, 1998). While actions are used to modify a given case regarding the demands of the query, preconditions ensure the technical validity of these modifications. Due to technical constraints not every action can be applied to each case, of course. For example, an Pentium IV CPU can only be installed if the motherboard installed supports this type of CPU.

Further, we have defined an initial similarity measure Sim_U consisting of the following elements:

[1] According to the technical standards in 2002. Of course, due to the rapid technical progress in the computer domain, this case model will be outdated in the near future.

Name	Range	Weight	Similarity
CD-Recorder	("4/4/24", "16/12/40", "24/10/40", "32/12/48", "40/12/40")	0.066	function
RAM-Size	{ 64, 128, 192, 256, 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 1024 }	0.105	function
HD-Size	{ 20, 40, 60, 80, 120, 30 }	0.118	function
Price	[0 .. 5000]	0.158	function
Mainboard	{ "Asus", "Asus_A_DDR", "Asus_A", "Asus_Celeron", "Epox", "Epox_A", "Epox_A_DDR", "Epox_Celeron", "Intel", "Intel_P4", "Microstar", "Microstar_P4", "MSI", "MSI_A_DDR" }	0.013	table
Graphic-Memory	{ 8, 32, 64, 128 }	0.026	function
RAM-Type	{ "SDRAM", "SDRAM100", "SDRAM133", "RAMBUS", "DDR", "Sync-DIMM" }	0.026	table
CPU-Clock	{ 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, 2400 }	0.132	function
CD-Drive	("0/48", "0/52", "6/24", "8/24", "12/40", "16/40", "16/48")	0.039	function
Casing	("Laptop", "Mini-Tower", "Midi-Tower", "Big-Tower")	0.263	table
CPU-Type	{ "AMD", "Duron", "Athlon", "Athon XP", "Intel", "P-III", "P-IV", "Celeron", "Tualatin" }	0.053	table

Table 8.1.: The PC Domain

- 11 attribute weights $w_1 \ldots w_{11}$ as enumerated in Table 8.1,
- 11 local similarity measures, where 4 of them are represented as similarity tables (cf. Definition 3.14) and 8 are represented as difference-based similarity functions based on linear difference (cf. Definition 3.15)[2],
- and a standard aggregation function in form of a weighted average aggregation (cf. Section 3.3.5).

During the definition of Sim_U, we have encoded knowledge about the meaning of technical properties for the recommendation process, however, we have ignored the impact of the provided adaptation rules. This means, Sim_U aims to approximate the utility of a given PC regarding a given query as it would be sufficient without adaptation.

8.2.2. Generation of Training Data

The general procedure of how to acquire training data required for realising a knowledge transfer between adaptation and similarity knowledge was already described in Section 6.5. Basically, single training examples can be obtained by performing the following five steps:

1. generation of a training query
2. start of case retrieval by using an initial similarity measure
3. adaptation of the n most similar cases
4. evaluation of adaptation results
5. construction of a training example by re-ranking the cases according to the outcome of step 4

Of course, the most interesting step is step 4, i.e., the evaluation of the results obtained during the precedent adaptation steps. This evaluation procedure can be executed automatically by using a special similarity measure which compares adapted cases with the query. Here, we are able to exploit the already defined similarity measure Sim_U because it measures the suitability of a given PC configuration regarding a particular query.

Figure 8.1 illustrates the procedure for generating training examples. First, a query is generated randomly and used to initiate the retrieval of all cases according to the initial similarity measure Sim_U. Due to the limited number of cases available

[2]Because some of the 8 symbolic attributes are modelled as ordered symbols (cf. Section 3.3.3), similarity functions could also be used for these attributes.

in our example domain, then all 15 cases are adapted by applying the provided adaptation rules. After that, for all adapted cases the similarity computation is repeated by using Sim_U again. This leads to a modified case ranking which has to be stored in form of a training example.

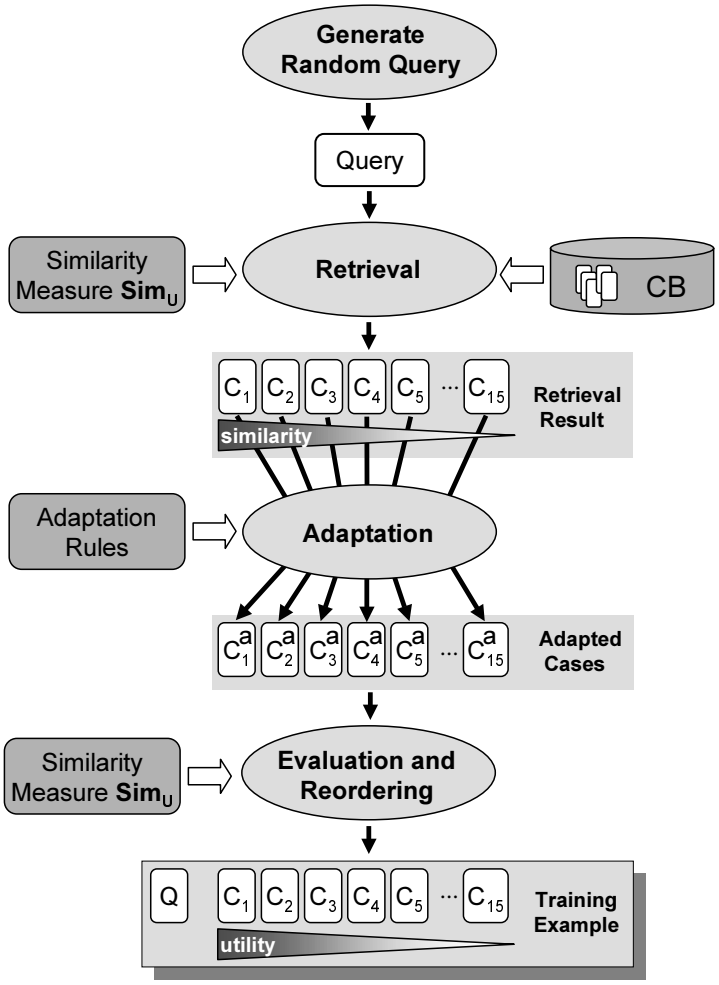


Figure 8.1.: Generation of Training Data

8.2.3. Learning

For the actual learning phase we have generated training data by applying the previously described procedure. For each repetition of the experiments we generated a new *training set* S_{train} consisting of 200 training examples. Further, we generated one independent *test set* S_{test} consisting of 200 training examples, too. While the training sets were used as actual input for the applied learning algorithms, the test set was only used for evaluation of all obtained learning results.

In order to investigate the influence of the training data size, each experiment was repeated several times by using increasing number of training examples from S_{train}. Here, the resulting subsets S_{train_i} with $i \in \{5, 10, 20, 35, 50, 70, 100, 140, 200\}$ were generated by adding new training examples to the previously used training set. Hence, for example, $S_{train_{10}}$ includes all training examples of S_{train_5} and 5 additional examples from $S_{train} \setminus S_{train_5}$. Each S_{train_i} was then used as input for a particular version and setting of one of our learning algorithms in order to optimise the initial similarity measure Sim_U and to obtain a new measure Sim_{A_i} which should also considers adaptation possibilities when determining the utility of cases. For evaluating the quality of each Sim_{A_i} learned, we have used the following three quality measures:

- the average index error on S_{test} (cf. Definition 4.6),
- the percentage of 1-in-3 sound retrieval results regarding S_{test} (cf. Definition 3.12),
- the percentage of 1-in-1 sound retrieval results regarding S_{test} (cf. Definition 3.12).

While the index error represents more an abstract measure for estimating the entire retrieval quality, i.e. the ranking quality with respect to all cases, the two other measures are more suitable to show the actual benefit of our learning framework in the underlying application scenario. Which of the two versions of the n-in-m soundness is the more accurate one in practice, depends on the complexity of the applied adaptation functionality. On the one hand, if we suppose that adaptation of 3 cases immediately after retrieval is feasible, it is sufficient to ensure that the most useful case is under the 3 most similar ones. On the other hand, if we want to minimise answer delays and therefore only adapt the most similar case, of course, this case should also be the most useful one.

With the described procedure we have then evaluated several versions and settings of our learning algorithms as summarised in table 8.2.

First, we have exclusively optimised the 11 attribute weights by applying the gradient descent algorithm (GDA) and the genetic algorithm (GA). Here, we have tested two different values for the *error-position weight* θ used within the employed error functions, namely the average similarity error $\hat{E}_S$ (cf. Definition 5.4) for the gradient descent algorithm, and the average index error $\hat{E}_I$ (cf. Definition 4.6) for the genetic algorithm. The resulting four experiments (exp. 1-4) mainly aimed to compare the two different algorithms. To get an impression of the robustness of the algorithms we have repeated each experiment 10 times with different training sets. However, to ensure the comparability of the results, we used the same training sets for the different learning strategies. This means, for example, that the 10 repetitions

No.	Algorithm	Strategy	Error Function	θ	Generations	Rep.
1	GDA	-	$\hat{E}_S$	0	-	10
2	GDA	-	$\hat{E}_S$	10	-	10
3	GA	GA_W	$\hat{E}_I$	0	60	10
4	GA	GA_W	$\hat{E}_I$	10	60	10
5	GA	Seq_{W-LS}	$\hat{E}_I$	10	60/400	5
6	GA	Seq_{LS-W}	$\hat{E}_I$	10	60/400	5
7	GA	$Par_{composed}$	$\hat{E}_I$	10	4400	5
8	GA	Par_{pseudo}	$\hat{E}_I$	10	60/400	5

Table 8.2.: Performed Experiments

of experiment 1 were based on the same training and test sets as the 10 repetitions of experiment 2.

In the second part of the evaluation (exp. 4-8), we have tested different strategies for combining learning of attribute weights and local similarity measures when applying the genetic algorithm. These four strategies, which we have already discussed in Section 5.3.5, are:

Seq_{W-LS}: Here, weights and local similarity measures are processed sequentially, where the weights are learned first and the local similarity measures are treated in a second optimisation loop. During this second loop, the 11 local similarity measures have been optimised by applying the pseudo parallel approach. We have restricted the experiments to 60 generations for learning weights and 400 generations for learning each local similarity measure.

Seq_{LS-W}: Similar to the previous strategy, here, weights and local similarity measures are also processed during two subsequent optimisation loops. In contrast to the Seq_{W-LS} strategy, here, all local similarity measures are learned first in the pseudo parallel manner. The number of processed generations was the same as for strategy Seq_{W-LS}.

$Par_{composed}$: Here, weights and all local similarity measures are learned in parallel by using composed individuals. For this strategy we allowed the genetic algorithm to process much more generations (4400) due to the much larger search space to be considered by composed individuals at once.

Par_{pseudo}: Here, twelve independent optimisation loops for learning weights and all local similarity measures are processed alternated. The total number of generations was the same as for the sequential strategies to ensure the comparability of the achieved results.

Due to the higher computational complexity we have repeated each of these experiments only 5 times using 5 of the training sets already employed during the experiments 1-4. During the experiments 5-8 we have always used the average index error with an error-position weight $\theta = 10$ since this value showed the best results in the experiments performed previously (see Section 8.2.4).

Concerning other important parameters of our learning algorithms we have used constant values during all experiments. For the gradient descent algorithm we have used the following parameters discussed in Section 5.2.3:

- As *initial weight vector* required by the gradient descent algorithm we have always used the weights of Sim_U (see Table 8.1).
- For the implementation of the *stop-criterion* we have applied a combined approach consisting of a maximal number of optimisation iterations failed (10) and a maximal number of iterations in total (100).
- $\Delta_{max} w = 0.5$
- λ-*reduction-rate* $= 2$

For the genetic algorithm we have selected the following values for the parameters discussed in Section 5.3.6:

- population Size = 20
- reproduction Rate = 0.5

8.2.4. Results

In this section, the results of the previously described experiments are presented. First, we compare the results of the gradient descent algorithm and the version of the genetic algorithm that optimises attribute weights only, i.e. the results of the experiments 1-4. After that, we present the results of the different versions of our genetic algorithm that optimise both, weights and local similarity measures, i.e. the results of the experiments 5-8. In this section we focus on the presentation of the results only. A detailed discussion of all results is given in Section 8.4.

Gradient Descent Algorithm Vs. Genetic Algorithm

The objective of the first four experiments was a comparison between the gradient descent approach and the genetic algorithm when learning attribute weights only. Figure 8.2 illustrates the results achieved with the gradient descent algorithm when using error-position weights $\theta = 0$ (exp. 1, left chart) and $\theta = 10$ (exp. 2, right chart). Here, the x-axis corresponds to the number of training examples used, the

left y-axis denotes percentage values for 1-in-3 and 1-in-1 sound retrieval results on the 200 test examples of S_{test}, and the right y-axis denotes values of the average index error on S_{test}.

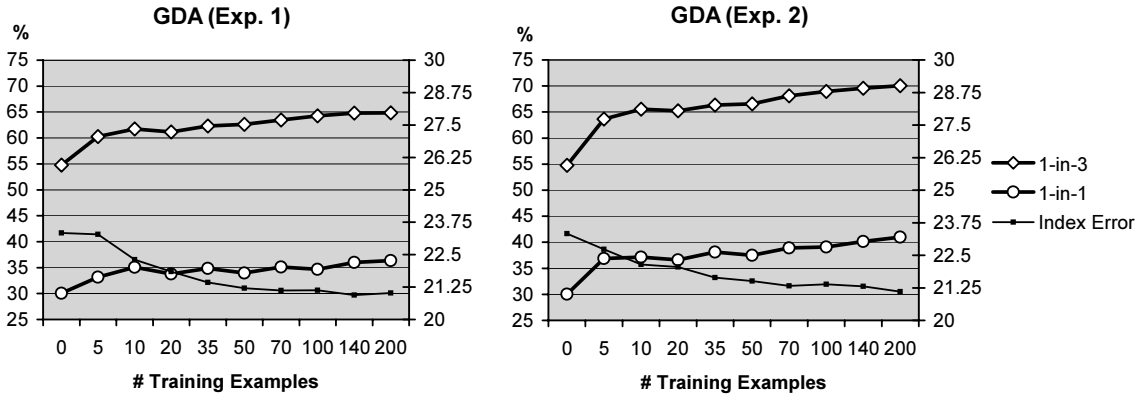


Figure 8.2.: Experiments 1 and 2: Dependency on Training Data Size

It is important to note that the x-axis is not scaled linearly. Due to the chosen scale, the learning curves are actually steeper for small training data sets as they appear in the charts. Further, the different characteristics of the three quality criteria must be considered. On the one hand, because the two versions of the n-in-m soundness of retrieval results (cf. Definition 3.12) represent quality measures, the corresponding values typically increase after learning, resulting in acceding curves. On the other hand, because the average index error measures retrieval failures, here improvements in retrieval quality lead to decreasing values, and therefore to descending curves.

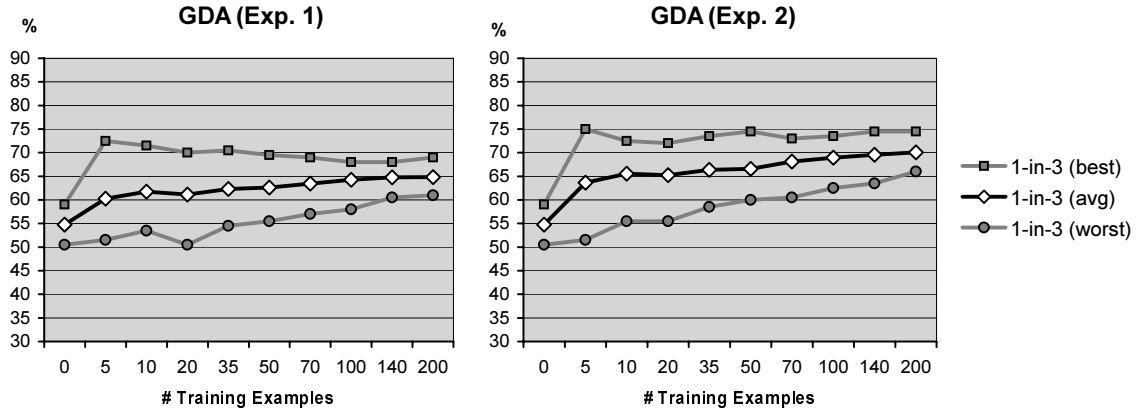


Figure 8.3.: Experiments 1 and 2: Variations of Results

All three curves shown correspond to average values obtained during 10 independent repetitions of the experiments. They show, that the quality of retrieval results

clearly could be improved by applying the described learning procedure. For example, the right chart states that the percentage of 1-in-3 sound retrieval results could be increased from around 55% prior to learning up to around 70% in average when exploiting the training set $S_{train_{200}}$, i.e. 200 training examples.

In order to get an impression of the variance of the learning results achieved during the 10 repetitions, in Figure 8.3 we have illustrated exemplarily also the best-case and worst-case values for the 1-in-3 soundness criterion. Since for the two other quality measures the curves are quite similar, we have restricted the charts on this meaningful measure. Again, the left chart shows the results for $\theta = 0$, and the right chart shows the results for $\theta = 10$.

Since the experimental settings were the same for the gradient descent and the genetic algorithm, the results of the genetic algorithm are illustrated in the same way as previously described for the gradient descent algorithm. The resulting charts are shown in Figure 8.4 and Figure 8.5.

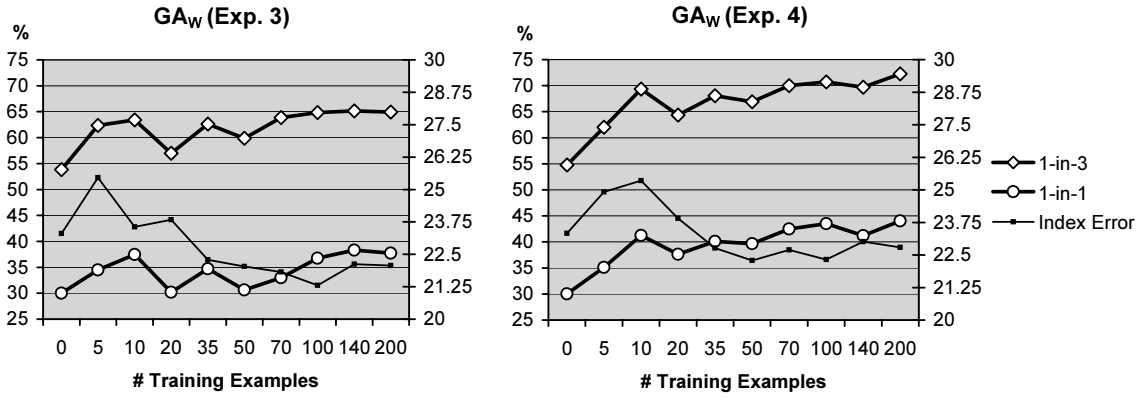


Figure 8.4.: Experiments 3 and 4: Dependency on Training Data Size

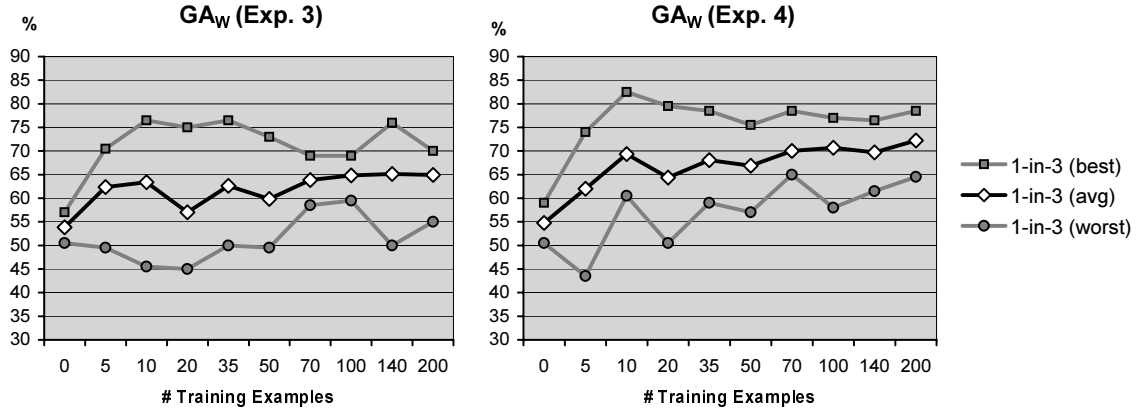


Figure 8.5.: Experiments 3 and 4: Variations of Results

Learning Weights and Local Similarity Measures

In the second part of this evaluation scenario we have investigated the performance of the four discussed strategies used to combine attribute weight learning and local similarity measure learning when applying the genetic algorithm. Figure 8.6 shows the average performance of these four strategies in dependency on the number of training examples exploited. The two charts at the top show the achieved results when applying the two sequential approaches Seq_{W-LS} (exp. 5, left chart) and Seq_{LS-W} (exp. 6, right chart). The two charts below show the outcome of the two parallel approaches $Par_{composed}$ (exp. 7, left chart) and Par_{pseudo} (exp. 8, right chart). The results are illustrated in the same way as already described for the experiments 1-4 with the exception that the curves correspond to average values obtained during 5 repetitions of the experiments.

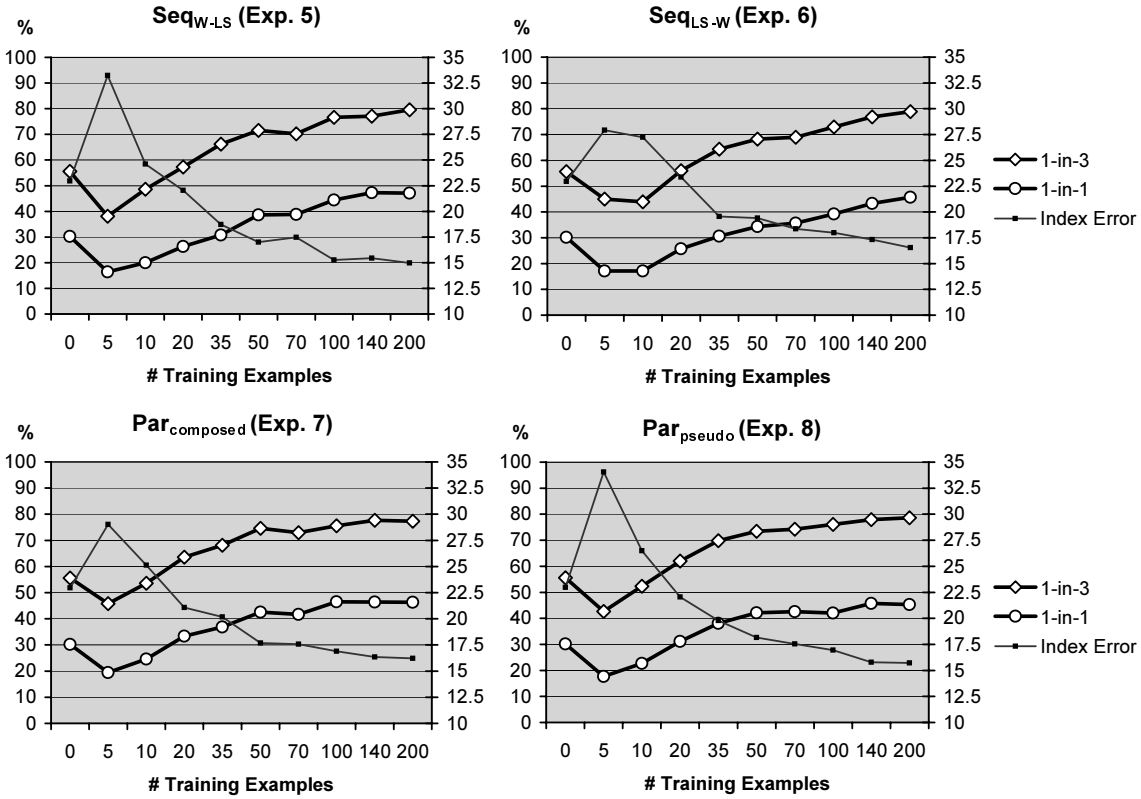


Figure 8.6.: Experiments 5-6: Dependency on Training Data Size

In order to get an impression how the results vary over the 5 independent repetitions of the experiments, in Figure 8.7 we have illustrated again the worst, the average and the best values for the 1-in-3 soundness criterion.

In general, when applying a genetic algorithm, the number of processed generations is a very important parameter. On the one hand, if it is chosen to small, the

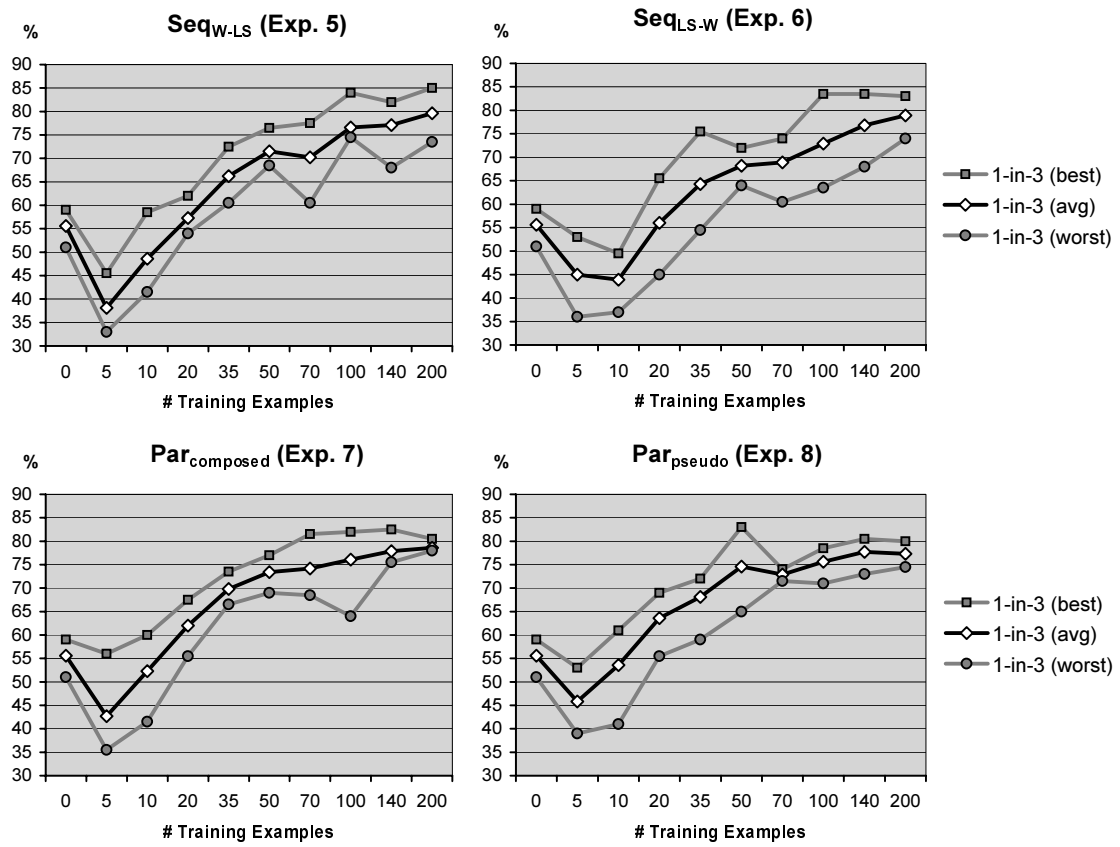


Figure 8.7.: Experiments 5-6: Variations of Results

optimisation process will be stopped before it could converge to an optimum. On the other hand, if using a very large value, one will probably obtain good learning results, but one will possibly also waste a lot of computation time, because the algorithm actually converges much earlier. Figure 8.8 illustrates the improvements concerning the three introduced quality measures (when exploiting $S_{train_{200}}$) depending on the number of generated generations for two exemplary strategies, namely the Seq_{W-SL} and the $Par_{composed}$ strategy. The shown results again represent average values for the 5 repetitions of experiments 5 and 7. While the learning procedure of the parallel strategy $Par_{composed}$ is a continuous process, the procedure of the sequential strategy Seq_{W-LS} can be divided into two parts, namely learning of weights and subsequent learning of local similarity measures. The two charts indicate that we have chosen the number of generations appropriately since a clear convergence for the final generations can be noticed.

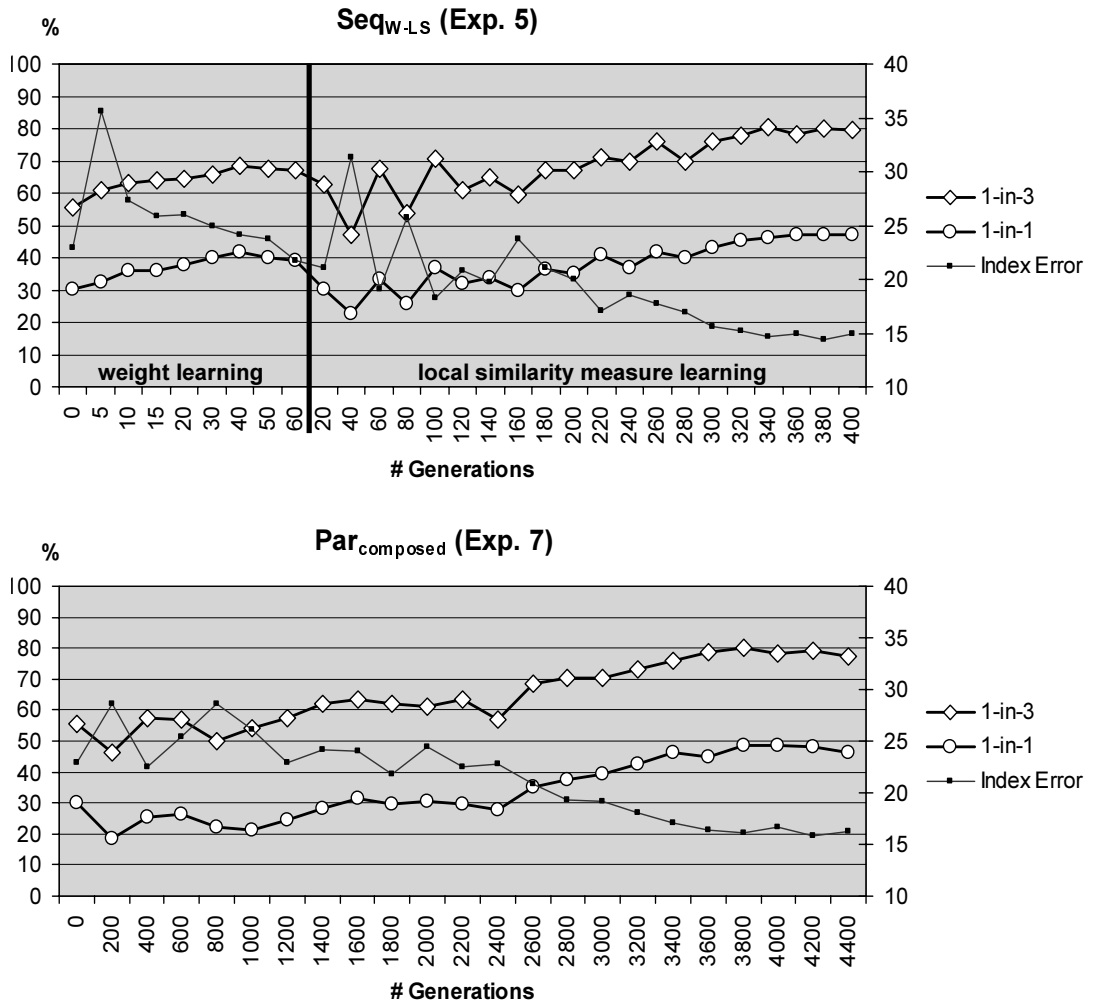


Figure 8.8.: Experiments 5 and 7: Dependency on the Number of Generations

Computation Times

Another interesting issue when comparing learning algorithms is, of course, their performance with respect to computation time. In Figure 8.9, a summary of the average computation times of all six[3] discussed algorithms is given. All measured times were obtained by using a PC with an 1800MHz Athlon CPU provided with 512MB DDR-RAM. All times represent the number of minutes required for the entire learning procedure when exploiting $S_{train_{200}}$.

[3]Since θ does not influence the computation times, neither for the gradient descent nor for the genetic algorithm, we have neglected a separation of the corresponding experiments.

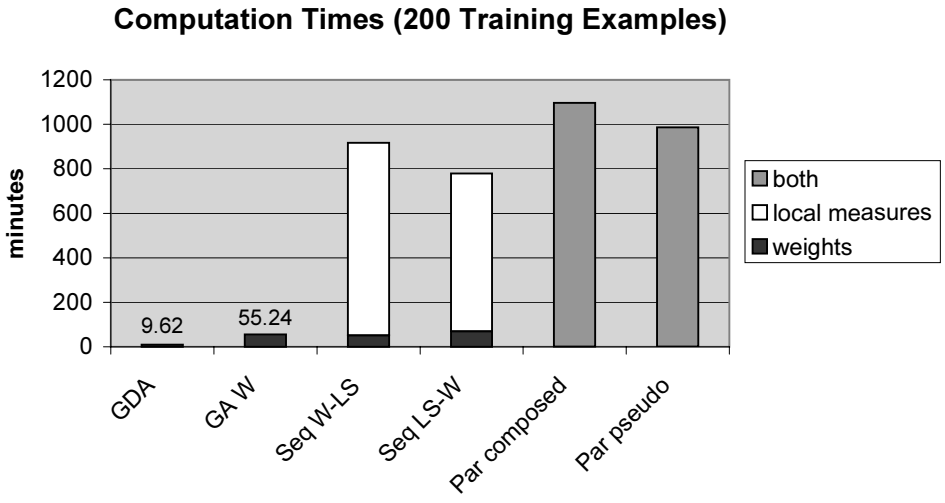


Figure 8.9.: Computation Times for Processing 200 Training Examples

8.3. Scenario 2: Learning Customer Preferences

As discussed in Section 6.4, our framework might be applied in order to learn preferences of individual customers or customer classes, respectively, in an eCommerce scenario. According to customer feedback about preferred products, the learning algorithms then have to generate a similarity measure which reflects these preferences as good as possible. However, one crucial issue of this application scenario is the kind of utility feedback that can be expected to be given by customers. On the one hand, customers usually will not be willed to rerank a large set of presented products. Thus, every single training example will contain information about less cases only. On the other hand, customers—as most human similarity teachers—will not always be consistent in their statements. One time they might prefer another product as to another point of time, though the demands are the same. This aspect plays a very important role if personalisation aims on customer classes and not on individual customers. Although if one supposes that all customers of a particular class have quite similar preferences, utility feedback of different customers will mostly also include contrary statements leading to noisy training data.

While little amount of information in single training examples might be compensated by a large number of training examples, noisy training data makes learning much more difficult. Thus, the experiment described in the following aims on a first investigation of the impact of noisy training data when applying our framework. Here, we have focused the evaluation on the genetic algorithm only.

Name	Range	Weight	Similarity
Erstzulassung	[1997.0 .. 2003.0]	0.156	function
Marke	{ "AUDI", "BMW", "CHRYSLER", "FIAT", "FORD", "MERCEDES-BENZ", "OPEL", "VW" }	0.067	table
Preis	[3000 .. 50000]	0.222	function
Km	[5000 .. 155000]	0.156	function
Variante	{ "Cabriolet", "Coupe", "Kastenwagen", "Kombi", "Limousine", "Pritsche", "Roadster" }	0.178	table
Farbe	{ "blau", "gelb", "grau", "grün", "rot", "schwarz", "silber", "weiß" }	0.089	table
KW	[33 .. 225]	0.111	function
PLZ	("0", "1", "2", "3", "4", "5", "6", "7", "8", "9")	0.022	table

Table 8.3.: The Used Cars Domain

8.3.1. The Domain: Recommendation of Used Cars

As domain we have chosen another typical eCommerce domain, namely recommendation of used cars. Since used cars usually cannot be customised regarding customer requirements, here, case adaptation is not required. When selling used cars, it can be noticed that one is mostly confronted with different customer classes that show very different behaviour in selecting preferred cars. On the one hand, there might be a class *"family fathers"* who search bigger, functional, but economical cars. For example, they would probably prefer a caravan instead of a limousine. On the other hand, there might be a class *"fun drivers"* who are rather looking for sports cars and who are characterised by a bigger budget. While a family father might rule out a car that costs 2500 EUR more than the demanded price, a fun driver might be willed to pay 10000 EUR more if the presented car represents her/his "dream car". Such different preferences can be encoded into accurate similarity measures.

Our domain model consists of 8 attributes (4 symbolic and 4 numeric) that describe the most important properties of a car as summarised in Table 8.3. As in the PC domain described in Section 8.3.1, each attribute is connected with its own specialised value range and local similarity measure. The corresponding case data was extracted from one of the numerous internet sites offering used cars[4]. Our exemplary case base contained descriptions of 100 used cars covering a wide spectrum

[4]http://www.e-sixt.de/apps/auto/boerse/geb_erw_suche.jsp

of different alternatives.

For evaluation purposes we have defined two different similarity measures Sim_I and Sim_U, each consisting of

- 8 attribute weights $w_1 \dots w_9$ with $\forall i w_i = \frac{1}{8}$ for Sim_I and 8 different weights for Sim_U as enumerated in Table 8.3,
- 8 local similarity measures (4 difference-based similarity functions and 4 similarity tables),
- and a standard aggregation function in form of a weighted average aggregation.

Sim_I was used as initial similarity measure to be optimised during learning. Here, we used a standard knowledge-poor similarity measure as already illustrated in Figure 3.7. Sim_U was used to generate training data as described in the following section.

8.3.2. Generation of Training Data

Since for our evaluation it was infeasible to acquire human similarity teachers that would be able to deliver the required training data, we have employed a simulation procedure. The idea of this simulation was to model the behaviour of a customer or customer class, respectively, by using the similarity measure Sim_U. During the definition of Sim_U we have encoded imaginable preferences of some customer class into the similarity measure. Then, Sim_U could be used to provide utility feedback that corresponds to the encoded customer preferences.

The resulting procedure for generating single training examples is shown in Figure 8.10. First, a randomly created query is used to start retrieval which returns a size limited retrieval result (cf. Definition 3.4) consisting of the 10 most similar cases according to similarity measure Sim_I. After that, the similarity measure Sim_U is used to select and rank the 3 most useful cases from the retrieval result (in our example the cases C_3, C_6, C_1). The idea here was to simulate a scenario where customers are able to comment retrieval results by selecting their 3 favourite products out of the 10 presented products in order to provide the system with information about their individual preferences. However, because the behaviour of customers is often unpredictable, we have also implemented a mechanism for introducing noise which relies on three different probability values:

- ρ_1 denotes the probability that one of the 3 selected cases is not chosen according to Sim_U, but randomly from the retrieval result. Whether this "failure" within the utility feedback is located at the first, the second, or the third case is determined randomly with a unique probability.

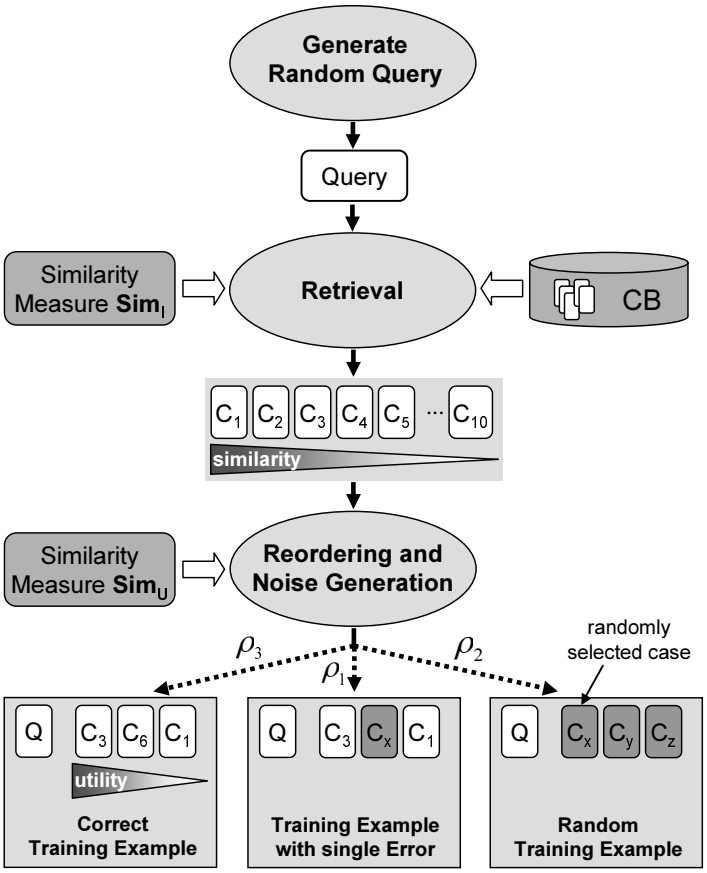


Figure 8.10.: Generation of Training Data

- ρ_2 denotes the probability that the entire utility feedback for the current query is inconsistent with the presumed utility function and thus meaningless. In this situation all three cases are selected randomly from the retrieval result.
- $\rho_3 = 1-(\rho_1+\rho_2)$ denotes the probability that optimal utility feedback without inconsistencies with respect to Sim_U is provided. In this case, all three cases are selected and ranked according to Sim_U.

In order to investigate the impact of this kind of noise on the outcome of our learning algorithms, we have generated samples of training data with the following characteristics:

- Each sample consisted of *three independent training data sets* $S_{train\_0/0/100}$, $S_{train\_2.5/7.5/90}$ and $S_{train\_7.5/22.5/70}$ where the three numbers $\rho_1/\rho_2/\rho_3$ corresponded to the percentage of introduced noise. This means, one training set was free of noise, another one contained around[5] 10% noisy training examples

[5]Since noise was introduced using probabilities, the exact amount of noise might differ slightly.

(2.5% random examples and 7.5% examples with minor errors), and the final one contained around 30% noisy training examples (7.5% random examples and 22.5% examples with minor errors).

- Each $S_{train\_\rho_1/\rho_2/\rho_3}$ consisted of *1000 training examples*, where the queries used were identical for each set. Here, we have chosen such large training sets since each training example contains utility feedback for 3 cases only. Hence, we can expect that much more training examples will be required to achieve good learning results compared to the experimental scenario described in Section 8.2.

- Further, each sample included *one independent test set* S_{test} consisting of 250 noise-free training examples. In contrast to the training sets, where each training example contained feedback about 3 cases only, the examples of S_{test} included feedback about all 100 cases. This enabled us to measure the achieved retrieval quality more exactly.

8.3.3. Learning

In this experimental scenario it was not our aim to investigate different settings of our learning algorithms, because this was already the objective of scenario 1 described in Section 8.3. Therefore, we only used one version of the genetic algorithm that showed good results in the first experiment, namely Seq_{W-SL}. Concerning the employed error function we have used the average index error with an error-position weight $\theta = 10$.

For measuring the quality of the achieved learning results, we used the same quality measures as already introduced in Section 8.2.3, namely the average index error, and the percentage of 1-in-1 and 1-in-3 sound retrieval results. Additionally, we have measured the percentage of 1-in-10 sound retrieval results. This soundness criterion seems to be more meaningful since we simulate an eCommerce scenario where the 10 most similar products are presented to the customer. Hence, it is at least our aim that this return set contains the preferred product. In order to obtain average values, we have repeated the experiment 3 times with newly generated samples of training data. Each of these 3 experiments consisted of 3 sub-experiments, each using one of the training sets $S_{train\_0/0/100}$, $S_{train\_2.5/7.5/90}$ or $S_{train\_7.5/22.5/70}$.

In order to investigate the influence of the amount of available training data, we have again applied an iterative procedure that used an increasing number of training examples, leading to actual training sets $S_{train\_\rho_1/\rho_2/\rho_3\_i}$ with $i \in \{50, 100, 250, 500, 1000\}$, where i denotes the number of training examples used.

8.3.4. Results

Figure 8.11 shows an overview of the achieved learning results in dependency on the number of training examples used. The charts can be interpreted in the same way as already described in Section 8.2.4. The only differences are the additional 1-in-10 quality criterion, and the different scale of the x-axis and the second y-axis required for the average index error. Here, the values of the index error are much larger compared to the first scenario since the corresponding test set S_{test} consisted of more cases. Again, it is important to notice that the x-axis is not scaled linearly, which leads to an alienated appearance of the actual shape of the learning curves.

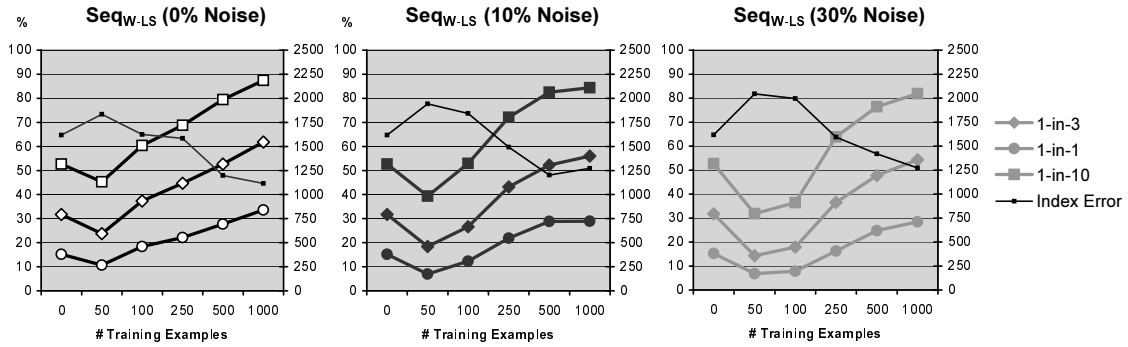


Figure 8.11.: Learning Weights and Local Similarity Measures: Overview

The experiment led to three similar charts for the different training sets corresponding to different amounts of introduced noise. Again, all curves correspond to average values obtained during the 3 repetitions of the experiments.

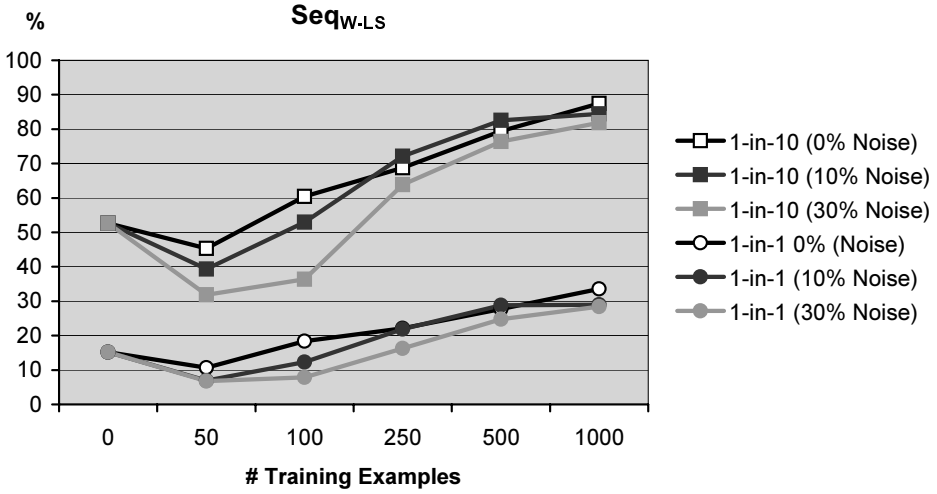


Figure 8.12.: Learning Weights and Local Similarity Measures: Impact of Noise

In order to enable a better investigation of the actual impact of noisy training data, in Figure 8.12 we have summarised the curves for the 1-in-1 and the 1-in-10 soundness criteria in a single chart.

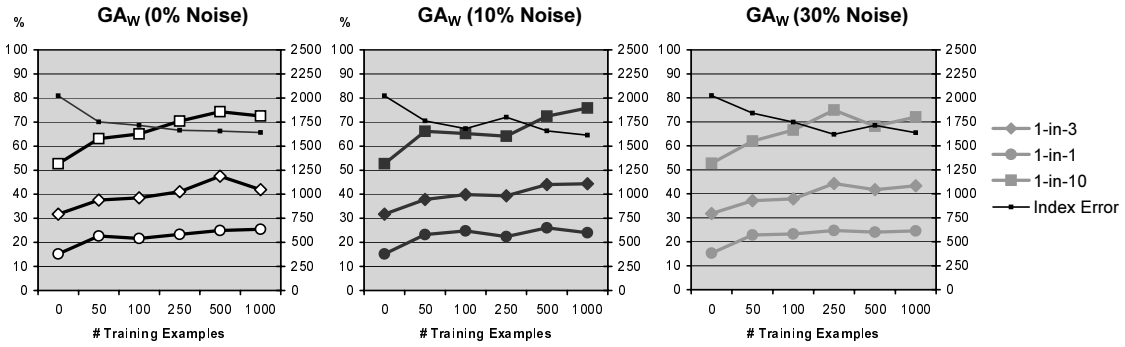


Figure 8.13.: Learning Weights Only: Summary

Since we have applied the Seq_{W-LS} strategy for combining weight and local similarity measure learning, we were able to extract the results of pure weight learning from the obtained data, too. Therefore, we included an additional evaluation step which measured the achieved retrieval quality after optimising weights only. Figure 8.13 and Figure 8.14 show the corresponding results in the same manner as previously described for the final outcome of the learning algorithm .

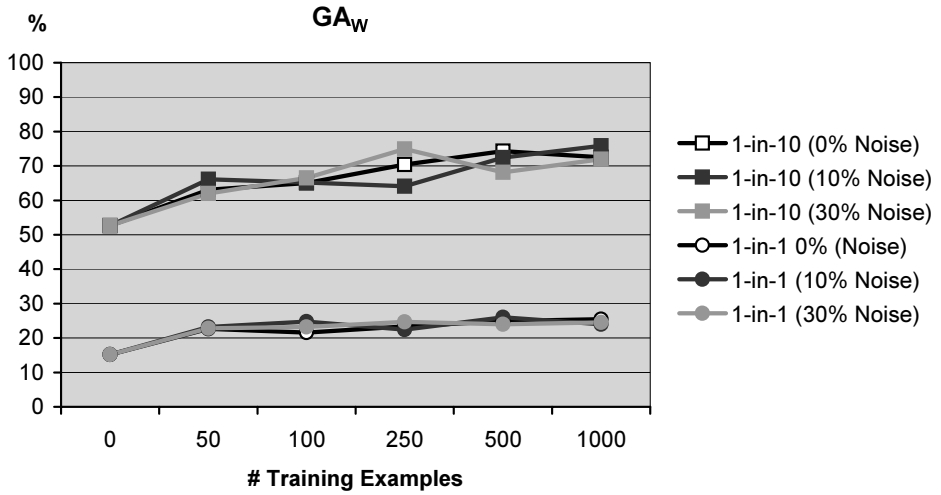


Figure 8.14.: Learning Weights Only: Impact of Noise

	no learning	weights only			weights and local measures		
criterion	Sim_U	Sim_{50}	Sim_{100}	Sim_{200}	Sim_{50}	Sim_{100}	Sim_{200}
1-in-1	30%	40%	44%	44%	34-43%	39-47%	45-47%
1-in-3	55%	67%	71%	72%	68-75%	73-77%	77-80%

Table 8.4.: Summary of Retrieval Improvements achieved in Scenario 1

8.4. Discussion of Evaluation Results

In this section we discuss the experimental results presented in Section 8.2.4 and 8.3.4. First we summarise the most important observations that show the fundamental capability of our learning approach. After that, we discuss some interesting aspects in more detail.

8.4.1. Summary

When applying our learning technique, the first question is, of course, which improvements in retrieval quality can be achieved.

Basic Conclusions for Scenario 1

For our first evaluation scenario (cf. Section 8.2), Table 8.4 gives an overview of the average retrieval quality measured before learning, after learning attribute weights only, and after learning both, local similarity measures and weights. In the last case, we have also considered the variance between the different versions of our genetic algorithm. Moreover, we have shown the corresponding values for three different amounts of exploited training examples. Here, Sim_i denotes the similarity measure learned when using i training examples. All values shown here were obtained by employing the genetic algorithm. A discussion of the differences between the gradient descent and the genetic algorithm is given in Section 8.4.2

These values indicate clearly that our learning algorithms were able to find similarity measures leading to significant better retrieval results compared to the initial similarity measure used. Here, the special characteristics of the addressed application scenario must be pointed out. Since this application scenario allows to generate training data automatically, the achieved improvements require computation time only. Additional manual effort for acquiring training data is not required, and therefore, the measured improvements might be obtained just by "a mouse click". The only requirement is the existence of an initial similarity measure Sim_U which approximates the utility of a given case (here, a PC) w.r.t. the given user demands without considering the adaptability of cases. However, this requirement is not really a problem, because such a similarity measure is required in any case. Our approach

	no learning	weights only			weights & local measures		
criterion	Sim_I	Sim_{100}	Sim_{500}	Sim_{1000}	Sim_{100}	Sim_{500}	Sim_{1000}
0% noise							
1-in-1	18%	22%	25%	25%	18%	28%	34%
1-in-3	32%	39%	57%	42%	37%	53%	62%
1-in-10	53%	65%	74%	73%	60%	79%	87%
10% noise							
1-in-1	18%	25%	26%	24%	12%	29%	30%
1-in-3	32%	40%	44%	44%	27%	52%	56%
1-in-10	53%	65%	72%	76%	53%	83%	84%
30% noise							
1-in-1	18%	23%	24%	25%	8%	25%	28%
1-in-3	32%	38%	42%	43%	18%	48%	54%
1-in-10	53%	67%	68%	72%	36%	76%	82%

Table 8.5.: Summary of Retrieval Improvements Achieved in Scenario 2

then allows a self-optimisation of this initial similarity measure by the CBR system itself, in order to consider the adaptability of cases during retrieval, too.

When judging the presented improvements in retrieval quality, one must also be aware that optimal retrieval results probably cannot be achieved with the presumed vocabulary and similarity measure representation of our experimental domain. To estimate the adaptability of cases exactly, also dependencies between attributes are important. For example, an alternative hard disk cannot be installed if someone asks for a laptop. So, the underlying hypotheses space does not include the optimal similarity measure, and hence, it is impossible to reach the 100% values of the introduced quality measures at all. Therefore, an increase of the percentage of correct retrievals from 55% up to 80%, when demanding that the most useful case should appear during the 3 most similar cases, represents a quite good result. Concerning both quality measures relative improvements of about 50% can be recognised. Thus, finally one may conclude that the application of our learning framework brings obvious benefits in the assumed application scenario (cf. Section 6.5).

Basic Conclusions for Scenario 2

Similar to scenario 1, also in the experimental scenario 2 clear improvements in retrieval quality by applying our learning algorithms can be recognised. Table 8.5 again gives an overview of the average retrieval quality measured before learning, after weight learning, and after learning weights and local similarity measures.

Here, relative improvements in all quality measures about 65% up to nearly 100%

have been achieved. For example, when dealing with 1000 noise-free training examples, the values for the 1-in-3 soundness criterion could be increased from 32% to 62%, i.e. the value could be nearly doubled.

Due to the differences to the first experimental scenario, here, the reported improvements in retrieval quality must be seen from a different point of view. At least for the noise-free training data one might expect better retrieval results because the optimal similarity measure should lay in the considered hypothesis space. This optimal similarity measure is obviously the similarity measure Sim_U used to simulate customer feedback. Hence, one might be surprised that the achieved quality values are still far away from the optimum.

This observation can be explained in two ways. Firstly, one must consider that local similarity functions are only represented with a limited set of sampling points during learning. Hence, the optimal similarity measure can only be approximated but cannot be represented exactly. Secondly, when comparing the charts of Figure 8.6 and Figure 8.11, one can observe an important difference. While the learning curves for the first experimental scenario show a clear convergence for the maximal number of used training examples, such a convergence is missing in Figure 8.11. This might be an evidence that the investigated number of training examples was not sufficient to obtain the results that are feasible, in principle.

Although 1000 training examples seem to be an unrealistic number for practical usage on the first look, it must be noticed that this experimental scenario, of course, relies on much more training examples compared to scenario 1. Since here single training examples contain utility information about 3 cases only, significantly more training examples are required to obtain satisfactory learning results. Nevertheless, in our point of view this requirement should not prevent the application of our approach in this application scenario. At least when learning the preferences of a certain customer class and not of a single customer, it should be feasible to acquire the required amount of training examples. Nowadays, successful eCommerce applications (e.g., www.amazon.de) are used very frequently by many customers so that enough utility feedback may be obtained.

8.4.2. Gradient Descent vs. Genetic Algorithm

In our first experimental scenario we have compared the gradient descent algorithm with the version of the genetic algorithm that learns attribute weights only. The corresponding results presented in Section 8.2.4 indicate that both algorithms are able to achieve approximately the same learning results with respect to the 1-in-3 quality criterion. Surprisingly, regarding the two other quality measures slight differences between both algorithms can be noticed. On the one hand, the gradient descent algorithm was able to achieve better results for the index error, i.e. a criterion that measures the quality of the entire retrieval result. On the other hand, the genetic algorithm performed better with respect to the 1-in-1 quality criterion

which focuses on the retrieval of the most useful case only. Up to now we have no idea of how these differences can be explained. Here, further experiments are necessary to investigate this phenomenon in detail.

A crucial difference between both algorithms can be recognised in Figures 8.3 and 8.5, which illustrate the variance of the achieved results over the 10 repetitions of the experiments. Here, it becomes clear that the gradient descent algorithm is much more stable concerning the achieved improvements in retrieval quality compared to the genetic algorithm. When using the gradient descent algorithm, the exploitation of more training examples leads to better learning results continuously. In contrast, the genetic algorithm shows significantly more variations. This property becomes a crucial problem when dealing with few training examples only. Here, it can be observed that 5 training examples were always sufficient to achieve improvements in the retrieval quality when applying the gradient descent algorithm (see the worst values measured for the 1-in-3 criterion shown in Figure 8.3). In contrast, here the genetic algorithm required 35 (for $\theta = 10$) or over 140 (for $\theta = 0$) training examples in the worst case.

This undesired behaviour of the genetic algorithm might be explained by the particular search strategy used. Naturally, the genetic algorithm relies on much more random effects compared with the gradient descent algorithm, which become also obvious in the results finally achieved. This effect might be decreased by allowing the genetic algorithm to process more generations.

Another crucial difference between both algorithms is the performance with respect to computation time. Since the gradient descent search is much more goal directed compared to the search strategy genetic algorithms are based on, the entire learning process is much faster. Figure 8.9 illustrates that the gradient descent algorithm is about 5-6 times faster compared with the genetic algorithm when learning weights only.

8.4.3. Influence of θ

In Definitions 4.5 and 5.3 we have introduced a so-called *error-position weight* θ. The basic idea of this parameter was to "penalise" retrieval failures that concern the more useful cases additionally, since we are mostly interested in a correct retrieval concerning these cases.

In order to validate whether this error-position weight helps to influence the learning process so that it focuses on finding a similarity measure that retrieves the most useful cases correctly, in our first experimental scenario we have tested two different values for θ. In fact, the corresponding results shown in Figures 8.3 and 8.5 indicate that a value of $\theta = 10$ significantly increased the achieved values for the quality criteria that measure the success in retrieving the most similar case. As expected, the achieved values for the criterion that measures the quality of the entire retrieval result, i.e. the index error which also considers retrieval failures concerning very

useless cases, simultaneously decreased slightly.

Hence, the error-position weight seems to be appropriate to learn similarity measures that in particular ensure correct retrieval of the most useful cases. In future experiments it has to be investigated which concrete values of θ lead to optimal results.

8.4.4. Required Amount of Training Data

One of the most important aspects when applying learning procedures is, of course, the amount of training data required to obtain reasonable learning results. Basically, we can distinguish two important thresholds concerning the number of training examples required:

te_{min}: When using only few training examples, learning procedures often tend to *overfitt* the presented examples. The learning algorithms may find a similarity measure that works quite well for the training examples presented. However, for other queries and corresponding cases, this similarity measure might led to very poor retrieval results. To ensure learning of similarity measures that are general enough to improve retrieval results for arbitrary queries and cases, a minimal amount of training data is required. Hence, we investigate the number of training examples te_{min} that are at least necessary in order to ensure that the learning procedure brings a benefit.

te_{suf}: As typically for learning curves, they become more flat for increasing number of training examples. This means, after using a certain amount of training data, more training examples do not improve the results significantly any more. Therefore, we also investigate the number of training examples te_{suf} that are sufficient to achieve approximately the optimum of the learning process.

Table 8.6 gives an overview of the average values of te_{min} and te_{suf} for our two experimental domains that can be determined by analysing the presented learning curves (cf. Sections 8.2.4 and 8.3.4). Here, we have not distinguished between different versions of our learning algorithms, but have selected the values achieved with the most accurate version with respect to the particular scenario. For the second experimental scenario te_{suf} could not be determined appropriately, since the maximally number of training examples used was not sufficient.

The presented values show clearly, that learning of attribute weights requires much less training examples than learning of local similarity measures. This is not surprising since local similarity measures represent a much larger search space compared with a weight vector. Due to the various number of possibilities for tuning local similarity measures by changing table entries and sampling points, here, the risk of overfitting the training data is much higher as when learning weights only.

	weights only		weights and local measures	
scenario	te_{min}	te_{suf}	te_{min}	te_{suf}
1	<5	70-100	∼20	140-200
2				
0% noise	<50	∼500	50-100	?
10% noise	<50	∼500	∼100	∼1000(?)
30% noise	<50	∼250	100-250	∼1000(?)

Table 8.6.: Required Number of Training Examples

8.4.5. Impact of Noisy Training Data

The impact of noise investigated in our second experimental scenario was very surprising. On the one hand, noise influences the learning procedure clearly when learning both, attribute weights and local similarity measures. However, when looking at Figure 8.12 it can be seen, that the amounts of noise investigated in our experiments only seem to impede the learning process significantly when exploiting less than 250-500 training examples. If the algorithm is provided with 500 or more examples, noise seems to have only a minor impact on the results achieved.

On the other hand, when analysing the results obtained for pure weight learning, the situation is different. Here, the amount of noise investigated in our experiments, seems to have no clear impact on the learning procedure at all (cf. Figure 8.14). In order to be able to explain this surprising observation, more experiments, for example, with a higher percentage of noisy training examples are necessary.

8.4.6. Sequential vs. Parallel Processing

Finally, we want to compare the differences between the strategies applied to combine learning of attribute weights and learning of local similarity measures. In our first experimental scenario we have evaluated four different strategies, namely the two sequential approaches Seq_{W-LS} and Seq_{LS-W}, and the two approaches towards parallel processing Par_{pseudo} and $Par_{composed}$.

When comparing the results achieved with these approaches, major differences cannot be recognised. All strategies achieve approximately the same results when exploiting the maximal number of available training examples. However, several minor differences can be observed between the sequential and the parallel strategies. For example, it seems that the parallel strategies lead to steeper learning curves. Therefore slightly less training examples are required in order to avoid a negative outcome of the learning procedure caused by overfitting the training data (cf. Figure 8.6). Hence, the te_{min} values of the parallel strategies (cf. Section 8.4.4) are a little bit smaller (10-20) compared to the te_{min} values of the sequential strategies

(20-35).

Although, both approaches finally reach approximately the same values for the introduced quality measures, the sequential approaches show no clear convergence, even for the maximal number of training examples investigated. This might be an evidence that further improvements in the retrieval quality might be possible here, if more training examples are presented. In contrast, the parallel approaches seem to converge to an optimum when being provided with about 140 and more training examples.

Another minor difference between the parallel and the sequential approaches can be observed when analysing the variance of the results achieved during the 5 repetitions of the experiments (cf. Figure 8.7). It seems that the parallel approaches produce more stable results compared to the sequential approaches. At least for training data sets consisting of more than 70-100 training examples, the variance of the values of the 1-in-3 quality measure are significantly lower when applying the parallel strategies. However, for smaller training data sets such a difference cannot be noticed.

The computation times of the different learning strategies when exploiting all 200 training examples (cf. Figure 8.9) show, that the parallel approaches are more expensive (with a factor about 1.2-2). However, the comparison and interpretation of these differences is difficult, since the strategy $Par_{composed}$ requires a totally different number of generations due to the properties of this strategy. Although it requires much more generations as the three other strategies, the required computation time is quite similar. This can be explained by the similar number of needed procedures for evaluating individuals which represents the most costly steep during learning (each evaluation requires a case retrieval!). During the sequential approaches 20 individuals (population size) have to be evaluated 400·12 times[6]. In contrast, the real parallel approach $Par_{composed}$ requires 4400·1 evaluation procedures since all 12 elements of the similarity measure are treated simultaneously.

[6] number of generations · number of elements of the similarity measure to be optimised separately (11 local measures + 1 weight vector)

Part IV.

Related Work and Conclusions

9. Related Work

Although learning of similarity measures was not in focus of CBR research yet, numerous publications that can be seen as first steps into this direction have been published in the last years. In this chapter a short overview on existing work that is related to the topic of learning similarity measures is given.

9.1. Classic Attribute Weight Learning

The first approaches to improve the similarity assessment in CBR by using machine learning techniques all focused on the definition of accurate attribute weights. The major reason for this restriction is the structure of the similarity measures traditionally used in CBR systems (cf. Section 3.3.1) and foregoing Nearest-Neighbour classifiers (cf. Section 2.3.1). Here, attribute weights represent the only possibility to encode meaningful domain knowledge into the similarity measure.

So, the most early publications that deal with learning of attribute weights[1], can be found in the instance-based learning and classification community. An excellent overview on approaches to attribute weighting is given by Wettschereck and Aha (1995). The authors also present a framework for categorising such approaches based on the following five criteria:

1. **Feedback:** This criterion distinguishes between so-called *feedback* and *ignorant* approaches[2]. While weight modifications of feedback methods are based on feedback from the used classification algorithm, ignorant methods only perform a statistical analysis of the available case data in order to identify correlations between the distribution of attribute values and corresponding classes.

2. **Weight Space:** The weight space represents the values allowed to be assigned to attribute weights. For example, it is possible to distinguish between binary weights (i.e., a weight can only be 0 or 1) and more fine-grained weight models. Which weight models are more accurate is a controversially discussed issue. For example, experiments reported by Kohavi et al. (1997) indicate that very fine-grained models often lead to overfitting the training data during learning.

[1] Often also denoted as *feature weights*.
[2] Other researchers also denote them as *wrapper* and *filter* approaches.

3. **Representation:** Concerning the underlying case representation, some weighting approaches assume a binary attribute-value based representation. This means, only attributes with a binary value type are allowed. To apply these approaches on more flexible case representations with arbitrary symbolic and numeric attributes, a pre-processing step in order to *binarise* the attributes is necessary.

4. **Generality:** This criterion distinguishes between global weight models, where one single weight vector is used for the entire case space and more local models where weights may differ for different regions of the case space (cf. Section 3.3.4).

5. **Knowledge:** This final criterion has been introduced to distinguish between methods that exploit domain-specific knowledge when determining accurate weights and more knowledge-poor methods.

Concerning the weighting approaches that receive feedback from the used classification algorithm, the authors further distinguish between *incremental hill-climbers* and *continuous optimizers*.

9.1.1. Incremental Hill-Climbers

These types of learning algorithms modify weights incrementally after each training example is presented in order to correct occurring classification failures and to improve the classification accuracy in the future. Here, it is possible to distinguish *failure-driven* and *success-driven* strategies. While failure-driven strategies only modify weights as a result of a retrieval failure, success-driven strategies seek to update weights as a result of a retrieval success. Bonzano et al. (1997) enumerate four corresponding *learning policies*:

1. There has been a retrieval *success* and the weights of *matching* attributes are *incremented*.

2. There has been a retrieval *success* and the weights of *mismatching* attributes are *decremented*.

3. There has been a retrieval *failure* and the weights of *mismatching* attributes are *incremented*.

4. There has been a retrieval *failure* and the weights of *matching* attributes are *decremented*.

In principle, a learning algorithm might apply arbitrary combinations of these policies, however, the most common method is to employ all four strategies. Besides

pure failure-driven or pure success-driven strategies, respectively, sometimes also very specific combinations might be used. For example, Muñoz-Avila and Hüllen (1996) describe an algorithm that only modifies mismatching attributes, i.e. it applies only the policies 2 and 3.

The basic idea of incremental hill-climbers is to *push* wrongly retrieved cases away from the current target case T, and to *pull* useful cases closer to T (Bonzano et al., 1997; Zhang and Yang, 1999). For determining the magnitude of these pushing and pulling operations, different methods have been developed. Salzberg (1991) uses a quite simple method in EACH, where all four learning policies are applied with a fixed in/decrement Δ_f. The author reported that the selection of an accurate Δ_f depends on the domain and so has to be determined experimentally. The IB4 algorithm described by Aha (1991); Wettschereck and Aha (1995) and the algorithm RELIEF (Kira and Rendell, 1992; Kononenko, 1994) employ more sophisticated update rules that also consider the difference of mismatching feature values. Other strategies are, for example, decaying update rules that decrease the weight changes over time in order to guarantee convergence of the learning procedure (Muñoz-Avila and Hüllen, 1996; Bonzano et al., 1997).

As typical for incremental learning techniques, all these incremental hill-climbing approaches are sensitive to the presentation ordering of the training examples.

9.1.2. Continuous Optimisers

In contrast to the previously described incremental methods, continuous optimisers process a selected number of training examples at once during a batch process, instead to update weights on the basis of a single retrieval result. Therefore, a *leave-one-out cross-validation (LOOVC)* with the available training examples is typically performed in order to measure the average classification accuracy obtained by using a particular weight vector. A continuous optimiser than tries to modify the currently used weight vector in order to improve the measured average accuracy.

To guide this modification process different strategies can be applied. We have already discussed this issue in Section 4.3.4. On the one hand, modifications might be driven by *chance* or they might be driven by some *knowledge about expected changes in the classification accuracy*. In both cases, the classification accuracy is modelled as an *error function* of the weights and the available training data. On the one hand, if modifications are based on random operations, this function is only used to evaluate the outcome of a modification. On the other hand, such a function can also provide some knowledge to select promising modifications.

Typically, continuous optimisers represent more robust learning algorithms than incremental hill-climbers because they are not sensitive to the presentation ordering of training examples.

Lowe (1993) pointed out that the accuracy of a k-Nearest-Neighbour classifier strongly depends on the quality of the used distance metric when dealing with case

models that include irrelevant attributes. Therefore, the author presents a learning strategy—called *variable-kernel similarity metric (VSM)*—that is inspired by learning techniques of neural networks. The particular technique used is a *conjugate gradient* algorithm that exploits knowledge about the gradient of the error function. Therefore, the LOOCV error E is defined as the sum over all training examples t, and all possible class labels i, where s_{ti} is the known probability that the training example t belongs to class i (i.e., usually it holds $s_{ti} \in \{0, 1\}$), and p_{ti} is the corresponding probability determined by the classification algorithm:

$$E = \sum_t \sum_i (s_{ti} - p_{ti})^2$$

While the algorithm presented by Lowe (1993) also optimises a parameter of the classification algorithm, Wettschereck and Aha (1995) introduced a variant of VSM, namely $k - \mathrm{NN}_{VSM}$ that only focuses on attribute weights. However, it employs a similar error function based on LOOCV training error.

Other continuous optimisers that do not exploit knowledge about the error function rely on random search strategies. For example, Kelly and Davis (1991) have presented an approach that applies a genetic algorithm or a evolution program, respectively[3], in order to learn accurate attribute weights (cf. also Section 5.3). Therefore, the authors introduce five genetic operators used to create new individuals. Since they also address classification domains, their GA-WKNN algorithm employs a fitness function that is based on LOOCV again. A similar algorithm is presented by Demiröz and Güvenir (1996). However, here the search process is exclusively based on a single, very specific crossover operator—called *continuous uniform crossover*—that avoids the need of a normalisation of the weights after applying crossover. Experiments with another random search strategy called *random mutation hill climbing* are reported by Skalak (1994). This algorithm might be characterised as a genetic algorithm that operates with one individual only to which only random mutation is applied.

The approach presented by Jarmulak et al. (2000a,b) also applies a genetic algorithm in order to optimise CBR retrieval in the tablet formulation domain. Here, the underlying CBR system performs a two-step retrieval consisting of a case selection phase realised by a decision-tree, and the actual similarity computation phase for selected cases. Therefore, the GA does not exclusively optimise attribute weights used to compute similarities, but also other parameters required for the decision tree generation and the parameter k of the finally applied k-Nearest-Neighbour voting. The fitness function used to evaluate generated individuals is based on a variant of the LOOCV, namely *leave-n-out crossvalidation* on existing case data. Because of the usage of a binary weight model, the entire approach is rather a feature selection task, than an actual weight learning task. Nevertheless, the authors report about

[3]Here, attribute weights are not represented as bit strings but as real-valued numbers.

very promising results of an experimental evaluation in their tablet formulation domain. This evaluation also includes an interesting investigation of effects concerning overfitting the training data.

9.2. Extensions of Classic Weight Learning Algorithms

In the previous section a rough overview on classic attribute weight learning algorithms was given. In this section some further approaches to learning attribute weights that introduce some very specific aspects are discussed.

9.2.1. Learning of Asymmetric Similarity Metrics

Ricci and Avesani (1995) have introduced a novel approach to compute similarities called *asymmetric anisotropic similarity metric (AASM)*. On the one hand, this similarity metric uses case specific weights (cf. Section 3.3.4). On the other hand, it also introduces two separate weights for numeric attributes in order to enable the definition of asymmetric similarity measures (cf. Definition 3.7). The first weight is used, if the query value of the respective attribute is larger than the corresponding case value. The second weight is used in the remaining situations, if the query value is smaller than the case value.

Moreover, the authors present an LOOCV-based incremental hill-climbing algorithm in order to learn the different weights of AASM. Therefore, they apply all four variants of the failure-driven and success-driven learning policies already mentioned in Section 9.1.1. For updating the weights they introduce sophisticated update magnitudes that are based on the difference of numeric attributes as well as on so-called *reinforcement* and *punishment rates* that are represented by two parameters $\alpha, \beta \in [0, 1]$. To show the power of AASM and the corresponding learning algorithm the authors also report about results of an experimental evaluation on four popular data sets. These results indicate that the use of AASM may improve the accuracy of Nearest-Neighbour classifiers significantly.

9.2.2. Considering Decision Cost

All learning approaches discussed so far aim to increase the pure classification accuracy of a classification system. Motivated by two real-world applications, Wilke and Bergmann (1996) have presented an alternative learning goal. Instead to consider only the pure correctness of classifications, they argue that also *decision costs* play a major role in several domains. For example, in a credit scoring domain, different classification failures are coupled with very different costs for a bank. If the CBR systems classifies a creditworthy customer as not creditworthy, the bank looses only

the possible income. However, if a not creditworthy customer is classified as creditworthy, the bank probably looses the whole credit sum, i.e. a considerably higher amount.

To consider such decision costs during learning of attribute weights, the authors present the following specific error function to be used by a continuous optimiser:

$$E_{cost} = \sum_{q \in CB} \sum_{t \in T} sgn(C_{t_q,t}) \cdot C_{t_q,t}^2 \cdot p_{q,t}^2$$

Here, $p_{q,t}$ denotes the probability that the query q is classified as class t (T denotes the set of all occurring classes), and $C_{t_q,t}$ denotes the corresponding cost. If q is classified correctly the value $C_{t_q,t}$ represents rather a profit than a cost, and therefore $C_{t_q,t}$ will be negative. Since this function is partially differentiable with respect to attribute weights—they occur in the formula for computing the probability $p_{q,t}$—it can be minimised by employing a conjugate gradient algorithm. The authors also present an experimental evaluation for the credit scoring domain that indicates that this algorithm allows to improve the decision cost of a CBR system significantly, compared to a learning algorithm that focuses on classification accuracy only.

9.2.3. Exploiting a Declarative Domain Model

Fox and Leake (1995a,b) have presented an incremental approach towards the selection of attributes—this situation can be interpreted as a binary weight space—by using introspective reasoning. Since the addressed application domain is route planning (cf. Section 2.3.6), here, the training data is not obtained by exploiting pre-classified data. The presented planning system ROBBIE contains a declarative model of the ideal reasoning behaviour of the system that provides the feedback whether a retrieved case is suitable or not. If it becomes clear that an inaccurate case was retrieved, ROBBIE's introspective reasoner identifies attributes which would lead to retrieval of more adaptable cases, and refines the indexing criteria in order to include the needed features to avoid similar failures in the future.

9.2.4. Exploiting User Feedback

An interesting variant of an incremental hill-climber is presented by Zhang and Yang (1999). Here, a two-layer network architecture composed from attribute-value pairs, attribute weights and cases is used to model a case base. Due to this structure it is then possible to resemble the learning process of a back-propagation neural network (Mitchell, 1997). A special feature of the applied update policy is that it exploits feedback of the user about the desired similarity of cases. The update magnitude is then determined based on the difference between the actually computed similarity and the desired similarity. This approach is a novelty compared to the

other traditional learning techniques where the required feedback is obtained from pre-classified data. Therefore, the approach presented by Zhang and Yang (1999) is not restricted to classification domains only. It rather might be applied in different application scenarios like Knowledge Management (cf. Section 2.3.3) or eCommerce (cf. Section 2.3.4).

9.3. Learning of User Preferences

Recently, the development of *user adaptive systems* becomes more and more popular (Göker and Thompson, 2000). In CBR, user adaptive functionality can be realised by considering particular user preferences during case retrieval (cf. also Section 6.4). Branting (1999) has already pointed out the importance to consider individual user preferences. Here, the author presents an approach to learning preference criteria through active exploration when using an instance-based ranking algorithm (1ARC). However, up to now less publications concerning learning of user preferences in the CBR context have been published.

An approach in particular addressing learning customer preferences in case-based product recommendation systems is presented by Branting (2001). Here, an incremental hill-climber is used to learn the mean customer preferences of some customer population by exploiting *return-set selections* of the individual customers. A return-set selection represents a case selected by a customer among a set of retrieved cases considered to be most similar to the customer's requirement specification. In order to improve the retrieval quality, the incremental hill-climber then adjusts the weights based on attribute differences between the query and the cases that wrongly have been considered to be more similar than the return-set selection. This means, the algorithm realises a failure-driven update policy (cf. Section 9.1.1). The actual magnitude for updating the weights, however, does not consider attribute differences, but only depends on a fixed learning rate to be defined prior learning. In order to demonstrate the relevance for real-world applications, the author also presents an excellent experimental evaluation where several important influences on the learning procedure are investigated. For example, it is assumed that return-set selections are based on individual preferences of the particular customer that are normally distributed around some mean preferences with some standard deviation. Changes in this standard deviation as well as influences of the size of the return-set are investigated in detail. The results of the experiments indicate that the learning approach brings clear benefits, in particular if customers tend to assign a minority of attributes high importance and the majority of attributes low importance.

An approach that focuses on personalised recommendation service and learning of user preferences represented by personalised similarity measures is described by Coyle and Cunningham (2003). Here, the authors discuss how existing learning algorithms like introduced by Branting (2001); Stahl (2001, 2002a); Stahl and Gabel

(2003) might be employed to exploit re-ranking information in a case-based personal travel assistant. In this domain several requirements motivate learning strategies in order to optimise case retrieval. On the one hand, each retrieval is connected with a considerably network delay, so that repeated retrievals will probably not be tolerated by the users. On the other hand, the application is targeted at mobile web users, e.g. WAP enabled phones and mobile-PDAs users. Due to the constrained screen-size of these devices, only small return-sets can be displayed. Thus, the utility of the limited number of retrieved and displayed cases has to be maximised.

The application of a genetic algorithm for learning user preferences in case-based software reuse is described by Gomes and Bento (2000). The authors argue that the utility of software components to be reused is also influenced by individual preferences of the users, for example, the programming style of a particular programmer or designer. In order to estimate the utility of software components in their CREATOR II system, they present a sophisticated similarity measure that can be parameterised using several special weights. To optimise these weights a genetic algorithm is applied where the fitness of particular weight settings is determined by an evaluation function which compares achieved retrieval results with case re-ranking information provided by the user.

9.4. Learning of User Context in Information Retrieval

As already discussed in Section 3.1.1, Information Retrieval is another research field that is interested in retrieving useful documents. Here, *relevance feedback* acquired from the system users is used to improve the retrieval capabilities of the system. Usually relevance feedback only signs retrieved documents either as *relevant* or *not relevant*. A finer estimation of the utility, e.g. by comparing different items (cf. Section 4.2.2), is mostly not considered. The major difference to learning approaches applied in CBR is that the IR systems usually do not learn a general representation (comparable with a similarity measure). They rather try to extend given queries with additional key terms in order to obtain a more specific query. This approach in called *query expansion*.

The core assumption is that a user has some *information need* that is associated with some *context*. To get answers for this information need, a user then repeatedly submits queries to an IR system which hopefully returns relevant documents. However, the majority of users tend to provide queries that contain only few search terms. Therefore, such queries are not sufficient to capture the current context of the user, and thus, retrieval results are often not satisfying.

In order to learn the *actual user context* from subsequent queries, Göker (1999) has presented a so-called *Context Learner*. The basic idea of this approach is to

extract key terms from documents that the user has judged as relevant, or partially relevant. These key terms then represent the actual user context and can be used to extend new queries in order to retrieve more relevant documents. In principle, the actual user context, in our terminology, can be characterised as an individual and temporarily valid utility function.

Another approach, called *collaborative information retrieval*, that exploits relevance feedback in order to improve retrieval performance is presented by Hust et al. (1993). Here, feedback is used to change the weights of query- and document terms, used to compute document similarities in the common vector space model of IR. Therefore, the authors present a mathematical model that is mainly based on matrices operations.

10. Conclusions

This last chapter discusses the results presented in this thesis and points out some open questions that could not be answered in the scope of this work. These open questions can be seen as interesting issues for future research.

10.1. Objectives and Achieved Results

In this thesis we have presented a novel approach to learning *knowledge-intensive similarity measures* (cf. Section 3.4.1) in Case-Based Reasoning. In contrast to traditional knowledge-poor similarity measures, this interpretation of similarity leads to the well-known knowledge acquisition problem in AI. Thus, the main objective of this work was the development of a framework that helps to reduce the knowledge acquisition effort when employing knowledge-intensive similarity measures. Our framework is founded on machine learning techniques that allow to extract the necessary domain knowledge from training data that may often be acquired more easily than the actual knowledge itself. Employing learning techniques leads to several major advantages:

- The definition of similarity measures does not require the skill to deal with complex mathematical representation formalisms anymore. Instead, only the natural knowledge chunks of the domain, namely cases and queries, have to be handled by human experts.

- A deep analysis of the domain and, in particular, of the influences on similarity is not mandatory anymore. High level knowledge about the cases' utility for given queries is sufficient to obtain accurate similarity measures.

- Additional similarity knowledge may be acquired during the daily usage of the CBR system. Knowledge that is not available at all during the development phase of the system can be employed in order to improve the competence of the system continuously (cf. application scenarios described in Section 6.3 and 6.4).

- Similarity knowledge may be extracted from existing knowledge resources automatically (cf. Section 6.5 and 6.6).

- The maintenance of similarity knowledge can be performed during usage of the system automatically (cf. Section 6.7).

The idea to learn domain knowledge required for defining knowledge-intensive similarity measures seems to be straight forward because CBR systems are already popular for their capability to improve their competence by learning. However, most approaches to learning in CBR research focus on situation-specific case knowledge, and not on general knowledge as required by similarity measures. In Section 4.1.1, we have described that also learning of general knowledge, and particularly similarity knowledge, can be motivated from the cognitive point of a view as well as from the functional point of view.

10.1.1. Summary

The main idea of our framework presented in Chapter 4 is to acquire similarity knowledge on a different level of detail compared to the common method to defining similarity measures discussed in Section 3.5. Instead of analysing all influences on the utility of cases in detail, and modelling them with help of specific parameters of the similarity measure, our approach only relies on the estimation of the utility of cases on a high level for some given queries. We called this kind of similarity knowledge *utility feedback*. This approach can be described as a top-down procedure since only the desired outcome of the similarity assessment has to be defined. The definition of a particular similarity measure to achieve this outcome is then left to the employed learning algorithm. The advantage of this approach is that it allows to hide the formal and difficult to handle representation of similarity measures and that a detailed analysis of the domain is not necessarily required. Instead, our approach relies on some *similarity teacher*, e.g. a domain expert, who is able to estimate the utility of cases for particular queries in a relative manner. This means, the utility of a case has not to be expressed absolutely, but only relatively to other cases.

In order to exploit utility feedback for the definition of similarity measures, we introduced the concept of the *retrieval error* (cf. Section 4.3.3). This error computed by a particular *error function* (e.g. the *index error* introduced in Definition 4.6) can be seen as a measure for estimating the quality of a given similarity measure. Therefore, the retrieval result, i.e. a case ranking determined by the similarity measure has to be compared to given utility feedback of some similarity teacher. The retrieval error is then a measure for the deviation of the computed case ranking from the ideal ranking defined by the teacher.

Besides estimating the quality of similarity measures, such an error function also represents a powerful instrument to realise learning functionality. It can be seen as an objective function for evaluating the quality of arbitrary similarity measures and hence, it can be used to guide a search or optimisation process (cf. Section 4.3.4), respectively, within the space of all representable measures. How this search process

Algorithm	Pro	Contra
GDA	very fast, quite stable	only suited for learning weights, problems with particular case data
GA	suited for entire similarity measure, flexible applicability	poor computational performance, less stable

Table 10.1.: Comparison of Learning Algorithms

is then actually realised depends on the employed learning algorithm. In Chapter 5 we have presented two respective learning algorithms, namely a *gradient descent algorithm (GDA)* and a *genetic algorithm (GA)*. While the gradient descent algorithm is only suited to learn attribute weights, the genetic algorithm, in principle, allows to learn the complete similarity measure representation, i.e. the attribute weights, the local similarity measures and the aggregation function. However, in this work we have focused on learning weights and local measures because they usually contain the major portions of similarity knowledge.

Table 10.1 shows an overview of the characteristics of the two learning algorithms presented in this work. On the one hand, the GDA is a very fast algorithm because it exploits knowledge about the shape of the error function. Further, in our experiments it has turned out to be a more stable algorithm compared to the GA since the variations in the achieved results were smaller. However, the GDA has also some crucial drawbacks. Besides that it is only suited for learning weights, in certain situations it also delivers bad results due to the characteristics of the required error function (cf. Section 5.2.5). Thus, it can only be applied, if the distribution of attribute values within the case data does not lead to undesired minima in the error function. However, this drawback of the GDA might be compensated if it would be possible to define an alternative error function which avoids such undesired minima. Whether this is principally possible or not, is still an open question.

In contrast, the genetic algorithm can be applied very flexibly without such problems and in our experiments it achieved similar improvements in retrieval quality as the GDA. Further, it can be employed to learn the entire similarity measure, and not only the attribute weights. However, the GA showed to be the less stable learning algorithm resulting in more variations in the learning results. Moreover, compared to the GDA, the requirements with respect to computation time are much higher when applying the GA. But this is not a crucial problem since exclusively learning of weights can be performed quite fast anyway.

Of course, similar to other learning approaches, the acquisition of enough training data with reasonable quality is one of the most crucial problems when applying our framework. Here, training data is represented by utility feedback and has to be

provided by some similarity teacher. To illustrate how this similarity teacher might be realised in practice, in Chapter 6 we have described several different scenarios for applying our framework. We have shown that the similarity teacher has not necessarily to be represented by a human being. Depending on the particular scenario, the similarity teacher might also be realised in form of an automated procedure.

In order to be able to perform an experimental evaluation of our framework and the developed learning algorithms, a prototypical implementation was required. In Chapter 7, we have described such an implementation developed in form of an additional software module for the commercial CBR tool CBR-Works.

In order to demonstrate the capabilities of our framework, in Chapter 8 we have described an experimental evaluation for two of the application scenarios described in Chapter 6. The objective of this evaluation was mainly a demonstration of the principle capabilities and properties of the developed learning algorithms. For two exemplary application domains we evaluated how much training data is required to achieve reasonable learning results. Further, we investigated the impact of some important parameters to the learning algorithms employed. Basically, the results of the experimental evaluation led to the following important conclusions:

- Assumed that enough training data is available, our framework and algorithms are able to improve the retrieval quality of a CBR system significantly.
- Learning of weights is much easier than learning of local similarity measures due to the smaller search space. Because local similarity measures provide various tuning possibilities, here, the risk to overfitt the training data is much higher resulting in much more training examples required to obtain reasonable learning results. In contrast, optimisation of attribute weights led to clear improvements of the retrieval quality by exploiting few training examples. Nevertheless, if enough training examples are available, learning of local similarity measures will represent a powerful approach to improve the retrieval quality of a CBR system beyond learning of attribute weights.
- In order to focus on the correct retrieval of the most useful cases, the error-position weight θ used by our error functions seems to be a powerful instrument.
- In our experiments, the genetic algorithm showed to be quite robust against noisy training data.

10.1.2. Novelty of this Work

Compared with already existing work towards the learning of similarity measures described in Chapter 9, in essence, this thesis includes the following important new contributions.

Broad Applicability

One novelty of our learning framework is that it is not restricted on classification domains as the most existing approaches to learning similarity measures or attribute weights, respectively. As already described in Chapter 9, these approaches usually rely on pre-classified cases. Training data is then obtained by performing a leave-one-out-test with such available pre-classified case data. Hence, the application scenario of existing approaches can be characterised as a specialisation of our solution similarity scenario introduced in Section 6.6. However, here "similarity" between solutions is only an exact match between the correct classification and the classifications proposed by the most similar cases. This means, cases that propose the correct classification are considered to be maximal useful, while cases corresponding to an incorrect classification are considered to be maximal useless. A more detailed differentiation of the cases' utility is mostly not performed.

In contrast, our framework is based on the concept of utility feedback that allows to validate achieved retrieval results much more detailed. Further, it does not presume a certain procedure to obtain utility feedback, but it can be acquired in very different ways like described in Chapter 6. Thus, our framework is not restricted to classification domains. Moreover, it even addresses in particular application domains that have come into focus of CBR research and application just recently. For example, the eCommerce and knowledge management domains clearly differ from the traditional classification and diagnosis domains. Here, a distinction between a problem and a solution part of cases is mostly not obvious. So, the utility of a case cannot be derived from the degree of correctness of some solution. Instead the cases' utility is often determined by—maybe subjective—judgements of the users.

The idea to estimate the utility of cases very detailedly also distinguishes our framework form the most existing learning approaches developed in the research field of Information Retrieval (cf. Section 6.6). Here, relevance feedback usually only distinguishes between *relevant* and *irrelevant* documents.

The core functionality of CBR is, of course, a similarity-based retrieval and the quality of this retrieval, i.e. the capability to select useful cases, strongly influences the competence of the entire CBR system. Our approach tries to improve the retrieval quality by employing feedback about retrieval failures. But the structure of cases and the processing of retrieved cases is not a crucial aspect for the applicability of our framework. Hence, in summary, one strength of our framework is, in principle, that it might be applied in all application domains addressed by CBR. However, in which domains our framework is able to improve the definition of similarity measures significantly, can only be validated during practical usage.

Suited for Complex Similarity Measures

Another important novelty of our work is that it addresses complex similarity measures, namely those defined according the local-global principle. Here, particularly *local similarity measures* encode a lot of knowledge about the utility of cases. In contrast, existing approaches commonly are only suited to adjust attribute weights and local similarity is only interpreted as exact match or is modelled in form of a fixed, quite simple distance metric. However, such similarity measures where particular domain knowledge can only be encoded in form of attribute weights, are not suited to approximate the cases' utility sufficiently in many domains. Unfortunately, when introducing more sophisticated local similarity measures in form of distance-based similarity functions and similarity tables (cf. Section 3.3.3), the effort for modelling an adequate similarity measure increases dramatically.

Here, our framework in combination with the presented genetic learning algorithm represents a novel and powerful approach to facilitate this modelling procedure. Through the general flexibility of genetic algorithms or evolution programs, respectively, not only learning of attribute weights, but also learning of local similarity measures is possible. Therefore, in this thesis we have presented approaches to represent local similarity measures in a way that an evolution program is able to optimise them. Further, we have introduced accurate genetic operators for realising the respective evolutional search process.

By considering the entire representation of the assumed similarity measures, our approach allows to learn measures that represent a much better approximation of the unknown utility function compared to existing approaches. However, due to the significantly larger search space, this also requires more training data of reasonable quality. Whether this training data can be acquired in real world application scenarios has to be considered before applying our framework.

Methodology for Evaluating Similarity Measures

Today, a detailed validation of the similarity measure employed in a particular CBR system is still neglected. Mostly, only a leave-one-out-test is performed to evaluate the competence of a CBR system previously developed. However, a leave-one-out-test does not focus on the quality of the similarity measure employed, but measures the quality of the entire system that is also influenced, for example, by the quality of the available case data or the effectiveness of adaptation procedures. While a lot of approaches have been developed to estimate and improve the quality of case data, today nearly no research work addresses the evaluation of similarity measures.

By introducing the concept of retrieval error and a respective function to measure this error, our framework brings a further benefit. The retrieval error represents a well-founded and powerful instrument to measure the quality of a defined similarity measure in a systematic way. Here, it is not important whether this similarity

measure was defined manually or whether it was learned by some learning procedure.

If it was defined manually, the training data required to compute the retrieval error can be characterised as independent test data. It defines the desired outcome of the similarity assessment for some exemplary queries. This desired outcome is then compared with the outcome achieved with the similarity measure defined manually. A significant difference between the desired and the achieved results, then might be an evidence for an insufficiently modelled similarity measure. In this case, the actually redundant training data, of course, can also be used as the foundation of a subsequent learning procedure in order to optimise the suboptimal similarity measure.

10.2. Open Questions and Future Research Directions

As already mentioned, the work presented in this thesis represents only a first step towards a general approach to learning complex knowledge-intensive similarity measures. It introduced a basic framework and methodology, and two concrete learning algorithms. However, there are still several open aspects that represent interesting research issues for future work.

10.2.1. Improved Acquisition of Training Data

As typical for machine learning approaches, the success of our learning framework strongly depends on the quality of the available training data. In Chapter 4 we have discussed several principle techniques to acquire training data and in Chapter 6 we have shown how training data might be obtained in real world applications. However, concerning the acquisition of accurate training data the following ideas might improve the actual learning process.

Selecting Accurate Training Data

In this work we have only assumed that some similarity teacher provides feedback about the relative utility of some arbitrary cases previously retrieved. However, we have not discussed how to select cases within the retrieval result for which an evaluation seems to be promising in order to obtain "good" training data. Instead we assumed that some human similarity teacher selects accurate cases her/himself due to her/his experiences in the domain. Of course, when employing an artificial similarity teacher, for example, as discussed in Section 6.5, this is more difficult. Here, we have supposed that the n most similar cases (according to the initial similarity measure) will be evaluated automatically.

The same problem arises with respect to the selection of appropriate queries to be used as the foundation of training examples. Here, we have assumed again that

"good" training queries are provided by a human similarity teacher, or that they are generated randomly when dealing with an artificial similarity teacher. Another possibility might be to use queries that occur during the daily application of the underlying CBR system. On the one hand, this approach might be infeasible when learning the similarity measure during the development phase of the CBR system. On the other hand, if the system is used very frequently, perhaps by many different users, one has to deal with the problem to select some cases from the large set of available "real world" queries.

When thinking about "good" training examples one important aspect is, of course, the coverage of the case space. If training data only contains training examples that are based on queries clustered in some region of the case space, this will probably lead to unsatisfactory learning results. The same problem holds if the evaluated cases are clustered. To solve this problem, one might propose to generate a set of training queries automatically that cover the case space uniformly. However, this might also be suboptimal because of significant differences between the theoretical case space and the space relevant in practice. Often, large areas of the theoretical case space are irrelevant since real world queries and cases typically do not fall into these areas. Thus, in the future this issue might be examined in more detail in order to develop better strategies for selecting accurate training queries and training cases.

Developing Comfortable And Intelligent User Interfaces

In Section 7.3.1 we have already presented a first design proposal for the implementation of a graphical user interface for acquiring utility feedback. Nevertheless, the development of comfortable and more intelligent user interfaces is still an important and interesting research issue. The interface presented in this work is only suited for the application scenario described in Section 6.2, i.e. for supporting a domain expert during the definition of similarity measures. For all other presented scenarios that deal with human similarity teachers, the introduced interface structure is not appropriate. In these scenarios, the actual users of the CBR system play the role of the similarity teacher and utility feedback has to be acquired when using the system.

In this situation it is very important to develop comfortable interfaces that allow to obtain utility feedback during the normal operation of the CBR system with minimal additional effort for the user. For example, in the described eCommerce scenario (cf. Section 6.4) a customer usually will not be willed to operate a complex interface in order to provide feedback about the utility of recommended products. Instead, customers will be interested to get good recommendations very quickly with minimal effort.

How much interaction between the system and the user will be tolerated strongly depends on the domain and the respective application scenario. To minimise the effort for the user one may use the non-invasive approach already discussed in Section 4.2.3. However, to get enough high quality training data, this approach might

require more intelligent user interfaces that acquire utility feedback by observing the users' behaviour. For example, in a knowledge management scenario the time that a user spends with analysing a case might be a hint for the case's utility. Other possibilities are the evaluation of the clicking behaviour or an analysis of the process how a user modifies and repeats her/his query in order to find a more useful case.

Probably, it will be difficult to develop general approaches for realising accurate interfaces used to acquire utility feedback from the systems' users. Special versions of such interfaces rather have to be developed for each application scenario in order to fit the particular requirements. For example, a particular knowledge management system probably might require another interface than a product recommendation system.

10.2.2. More "Intelligent" Learning Algorithms

The two learning algorithms presented in Chapter 5 can be characterised as relatively "stupid" algorithms. In principle, they are founded on a statistical analysis of the training data by the determination of retrieval errors caused by the similarity measure currently used. Although the gradient descent approach employs some knowledge about the "shape" of the error function, the search process of both algorithms is not based on a deeper analysis of the available training data.

To obtain high quality similarity measures from small training data sets, one could imagine learning algorithms that try to extract the influences on the actual utility of cases in a more "intelligent" way. For example, a learning algorithm might analyse very useful cases in order to find common properties of these cases. Such properties then might be employed to define accurate similarity measures in a more goal directed manner.

Dealing with Cardinal Utility Feedback

In Chapter 4, we have distinguished the principle possibility to acquire *ordinal* and *cardinal utility feedback*. However, in this work we have only introduced error functions that exploit ordinal utility information (cf. Definition 4.4). Nevertheless, it would also be interesting to investigate how cardinal information can facilitate the learning process. At least when applying the genetic algorithm, the definition of a corresponding error function should be simple because arbitrary functions can be used here .

Adaptation to Training Data

The versions of learning algorithms presented in this thesis treat all regions of the search space in the same way. For example, the genetic algorithm tries to optimise

every entry of similarity tables and every sampling point of difference-based similarity functions. However, particular training data often provides different amount of information about different areas of the search space. It might happen, that it does not contain certain combinations of query and cases values for some attributes, and therefore learning of corresponding table entries or sampling points will be viewless. In order to avoid the waste of computation time, training data might be analysed prior to the actual learning process in order to optimise the representation of the search space. We have already discussed this possibility in Section 5.3.3 where we mentioned the possibility to distribute sampling points for representing similarity functions dynamically. However, other possibilities to adapt the learning procedure on the available training data are thinkable, for example:

- In particular when dealing with symbolic attributes that allow many different values, similarity tables become huge, resulting in a large search space to be considered by the learning algorithm. This search space can be reduced, if table cells, for which no training information is available, are not considered during learning. This can be realised by a respective individual representation that has to be determined according to the available training data.

- Another problem is the influence of attribute weights on learning local similarity measures. For attributes with very small weights, computational intensive learning of corresponding local similarity measures might be needless, because possible improvements might have a minimal influence on the retrieval behaviour. If a rough estimation of the attributes' weights is given prior to learning of local similarity measures, the learning process might be improved by focusing the search on the most important attributes. Here, it is thinkable to assign computation time dynamically by processing different number of generations during learning of different local similarity measures.

Handling of Noisy Training Data

Another important issue is the treatment of *noise* in the training data. In particular, when acquiring utility feedback from several different users, the probability that the training data includes inconsistencies and contradictions is very high. Because utility feedback is often founded on more or less subjective decisions, even when asking one individual similarity teacher to repeat the evaluation of some cases, s/he might come to different results as in previous evaluation process.

So, it would be very helpful if a learning algorithm is able to detect obvious inconsistencies in the provided data. Information about such inconsistencies then, for example, might be used to ask the similarity teacher again, or to remove certain cases from training examples.

Avoiding of Overfitting

Another import aspect in machine learning is the well-known problem of *overfitting* the training data (Domingos, 1999; Cohen and Jensen, 1997). In particular, when dealing with small training data sets, the risk to learn a similarity measure that works very well for queries and cases contained in the training examples, but that produces insufficient results for new queries, is very high. Here, improvements in the presented learning algorithms or alternative algorithms might help to reduce this risk. For example, sometimes it might be useful to stop the learning process before the error function's actual minimum is found to avoid overfitting.

10.2.3. Consideration of Available Background Knowledge

Concerning the problem of overfitting, another approach seems to be promising, too. In Section 4.3.1 we have characterised a manual definition of similarity measures as a bottom-up procedure and our learning approach as a top-down procedure. This means, both approaches to modelling similarity measures follow a complementary strategy. While the bottom-up procedure relies on low-level knowledge about the influences on the utility of cases, the top-down approach exploits available high-level knowledge.

In spite of the complementary character of both strategies, it might be useful to combine both approaches in practice (Stahl, 2002a) in order to profit from the advantages of both. On the one hand, a manual definition of similarity measures based on well-founded domain knowledge usually will provide a good approximation of the unknown utility function. On the other hand, the learning approach might help to exploit domain knowledge difficult to acquire or to formalise.

In a first step, a knowledge engineer might encode some well-founded background knowledge into the similarity measure by following the bottom-up procedure. This means, any causal relationships of the application domain that are obvious to the expert should be encoded directly in particular local similarity measures and feature weights. For all other representation elements one may use standard similarity functions based on the geometrical distance of the values to be compared. Such standard functions are even used by common commercial CBR shells as an initial similarity measure.

In a second step, one can apply the top-down approach to encode additional, but perhaps difficult to acquire background knowledge into the similarity measure. Therefore, the knowledge engineer has to determine typical example queries that are used for a test retrieval based on the similarity measure defined before. By giving respective utility feedback, i.e. high-level knowledge about the utility of cases, the resulting training data then can be used to optimise the predefined similarity measure.

In order to combine the bottom-up with the top-down approach in order to op-

timise the definition of knowledge-intensive similarity measures, one might use the strategies described in the following.

Using Well-Founded Initial Similarity Measure

Depending on the applied learning algorithm, the definition of an initial similarity measure may have a crucial impact on the outcome of the entire learning procedure. A typical example of such a learning algorithm is the gradient descent approach (cf. Section 5.2). In order to obtain an accurate initial similarity measure hopefully leading to good learning results, one might exploit well-founded background knowledge. Therefore a similarity measure defined by a domain expert—maybe only partially—can be used as starting point. Such a well-founded initial similarity measure should increase the probability that the learning algorithm converges to an accurate similarity measure. Of course, under certain circumstances, the algorithm might also change the initial setting completely. Then, one has to answer the question whether there are significant inconsistencies between the defined similarity measure and the training data exploited by the learning algorithm. When working with relative small training data sets, such an effect might also be a hint, that the learned similarity measure significantly overfitts the training data.

Using Constraints

The previously described possibility for considering background knowledge during learning obviously does not ensure that the defined background knowledge still can be found in the final learning results. Further, this approach is also difficult to apply when employing learning algorithm that do not strongly rely on a good starting point represented by an initial similarity measure. For example, a genetic algorithm uses several starting points because each individual of the initial population can be seen as a different starting point. Although one might employ background knowledge when defining the initial population, this seems not to be very promising to influence the outcome of a genetic algorithm due to the very flexible search process.

In order to exploit available well-founded background knowledge, therefore, one might follow an alternative strategy. A domain knowledge might again define a similarity measure partially due to her/his expertise in the underlying domain. The parts (e.g. certain local similarity measures, or parts of them) of the entire similarity measure representation that have been defined by the expert have to be marked. The learning algorithm is then allowed to modify only the remaining representation elements in order to optimise the entire measure. This leads to a limitation of the search space and thus fewer training data should be required for learning.

Instead to predefine some parts of the similarity measure exactly, one may also apply a more flexible approach. One could enable the domain expert to express *constraints* on the entire similarity measure representation in order to incorporate

background knowledge. For example, the expert might express certain knowledge about the importance of attributes by ranking them. This information then can be interpreted as a constraint to be considered when learning attribute weights. Therefore, the learning algorithm has to determine concrete values for the weights that have to be consistent with the attribute ranking of the expert. e.g. it must hold $w_i > w_j$ if attribute a_i is considered to be more important than attribute a_j. Similar constraints might be used to express knowledge about the entries of similarity tables or the basic structure of similarity functions.

10.2.4. Extension for Handling other Case Representations

In this work we have assumed a flat attribute-value based case representation like introduced in Section 2.2.4. This particular representation leads to the assumed structure of similarity measures defined in Section 3.3.2. However, depending on the application domain sometimes other case representations are more suitable. In particular, object-oriented case representations that can be seen as an extension of flat attribute-value based representation also play an important role in commercial CBR applications. The advantage of this formalism is the possibility to represent case knowledge in a more structured way.

However, this also increases the effort and complexity when defining corresponding similarity measures. Although similarity measures commonly used for object-oriented representations are based on the same principles as the measures introduced in this work, the learning algorithms presented in Chapter 5 cannot directly be applied to learn such measures. On the one hand, also the structure of the entire similarity measure becomes more complex because the global similarity measure (cf. Definition 3.17) is extended in a recursive manner. The reason for this is, that local similarity measures may be represented again by global similarity measures instead of a simple similarity function or similarity table. On the other hand, the object-oriented structure provides some information that might be used to define more accurate similarity measures. How to employ this information for the similarity assessment is presented by Bergmann and Stahl (1998).

These differences in the similarity representation complicate or even prevent the direct application of our learning algorithms. On the one hand, concerning the gradient descent algorithm the more sophisticated structure of the global similarity measure leads to a much more complex function for computing the similarity error (cf. Definition 5.4) required for learning. Due to this complexity the partial derivation of the function cannot easily be derived. On the other hand, the application of our genetic approach would not lead to such fundamental problems. Nevertheless, this algorithm ought be extended to consider also the object-oriented structure in order to learn more powerful similarity measures.

When dealing with other case representations that are not based on attribute-value pairs (e.g. representations based on first order logic, graph representations,

etc.), our learning algorithms cannot be applied at all. For example, the application of a genetic algorithm then would require a completely new formalism for representing respective similarity measures in form of individuals. Although, the presented learning algorithms are not suited to be applied on other case representations directly, our general learning framework presented in Chapter 4 abstracts from the case representation, and thus, it is also valid for other representation formalisms.

10.2.5. More Detailed Evaluation

The experimental evaluation carried out in the scope of this work is only suited to demonstrate the principle functionality of our framework and the implemented learning algorithms. A more detailed evaluation would have been advisable, but was not feasible in the scope of this work. On the one hand, a statistical significant evaluation would require to repeat the experiments numerous times. However, concerning the application of the genetic algorithm a large number of single runs of the algorithm requires considerable CPU time. The computational complexity becomes a really crucial problem if the algorithm should also be evaluated concerning different parameters (e.g., different version of the error function). On the other hand, in particular the application scenarios that base on one or several human similarity teachers are difficult to evaluate, because such realistic similarity teachers are not easily available in the research field.

Because of these difficulties we have focused the evaluation on few experiments only. Although, these experiments are sufficient to demonstrate the principal applicability of our framework and algorithms, a more detailed evaluation is an interesting issue for future research. The following aspects might be examined in future evaluation experiments:

- In our experiments we have not employed training data of human similarity teachers. Since such training data probably contains much noise and therefore makes learning more difficult, a respective evaluation would be interesting.

- In our evaluation we have only considered two of the application scenarios discussed in Chapter 6. In the future it has to be validated if our framework is also suited for the other described scenarios in real world domains.

- Another interesting issue is the relationship between the complexity of the underlying domain size and the required amount of training data. The more attributes a case representation includes, the more weights and local similarity measures are required. This obviously leads to an increased search space and therefore more training data is necessary to learn an accurate similarity measure.

- The functionality of the presented learning algorithms can be influenced by several parameters like discussed in Section 5.2.3 and 5.3.6. Moreover, also the introduced error functions can be modified, for example, by the error-position weight introduced in Definition 4.5. A more detailed analysis of these parameter's impact on the quality of learning results would be interesting and helpful in order to optimise our learning algorithms. However, as already mentioned above, such an evaluation will be very time consuming due to the computational complexity of the genetic algorithm.

- In Section 5.3.3 we have discussed three possible approaches for distributing sampling points when representing difference-based similarity functions to be optimised by the described genetic algorithm. However, in the presented evaluation we have only used the simplest of these approaches, namely an equidistant distribution. Thus, further experiments in order to evaluate the power of the two more sophisticated strategies would be interesting.

- When developing learning algorithms one always has to consider the well-known problem of overfitting the training data. Although our experiments provide first evidences about the risk of overfitting when applying our framework, a more detailed analysis of this aspect would be interesting.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59.

Aha, D. (1991). Case-Based Learning Algorithms. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 147–158. Morgan Kaufmann.

Aha, D. W. and Wettschereck, D. (1997). Case-based learning: Beyond classification of feature vectors. In *Proceedings of the 9th European Conference on Machine Learning (ECML'97)*. Springer.

Althoff, K.-D. (1992). *Eine fallbasierte Lernkomponente als integrierter Bestandteil der MOLTKE-Werkbank zur Diagnose technischer Systeme*. Infix.

Althoff, K.-D., Auriol, E., Barletta, R., and Manago, M. (1995). *A Review of Industrial Case-Based Reasoning Tools*. AI Intelligence, Oxford, UK.

Althoff, K.-D., Bergmann, R., Wess, S. Manago, M., Auriol, E., Larichev, O., Bolotov, A., Zhuravlev, Y. I., and Gurov, S. I. (1998). Case-Based Reasoning for Medical Decision Support Tasks: The INRECA Approach. *Artificial Intelligence in Medicine*, 12:25–41.

Althoff, K.-D., Birk, A., and Tautz, C. (1997). The Experience Factory Approach. In *Proceedings of the 10th German Workshop on Machine Learning (FGML'97)*.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). Experience Factory. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering, Volume 1*, pages 469–476. John Wiley & Sons.

Bergmann, R. (1998). On the Use of Taxonomies for Representing Case Features and Local Similarity Measures. In *Proceedings of the 6th German Workshop on Case-Based Reasoning (GWCBR'98)*.

Bergmann, R. (2002). *Experience Management*. Springer.

Bergmann, R., Breen, S., Fayol, E., Göker, M., Manago, M., Schmitt, S., Schumacher, J., Stahl, A., Wess, S., and Wilke, W. (1998). Collecting Experience on the Systematic Development of CBR Applications Using the INRECA-II Methodology. In *Proceedings of the 4th European Workshop on Case-Based Reasoning (EWCBR'98)*. Springer.

Bergmann, R., Breen, S., Göker, M., Manago, M., and Wess, S. (1999a). *Developing Industrial Case-Based Reasoning Applications: The INRECA Methodology*. State-of-the-Art-Survey, LNAI 1612. Springer.

Bergmann, R., Richter, M. M., Schmitt, S., Stahl, A., and Vollrath, I. (2001). Utility-Oriented Matching: A New Research Direction for Case-Based Reasoning. In *Professionelles Wissensmanagement: Erfahrungen und Visionen. Proceedings of the 1st Conference on Professional Knowledge Management*. Shaker.

Bergmann, R., Schmitt, S., and Stahl, A. (2002). *E-Commerce and Intelligent Methods*, chapter Intelligent Customer Support for Product Selection with Case-Based Reasoning. Physica-Verlag.

Bergmann, R. and Stahl, A. (1998). Similarity Measures for Object-Oriented Case Representations. In *Proceedings of the 4th European Workshop on Case-Based Reasoning (EWCBR'98)*. Springer.

Bergmann, R. and Vollrath, I. (1999). Generalized Cases: Representation and Steps Towards Efficient Similarity Assessment. In *Proceedings of the 23rd annual German Conference on Artificial Intelligence (KI'99)*. Springer.

Bergmann, R., Vollrath, I., and Wahlmann, T. (1999b). Generalized Cases and their Application to Electronic Designs. In *Proceedings of the 7th German Workshop on Case-Based Reasoning (GWCBR'99)*.

Bergmann, R., Wilke, W., Vollrath, I., and Wess, S. (1996). Integrating General Knowledge with Object-Oriented Case Representation and Reasoning. In *Proceedings of the 4th German Workshop on Case-Based Reasoning (GWCBR'96)*.

Bonzano, A., Cunningham, P., and Smyth, B. (1997). Using Introspective Learning to Improve Retrieval in CBR: A Case Study in Air Traffic Control. In *Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR'97)*. Springer.

Branting, K. (1999). Active Exploration in Instance-Based Preference Modeling. In *Proceedings of the 3rd International Conference on Case-Based Reasoning (ICCBR'99)*. Springer.

Branting, K. (2001). Acquiring Customer Preferences from Return-Set Selections. In *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR'2001)*. Springer.

Burke, R. (1999). The Wasabi Personal Shopper: A Case-Based Recommender System. In *Proceedings of the 11th International Conference on Innovative Applications of Artificial Intelligence (IAAI'99)*.

Cohen, P. R. and Jensen, D. (1997). Overfitting Explained. In *Preliminary Papers of the 6th International Workshop on Artificial Intelligence and Statistics*.

Coyle, L. and Cunningham, P. (2003). Exploiting Re-ranking Information in a Case-Based Personal Travel Assistent. In *Workshop on Mixed-Initiative Case-Based Reasoning at the 5th International Conference on Case-Based Reasoning (ICCBR'2003)*. Springer.

Cunningham, P., Slattery, S., and Finn, D. (1993). Knowledge Engineering Requirements in Derivational Analogy. In *Proceedings of the 1st European Workshop on Case-Based Reasoning (EWCBR'93)*. Springer.

Dasarathy, B. (1991). *NN concepts and techniques. Nearest Neighbour (NN) Norms: NN Pattern Classification Techniques*, pages 1–30. IEEE Computer Society Press.

Demiröz, G. and Güvenir, H. (1996). Genetic Algorithms to Learn Feature Weights for the Nearest Neighbor Algorithm. In *Proceedings of the 6th Belgian-Dutch Conference on Machine Learning (BENELEARN'96)*. MATRIKS.

Domingos, P. (1999). The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425.

Ferrario, M. A. and Smyth, B. (2000). Collaborative Maintenance - A Distributed, Interactive Case-Base Maintenance Strategy. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Fox, S. and Leake, D. (1995a). Learning to Refine Indexing by Introspective Reasoning. In *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR'95)*.

Fox, S. and Leake, D. (1995b). Using Introspective Reasoning to Refine Indexing. In *Proceedings of the 13th International Joint Conference in Artificial Intelligence (IJCAI'95)*.

Fuchs, B., Lieber, J., Mille, A., and Napoli, A. (1999). Towards a Unified Theory of Adaptation in Case-Based Reasoning. In *Proceedings of the 3rd International Conference on Case-Based Reasoning (ICCBR'99)*. Springer.

Globig, C. (1997). *Fallbasierte Repräsentierbarkeit und Lernbarkeit*. Ph.D. Thesis, University of Kaiserslautern.

Göker, A. (1999). Capturing Information Need by Learning User Context. In *Working Notes of the Workshop on Learning about Users, 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*.

Göker, M., Roth-Berghofer, T., Bergmann, R., Pantleon, T., Traphöner, R., Wess, S., and Wilke, W. (1998). The Development of HOMER: A Case-Based CAD/CAM Help-Desk Support Tool. In *Proceedings of the 4th European Workshop on Case-Based Reasoning (EWCBR'98)*. Springer.

Göker, M. and Thompson, A. (2000). Personalized Conversational Case-Based Recommendation. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Gomes, P. and Bento, C. (2000). Learning User Preferences in Case-Based Software Reuse. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Hanney, K. and Keane, M. (1996). Learning Adaptation Rules From a Case-Base. In *Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96)*. Springer.

Hanney, K. and Keane, M. (1997). The Adaptation Knowledge Bottleneck: How to Ease it by Learning from Cases. In *Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR'97)*. Springer.

Hanney, K., Keane, M., Smyth, B., and Cunningham, P. (1996). When do you need Adaptation?: A Review of Current Practice. In *Proceedings of the AAAI-95 Fall Symposium on Adaptation in Knowledge Reuse*. AAAI Press.

Hayes, C., Cunningham, P., and Smyth, B. (2001). A Case-Based Reasoning View of Automated Collaborative Filterning. In *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR'2001)*. Springer.

Heister, F. and Wilke, W. (1997). An Architecture for Maintaining Case-Based Reasoning Systems. In *Proceedings of the 4th European Workshop on Case-Based Reasoning (EWCBR'98)*. Springer.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press.

Hust, A., Junker, M., and Dengel, A. (1993). A Mathematical Model for Improving Retrieval Performance in Collaborative Information Retrieval.

Jarmulak, J., Craw, S., and Rowe, R. (2000a). Genetic Algorithms to Optimise CBR Retrieval. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Jarmulak, J., Craw, S., and Rowe, R. (2000b). Self-Optimising CBR Retrieval. In *Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2000)*. IEEE.

Kelly, J. J. and Davis, L. (1991). A Hybrid Genetic Algorithm for Classification. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*. Morgan Kaufmann Publishers.

Kira, K. and Rendell, L. A. (1992). A Practical Approach to Feature Selection. In *Proceedings of the 9th International Conference on Machine Learning (ICML'92)*. Morgan Kaufmann Publishers.

Kohavi, R., Langley, P., and Yun, Y. (1997). The Utility of Feature Weighting in Nearest-Neighbor Algorithms. In *Proceedings of the 9th European Conference on Machine Learning (ECML'97)*. Springer.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kaufmann Publishers.

Kononenko, I. (1994). Estimating Attributes: Analysis and Extensions of RELIEF. In *Proceedings of the 6th European Conference on Machine Learning (ECML'94)*. Springer.

Leake, D. (1994). Experience, Introspection, and Expertise: Learning to Refine the Case-Based Reasoning Process. *Journal of Experimental and Theoretical Artificial Intelligence, special issue: The Psychology of Expertise: Human and Artificial*.

Leake, D., Kinley, A., and Wilson, D. (1995). Combining Rules and Cases to Learn Case Adaptation. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

Leake, D., Kinley, A., and Wilson, D. (1996a). Linking Adaptation and Similarity Learning. In *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

Leake, D., Kinley, A., and Wilson, D. (1996b). Multistrategy Learning to Apply Cases for Case-Based Reasoning. In *Proceedings of the 3rd International Workshop on Multistrategy Learning (MSL'96)*. AAAI Press.

Leake, D., Kinley, A., and Wilson, D. (1997a). Case-Based Similarity Assessment: Estimating Adaptability from Experience. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI'97)*. AAAI Press.

Leake, D., Kinley, A., and Wilson, D. (1997b). Learning to Integrate Multiple Knowledge Sources for Case-Based Reasoning. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*. Morgan Kaufmann Publishers.

Leake, D. and Wilson, D. (2000). Remembering Why to Remember: Performance-Guided Case-Base Maintenance. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Lenz, M. (1999). *Case Retrieval Nets as a Model for Building Flexible Information Systems*. Ph.D. Thesis, Humboldt University Berlin.

Lenz, M., Bartsch-Spörl, B., Burkhard, H., and Wess, S., editors (1998). *Case-Based Reasoning Technology: From Foundations to Applications*. LNAI: State of the Art. Springer.

Liebowitz, J., editor (1999). *Knowledge Management Handbook*. CRC Press.

Lowe, D. (1993). Similarity Metric Learning for a Variable-Kernel Classifier. *Neural Computation*, 7.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Moser, A. (1999). Distributed Genetic Algorithms for Feature Selection. Master thesis, University of Kaiserslautern.

Mougouie, B. and Bergmann, R. (2002). Similarity Assessment for Generalized Cases by Optimization Methods. In *Proceedings of the 6th European Conference on Case-Based Reasoning (ECCBR'2002)*. Springer.

Muñoz-Avila, H. and Hüllen, J. (1996). Feature Weighting by Explaining Case-based Planning Episodes. In *Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96)*. Springer.

Norvig, P. and Russell, S. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Novick, L. (1988). Analogical Transfer, Problem Similarity, and Expertise. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 14:510–520.

Ohnishi, H., Suzuki, H., and Shigemasu, K. (1994). Similarity by Feature Creation: Reexamination of the Asymmetry of Similarity. In *Proceedings of the 16th Annual Conference of Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

O'Sullivan, D., Wilson, D., and Smyth, B. (2002). Improving Case-Based Recommendation: A Collaborative Filtering Approach. In *Proceedings of the 6th European Conference on Case-Based Reasoning (ECCBR'2002)*. Springer.

Purvis, L. and Pu, P. (1995). Adaptation Using Constraint Satisfaction Techniques. In *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR'95)*.

Reinartz, T., Iglezakis, I., and T., R.-B. (2000). On Quality Measures for Case Base Maintenance. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Ricci, F. and Avesani, P. (1995). Learning a Local Similarity Metric for Case-Based Reasoning. In *Proceeding of the 1st International Conference on Case-Based Reasoning (ICCBR'95)*. Springer.

Richter, M. (2003). Learning Similarities for Informally Defined Objects. In Kühn, R., Menzel, R., Menzel, W., Ratsch, U., Richter, M., and Stamatescu, I.-O., editors, *Adaptivity and Learning*. Springer.

Richter, M. M. (1992). Classification and Learning of Similarity Measures. In *Studies in Classification, Data Analaysis and Knowledge Organisation Proceedings of the Jahrestagung der Gesellschaft für Klassifikation*. Springer.

Richter, M. M. (1995). The Knowledge Contained in Similarity Measures. Invited Talk at ICCBR'95.

Roth-Berghofer, T. (2002). *Knowledge Maintenance of Case-Based Reasoning Systems*. Ph.D. Thesis, University of Kaiserslautern.

Salzberg, S. L. (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6.

Schaaf, J. W. (1996). Fish and Shrink. A Next Step Towards Efficient Case Retrieval in Large-Scale Case Bases. In *Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96)*. Springer.

Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press.

Schank, R. and Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding*. Hillsdale, NJ: Lawrence Erlbaum.

Schmitt, S. (2003). *Dialog Tailoring for Similarity-Based Electronic Commerce Systems*. Ph.D. Thesis, Kaiserslautern University of Technology.

Schmitt, S. and Bergmann, R. (1999a). Applying Case-Based Reasoning Technology for Product Selection and Customization in Electronic Commerce Environments. In *Proceedings of the 12th Bled Electronic Commerce Conference*.

Schmitt, S. and Bergmann, R. (1999b). Product Customization in an Electronic Commerce Environment Using Adaptation Operators. In *Proceedings of the 7th German Workshop on Case-Based Reasoning (GWCBR'99)*.

Schmitt, S. and Stahl, A. (2002). A CBR Approach to Identification and Counteraction for Harmful Microorganisms in Wastewater Treatment Plants. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'02)*. CSREA Press.

Schulz, S. (1999). CBR-Works - A State-of-the-Art Shell for Case-Based Application Building. In *Proceedings of the 7th German Workshop on Case-Based Reasoning (GWCBR'99)*.

Schumacher, J. and Bergmann, R. (2000). An Efficient Approach to Similarity-Based Retrieval on Top of Relational Databases. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Segovia, J., Szczepaniak, P., and Niedzwiedzinski, M., editors (2002). *E-Commerce and Intelligent Methods*. Physica-Verlag.

Skalak, D. B. (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*. Morgan Kaufmann Publishers.

Smyth, B. and Cotter, P. (1999). Surfing the Digital Wave, Generating Personalised TV Listings using Collaborative, Case-Based Recommendation. In *Proceedings of the 3rd International Conference on Case-Based Reasoning (ICCBR'99)*. Springer.

Smyth, B. and Keane, M. T. (1993). Retrieving Adaptable Cases: The Role of Adaptation Knowledge in Case Retrieval. In *Proceedings of the 1st European Workshop on Case-Based Reasoning (EWCBR'93)*. Springer.

Smyth, B. and Keane, M. T. (1995a). Experiments on Adaptation-Guided Retrieval in Case-Based Design. In *Proceedings of the 1st International Conference on Case-Based Reasoning (ICCBR'95)*. Springer.

Smyth, B. and Keane, M. T. (1995b). Remembering to Forget: A Competence Preserving Case Deletion Policy for CBR Systems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*. Morgan Kaufmann Publishers.

Smyth, B. and McKenna, E. (1998). Modelling the Competence of Case-Bases. In *Proceedings of the 4th European Workshop on Case-Based Reasoning (EWCBR'98)*. Springer.

Stahl, A. (2000). Recursive Case-Based Reasoning and its Application in Electronic-Commerce Environments. Master thesis, Kaiserslautern University of Technology.

Stahl, A. (2001). Learning Feature Weights from Case Order Feedback. In *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR'2001)*. Springer.

Stahl, A. (2002a). Defining Similarity Measures: Top-Down vs. Bottom-Up. In *Proceedings of the 6th European Conference on Case-Based Reasoning (ECCBR'2002)*. Springer.

Stahl, A. (2002b). Statement of Interest: Learning Personalized Utility Requirements. In *Proceedings of the Workshop on Case Based Reasoning and Personalization, 6th European Conference on Case-Based Reasoning (ECCBR'2002)*.

Stahl, A. and Bergmann, R. (2000). Applying Recursive CBR for the Customization of Structured Products in an Electronic Shop. In *Proceedings of the 5th European Workshop on Case-Based Reasoning (EWCBR'2000)*. Springer.

Stahl, A., Bergmann, R., and Schmitt, S. (2000). A Customization Approach For Structured Products in Electronic Shops. In *Proceedings of the 13th International Bled Electronic Commerce Conference*. Moderna organizacija, Kranj, Slovenia.

Stahl, A. and Gabel, T. (2003). Using Evolution Programs to Learn Local Similarity Measures. In *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR'2003)*. Springer.

Stahl, A. and Schmitt, S. (2002). Optimizing Retrieval in CBR by Introducing Solution Similarity. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'02)*. CSREA Press.

Stolpmann, M. and Wess, S. (1999). *Optimierung der Kundenbeziehung mit CBR-Systemen*. Business and Computing. Addison-Wesley.

Suzuki, H., Ohnishi, H., and Shigemasu, K. (1992). Goal-Directed Processes in Similarity Judgement. In *Proceedings of the 14th Annual Conference of Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

Tautz, C. and Althoff, K.-D. (1997). Using Case-Based Reasoning for Reusing Software Knowledge. In *Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR'97)*. Springer.

Tversky, A. (1977). Features of Similarity. *Psychological Review*, 84(4):327–352.

Wess, S. (1993). *Fallbasiertes Problemlösen in Wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik*. Ph.D. Thesis, University of Kaiserslautern.

Wettschereck, D. and Aha, D. W. (1995). Weighting Features. In *Proceeding of the 1st International Conference on Case-Based Reasoning (ICCBR'95)*. Springer.

Wilke, W. (1999). *Knowledge Management for Intelligent Sales Support in Electronic Commerce*. Ph.D. thesis, University of Kaiserslautern.

Wilke, W. and Bergmann, R. (1996). Considering Decision Cost During Learning of Feature Weights. In *Proceedings of the 3rd European Workshop on Case-Based Reasoning (EWCBR'96)*. Springer.

Wilke, W. and Bergmann, R. (1998). Techniques and Knowledge Used for Adaptation During Case-Based Problem Solving. In *Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA'98)*.

Wilke, W., Bergmann, R., and Vollrath, I. (1996). Using Knowledge Containers to Model a Framework for Learning Adaptation Knowledge. In *A Workshop at the 12th European Conference on Artificial Intelligence (ECAI'96)*. Springer.

Wilke, W., Lenz, M., and Wess, S. (1998). *Case-Based Reasoning Technology: From Foundations to Applications*, chapter Case-Based Reasoning and Electronic Commerce. Lecture Notes on AI: State of the Art. Springer.

Zhang, Z. and Yang, Q. (1999). Dynamic Refinement of Feature Weights Using Quantitative Introspective Learning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*.

List of Figures

Curriculum Vitae

Name:	Armin Stahl
Address:	Davenportplatz 9
	67663 Kaiserslautern
Born:	April 7, 1974, in Kaiserslautern
Family Status:	unmarried
Nationality:	german

1980 – 1984	Attendance at Elementary School, Kaiserslautern
1984 – 1993	Attendance at Gymnasium am Rittersberg, Kaiserslautern
1993	Abitur
1993 – 2000	Study of Computer Science, University of Kaiserslautern
1998 – 2000	Student Assistant in the Research Group "Artificial Intelligence – Knowledge-based Systems", University of Kaiserslautern
04/2000	Diplom
05/2000 – 12/2003	Member of the Research Group "Artificial Intelligence – Knowledge-based Systems", University of Kaiserslautern
10/2003	Oktober 10: Thesis defense at the University of Kaiserslautern