Diploma Thesis

Learning Similarity Measures: Strategies to Enhance the Optimisation Process

Thomas Gabel
November 2003

Tutors:
Prof. Dr. Michael M. Richter
Dr. Armin Stahl

Artificial Intelligence – Knowledge Based Systems Group
University of Kaiserslautern
Postfach 3049
67653 Kaiserslautern

UNIVERSITÄT
KAISERSLAUTERN

This is to certify that I have written this thesis, implemented the algorithms used for my work and conducted the described experiments completely by myself and without ulterior help. External sources and used literature are referenced and can be found in the Bibliography.

Kaiserslautern, November 2003 *Thomas Gabel*

Contents

List of Figures

1 Introduction

Similar problems have similar solutions. This is the assumption on the basis of which the research field of *Case-Based Reasoning* (CBR) has emerged. Knowledge-based systems operating on the basis of CBR solve new problems by means of reusing old solutions to former, yet similar problems. But when are two problems considered to be similar? The answer to that question is of crucial importance for CBR systems to function properly.

On the one hand, the similarity assessment could be based on a syntactical match between the old and the current problem situation. On the other hand, the similarity might represent an indication on how useful an old solution will be, when it is applied to the problem at hand. Finding a concrete manifestation of the term "similarity" is usually the task of a knowledge engineer. However, the last decade has seen a number of approaches that aim at using Machine Learning techniques to adjust the similarity assessment in Case-Based Reasoning.

Recently, an innovative and comprehensive framework for learning similarity measures has been introduced. Among its capabilities is the ability to automatically induce sophisticatedly modelled similarity metrics (Stahl, 2003). The objective of the thesis at hand is the enhancement of the mentioned learning framework, primarily by incorporating various forms of background knowledge into its optimisation process.

1.1 Motivation

Evidently, there is a multitude of approaches to build up knowledge-based systems. These approaches differ in the types of application domains they are employable in, in the amount of factual and general knowledge they can cope with as well as in the computational resources they require. When creating a knowledge-based system, the initial situation is characterised by a certain amount of available general knowledge, application domain-related knowledge and problem-specific knowledge that must be processed. From an abstract, perhaps metaphorical point of view that processing can be considered as making a more or less substantial amount of "knowledge liquid" pass through a "processing funnel" resulting in a ready-to-use system.

The Cyc project (Lenat and Guha, 1990) started in 1984 aimed ambitiously at the creation of a comprehensive intelligent system using a manual knowledge acquisition approach. Here, the processing funnel was operated manually (and was thus very tight): Human beings entered millions of rules into a knowledge base. Besides that very slow, yet accepted costly process, the resulting system suffered from the enormous complexity caused by the interplay of the vast amount of provided knowledge items. The other extreme, the Machine Learning approach, tries to automatically induce system-usable knowledge (e.g. rules) from data. This approach has led to numerous successful applications in many domains. However, they often suffer from problems such as noisy input data or the inability to computationally handle the eventually voluminous amounts of input knowledge.

Generalising, we may say that the processing of the knowledge liquid (input) by the funnel represents a bottleneck: If the amount of data/knowledge to be passed through is too extensive

or if the funnel itself is too tight, it may not be feasible to generate a satisfactory outcome with reasonable effort. In other words, smaller problems can be handled while handling more voluminous and comprehensive ones remains impossible. This is especially problematic as the tasks to be coped with in practice have been and are growing continuously in complexity.

In the last years, Case-Based Reasoning (CBR) has gained popularity as an approach that also tackles the before mentioned problem. Case-based applications solve new problems by reusing old solutions to former, similar problems. These experiences, called cases (usually consisting of a problem and a solution part), are stored in a case base. Given a new problem situation, called query, the CBR system retrieves that case from the case base which is most similar to the query and whose application to the current problem seems to be advisable. Returning to the metaphor from above, the advantage of the case-based approach is clear: During system build-up the case knowledge does not have to be compiled or processed explicitly. Instead it can – separated from the knowledge liquid – be bypassed along the funnel.

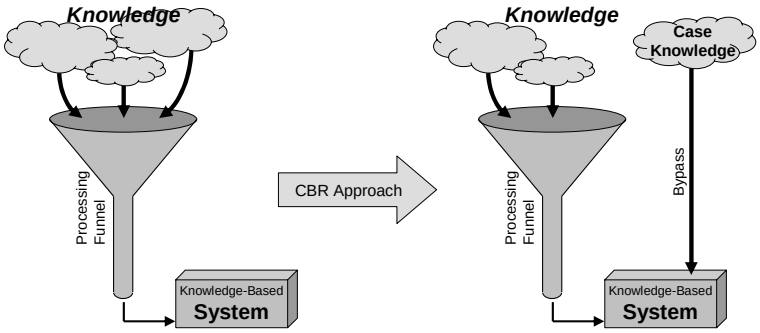


Figure 1.1: Metaphorical View on Building Knowledge-Based Systems

This advantage was one of the reasons why CBR as a new technology could succeed in being applied even in rather complex application domains throughout the previous decade.

The "price" to be paid for that advantage, however, can be prescribed as controlled inexactness in problem-solving. To keep that inexactness tolerable for possible users of the system, a first indispensable prerequisite for CBR to function properly is the availability of high-quality case data. Yet, the case data cannot be utilised appropriately, if it is not guaranteed that, given a new problem situation, the most useful cases are retrieved from the case base.

CBR systems employ so-called *similarity measures* to mathematically model the meaning of the rather informal term "similarity". Those measures represent heuristics to estimate the utility of old cases' solutions for the problem at hand. Since those measures depict the foundation on the basis of which cases are selected for reuse, a high quality of the employed similarity measures depicts another crucial prerequisite for any CBR system. As heuristics, similarity measures may contain more or less domain-specific knowledge. One extreme often used in traditional CBR systems is represented by simple distance metrics that do not consider any domain knowledge and interpret similarity as "similar look". Anyway, the experience of the previous years has shown that simple distance metrics are often insufficient, when applying CBR for more complex tasks. Hence, various modelling techniques to define more sophisticated similarity measures, i.e. those measures that may take domain knowledge into consideration and thus represent better heuristics for selecting useful cases, are used widely nowadays.

As a matter of fact, due to ever-growing requirements regarding the practical usage of a CBR system and growing sizes of problems to be solved, even experienced knowledge engi-

neers sometimes find it difficult to manually define appropriate similarity measures in complex application domains. In contrast to that state-of-the-art procedure for defining similarity measures, Stahl (2003) introduced a general learning framework that supports a better acquisition and formalisation of domain knowledge to be employed by machine learning algorithms for finding suitable similarity measures. Basically, it suggests the exploitation of easily acquirable feedback on the usefulness of cases for a number of queries. It then uses that information to adjust the similarity measures and thus to improve the quality of retrieval results they produce. In the language of our metaphor this work makes a second semi-automatic funnel available (the first funnel of course must be operated by a knowledge engineer), which uses some of the input knowledge liquid to find optimal similarity measures.

The intention of the thesis at hand is to enhance and to improve that framework for learning similarity measures, mainly by explicitly incorporating several forms of additional knowledge into the optimisation process. In so doing, we want to bridge or at least shorten the gap between both before mentioned oppositional approaches towards defining sophisticated similarity measures. As a result, we also intend to overcome some of the learning framework's weak points and, speaking metaphorically, mean to ameliorate the corresponding knowledge-processing funnel:

- The application of the learning framework's capabilities in certain application domains presupposes a number of specific extensions. We intend to achieve a broader applicability, i.e. to make the learning of similarity measures in further application scenarios applicable.
- The learning framework's optimisation process at its current stage of realisation is very time consuming. We intend to speed it up.
- By means of additional knowledge, given as hints or constraints to a learning algorithm, the overall performance may be improved on a grand scale. With respect to the mentioned learning framework, that incorporation of background knowledge ought to not only increase its general learning capabilities, but also reduce its susceptibility to the well-known problem of overfitting.

1.2 Thesis Outline

The availability of adequate similarity measures is of crucial importance for CBR systems. By means of the mentioned framework for learning similarity measures on the basis of "case order feedback" (which we will discuss in Chapter 2) we are in possession of the possibility to optimise similarity measures with various learning strategies. The thesis at hand attaches on that framework and aims at its enhancement. As a conclusion from the introductory motivation the following tasks shall be solved in the scope of this work.

Broader Applicability The current confinement to learning from training data in form of case order feedback, i.e. from information about the correctness of retrieval results, does not sufficiently support the usage of the learning framework for classification and solution prediction tasks. To gain a broader applicability and allow for the optimisation of similarity measures in those domains, too, we must develop an appropriate alternative way of acquiring feedback on the utility of cases as well as corresponding error functions.

This will give us the opportunity to learn similarity measures in numerous application domains, in which training data is amply available, e.g. in form of pre-classified cases. Consequently, the effort to acquire training data does not apply and so the realisation and experimental evaluation of the concepts to be introduced subsequently will be simplified.

Incorporating Knowledge The incorporation of additional knowledge can result in a selective restriction of the space of possible, syntactically correct similarity measures, that can be taken into consideration during learning, and thus significantly improve the learning process. The improvement may manifest itself in an increased learning speed and better learning results, respectively, as well as in a reduced amount of training data needed, which in turn may decrease the risk of overfitting. Intended strategies to incorporate background knowledge include, for example, heuristic constraints concerning the shape of similarity measures, the utilisation of concealed information within the case base or the acquisition and usage of partial expert knowledge in several forms and levels of confidence.

Systematic Optimisation and Evaluation The current prototypic realisation of the learning framework (as an extension to a commercial CBR tool) is only intended to represent a test and demonstration environment to show the principle functionality of the framework. We do not aim at advancing that prototype to a software usable for daily CBR practice. However, we intend to develop a number of sophisticated techniques by means of which the learning process can be sped up significantly. This is indispensable, especially when conducting an experimental evaluation of our extensions to the learning framework introduced in the scope of this work.

The remainder of this thesis is organised in accordance to the above-mentioned task definition. Beforehand, Chapter 2 gives an introduction to similarity measures and their modelling in Case-Based Reasoning. Furthermore, the principles and main features of the mentioned learning framework are illustrated. At the end of that chapter, setting up the way for the following ones, we motivate in detail why an enhancement of that framework is purposeful and in which manner those enhancements can be accomplished. Chapter 3 and 4 focus on a broader applicability and at the incorporation of background knowledge as described above. In Chapter 5, we start with some technical explanations and descriptions concerning the implementation carried out within this work. Moreover, we address several systematic (implementation-specific) optimisations that we introduced into the implementation of the learning framework. In order to show the capabilities of the concepts introduced in the previous chapters, in Chapter 6 we discuss the results of several experimental evaluations and address aspects such as overfitting and the learning improvement gained thanks to incorporated knowledge. The last chapter closes with conclusions and future work.

In general, we do not presume special previous knowledge from the reader of this thesis. However, some basic knowledge in Case-Based Reasoning and Machine Learning, in particular in Evolutionary Algorithms, would definitely help understanding the motivation and foundations of this work. Although we give a brief, technical description of the mentioned learning framework for similarity measures, the reader might be interested in a more detailed description of that framework. We therefore refer to the work of Stahl (2003). Some foundations on Case-Based Reasoning may be found in the respective literature, e.g. by Leake (1996a) or Lenz et al. (1998), and concerning the basics of Genetic and Evolutionary Algorithms we refer to Holland (1975) and Schwefel (1981).

2 Learning Similarity Measures

During the development of a CBR system for a specific application domain the definition of appropriate similarity measures is a difficult task. Yet, the availability of similarity measures, which sufficiently approximate the utility of cases for a given query, is an indispensable prerequisite for a proper working and a successful employment of any CBR application (Stahl, 2001). Hence, for a Case-Based Reasoning system to function in accordance to the frequently-cited paradigm "Similar problems have similar solutions" (Leake, 1996b) it is necessary to dispose of a reliable, trustworthy similarity assessment.

The work of Stahl (2003) introduces a methodology and a complete framework together with appurtenant algorithms, that aim at learning similarity measures. In this chapter we give a brief outline of "A Framework for Learning Similarity Measures in Case-Based Reasoning", since that work represents the foundation for the thesis at hand. In the following we will refer to that framework shortly by the abbreviation FLSM.

After some general remarks on similarity measures, we introduce their commonly used representation mechanisms and oppose two contrary approaches towards defining those measures. Then, we summarise the main features of the mentioned learning framework and its implementation. The following section focuses in detail on FLSM's learning algorithms as those will play a significant role throughout the remainder of this thesis. The last section of this chapter presents our motivation for enhancing FLSM.

2.1 Foundations

According to Richter (1995) the knowledge a CBR system makes use of is distributed over four containers: case knowledge, vocabulary knowledge, adaptation knowledge and retrieval knowledge (see Figure 2.1).

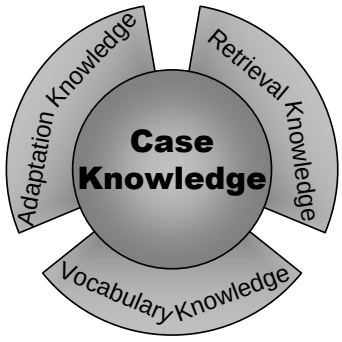


Figure 2.1: Knowledge Container Model

The case knowledge container of course contains cases, i.e. problem descriptions together with their respective solutions, that represent experiences made in the past. The vocabulary

used to characterise cases strongly depends on the application field. To fill that container, a knowledge engineer has to choose an appropriate case representation (e.g. attribute-value based representation[1] or graph representation) and to decide for associated data structures (e.g. data types of attributes and their value ranges). The third container comprises all the functionality required to adapt retrieved cases (e.g. transformational analogy using adaptation rules) in order to make them more useful for the problem situation at hand. Finally, the retrieval knowledge covers the similarity measure(s). These are mathematical representations of the very broad term "similarity" that are used to express the degree of similarity between two entities numerically. We will revert to the knowledge container model several times in the scope of this work, since it provides a good overview of the knowledge sources a CBR system may make use of and because with its help we can easily illustrate on which kind of knowledge a CBR system's competence relies on.

Definition 2.1 (Similarity Measure)
*A **similarity measure** is a real-valued function* $sim : \mathcal{M}^2 \rightarrow [0; 1]$*, where* $\mathcal{M}$ *is the set of all problem descriptions.*

Concerning the types of similarity measures used in CBR applications Stahl and Gabel (2003) differentiate between *knowledge-poor* and *knowledge-intensive similarity measures*, that differ from each other in the amount of domain-specific knowledge encoded into the measure.

The first ones are typically simple distance metrics whose similarity assessment can be characterised as a syntactical comparison. Typical examples are the hamming distance measure, the simple matching coefficient, or an Euclidian distance measure. Those measures include no or only little domain-specific knowledge and thus merely provide a poor approximation of a case's utility regarding a specific query. However, knowledge-poor similarity measures have been frequently used in many CBR applications developed during the past years. On the one hand, this is due to the fact that they inhere the advantage to be easily definable, i.e. the knowledge engineering effort is comparatively low. On the other hand, they lead to sufficiently correct retrieval results in many application areas – especially in rather simply structured domains.

However, intending to apply a case-based approach to domains that are structured more complicated, the use of more sophisticated similarity measures that consider as much domain knowledge as possible is inevitable. Furthermore, the use of those knowledge-intensive similarity measures may lead to even better problem solving capabilities in application fields, in which CBR has already been employed successfully. Unfortunately, the definition of knowledge-intensive similarity measures reconstitutes the knowledge acquisition problem, which ought to be avoided or at least attenuated by the use of Case-Based Reasoning. Their accurate definition in general requires a skilled domain expert providing the mandatory knowledge manually – which is of course a time-consuming and expensive process.

In Table 2.1 the characteristics of knowledge-poor and knowledge-intensive similarity measures are opposed to each other. The framework for learning similarity measures primarily targets to overcome the problems of knowledge-intensive similarity measures that are emphasised in the lower two lines of the right column of Table 2.1.

[1] This kind of case representation is the one used most commonly. Thus, in the scope of this work we assume an attribute-value based representation of cases.

Characteristic	kpSM	kiSM
similarity assessment paradigm	syntactical comparison	semantical comparison
knowledge contained	none	much
quality of retrieval results	sufficient in simply-structured domains, poor in more complicated ones	high
representation formalism	arbitrary (both types of similarity measures may employ the same formalisms)	
modelling method	usage of default measures, setting of a few parameters	*complex task, performed manually, reliant on human experts*
modelling effort	low	*high*

Table 2.1: Comparison Between Knowledge-Poor and Knowledge-Intensive Similarity Measures

2.2 Settling Similarity

As pointed out in the previous section, determining similarity measures that approximate the respective domain's underlying utility function as good as possible is a crucial task, when developing a CBR application. In this section we want to describe two contrary approaches to defining similarity measures. Before, we review common representation mechanisms for similarity measures.

2.2.1 Representation Mechanisms

For the attribute-value based case representation that we are assuming here similarity measures are usually modelled according to the so-called *local-global principle*. This principle allows to disassemble the overall similarity calculation into several parts corresponding to the elements of the case representation.

- First, for each attribute of a case exists a *local similarity measure* computing the local similarity between the case's and the query's value for the respective attribute.
- Second, each attribute is assigned a *feature weight* (also called attribute weight) that determines the respective attribute's importance regarding the approximation of the case's utility for the given query.
- The third part of this similarity measure definition method represents an *amalgamation function* which combines the local similarities with the feature weights to output a similarity value within $[0;1]$ (cf. Definition 2.1). Although several realisations of that amalgamation function are imaginable, in practise mostly a weighted average of the local similarity values is used.

Definition 2.2 (Local Similarity Measure, Feature Weight)
Given an attribute A and a type T_A, a ***local similarity measure*** *for A is defined as a function: $sim_A : T_A \times T_A \rightarrow [0;1]$.*

Moreover, attribute A is associated with a ***feature weight*** $w_A \in \mathbb{R}^+$ *that reflects A's relative importance compared to other attributes.*

In general, the representation of local similarity measures depends on the data type of the corresponding attribute on a grand scale. The two mostly used types are *numeric* types, like Integer or Real, and *symbolic* types, like unordered/ordered symbols or taxonomies. Two basic representation formalisms that can be used to represent local similarity measures for numeric and symbolic types are due to the following definitions:

Definition 2.3 (Similarity Function)
Let A be a numeric attribute with a value range of $D_A = [A_{min}, A_{max}]$. Under a ***similarity function*** *for D_A we understand a function*

$$sim_A : [(A_{min} - A_{max}), (A_{max} - A_{min})] \rightarrow [0; 1]$$

that computes a similarity value out of the interval $[0; 1]$ based on the difference between the case's value c and the query's value q of A.

Definition 2.4 (Similarity Table)
Let A be a symbolic attribute with a list of allowed values $D_A = (d_1, d_2, \ldots, d_n)$. A $n \times n$-matrix with entries $x_{i,j} \in [0; 1]$ that represent the similarity between the query's value $q = d_i$ and the case's value $c = d_j$ is called a ***similarity table*** *for D_A.*

There are other methods for defining similarity measures as well. For example, Richter (2001) speaks about the assignment of class-specific weights and Ricci and Avesani (1995) learn and use case-specific weights to reduce the size of the case base and to speed up the retrieval.

However, similarity measures modelled after the local-global-principle as introduced above, have reached a broad dispersion due to their modelling power. Therefore, the framework for learning similarity measures FLSM described here also works with similarity measures modelled after the local-global principle. Hence, FLSM might take up the optimisation of similarity measures at each of the three components of this similarity representation as listed above. In fact, up to now the functionality provided by FLSM covers the learning of attribute weights as well as the optimisation of local similarity measures (see Section 2.3), while the amalgamation function according to Definition 2.5 remains unchanged.

Definition 2.5 (Global Similarity Measure, Weighted Average Amalgamation)
Let cases be characterised by n attributes $A_1, \ldots, A_n$, for which local similarity measures sim_{A_i} are defined. Further, be $(w_1, \ldots, w_n)$ with $w_i \in \mathbb{R}^+$, a vector of corresponding feature weights. Then, the ***weighted average amalgamation*** *function computes the similarity between two cases c_1 and c_2 after*

$$Sim(c_1, c_2) = \frac{\sum_{i=1}^{n} w_i \cdot sim_{A_i}(c_1.A_i, c_2.A_i)}{\sum_{i=1}^{n} w_i}$$

In contrast to many other works we do not assume the feature weights to be normalised, i.e. we do not presume that it holds $\sum_{i=1}^{n} = 1$. The abandonment of that property results in a number of benefits for the implementation of feature weight individuals within FLSM's evolutionary algorithm (see Section 2.4.2). Yet, through the denominator a normalisation is contained in the definition of Sim anyway, so that it always holds $Sim(c_1, c_2) \in [0; 1]$.

2.2.2 Usage of Default Similarity Measures

Apart from defining sophisticated similarity measures – no matter in which way – a user of a CBR system may also employ so-called *default measures*. Without any doubt those can be characterised as knowledge-poor similarity measure, since no knowledge engineering effort is required prior to their usage. Consequently, they bear all the advantages and disadvantages described in the previous section.

Though it is possible to construct more complex similarity measures, many commercial CBR tools are able to use some kind of default measures by default. Since we will use the quality of a retrieval achieved by employing default measures as a reference value in the scope of this work, we give an accurate definition of default similarity measures (see also Figure 2.2).

Definition 2.6 (Default Similarity Measures)
*The **default similarity function** for a numeric attribute N, is defined as:*

$$\begin{aligned} sim_{N,default} : [(n_{min} - n_{max}), (n_{max} - n_{min})]^2 &\rightarrow [0;1] \\ (q,c) &\mapsto 1 - \frac{|q-c|}{n_{max}-n_{min}} \end{aligned}$$

*The **default similarity table** for a symbolic attribute S, is defined as:*

$$\begin{aligned} sim_{S,default} : D_S \times D_S &\rightarrow [0;1] \\ (q,c) &\mapsto \begin{cases} 1 & \text{if } q = c \\ 0 & \text{else} \end{cases} \end{aligned}$$

*The **default similarity measure** $Sim_{default}$ for a specific application domain is assembled by default similarity functions and tables, depending on the type of the respective attribute, as local similarity measures and attribute weights set to 1 for each w_i, while employing the weighted average amalgamation function to compute the global similarity.*

2.2.3 Bottom-Up Approach

The bottom-up approach to defining similarity measures represents so to speak the conventional way to perform that task. Commercial CBR tools (for a detailed evaluation of several tools see (Althoff, 1997)) dispose of complex and powerful mechanisms, in particular also of appropriate graphical tools, to model similarity measures. With their help human experts may define similarity measures according to their needs and understanding of the domain, respectively: They decide for appropriate local similarity measures, tune their parameters, assign feature weights to the attributes and choose a suitable amalgamation function.

In the upper part of Figure 2.2 we show two examples of local similarity measures that are defined manually and that might be used in a product recommendation system for personal computers. The first measure is a similarity table for the symbolic attribute RAMType, the second for the numeric attribute CPUClock. Both measures are supposed to estimate the utility of the technical properties of a given PC with respect to a user's requirements. In the lower part of the figure we contrast the default similarity measures for the respective types. Obviously, the upper, knowledge-intensive measures contain more domain knowledge. For example, the measure for attribute CPUClock "knows" that a customer will probably be satisfied, if he/she gets an even faster PC than desired (provided that other properties continue to match his/her demands), since it assigns a similarity value of 1.0, if a PC's clock rate is higher than the one specified in the query ($q < c$).

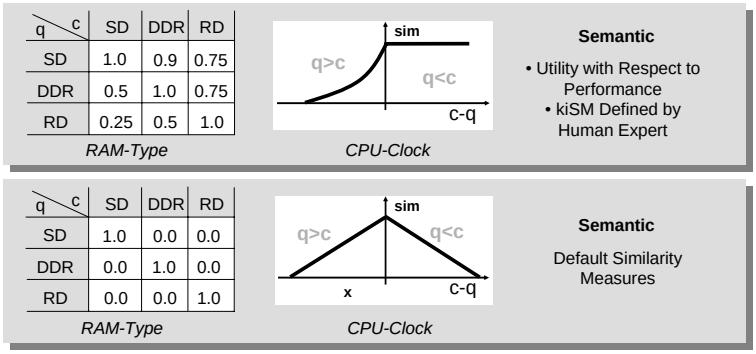


Figure 2.2: Knowledge-Intensive Local Similarity Measures and Default Measures

But who is considered to be a "human expert" defining a CBR system's similarity measure manually? On the one hand, this catchphrase could mean a knowledge engineer who is familiar with the characteristics and methods of CBR and who also knows how to define accurate similarity measures. On the other hand, it could be interpreted as referring to a domain expert, who fully understands the application domain at hand, knows exactly about its peculiarities and who is able to estimate which cases would be useful for a given query. However, each of those imaginary experts does not know much about the area of expertise of his/her counterpart.

That fact is a primary reason for some of the disadvantages of the bottom-up approach to defining similarity measures:

- A domain expert often finds it difficult to formalise his/her domain knowledge to make it usable for the definition of appropriate similarity measures. A CBR expert, on the other hand, cannot define a similarity measure without knowledge about the underlying utility function. Therefore, the problem of defining similarity measures manually bottom-up induces a communication problem between two experts of two different fields.
- The bottom-up procedure is a very time-consuming and, consequently, very expensive task.
- In complex application domains it is impossible to estimate the influence of each attribute and/or the influence of a single parameter of a measure sufficiently due to the lack of reasonable domain knowledge. Despite that a domain expert may be able to estimate the utility of a case as a whole for a specific query.
- Usually the quality of the similarity measure defined completely manually in a bottom-up manner is not validated in a systematic way. System users as well as experts often tend to blindly trust the retrieval results with their corresponding exact similarity values returned by a CBR application.

2.2.4 Top-Down Approach

An alternative strategy for defining knowledge-intensive similarity measures, that uses a Machine Learning approach, has been introduced by Stahl (2002). Opposed to the bottom-up approach of defining similarity measures manually as described in the previous section, this technique is called *top-down approach*. Its basic idea is to acquire only top-level knowledge

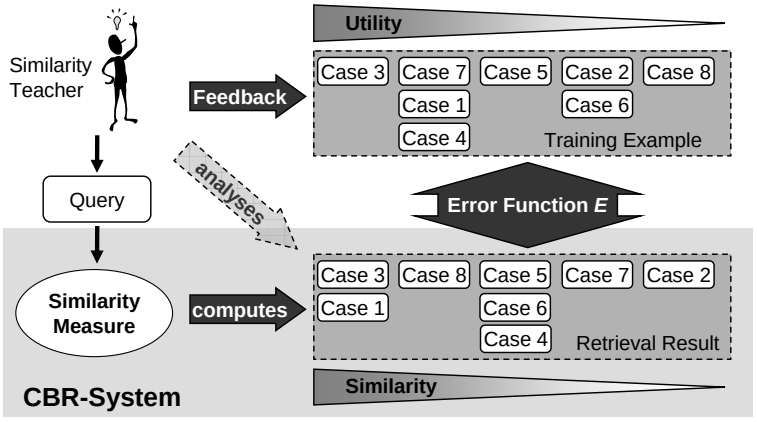


Figure 2.3: Top-Down Approach to Defining Similarity Measures

about the utility of cases for some set of problem situations. The low-level knowledge, with the help of which the accurate similarity measure is defined, shall be obtained by employing machine learning techniques.

In Figure 2.3 we illustrate the basic ideas of the top-down approach. Obviously, the approach is based on four key elements:

- The first key element is the concept of the so-called *similarity teacher*, who or which[2] is able to give feedback about the utility of cases for a specific problem situation. The teacher possesses the knowledge about the correct case order that should be returned by the CBR system for a given query – we call that knowledge *utility feedback*. Of course, the teacher does not necessarily have to give feedback about the utility of all cases contained in the case base, sometimes even information about a single case may be useful, for example, in an e-commerce scenario where the user does not buy the most similar product, but another one contained in the retrieval result.

- Second, a *CBR system* is part of the top-down scenario. Its functionality is used to acquire retrieval results for given queries on the basis of different similarity measures.

- Furthermore, an *error function* that computes the "difference" between the retrieval result based on the similarity measure used currently and the utility feedback given by the similarity teacher plays a major role. Several definitions for the error function are possible, the question which one to use for a particular learning task depends on the application scenario, on the amount of available training data, as well as on the circumstantiality of the utility feedback the similarity teacher gives (see Sections 2.3.2 and 3.3).

- The fourth key element (not shown in Figure 2.3) is represented by the *learning algorithm* whose task is to minimise the error function by modifying the similarity measure to be used.

[2]The similarity teacher can be represented by either a human expert or by some kind of software module disposing of the necessary knowledge.

Generally, the actual learning task, i.e. minimising the value of the error function by making use of other, optimised similarity measures, may be performed by any accurate learning algorithm. Within in the learning framework FLSM the focus (concerning the algorithmic realisation) is laid on a gradient descent approach and especially on evolutionary algorithms to perform that task.

The top-down approach of defining similarity measures can easily be integrated into the process model commonly accepted for Case-Based Reasoning. The *CBR cycle*, as introduced by Aamodt and Plaza (1994), may be refined so that it includes the functionality that is required to learn similarity measures from utility feedback (Stahl, 2003).

2.3 Realisation of the Learning Framework

As already described, FLSM assumes a similarity teacher that is able to provide high-level knowledge in a specific form about the utility of cases. Then, it is the goal of an appropriate learning procedure to construct a similarity measure that approximates the cases' utilities for all (or at least for most) queries as good as possible. In this section we introduce the necessary concepts and formalisations that are mandatory for learning similarity measures from utility feedback. Furthermore, we focus on the general learning scenario that is realised by FLSM.

2.3.1 Training Data

Utility feedback can be understood as a function u that is only partially known. The only available information about that function comes from the similarity teacher having some implicit knowledge of u. Moreover, utility feedback always refers to one single query q, that way stating how the "correct" retrieval result (at least in part) should look like, when accomplishing a retrieval for q.

Definition 2.7 (Training Example)
*For a query q, a case base CB and a utility function u, implicitly given by the similarity teacher, a **training example** is defined as an ordered set*

$$TE_u(q) = [(c_1, u(q, c_1)), (c_2, u(q, c_2)), \ldots, (c_f, u(q, c_f))]$$

where f is the number of cases for which feedback is given. For each $i \in \{1, \ldots, f\}$ it holds:

- $c_i \in CB$
- $u(q, c_i) \geq 0$
- $u(q, c_i) \geq u(q, c_j)$ *for all* $j > i$

FLSM distinguishes between two types of utility feedback, *cardinal* and *ordinal feedback*. For the first type it holds $u(q, c_i) \in [0; 1]$ for all $i \in \{1, \ldots, n\}$. In the latter case, however, $u(q, c_i) \in \mathbb{N}$ is used as an index only, determining the (partial or complete) correct case order.

For the learning algorithm, whose task is to approximate the utility function, it is insufficient to get information about one point of the problem space, i.e. for one single query q only. Instead, learning must be based on a larger set of training examples in order to obtain reasonable learning results. This leads to the following definition.

Definition 2.8 (Training Data) *Given a set of queries $Q = \{q_1, \ldots, q_s\}$, a case base CB and a utility function u, a set of training examples $TE_u(q_i)$*

$$TD_u(Q) = \{TE_u(q_1), \ldots, TE_u(q_n)\}$$

*is called **training data**.*

2.3.2 Error Functions

The basic idea of defining similarity measures in a top-down manner (see Section 2.2.4) is to minimise the deviation of the retrieval result that can be obtained, when carrying out a retrieval based on a specific similarity measure, and the (partially known) correct case order as given by the training data (based on the similarity teacher's utility feedback). So, a simple training example can be considered as a partial definition of an optimal retrieval result, i.e. the cases in a retrieval result based on an optimised similarity measure should be in the same order as they are in the training example.

In order to enable a learning algorithm to minimise the deviation mentioned above, an adequate definition of an error function is indispensable, measuring that deviation and thus expressing the quality of the respective retrieval result and, therewith, of the corresponding similarity measure that lead to the retrieval result.

Here, we give a simplified[3] definition of an error function performing that task. Its objective is the computation of an error value measuring the differences between a retrieval result determined by the similarity measure at hand and a training example based on the feedback of the similarity teacher.

Definition 2.9 (Ordinal Error Unit)
*Let q be a query, $TE_u(q)$ be the corresponding training example, where $(c_i, u(q, c_i)), (c_j, u(q, c_j)) \in TE_u(q)$. Further, let Sim be a similarity measure. An **ordinal error unit** for cases c_i and c_j $(i \neq j)$ is defined as*

$$eu_o(TE_u(q), Sim, (c_i, c_j)) = \begin{cases} 1 & \text{if } Sim(q, c_i) < Sim(q, c_j) \text{ and } u(q, c_i) \geq u(q, c_j) \\ 0 & \text{else} \end{cases}$$

With the help of the definition of an error unit, we can define the desired error function:

Definition 2.10 (Index Error)
*For a similarity measure Sim, a query q and a corresponding training example $TE_u(q)$, the **index error** is defined as*

$$E_I(TE_u(q), Sim) = \sum_{i=1}^{f-1} \sum_{j=i+1}^{f} eu_o(TE_u(q), Sim, (c_i, c_j)) \cdot \frac{i + \alpha}{i}$$

where $\frac{i+\alpha}{i}$ is called error-position weight to be influenced by the parameter α. f represents the number of cases for which feedback is contained in the training example.

[3] Simplified with respect to the "Ordinal Evaluation Function" as defined by Stahl (2003).

Since the index error refers to the quality of a retrieval for one single query only, we need an extended definition, allowing the evaluation of an entire training data set:

Definition 2.11 (Average Index Error)
Given a set of training queries $Q = \{q_1, \ldots, q_s\}$*, corresponding training data* $TD_u(Q) = \{TE_u(q_1), \ldots, TE_u(q_s)\}$ *and a similarity measure* Sim*, the* ***average index error*** *induced by* Sim *with respect to* Q *is defined as*

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

The average index error represents a measure that is able to reflect the quality of a number of retrieval results based on a specific similarity measure Sim and some set of training queries Q. The term "quality" here refers to the degree to which the retrieval results accord to the utility feedback specified by the similarity teacher.

The error functions given here make use of ordinal case order information contained in a training example only. We will describe an extension facilitating the employment of *cardinal feedback*, too, in Section 3.3. There, we will focus on the concept of *solution similarity* (see also (Stahl and Schmitt, 2002)) as well as on classification domains, aiming at a broader applicability of FLSM.

2.3.3 Implementing the Framework

In this section we want to shortly review FLSM's realisation of the approach to improve similarity measures with the help of the training data defined in the previous section, whereas in the following section we will focus in depth on the machine learning algorithms performing that task.

Motivation

The learning process can be considered as a compilation process that transfers knowledge from the training data into the similarity measures. To stress the possible advantages of optimising similarity measures, we want to present an exemplary application scenario.

Assume a case-based product recommendation system for personal computers (e-commerce scenario). Given a user-specified query, that system can retrieve several base products, i.e. base PCs, that match the query best. Moreover, the system disposes of a set of *adaptation rules* with which the base PCs may be customised with respect to the query. For instance, a bigger hard-disc may be installed or a CD drive may be exchanged for a DVD-CD combo drive.

Concerning the similarity measures used in this scenario it is important to note, that when a CBR system provides possibilities of case adaptation, it is in general insufficient, if the system just retrieves those cases that match the query best. Instead, to obtain reasonable, high-quality retrieval results, the similarity measure has to pay attention to the adaptability of cases, too. Hence, a transfer or compilation process, respectively, that carries forward knowledge from the adaptation container (see Section 2.1) to the retrieval knowledge container is needed, as shown in Figure 2.4.

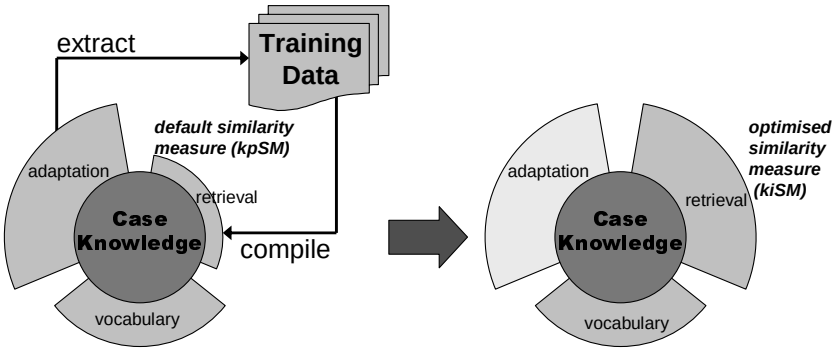


Figure 2.4: Knowledge Compilation: From the Adaptation to the Retrieval Container

What is the advantage of that knowledge transfer? One might argue that the system can apply adaptation to *all* of its retrieved cases and that way – so to say in a brute force manner – determine the case that has the highest utility with respect to the current query after having been adapted. However, case adaptation is in general a rather time-consuming task. Assuming an e-commerce scenario with hundreds or thousands of cases (base products), the process of applying case adaptation to all of them in order to find the most useful one, may take quite a long time. During that time, which the visitor of the respective web shop perceives as an annoying delay, the user may get frustrated waiting for his/her search result and leave the web site.

Apparently, an improved similarity measure in this scenario would rank those cases higher, i.e. among the top cases, in the retrieval result which have the highest utility after having been adapted. As a result, then it is sufficient to apply case adaptation to only a few cases of the retrieval result returned – in the best case only to the retrieval result's top case. Accordingly, the time needed for case adaptation and thus to find the really most useful case for the query at hand can be decreased drastically.

For a more detailed description of this application scenario and of the procedure to create appropriate training data sets, i.e. the extraction of knowledge from the adaptation knowledge container into a set of training examples (see Figure 2.4), the reader is referred to Stahl (2002).

The General Learning Scenario

The realisation of the learning process, as illustrated in Figure 2.5, can roughly be divided into two parts. The first part is represented by the similarity teacher already introduced, with which the mandatory training data is created. We assume that it has some implicit knowledge about the utility of one or more cases for a specific query (utility feedback). The queries for which the training examples are built may be generated by the similarity teacher as well or may be provided by the application environment. Depending on the application domain and on the realisation of the similarity teacher it might be necessary that a similarity-based retrieval has to be carried out in order to enable the teacher to give feedback – if not, then the similarity teacher must have access to the cases in the case base, at least (gray, dashed arrow). The result of the similarity teacher's feedback is a training data set consisting of one training example for each query.

The second key part is depicted by the concept of the *similarity learner*. Provided with

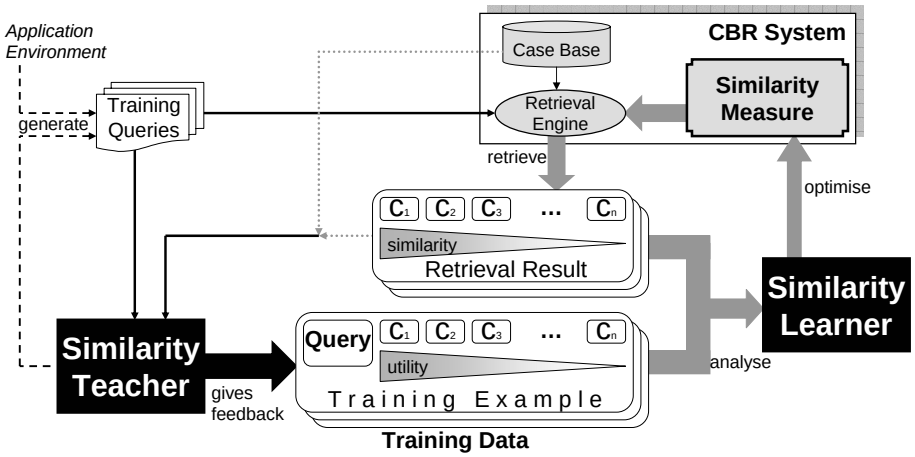


Figure 2.5: The General Learning Scenario

a set of training examples, the learner tries to minimise the deviation between the correct case orders (partial orders) contained in the training examples and the case orders that arise when the CBR system carries out a retrieval based on some similarity measure. It does so by modifying the similarity measure, that is used by the CBR system, according to some strategy. The accurate learning algorithms used for changing the similarity measures are described in detail in Section 2.4.

The informal term "deviation" used above, here, of course refers to the value of an appropriate error function that shall be minimised, e.g. the average index error as formally defined in Definition 2.11. Consequently, the quality (with respect to the utility function implicitly given by the similarity teacher) of the similarity measure, and so the quality of the retrieval results it returns, is supposed to increase.

Optimisation Loop

The optimisation procedure yielding to improve the similarity measure can according to Stahl (2003) be divided into the following steps.

1. **Initiate Retrievals**
   Based on the current similarity measure a retrieval for each query contained in the training data is carried out.

2. **Evaluate Similarity Measure**
   The retrieval results obtained in Step 1 are compared to the (partially known) correct retrieval results as contained in the training data. That means, with help of the error function used, the quality of the currently considered similarity measure is evaluated.

3. **Check Stop Criterion**
   Implemented as an optimisation loop, the learning process may stop, if a certain quality of the similarity measure has been reached (e.g. the value of the error function falls below a certain threshold) or if it has run for a specified time (e.g. a maximal number of modifications has been applied to the similarity measure used).

4. Modify Measure

If the stop criterion is false, the current similarity measure needs further modifications improving its quality with respect to the training data. The respective software component performing the modifications my also employ information about the measure's structure as well as information about the error function to guide the modification process.

After having introduced FLSM's optimisation loop to learn similarity measures, we now can integrate the main subject of this thesis into that loop: We intend to change and enhance the optimisation loop's fourth step by the means of incorporating background knowledge so that we enable FLSM to achieve better quality improvements or the same improvements in a shorter time (stop criterion becomes true earlier) or with a smaller amount of training data, respectively.

In Figure 2.6 we show an illustration of that enhanced optimisation loop. In particular, the modification of the current similarity measure shall be based on several kinds of additional knowledge – the questions on what kind of knowledge to use[4] and on how to acquire that background knowledge will be discussed later.

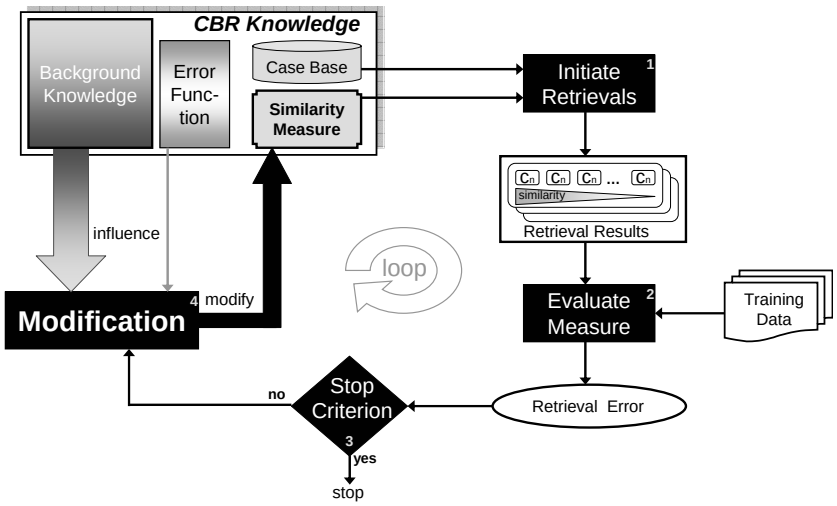


Figure 2.6: Optimisation Loop – Enhanced by Background Knowledge

2.4 Learning Algorithms

As denoted in the previous section, the crucial element within the general learning scenario performing the modification of similarity measures is represented by the similarity learner. The framework for learning similarity measures FLSM focuses on three different types of learning algorithms to be used as similarity learner.

On the one hand, it employs a *gradient descent approach* to learn the similarity measure's global attribute weights. This technique makes use of information about the shape of the error

[4]Information on the shape of the error function, as already employed by FLSM can also be interpreted as additional knowledge guiding the learning process.

function's gradient in order to direct the search into regions of the search space that promise to feature smaller values of the error function.

On the other hand, it makes use of a standard genetic algorithm (see also Section 2.4.1) to learn the attribute weights.

Third, FLSM suggests the usage of *evolution strategies* (ES). With the help of that learning strategy it facilitates not only the learning of feature weights, but also the learning of local similarity measures. In the scope of this work we will only concentrate on the latter learning technique due to the following reasons:

- The application of evolution strategies allows the optimisation of more than one element of the similarity measure representation (not restricted to feature weights). Though currently only implemented to learn attribute weights and local similarity measures, FLSM may be easily extended to also change and improve the amalgamation function (see Definition 2.5) as the third part of the similarity measure representation formalism.

- While the gradient descent approach already incorporates some kind of background knowledge – it uses information on the error function's shape to steer the search process –, evolutionary algorithms are more reliant on improvements occurring by chance. Hence, the idea of incorporating background knowledge to improve the learning process seems to be very promising, when applied to evolution strategies.

- Although it cannot be guaranteed, evolutionary learn techniques are in general less fragile to end up in a local minimum of the error function. In other words, learn strategies based on the principle of evolution have proved to provide powerful and robust mechanisms for the search for an optimum in complex search spaces.

2.4.1 Modelling Evolution

Though not knowing about the underlying genetics, Charles Darwin (Darwin, 1859) in the middle of the 19th century already identified the basic principles of natural evolution:

- the reproduction cycle
- the natural selection
- the diversity by variation

In Figure 2.7 we present an abstraction of the generational loop as implemented in FLSM. Apparently, all of Darwin's principles of evolution have found entrance into FLSM as well.

After an initial population has been created and evaluated (we will specify what the individuals within a population are in Section 2.4.2) an evolutionary loop corresponding to the reproduction cycle of natural evolution is entered. In its first stage a set of descendants is generated. Here, the diversity of variation is not only reached by mutations, but also by means of bisexual reproduction. Bisexual inheritance combines the ancestors' genomes in some way to create offspring – mostly by setting one or more crossover points in the genome of the parents. After the subsequent evaluation stage, a selection according to Darwin's principle of "survival of the fittest" takes place. Finally, a stop criterion determines whether the evolutionary process ought to be stopped or continued.

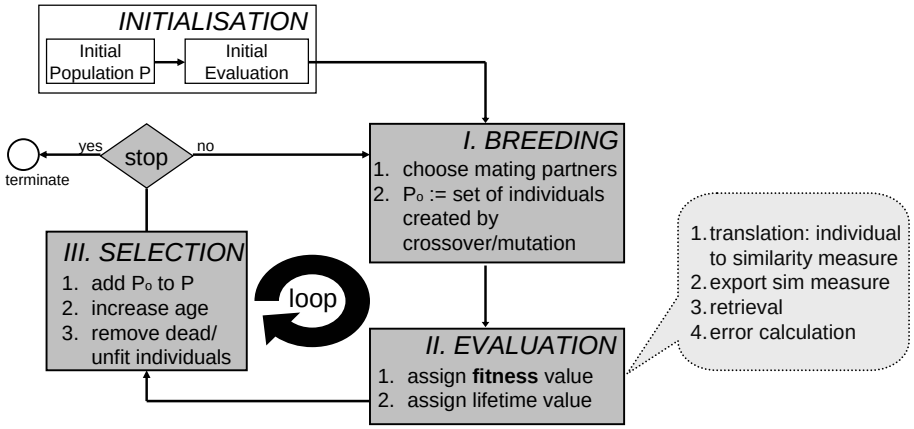


Figure 2.7: Evolution Strategy: Control Algorithm

Evolutionary Algorithms

According to Heitkoetter and Beasley (2001) "Evolutionary Algorithms" is an umbrella term that is used to circumscribe computer-based problem solving systems that make use of computational models of some of the known mechanisms of *evolution* as key elements in their design and implementation. In Figure 2.8 the variety of evolutionary algorithms that has been proposed over the past decades is outlined.

Of course, all these specialisations of evolutionary algorithms share a lot of commonness, many base principles are the same for each kind of EA – they are search algorithms based on the mechanics of natural selection, natural genetics, and the principle of the "survival of the fittest". Nevertheless, there are several differences, too, making each form of EA unique.

Genetic Algorithms (GA) represent a model of Machine Learning that derives its behaviour from some of the mechanisms of evolution in nature. Here, a population of individuals represented by their chromosomes is maintained, essentially a set of bit strings. Those individuals go through a process of simulated evolution, using crossover and mutation to create offspring.

Evolutionary Programming (EP) is a stochastic optimisation strategy similar to GAs. However, its main focus is placed on the behavioural linkage between parents and offspring, instead of seeking to emulate specific genetic operators as observed in nature.

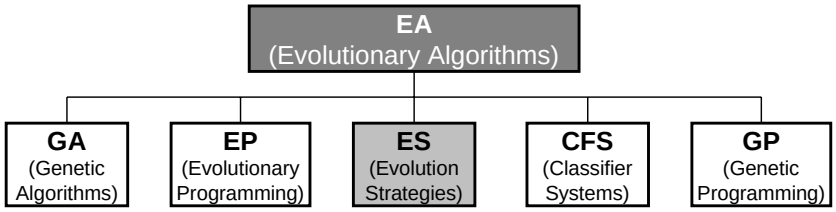


Figure 2.8: Variety of Evolutionary Algorithms

Classifier Systems (CFS) can be characterised as an offspring of John Holland's basal book on "Adaptation in Natural and Artificial Systems" (Holland, 1975). They can be seen as one of the early applications of GAs. One basic property of this type of evolutionary algorithms is that they adapt their behaviour toward a changing environment.

Genetic Programming (GP) depicts an extension to the genetic model of learning into the space of computer programs. Thus, the objects that constitute the population are not individuals composed of chromosomes that encode possible solutions to the problem at hand, they are programs themselves, which (when being executed) are the candidate solutions to the problem.

The evolutionary learning algorithms used by FLSM can be best classified as an *evolution strategy* (ES)[5]. Michalewicz (1996) refers to evolution strategies also as "evolution programs". The basic idea here is to use more sophisticated data structures (very often real numbers) as elements of an individual's genome and to apply specialised genetic operators for offspring creation, while still exploiting the evolutionary principle of (standard) genetic algorithms. An obvious advantage of using ES is that they allow to optimise the original data structures directly, so that a complex mapping to or from bit strings, respectively, is excrescent. In the following paragraphs we want to take a closer look at evolution strategies and on how they are used within FLSM. For more details on the basics and several applications of evolution strategies the reader is referred to Kursawe (1993), Michalewicz (1996) and Schwefel (1977).

Evolution Strategies

It is a main characteristic of evolution strategies to model evolution at the level of phenotypes – in contrast to, for instance, genetic algorithms which model evolution at the level of genotypes. As described, thereby the advantage arises that a mapping to or from genetic information strings is not necessary.

Another important feature of ES is the ability to encode (and optimise as well) so-called *strategic parameters*. Those mainly influence the behaviour of the mutation operators, they may change their reach, their orientation[6] or their frequency of use. That way the evolution strategy is equipped with a self-adapting mechanism that to some extent models the environment and corresponds to the speed or to changes in the evolutionary development of a population. As a consequence, descendants with a better internal model, i.e. with a better set of strategic parameters, have an advantage regarding fitness and selection. Thus, the search for even fitter individuals may be accelerated. In FLSM, however, the optimisation of strategic parameters has been neglected so far, i.e. a deterministic model of the environment has been employed.

Natural selection – as the second of Darwin's principles – is a pure random process. However, in the past many modelers were misled to model selection as mating selection only (Schwefel and Baeck, 1998): The fitter individuals produce more offspring than the less fit ones. So, EP as well as GA mostly model the environmental selection by means of "tournaments" between old and new individuals (parents and offspring). Evolution strategies, on the other hand, follow a different selection paradigm. In Section 2.4.3 we focus on aspects that characterise FLSM as an evolution strategy. There we will also describe FLSM's selection strategy.

2.4.2 Representing Individuals

Obviously, for FLSM the individuals to be maintained in a population for an evolution strategy have to correspond to similarity measures. So, we need to find an appropriate representation

[5] Note, that FLSM is also capable of using a standard genetic algorithm to learn the feature weights.

[6] Here, reach refers to the distance covered by the mutations, and orientation means the direction into which the mutation changes an individual.

for attribute weights as well as an appropriate representation for local similarity measures as individuals. In this section we want to show, that we can define an adequate representation as individuals for both elements of similarity measures, to be used within an evolutionary process.

In standard genetic algorithms individuals are commonly represented as bit strings – lists of 0s and 1s – to which evolutionary operators, such as crossover or mutation, are applied. One drawback of that representation is the difficulty to incorporate constraints on the solution that a single individual stands for. Since it is one main goal of this work to improve the process of optimising similarity measures by incorporating additional knowledge during learning, it is crucial that constraints on possible solutions can be processed easily. Moreover, the calculation of the similarity between a query and a case (as depicted in Section 2.2) is mainly based on real-valued numbers. Apparently, if we had used a standard genetic algorithm with individuals represented as bit strings, a complex mapping from and to real numbers would have been indispensable. For these reasons FLSM employs an evolution strategy that makes use of "richer" data structures and applies appropriate genetic operators.

Representing Local Similarity Measures

In order to be able to learn local similarity measures as defined in Definition 2.3 and 2.4, i.e. similarity functions and tables, we have to settle how to represent the respective individuals. Further, we need a formalism that maps those individuals, as used in the evolutionary algorithm, to a local similarity measure.

Consider an arbitrary distance-based similarity function sim_A. Since sim_A is continuous in its value range D_A it is generally not possible to describe it with a finite number of parameters (in certain cases that may be possible, but in general it is not). Thus, we employ an approximation, using piecewise linear functions based on a number of sampling points:

Definition 2.12 (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 space of all similarity function individuals is denoted by $\mathbb{V}$.

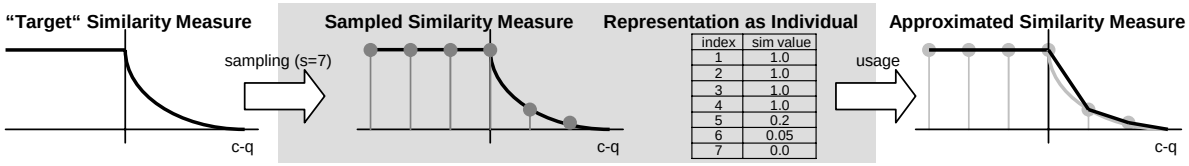


Figure 2.9: Representation of Similarity Vectors as Individuals

The sampling points are distributed equidistantly over the value range $D_A = [(A_{min} - A_{max}), (A_{max} - A_{min})]$ of attribute A. Figure 2.9 illustrates how a local similarity measure for a numeric attribute is modelled. The length s of the similarity vector 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.

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

The space of all similarity table individuals is denoted by $\mathbb{M}$.

This definition corresponds to the representation of similarity tables (see Definition 2.4), i.e. the original representation of this type of local similarity measure is used without modification by the evolution strategy. Hence, Definition 2.13 is only required to introduce the necessary notation.

Representing Weights

Although FLSM also disposes of a standard genetic algorithm to learn the feature weights and thus uses a bit string representation for the weights, the usage of an evolution strategy allows us to represent the attribute weights as a vector of real numbers. Both representations are somehow analogous, the latter one, however, is more fine-grained because no weight discretisation due to the bit string representation is necessary.

Definition 2.14 (Feature Weight Individual, Weight Vector)
Let the cases be characterised by n attributes $A_1, \ldots, A_n$. Given a global weight vector $\vec{w} = (w_1, \ldots, w_n)$, an individual I representing $\vec{w}$ is coded as a vector $V_{\vec{w}}^I$ of real numbers from the interval $[0; 1]$, i.e. $V_{\vec{w}}^I = (v_1^I, \ldots, v_n^I)$, where

$$v_i^I = \frac{w_i}{\max_{j \in \{1,\ldots,n\}} \{w_j\}}$$

The space of all feature weight individuals is denoted by the symbol $\mathbb{W}$.

That definition reveals that the representation of feature weights as individuals is essentially the same as the representation used for distance-based similarity functions, namely a vector of real numbers. We have introduced feature weights to be positive real numbers (see Definition 2.2). When creating a corresponding individual from a vector $\vec{w}$ of feature weights, the division of each weight w_i through the maximum weight within that weight vector creates a "normalised" weight vector whose entries are all from the interval $[0; 1]$. Although the entries of those vectors are interpreted differently for similarity function individuals and feature weight individuals, respectively, the specialised genetic operators we describe in the following section are exactly the same for both kinds of individuals.

2.4.3 FLSM as Evolution Strategy

In this section we want to focus on the specifics of evolution strategies. We intend to show why the evolutionary algorithm used by FLSM has to be classified as an evolution strategy, and which ES-specific settings we employ.

The $(\mu, \kappa, \lambda, \rho)$ Evolution Strategy

We want to emphasise that many, usually specialised versions of evolution strategies exist and have been applied successfully. An overview may be found in T. Baeck (1997).

The $(\mu, \kappa, \lambda, \rho)$ strategy (Schwefel and Baeck, 1998) represents an extension of the $(\mu + \lambda)$ strategy introduced by Schwefel (1981). The symbol μ denotes the number of individuals (and thus possible parents), appearing at a time in a population, and can be equated with the size of the population. The symbol λ stands for the number of all offspring created within one (synchronised) generation. In the $\mu + \lambda$ strategy the λ offspring and the μ individuals of the preceding generation are united, before – according to a given criterion – the μ fittest individuals are selected from this set of size $\mu + \lambda$.

The $\mu + \lambda$ strategy implies that each parent may live eternally, if no child achieves a better or at least the same quality. The $(\mu, \kappa, \lambda, \rho)$ strategy introduces a maximal life span of $\kappa \geq 1$ reproduction cycles (iterations) for the individuals. Moreover, that strategy allows a free number ρ of parents to be involved in a reproduction. FLSM confines itself on selecting no more than two parents to be involved in a reproduction ($\rho \leq 2$): In the case of reproducing via *reproducing mutation* FLSM randomly picks one parent and in the case of breeding via one of the *crossover operators* (see Section 2.4.4) it randomly chooses two individuals as ancestors and creates exactly one descendant.

Formally, the $(\mu, \kappa, \lambda, \rho)$ evolution strategy is defined as a 19-tuple (according to Schwefel and Baeck (1998)):

$$(\mu, \kappa, \lambda, \rho)ES := (P^{(0)}, \mu, \kappa, \lambda, \mathbf{rec}, p_r, \rho, \gamma, \omega, \mathbf{mut}, p_m, \tau, \tau_0, \delta, \beta, \mathbf{sel}, \zeta, t, \epsilon) \tag{2.1}$$

Formalising ES Parameters

The objective function, in the context of evolutionary algorithms often also called fitness function, shall be denoted by $f(I)$, where I stands for an individual. How to assign appropriate fitness values to individuals is described in Section 2.4.5.

Definition 2.15 (Optimisation Problem)
*Let $\mathbb{I} = \{\mathbb{V}, \mathbb{M}, \mathbb{W}\} \subset [0;1]^n$ the space of all producible individuals in the context of similarity measure optimisation. Further, be $f(I)$ the **objective function** assessing the fitness of individual I. Then, the **optimisation problem** to be solved by the evolution strategy is defined as follows:*

$$optimise\{f(I)|I \in \mathbb{I}\}$$

In the following, we want to discuss in short how we implemented, slightly changed (w.r.t. their original definition) or extended each of the 19 elements, specified in Equation 2.1, thereby not focusing on the respective notion in detail.

a) Start Population $P^{(0)}$ The start population, that is used for the optimisation of a particular part of the entire similarity measure (e.g. a population for a local similarity measure) is represented by a set of μ corresponding individuals, that are generated randomly. Thus, it holds: $P^{(0)} = (I_1, \ldots, I_\mu) \in \mathbb{I}^\mu$.

b) Main Parameters μ, λ, κ, ρ As already mentioned, μ stands for the number of individuals within a population, λ for the number of offspring created per generation and ρ for the number of individuals that is (maximally) allowed to partake in a reproduction.

Concerning the life span κ that is assigned to an individual, FLSM follows a dynamic strategy (Michalewicz, 1996): It specifies two thresholds, κ_{min} and κ_{max} (with $\kappa_{min} \leq \kappa_{max}$), that correspond to the minimal and to the maximal lifetime that can be allocated to an individual. Based on these thresholds, on its *lifetime assignment policy* and on the individual's fitness, FLSM determines an adequate lifetime value κ.

We distinguish between *proportional lifetime allocation policy* and *linear lifetime allocation policy*. The former computes κ for an individual I according to

$$\kappa := \frac{f(I)}{2 \cdot f_{avg}} \cdot (\kappa_{min} - \kappa_{max}) + \kappa_{max} \tag{2.2}$$

and the latter one according to

$$\kappa := \frac{f(I) - f_{best}}{f_{worst} - f_{best}} \cdot (\kappa_{min} - \kappa_{max}) + \kappa_{max} \tag{2.3}$$

At this, f_{worst} and f_{best} correspond to the fitness of the most unfit and fittest individual in the current population, respectively. f_{avg} stands for the average fitness of all individuals in the population: $f_{avg} := \frac{1}{\mu} \cdot \sum_{I \in P} f(I)$. Note, that a very non-uniform distribution of $f(I)$ over the interval $[f_{best}, f_{worst}]$ might impair the linear lifetime allocation policy. If an uneven distribution distorted the lifetime allocation too much (we did not encounter that case during testing), an appropriate heuristic to disregard the responsible fitness outliers causing the imbalance would have been necessary.

Moreover, it should be noted that, on the one hand, a setting of $\kappa_{max} = 1$ results in discarding all parents regardless of their quality. This complies with the μ, λ-selection as used in traditional evolution strategies. On the other hand, a setting of $\kappa_{min} = \infty$ guarantees that parents are only to leave the population, if they are outperformed by their children, complying with the $\mu + \lambda$-selection in traditional evolution strategies.

c) Genetic Operators rec and mut Since the definition of appropriate specialised genetic operators influences the behaviour and the capabilities of any evolutionary technique on a grand scale, we devote a separate section to that topic. The recombination operators **rec** – we will call them *crossover operators* – and the mutation operators **mut** that we designed and implemented for FLSM are described in detail in Section 2.4.4.

d) Parameters for Genetic Operators p_r, p_m, γ and ω The first two parameters represent vectors of probabilities to influence the behaviour of the recombination and mutation operators. While ω is responsible for selecting a specific type of recombination operator, γ determines the number of crossover sites in a chromosome. Our implementation does not make use of those parameters explicitly, instead in Section 2.4.4 we point out by which means the behaviour of genetic operators is influenced and by which probabilities they are chosen.

e) Strategic Parameters τ, τ_0, δ and β As already pointed out, FLSM desists from using and optimising strategic parameters.

f) Selection Criterion sel and Tournament Participators ζ The selection operator **sel** : $\mathbb{I}^{\mu+\lambda} \rightarrow \mathbb{I}^{\mu}$ implemented in FLSM first creates the union $\hat{P}^{(T)}$ of the current population $P^{(T)}$ and the λ offspring created via applying crossover and mutation operators. Then, the selection operator forms the population of the next generation via $P^{(T+1)} := \mathbf{sel}(\hat{P}^{(T)})$. The application of **sel** selects the μ best (fittest) individuals from $\hat{P}^{(T)}$, that still have a positive remaining lifetime.

An alternative realisation of the selection operator – based on *tournaments* in which the new individuals and their ancestors have to compete with each other over a couple of turns – is presented by Schwefel and Baeck (1998). For that realisation the parameter ζ (number of participators in a tournament) plays a significant role.

g) Termination Criterion t and Accuracy ϵ The termination criterion of FLSM may become true, if the value of the objective function falls below a certain threshold, if the search process has converged so that no further improvement is likely or simply if a specific amount of computation time has been consumed (i.e. a certain number of evolutionary generations has been processed). For the purpose of comparing sundry runs of optimisation processes using different prerequisites we will mainly employ the last-mentioned realisation of a stop criterion.

The parameters for numeric accuracy ϵ is disregarded by FLSM.

2.4.4 Specialised Genetic Operators

Genetic operators are responsible for the creation of new individuals and thus have a significant influence on the way a population develops. Using parts of the genome of two or more parent individuals, new ones are composed. Further, random mutations can be applied to a recently generated individual (adapting mutation) or to an existing individual in order to form a new one (reproducing mutation). The operators used in FLSM are quite different from classical ones since they operate in a different domain (real-valued instead of binary representation). However, because of underlying similarities, we divide them into the two standard groups: mutation and crossover operators.

Mutation Operators

Operators of this class are the same for all kinds of individuals (similarity function individuals, similarity table individuals and feature weight individuals) we are dealing with. They change one or more values of a similarity vector V_A^I, weight vector $V_{\vec{w}}^I$ or similarity 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. Since any matrix can be understood as a vector, we describe the mutation operators' functionality for similarity vectors (for a local similarity measure for attribute A) 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 chance 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 that are 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

after $\hat{v}_j^I = v_j^I \pm (1 - r^{(1-\frac{t}{T})^2})$, where t is the current age of the population at hand, T its maximal age, and r a random number from $[0, 1]$ (if $\hat{v}_j^I > 1.0$ or $\hat{v}_j^I < 0.0$, then of course the new value for v_j^I is restricted to 1.0 or 0.0, respectively). Hence, this property makes the operator search the space more uniformly at early stages of the evolutional process (when t is small) and rather locally at later times[7].

- *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. Assume, the local similarity measure for the attribute CPUClock (see Figure 2.2) as the optimisation goal and an individual I for whose similarity vector it holds: $v_i^I < 0.9$ for all $i > \frac{s}{2}$. Here, a mutation, that increases the similarity value for several neighbouring sampling points rights of the y-axis, will bring I nearer to its optimisation goal.

Crossover Operators

Applying crossover operators, a new individual in the form of a similarity vector or matrix is created using elements of its parents. Though there are variations of crossover operators described that exploit an arbitrary number of parents (cf. Section 2.4.3), we rely on the traditional approach using exactly two parental individuals, I_a and I_b.

- *Simple crossover* is defined in the usual way: A "split point" for the particular similarity vector or matrix is chosen randomly. The new individual is assembled by using the first part of the parent I_a's similarity vector or matrix and the second part of parent I_b's.
- *Arbitrary crossover* represents a kind of multi-split-point crossover with a random number of split points. Here, we decide by random chance for each component of the offspring individual whether to use the corresponding vector or matrix element from parent I_a or I_b.
- *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 $m_{ij}^{I_{new}} = \frac{1}{2} m_{ij}^{I_a} + \frac{1}{2} m_{ij}^{I_b}$ for all $i, j \in \{1, \ldots, d\}$.
- *Line/row crossover* is employed for similarity tables, i.e. for symbolic attributes, only. Lines and rows in a similarity matrix contain coherent information, since their similarity entries refer to the same query or case value, respectively. Therefore, splitting a line/row by simple or arbitrary crossover may lead to less valuable lines/rows for the offspring individual. So, we define line crossover as follows: For each line $i \in \{1, \ldots, n\}$ we randomly determine individual I_a or I_b 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.

2.4.5 Assessing Individual Fitness

Since any kind of optimisation technique that tries to model characteristics of natural evolution is based on the principle "survival of the fittest", the choice of an appropriate *fitness function*

[7] The sign $\pm$ indicates, that the alteration is either additive or subtractive, the decision about that is made randomly as well.

is crucial. For the representation of individuals that we have chosen and introduced in Section 2.4.2 we must associate each weight vector and each local similarity measure, respectively, with a fitness value. Intending to optimise the individuals' fitness, our evolution strategy is then supposed to evolve its populations of weight, similarity vector and similarity matrix individuals to be of higher quality.

For FLSM's evolution strategy the presumed error function defined in Definition 2.11 already represents a convenient fitness function. The lower the average index error, the higher the fitness of the respective individual, here a particular similarity measure. Of course, the average index error is not the only adequate fitness function. Evolutionary algorithms do not require fitness functions with certain properties – any error function corresponding to the retrieval error induced by a specific similarity measure may be used to assess an individual's fitness. We will focus on adequate possibilities of assessing an individual's fitness again in Section 3.3.4.

Concerning the proceeding to compute the fitness of a specific individual, we distinguish between four successive steps (already illustrated in Figure 2.7):

1. The respective individual has to be translated to a corresponding element of the similarity measure representation. So, for example, a similarity function individual (cf. Definition 2.12) is translated into a distance-based similarity function, whereas a feature weight individual (cf. Definition 2.14) is translated into a vector of feature weights.

2. The weight vector or local similarity measure created in Step 1 must be exported to the CBR tool CBR-Works[8] that we are using.

3. Based on that similarity measure, a retrieval for all the queries, that are contained in the training data at hand, is carried out.

4. The quality of the retrieval results returned is determined. That means that we analyse how much these retrieval results differ from the correct retrieval result which are contained within the training data.

As already mentioned, the error value computed that way, is used directly as the respective individual's fitness.

2.4.6 Sequential vs. Parallel Processing

Given a case model for a particular application domain and a set of training data, the evolutionary algorithm designed and implemented in FLSM is capable of optimising two of the three major elements of the similarity measure representation, namely the attribute weights and all local similarity measures required for the attributes. The optimisation of the amalgamation function, however, is neglected. Instead, always a weighted sum according to Definition 2.5 is employed, as that function is accurate in most domains.

It is important to note, that FLSM is able to learn several local similarity measures as well as the corresponding feature weights *simultaneously*. Given a set of attributes $A_1, \ldots, A_n$, for which local similarity measures and weights are to be learnt, a population P_i is generated for each attribute A_i as well as an additional population $P_{\vec{w}}$ of weight individuals. Thus, it is

[8] CBR-Works and its successor orenge are products of the knowledge management enterprise empolis, formerly tec:inno.

possible to optimise the entire similarity measure even in more complex domains, where the global similarity between a case and a query is composed of several weighted local similarities.

Since the effects of feature weights and local similarity measures interact with each other, one has to thoroughly think about possible strategies to integrate the respective optimisation process. Stahl (2003) basically distinguishes two approaches for an integration – sequential and parallel processing – and describes the results obtained, when comparing them with each other using FLSM.

For *sequential processing* the optimisation of weights and local measures is separated. Hence, FLSM first optimises the feature weights using fixed local measures. Afterwards the local measures are learnt on the basis of the optimised weight vector obtained before. The order of processing may, of course, be inverted.

Concerning the learning of local similarity measures, a sequential learning approach, that optimises all local similarity measures one after another, seems to be a promising idea. An advantage of that proceeding would be the opportunity to decompose the entire optimisation process into several optimisation loops that are responsible for learning feature weights and single local similarity measures, respectively, and that could be executed one after another. However, the major drawback of the sequential approach is the need of some initial measure. Optimising only one single part of the entire similarity measure, the similarity learner already needs the remaining parts (e.g. all other local measures and an appropriate weight vector), in order to evaluate the fitness of an individual, i.e. when computing the retrieval error (see Section 2.4.5).

The alternative approach to learning the different elements of the similarity measure representation is called *parallel processing*. Here, the evolutionary optimisation technique has to manage *meta individuals* consisting of one atomic individual for the weight vector (cf. Definition 2.14) and one atomic individual for each local similarity measure (cf. Definition 2.12). The corresponding genetic operators for meta individuals are supposed to employ the respective atomic operators (see Section 2.4.4) for the atomic individuals involved.

Unfortunately, the usage of meta individuals leads to a very huge search space and implies that a much higher number of evolutionary generations has to be processed in order to find an optimum and thus to obtain satisfying results. This may lead to infeasible demands regarding computational resources.

Due to the disadvantages of both forms of processing described above, we suggest to make use of a mixture of them that we call *round-robin processing*. Here, each element of the similarity measure representation, and hence each population that is about to be enhanced, is optimised for a fixed and relatively small number of evolutionary generations – called round-robin size ϱ – only. After that the control algorithm proceeds to the next population and allows it to evolve for the same number of generations. When all populations involved have undergone the same number of evolutionary cycles ($k \cdot \varrho$ with $k \in \mathbb{N}$), the control algorithm starts over with the first of its populations again. So, the entire optimisation process is decomposed into several optimisation loops that run on a time sharing principle.

We need to stress that, when setting ϱ to the number of generations a population is maximally permitted to evolve, the round-robin processing is the same as sequential processing. On the other hand, choosing the minimal value for ϱ ($\varrho = 1$) leads to an increased overhead due

to the higher number of "context switches"[9] that occur when the control algorithm stops one and continues another optimisation loop. We will focus in more detail on that issue in Section 5.4 as well as in Section 5.5 where we introduce a refinement of the round-robin processing strategy.

2.5 Motivation for Enhancing the Learning Process

The framework for learning similarity measures (FLSM) presented so far, with its employment of evolution strategies as learning technique, represents a promising approach to learn accurate similarity measures. It must be stressed that it is capable – unlike many previous approaches that mostly focused on learning feature weights – of learning all fundamental parts of the entire similarity measure representation. While a weighted average is sufficient as amalgamation function in most cases, especially the optimisation of local similarity measures is of major impact.

According to Stahl (2001, 2002) a similarity assessment for a given problem situation and a set of cases, that really reflects a case's *utility* for a query, is of crucial importance for a successful application of any CBR system. Therefore, an automated acquisition of a CBR system's similarity knowledge via FLSM – and that way avoiding the drawbacks of defining similarity measures manually with the bottom-up approach (cf. Section 2.2.3) – may become an important aspect, when developing a CBR application.

Broader Applicability

Being of such high importance, it is desirable that the application of FLSM is not restricted to a couple of application domains only. Instead, it should be guaranteed, that FLSM can handle and learn similarity measures in many, if not all, application domains. The work of Stahl (2003) introduces a the successful employment of that framework for learning similarity measures for a product recommendation and customisation scenario (see also Section 2.3.3). Moreover, it is described how FLSM can be applied to learn customer preferences in an e-commerce scenario (web shop selling cars). In both cases the feedback employed is ordinal, i.e. the similarity teacher provides (partially known) correct case orders as utility feedback (cf. Section 2.3.2).

In practice, however, CBR has been and is very often used for classification tasks. Here, the class membership of a given query has to be predicted. Accordingly, the feedback from a teacher can only be represented by the actual class membership, which means that the feedback is less substantial and voluminous than in the cases mentioned above. In the next chapter we want to present several ideas on how that less yielding feedback can be employed anyway in order to obtain qualitative learning results for classification domains as well.

Improved Performance

The advantages of applying machine learning techniques that try to model some of the aspects of natural evolution are well-known. They are robust and powerful search strategies, they can be employed for various types of optimisation tasks, they do not make considerable demands

[9] An analogy to the context switches as known from field of operating systems and system software cannot be denied.

on the error function to be minimised and, finally, the learning results depend mainly on the quality of the training data only.

Regrettably, evolutionary algorithms are also afflicted with a number of shortcomings. As natural evolution teaches, lots of time are required for life to begin and to evolve. This means, evolution based on the metaphorical "struggle for life" (Darwin, 1859) is a long-lasting process. This is also true for simulated evolution modelled by an evolution strategy. Of course, we do not deal with millions of years – but a single evolutionary generation for the scenario described in Section 2.3.3, in which all the involved elements of the similarity measure representation, that are represented as individuals, are brought forward by one cycle, takes about 2.5 minutes (on a P-IV 1.8 GHz machine, on the basis of $|TD| = 200$).

Thus, a reduction of the number of evolutionary cycles required to achieve clear improvements or a reduction of the time needed to conduct one single generation is a desirable target of this work.

Another drawback not just of evolution strategies, but of many machine learning algorithms is, that finding the global optimum of the objective function cannot be guaranteed in general. Furthermore, (in the case of using EA) a found solution yielding an extraordinarily good value of the error function may go lost during the evolutionary process due to the fact that the corresponding individual dies without having bequeathed its genetic heir appropriately. Nevertheless, on the one hand this is part of the evolution paradigm: Specialised individuals die out, giving way for other, yet underdeveloped individuals, which after several reproduction cycles may have even produced offspring that gets ahead of the formerly best, but extinct individuals. On the other hand, due to the robustness of this learning technique it is probable that a population is supposed to proceed to an optimum (w.r.t. the error function), even if some top individuals have vanished. However, a thorough examination of these aspects has not yet been carried out.

Overfitting

A further problem machine learning methods have to struggle with is called *overfitting*. The critical issue in searching a model, i.e. here, a similarity measure, that really emulates the domain's underlying utility function is generalisation: How well will the learnt similarity measure suit for queries that are not contained in the training data set? That means, how noticeable is the discrepancy of the learnt similarity measure's quality concerning the retrieval error it generates using, on the one hand, the training data set and, on the other hand, some test data set of independent training examples? Of course, the best way to avoid overfitting is to use lots of training data. Unfortunately, in many application domains the amount of available training data is limited, for example, if the training examples must be acquired manually with the help of a human expert. Apart from that, the number of training examples used for learning is linearly proportional to the computation time needed. Hence, if it was possible to obtain learning results with a smaller set of training examples while avoiding overfitting, the computational effort might be reduced as well.

For these reasons, we also want to concentrate on strategies that reduce or avoid overfitting, that are based on the incorporation of additional background knowledge and that thus allow the utilisation of FLSM for smaller training data sets as well.

Bringing two of the previous issues together – namely the enormous effort regarding computation time needed and the danger of overfitting due to constructing overly complex models –, we need to point out that the control algorithm implemented for the evolution strategy described above allocates the same amount of computation time for each of the parts of the similarity measure representation, that are about to be learnt, i.e. for each population to be optimised. It becomes clear that this strategy is not optimal, when considering two symbolic attributes A_1 and A_2 whose value ranges comprise 3 and 10 elements, respectively. In the first case a 3×3 similarity table has to be optimised (thus 9 entries), whereas in the second case a 10×10 table (thus 100 entries).

Obviously, when allowing both corresponding populations of similarity table individuals (cf. Definition 2.13) to evolve for the same number of evolutionary generations, this may result in difficulties: On the one hand, the 3×3 matrix may become overfitted (presumed the training data set is relatively small) or the 10×10 matrix may be underfitted, respectively. On the other hand, computational resources are eventually wasted. Hence, we intend to address these problems by defining appropriate *scheduling strategies*.

3 Classification Domains and Solution Similarities

In the past years Case-Based Reasoning systems have often been applied in domains in which the class membership of a new problem situation or query, respectively, has to be predicted. For some examples of systems applying CBR for classification tasks successfully, the reader is referred to Yang and Honavar (1997), Jarmulak and Craw (1999), Branting (2001) and Gabel and Veloso (2001).

When considering to employ FLSM's capabilities to learn similarity measures in domains where the value of an attribute (e.g. a class membership or a numeric value of some target attribute) shall be predicted, we have to think about particular ideas and concepts how the mandatory utility feedback (cf. Section 2.2.4) can be expressed.

In this chapter we first focus on the basics of the so-called k-nearest neighbour retrieval and classification and introduce the concept of solution similarity. Then, we define specialised error functions building the foundation for applying FLSM's learning functionality to domains of the type mentioned. Afterwards, we concentrate on the specifics of our enhancement of FLSM's strategies to optimise similarity measures.

In Chapter 6 we present experimental learning results for some – artificial as well as real-world – classification and outcome prediction domains .

3.1 The k-Nearest Neighbour Retrieval

In a CBR system for prediction tasks a case characterisation consists on n describing attributes $A_1, \ldots, A_n$ and of a solution attribute A_s. A case may be written as $c = (c_p, c_s)$ making the distinction between problem and solution part clear. If the value range of A_s is finite, i.e. A_s is a symbolic attribute with $D_{A_s} = \{g_1, \ldots, g_m\}$, we also speak about a *classification task* since here a case's membership to one of the m classes has to be predicted. In the case of A_s being a numeric attribute, it is the system's task to project a query's accurate value for the solution attribute A_s.

According to Richter (2001) a similarity measure along with a case base defines a classifier for a CBR system:

Definition 3.1 (Nearest Neighbour Classification)
Given a case base CB, a similarity measure Sim and a query $q = (q_1, \ldots, q_n) \in \mathcal{M}$, the ***nearest neighbour*** *for q is a case $c = (c_p, c_s) \in CB$, so that it holds:*

$$\forall c^\star(c_p^\star, c_s^\star) \in CB : Sim(q, c_p) \geq Sim(q, c_p^\star)$$

Then, this principle of the nearest neighbour defines a classifier according to:

The class of each $q \in \mathcal{M}$ is determined by the class of q's nearest neighbour.

Of course, Definition 3.1 may easily be extended for solution prediction tasks, when saying:

If $c = (c_p, c_s) \in CB$ is q's nearest neighbour, then q's value for the solution attribute A_s will be predicted to c_s (for each $q \in \mathcal{M}$).

In practice, however, CBR systems usually make use of the more general approach to project the class/solution of a query by retrieving q's *k-nearest neighbours* $K = \{r_1, \ldots, r_k\}$ and apply some kind of majority vote on them.

For example, Michie et al. (1994) or Weiss and Kulikowski (1991) use the *single majority vote* that only considers the frequencies of the different classes in the set K of nearest neighbours. Wilke and Bergmann (1996), for instance, employ a weighted majority vote. As it has the advantage that the distance between neighbours collected in K is taken into account for the class prediction, it has also found entrance into our extensions to FLSM. Nevertheless, for a CBR systems it remains a difficult task to settle on a class decision or a solution proposal, based only on these k-nearest neighbours to the problem situation.

For classification tasks (symbolic solution attribute A_s) we perform the k-nearest neighbour classification for a given query q by computing a *class membership probability* for each of the m possible classes (Wilke and Bergmann, 1996). Then, the prediction of the CBR system is the class yielding the highest probability. We have defined three different strategies to calculate the class membership probabilities:

Definition 3.2 (Class Membership Probability Calculation)
*Let $K = \{r_1, \ldots, r_k\}$ be the ordered set of k-nearest neighbours ($Sim(q, r_i) \geq Sim(q, r_j)$ for all $i < j$) retrieved from a case base CB for a query q, based on a similarity measure Sim. The set of possible classes is given by the value range of the solution attribute A_s, $D_{A_s} = \{g_1, \ldots, g_m\}$. Then, for each class $g \in D_{A_s}$ the **class membership probability** is calculated*

- *for linear class membership prediction strategy according to:*

$$p^1_{Sim}(g|q) = \frac{\sum_{r \in K} \delta_{r,g} \cdot Sim(q, r)}{\sum_{r \in K} Sim(q, r)}$$

- *for exponentiated class membership prediction strategy (influenced by $\beta > 1$) according to:*

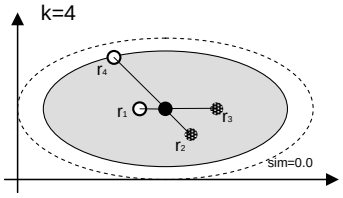
$$p^\beta_{Sim}(g|q) = \frac{\sum_{r \in K} \delta_{r,g} \cdot Sim(q, r)^\beta}{\sum_{r \in K} Sim(q, r)^\beta}$$

- *for position-weighted class membership prediction strategy (influenced by $\alpha \geq 0$) according to:*

$$p^\alpha_{Sim}(g|q) = \frac{\sum_{i=1}^{k} \delta_{r_i,g} \cdot \frac{\alpha+i}{i}}{\sum_{i=1}^{k} \frac{\alpha+i}{i}}$$

Thereby, $\delta_{x,y}$ is defined as follows: $\delta_{x,y} = \begin{cases} 1 & \text{if } class(x) = y \\ 0 & \text{else} \end{cases}$

The linear class membership prediction strategy can be seen as a specialisation of the second strategy: For $\beta = 1$ the exponentiated class membership prediction strategy is equal to the linear one.



query q
case of class 1
case of class 2

neighbour r_i	r_1	r_2	r_3	r_4
Sim(q,r_i)	0.8	0.6	0.5	0.2
class(r_i)	1	2	2	1

Predictions:

strategy	$p_{q,class1}$	$p_{q,class2}$
linear	47.6%	52.4%
exp. (α=2)	52.7%	47.3%
pos.w.(β=5)	57.2%	42.7%

Figure 3.1: Examples for Classification: To Which Class Belongs the Query q?

Figure 3.1 illustrates the fact that depending on the strategy chosen, very unequal classification decisions can be made. That way it emphasises the difficulty for a software system to predict the class membership based on a k-nearest neighbour retrieval.

Our motivation for the definition of different prediction strategies is to cover several heterogeneous approaches to class/solution prediction: The linear prediction strategy simply looks at the average similarity between q and its k-nearest neighbours separated by classes. The exponentiated strategy favours the class of those neighbours that have a relatively high similarity value (assumed that $\beta > 1$) while repelling the influence of neighbours with little similarity to q. Thus, the more similar neighbours have a higher impact on the classification decision made. That is true for the position-weighted class membership prediction strategy as well. However, here the accurate similarity values between q and its neighbours are disregarded. Instead this strategy only considers the positioning of a neighbour r_i in the retrieval result obtained for q. For $\alpha > 0$ the nearer neighbours' classes gain a more eminent influence on the classification decision. Kelly and Davis (1991) describe an interesting, more general and dynamic version of this position-weighting strategy. Their approach, called GA-WKNN, learns each of the weights for each position within the set of the k-nearest neighbours with the help of a genetic algorithm.

In Section 6.1 we compare the three before-mentioned strategies regarding their performance in several (real-world) application domains.

In the case of solution prediction tasks where A_s is a numeric attribute we differentiate between the same three prediction strategies as mentioned above (for the same reasons as mentioned above). However, since the CBR system has to give an accurate (here: numeric) prediction for the value of A_s, we do not determine probabilities, but concrete numerical values directly.

Definition 3.3 (k-NN Solution Prediction)
*Let $K = \{r_1, \ldots, r_k\}$ be the ordered set of k-nearest neighbours ($Sim(q, r_i) \geq Sim(q, r_j)$ for all $i < j$) retrieved from a case base CB for a query q, based on a similarity measure Sim. The interval of possible solutions is given by the value range $[s_{min}, s_{max}]$ of the solution attribute A_s. Then, the **predicted solution** s_q for query q is calculated*

- *for linear solution prediction strategy according to:*

$$s^1_{Sim}(q) = \frac{\sum_{r \in K} s_r \cdot Sim(q, r)}{\sum_{r \in K} Sim(q, r)}$$

- *for exponentiated solution prediction strategy (influenced by $\beta > 1$) according to:*

$$s^{\beta}_{Sim}(q) = \frac{\sum_{r \in K} s_r \cdot Sim(q,r)^{\beta}}{\sum_{r \in K} Sim(q,r)^{\beta}}$$

- *for position-weighted solution prediction strategy (influenced by $\alpha \geq 0$) according to:*

$$s^{\alpha}_{Sim}(q) = \frac{\sum_{i=1}^{k} s_{r_i} \cdot \frac{\alpha+i}{i}}{\sum_{i=1}^{k} \frac{\alpha+i}{i}}$$

Without any doubt the parameter k has an remarkable impact on the classifier or solution predictor, respectively, defined so far. In many practical applications k is simply set to 1 (see, for example, Skalak (1993) or Jarmulak and Craw (1999)). Other approaches allow the specification of an appropriate value for k by the user or by a domain expert (e.g. Wilke and Bergmann (1996)) or determine a good choice for k prior to the actual learning process with the help of some heuristic tests (e.g. Wettschereck and Aha (1995)). Jarmulak et al. (2000) report on an approach that tries to optimise feature weights as well as the parameter k to improve the quality of a k-nearest neighbour retrieval in the application domain of tablet formulation.

Our extensions to FLSM combine the afore mentioned strategies, in order to facilitate the incorporation of background knowledge into the learning process. The user is allowed to specify a lower bound k_{min} and an upper bound k_{max} for the interval of possible values for k to be searched within the context of FLSM's evolution strategy. That way a knowledge engineer knowing the best choice x for k may set $k_{min} = k_{max} = x$. On the other hand, he/she might define a more or less large interval $[k_{min}; k_{max}]$ to be searched depending on his/her degree of certainty about the actually optimal value for k.

Note, that in the following we mostly make use of a fixed (expert) value for k to guarantee a better comparability of experimental results, when performing evaluation of different learning procedures with varying parameter settings.

3.2 Utility Feedback Revisited

In Section 2.3 we have emphasised that the availability of appropriate training data is an essential precondition to the use of the framework for learning similarity measures. A single training example – no matter, if acquired manually or generated automatically – is supposed to contain certain information on the utility of one or several cases for a specific query (utility feedback). Now speaking about application domains with a predictive nature, i.e. classification and solution prediction tasks, we have to reflect on the question, how the necessary utility feedback may be represented.

At first glance one can say, that the only feasible feedback for a case must be based on the correct solution for that case. Obviously, for a given query q, of which it is known that the correct value of its solution attribute A_s is q_s, all those cases $c \in CB$ should yield the maximal utility of 1.0 which possess the same solution. The remaining cases from the case base should have a utility of less than 1.0, accordingly. Thus, one might be guided to decode utility in a binary way. But, if we followed the idea to set

$$u(q,c) = \begin{cases} 1.0 & \text{if } q_s = c_s \\ 0.0 & \text{else} \end{cases} \tag{3.1}$$

this would divide the case base into two sets within which all cases obtain the same utility. As a consequence, the feedback would represent two clusters of cases – one cluster of cases with maximal and another with minimal utility. In practical tests, however, we found that the learning algorithms we apply perform much better, if we provide them with error functions that make use of more substantial, i.e. a more fine-grained, utility feedback.

Introducing Solution Similarity

One way out of this dilemma is represented by introducing a so-called *solution similarity*. This concept has already been used in the domain of robotic soccer (Gabel and Veloso, 2001) and was more formally introduced by Stahl and Schmitt (2002). The basic idea here is to explicitly define an additional similarity measure sim_s for the cases' solution parts, i.e. for the solution attribute A_s (see Figure 3.2). Then, the similarity assessments received from sim_s can be used to generate the mandatory training data for optimising the similarity measure Sim for the problem parts: If the similarity between the solutions of two cases is very different from the similarity between their problem parts, this may indicate that Sim is not defined optimally.

The solution similarity measure may either be a rather simple, distance-based syntactical similarity measure (this approximation is at least sufficient in many application domains) or a more sophisticated one defined by an expert. The fundamental assumption, however, is that it is in general by far easier to settle on a similarity measure for the cases' solution part than to define an appropriate similarity measure for the problem part on the basis of which the retrieval will be carried out. Instead, the former one ought to be used to learn the latter one.

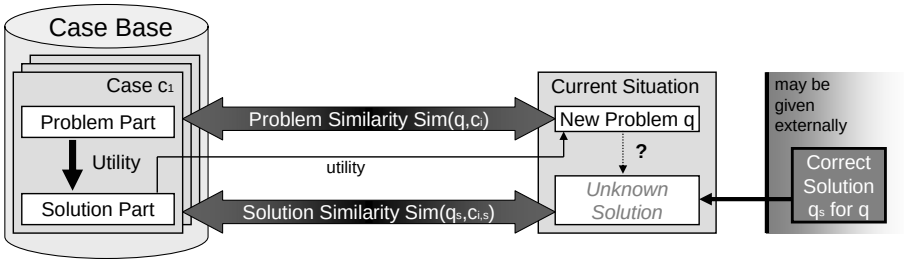


Figure 3.2: Solution and Problem Similarity

This is in particular true for domains we are dealing with in this chapter. Of course, we have to acknowledge that in practice there are also application areas where a case's solution is a highly complex entity for which an appropriate similarity measure cannot be defined easily (for an example see Gabel and Veloso (2001)). Furthermore, we need to stress that the usage of a solution similarity measure also depends on the prerequisite that the cases within the case base can be characterised as "optimal" cases. This means the solution contained in a single case should represent a correct or at least a highly appropriate solution to the problem given by the case's problem (query) part. In the following, we proceed on that assumption.

With the help of a solution similarity measure we are able to determine the correct case order that ought to be returned, when performing a retrieval for a query q from case base CB. In other words, the solution similarity represents a comfortable and sophisticated opportunity to realise an approximative, yet proper similarity teacher for the domains at hand. The following definition clarifies how we can get utility feedback based on a solution similarity measure.

Definition 3.4 (Utility Feedback via Solution Similarity)
Given a case base CB, a query q, whose value q_s of its solution attribute A_s is known, and a solution similarity measure sim_s for A_s, the utility of a case $c \in CB$ for query q is defined according to

$$u(q, c) = sim_s(q_s, c_s)$$

where c_s denotes the solution of case c.

Based on that definition, we are now enabled to "animate" the rather abstract Definition 2.7 (Training Example), at least for domains, for which an appropriate solution similarity measure can be defined (and assumed that concrete solutions of a sufficient number of problem situations are available).

Definition 3.5 (Solution Similarity Training Example)
For a query q, whose value q_s of its solution attribute A_s is known, a case base CB and a solution similarity measure sim_s approximating the underlying utility function, the concept of the similarity teacher is realised by dint of providing training examples through:

$$TE_s(q) = [(c_1, sim_s(q_s, c_{1,s}), \ldots, (c_f, sim_s(q_s, c_{f,s})]$$

where f is the number of cases for which feedback is given. For each $i \in \{1, \ldots, f\}$ it holds:

- $c_i \in CB$ *and* $c_{i,s} \in D_{A_s}$
- $sim_s(q_s, c_{i,s}) \in [0; 1]$
- $sim_s(q_s, c_{i,s}) \geq sim_s(q_s, c_{j,s})$ *for all* $i < j$

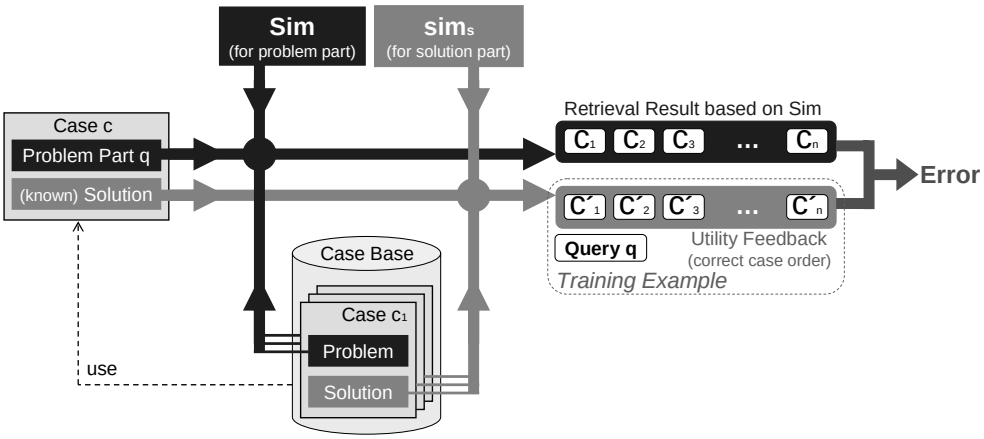


Figure 3.3: Employing Solution Similarity as Similarity Teacher

In Figure 3.3 we illustrate how a solution similarity measure can be employed as similarity teacher, i.e. to obtain the mandatory utility feedback that is required for the learning process. To create a single training example a solved case c consisting of its problem part (query) q and its known solution is necessary. Those solved cases may, for example, be taken from the case base (see Section 3.4). According to Definition 3.5 the correct case order, that should be

retrieved for q, is given by the retrieval result that is obtained, when a retrieval based on the solution similarity measure sim_s is carried out. This kind of utility feedback, together with the query q represents one single training example.

On the other hand, a retrieval based on the problem similarity Sim – that is to be optimised – may be conducted. If the corresponding retrieval result matches with the utility feedback, then Sim can already be characterised to be optimal. If there are discrepancies between both case orders, however, the problem similarity measure needs further adjustments that can be realised with the learning techniques described in the previous chapter, guided by an appropriate error function.

Reverting to "Simple Classification"

Regarding the utility feedback that can be provided, classification domains – in the simplest case a decision for one out of two classes has to be made – may be considered as a special case of domains in which a solution similarity is applicable.

The "simplest" prediction task occurring rather often in practice is represented by a binary classification, i.e. a case's membership to one out of two classes shall be predicted ($|D_{A_s}| = 2$). Concerning the utility feedback that can be gained in such a scenario, we are thrown back to Equation 3.1: Either the utility of case c for the query q is 1.0 or it is 0.0, depending on the real class membership, i.e. the actual solution, for the problem that q represents. Therefore, the utility feedback cannot be circumscribed to be very substantial, since a training example for q will consist of two "clusters" of entries – the first one showing the maximal utility of 1.0, the rest a utility of 0.0. Of course, this fact does not prevent the employment of this type of utility feedback within the learning process. However, we think that the success of the optimisation process can be increased, when that feedback can be "enriched" somehow. In the next section, for example, we will do so by making use of the class membership prediction probabilities as well. Generalising that, we can say that the introduction of a well-defined solution similarity measure may in certain application scenarios not lead to optimal results, if the value range D_{A_s} of the solution attribute consists of a few elements only.

Another drawback of obtaining utility feedback from a solution similarity measure is that this method cannot be described to be maximally goal-oriented in terms of the predictive character of the domains we are currently dealing with. Stated differently, we think that, when learning a similarity measure in solution prediction domains, it must be the overall goal to optimise the correctness of the actual solution prediction (and not just the correctness of a retrieved case order). In that case, of course, the utility feedback and the training examples containing it have a rather elementary structure:

Definition 3.6 (Solution Prediction Training Examples)
For a query q, whose value q_s of its solution attribute A_s is known and a case base CB training examples are defined as: $TE_p(q) = [q, q_s]$

As a consequence, we are in need of a strategy that enriches that feedback to make it more suitable for its employment by a learning algorithm. The following section addresses this task.

3.3 Readapted Error and Fitness Functions

In this section we want to discuss three forms of error functions that are able to measure the quality of a retrieval result with respect to some training data and that shall be used as fitness

functions for FLSM's evolution strategy (see Section 2.4) as well.

Here, we focus on domains in which the application of a solution similarity measure seems to be promising, i.e. the value range of a discrete solution attribute A_s is rather large or A_s is numeric. Furthermore, we also want to aim at those prediction domains in which the solution attribute is symbolic with a rather limited set of possible values making the usage of a solution similarity measure somewhat troublesome (cf. Section 3.2).

The remaining part of this section examines the following possibilities to define adequate error functions:

- retrieval error based on ordinal as well as on cardinal feedback in domains with a solution similarity measure
- prediction error based on outcome prediction results
- classification error based on classification probabilities in domains with a symbolic solution attribute

3.3.1 Ordinal/Cardinal Retrieval Error with Solution Similarity

Since utility feedback provided with help of a solution similarity measure represents a correct case order (a partial order) and because there is no substantial difference between Definitions 2.7 and 3.5 (Training Example and Solution Similarity Training Example, respectively), a first possibility for defining an appropriate error function is to adopt the error function from Section 2.3.2, which is based on ordinal utility feedback. That means, the index error function E_I (see Definition 2.10) may be applied here as well:

Definition 3.7 (Index Error From Solution Similarity)
*For a similarity measure Sim, a query q and a corresponding training example $TE_s(q)$ built upon utility feedback from a solution similarity measure sim_s, the **index error from solution similarity** is defined as*

$$E_{IS}(TE_s(q), Sim) = \sum_{i=1}^{f-1} \sum_{j=i+1}^{f} eu_o(TE_s(q), Sim, (c_i, c_j)) \cdot \frac{i+\alpha}{i}$$

where $\frac{i+\alpha}{i}$ is called error-position weight to be influenced by the parameter α. f represents the number of cases for which feedback is contained in the training example and eu_o is the ordinal error unit.

For that definition it seems appropriate to identify parameter f with the number k of nearest neighbours that are retrieved from a k-NN retrieval, i.e. $f = k$. This would make sense for reasons of comparability with the error functions that we will introduce in the following. In spite of that, we point out that a higher value for f should yield improved learning results, on the one hand, but increase the computational effort, on the other hand.

Due to the accurate similarities between solutions that the solution similarity measure sim_s provides, however, we may also employ a more sophisticated error function E_s. This function exploits the numeric similarity values between the query's correct solution and solutions of other cases as contained in the training data. For this purpose we redefine the error unit – now utilising cardinal, instead of ordinal information.

Definition 3.8 (Cardinal Error Unit Based on Solution Similarity)
*Let q be a query with a known correct solution q_s, $TE_s(q)$ be the corresponding training example, where $(c, sim_s(q_s, c_s)) \in TE_s(q)$. Further, let Sim be a similarity measure. An **cardinal error unit** on the basis of the solution similarity measure sim_s is defined as*

$$eu_c(TE_s(q), Sim, c) = |Sim(q, c) - sim_s(q_s, c_s)|^\gamma$$

where $\gamma \in \mathbb{R}^+$.

The cardinal error unit based on a given solution similarity measure analyses the correctness of an arbitrary case's similarity to q with respect to the "target" similarity that it should have according to the training data. With the help of the parameter γ larger deviations may be punished harder, e.g. when setting $\gamma = 2$.

With help of the cardinal error unit defined previously we can now introduce the definition of the cardinal retrieval error from solution similarity.

Definition 3.9 (Cardinal Retrieval Error from Solution Similarity)
*For a similarity measure Sim, a query q and a corresponding training example $TE_s(q)$ built upon utility feedback from a solution similarity measure sim_s, the **cardinal retrieval error from solution similarity** is defined as*

$$E_S(TE_s(q), Sim) = \sum_{i=1}^{f} eu_c(TE_s(q), Sim, c_i) \cdot \frac{i + \alpha}{i}$$

where $\frac{i+\alpha}{i}$ is called error-position weight to be influenced by the parameter α. f represents the number of cases for which feedback is contained in the training example.

An obvious advantage of the cardinal retrieval error is that its application is not restricted to certain application domains: As long as a solution similarity measure can be specified, that error function is usable, i.e. it does not make any demands on the type of the solution attribute. However, as already pointed out a deficiency of E_S is its lack of goal-orientation concerning the solution that has to be predicted for a given query, based on a k-nearest neighbour retrieval.

3.3.2 Solution Prediction Error

It is important to keep in mind that the goal of optimising a similarity measure is to improve the overall performance of the CBR system. So, when the CBR system is employed to predict a solution for a new query or to classify it, the similarity measure, that is about to be learnt by FLSM, ought to be optimised in such a way that the CBR system's actual task will be ameliorated.

For these reasons, an optimisation on the basis of the ordinal or cardinal retrieval error induced by a solution similarity measure may be suboptimal – because it mainly corresponds to the correct case order provided by the solution similarity and tries to minimise the discrepancy between that order and the case order obtainable from a retrieval based on the measure to be learnt. That way the precise prediction or classification decision arising from a certain similarity measure are not taken into consideration directly. We argue that it is advantageous to not neglect that information and to use it for guiding the optimisation process. To do so we need to encapsulate it into an appropriate error function.

In Section 3.1 we have defined several strategies to predict a solution for a given query based on a k-nearest neighbour retrieval. In the case of a numeric solution attribute we get a concrete value, representing the prediction. In the case of a symbolic solution attribute, we obtain a class prediction. If we assume to be in possession of one or several queries together with their belonging correct solutions (no matter if these correct solutions are numerical values or class memberships), we can express the quality of the similarity measure with means of a comparison between correct, predetermined and predicted solutions.

Definition 3.10 (Solution Prediction Errors)
*Given a solution prediction training example $TE_p(q) = [q, q_s]$ for query q, a case base CB, a similarity measure Sim, on the basis of which a k-nearest neighbour retrieval is carried out, and a similarity measure sim_s for the solution attribute, the **solution prediction similarity error** is defined as*

$$E_{P_s}(TE_p(q), Sim) = 1 - sim_s(q_s, s^x_{Sim}(q))^\gamma$$

where $s^x_{Sim}(q)$ stands for the predicted solution, $x \in \{1, \alpha, \beta\}$ determines the strategy by which the solution is predicted (cf. Section 3.1) and $\gamma \in \mathbb{R}^+$ is a parameter to in/decrease the effects of larger deviations from the correct solution.

*Additionally, for solution attributes A_s with a numerical value range, the **solution prediction distance error** is defined according to*

$$E_{P_d}(TE_p(q), Sim) = |q_s - s^x_{Sim}(q)|^\gamma$$

where γ and $s^x_{Sim}(q)$ have the same meaning as before[1].

When employing these solution prediction error function as objective function, FLSM is supposed to directly optimise its prediction accuracy. A drawback of the second error function is that it is explicitly designed for those domains in which the value of a numeric solution attribute shall be predicted (since the difference between the predicted and the real solution is calculated). The first error function, however, can be applied as long as an appropriate solution similarity measure is available – unfortunately, its competence is very reliant on the accuracy of that similarity measure.

3.3.3 Classification Errors

If it is the CBR system's task to classify new problem situations q, i.e. to assign one out of m classes $\{g_1, \ldots, g_m\}$ to it, a solution prediction training example contains the query's correct class only. Hence, an error function corresponding most to the system's actual optimisation task (improving the classification accuracy) and thus being most goal-oriented ought to make use of binary utility feedback as denoted in Equation 3.1.

Of course, with the definitions introduced so far, we already have the possibility to employ an error function that directly corresponds to the system's classification accuracy: Using, for example, a trivial, binary solution similarity measure $sim_s(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{else} \end{cases}$ and assuming that $s^x_{Sim}(q)$ represents the class that yields the highest class membership probability (cf. Definition 3.2), the solution prediction similarity error E_{P_s} as given in Definition 3.10 depicts an error function that seems to be very appropriate for classification tasks.

[1] For reasons of simplicity, in the following we will refer to both error functions via the function symbol E_P. If we want to address one of them specifically, that will be mentioned explicitly.

Characterising that error function informally – in the following, we refer to it as *simple classification error* –, it simply verifies whether Sim is able to predict the correct class membership for q. If it is so, the error value will be 0, otherwise 1. Consequently, the simple classification error can be qualified to be most goal-oriented, since it consistently judges a similarity measure to make the right or the wrong classification. Unfortunately, this implies that this error function is only a very coarse measure allowing no fine-grained nuances concerning the assessment of different similarity measures.

For example, assume a classification task where the class membership of cases to one out of two classes g_1 and g_2 has to be predicted and a query q that actually should be classified as belonging to g_1. A first similarity measure Sim_1 calculates the class membership probabilities of 0.4 for class g_1 and 0.6 for g_2, thus voting for class g_2 (wrong classification). A second measure computes class membership probabilities of 0.1 and 0.9 for g_1 and g_2, respectively (wrong classification as well). In this scenario, obviously, the first similarity measure ought to be preferred to the second one – and thus should obtain a smaller value for the classification error. The simple classification error, however, makes no difference between both measures and assigns an error value of 1 in each case.

In the previous section we have presented one way out of this dilemma: The definition of a sophisticated solution similarity measure may improve the error assessment significantly since it allows a more fine-grained evaluation of similarity measures. Unfortunately, if the number of classes m is rather small, the benefit gained from using a solution similarity measure vanishes in part: In the extreme case of having only a two-class classification task, there is no solution similarity definable that would increase the error function's granularity (as the informal introduction of the simple classification error shows).

Here, we want to suggest an alternative approach for defining an adequate error function: We intend to employ the class membership probabilities to evaluate the quality of a certain similarity measure.

Definition 3.11 (Simple and Advanced Classification Error)
Let $TE_p(q) = [q, q_s]$ be a solution prediction training example for query q where $q_s \in D_{A_s} = \{g_1, \ldots, g_m\}$ is the correct class of q. Further, CB be a case base and Sim a similarity measure on the basis of which a k-nearest neighbour retrieval is carried out. Based on that retrieval, $h \in D_{A_s}$ be the class yielding the highest class membership probability p_{max}, so that it holds

$$p^x_{Sim}(h|q) = p_{max} = \max_{g \in A_s}\{p^x_{Sim}(g|q)\}$$

Then, the classification errors are defined as follows:

- ***simple classification error*** *(derived from E_{P_s})*

$$E_{C_s}(TE_p(q), Sim) = \begin{cases} 0 & \text{if } q_s = h \\ 1 & \text{else} \end{cases}$$

- ***advanced classification error***

$$E_{C_a}(TE_p(q), Sim) = \sum_{g \in D_{A_s}} \begin{cases} |1 - p_{max}|^\gamma & \text{if } g = q_s \\ (p^x_{Sim}(g|q))^\gamma & \text{else} \end{cases}$$

The parameter x may be chosen from $\{1, \alpha, \beta\}$ and $\gamma \in \mathbb{R}^+$.

The advanced classification error assesses the quality of a class membership prediction in a more sophisticated way allowing an infinite number of error shades. This bears the advantage that, when using E_{C_a} as error function, it is easier for the machine learning algorithm to seek for an optimum due to the advanced classification error's gradient: This error function is continuous throughout the entire search space, while the simple classification error can rather be understood as consisting of plenty of plateaus and discontinuities.

However, one should not surmise that E_{C_a} is in general to be preferred to E_{C_s}. Reverting to the example from above and this time presuming q's correct class was g_2, we have to realise that the simple classification error (E_{C_s}) is optimal with respect to the classification decision made: The k-NN classifier decides for g_2 (on a $0.4 : 0.6$ vote) and the correct solution *is* indeed g_2.

As a consequence of the error functions' advantages and disadvantages discussed, we suggest to employ a combination of both. With the help of the combined classification error the orientation towards the optimisation goal (improve classification accuracy) of the simple error can be preserved, while the higher granularity of the advanced classification error is incorporated. We experimented with several settings for ω and obtained convincing learning results for values $\omega \approx 0.5$ (see Section 6.1).

Definition 3.12 (Combined Classification Error)
*For a solution prediction training example $TE_p(q)$ for query q and a similarity measure Sim the **combined classification error** is defined according to*

$$E_C(TE_p(q), Sim) = \omega \cdot E_{C_s}(TE_p(q), Sim) + (1 - \omega) \cdot E_{C_a}(TE_p(q), Sim)$$

where ω is chosen from $[0; 1]$.

3.3.4 Settling Fitness

In analogy to the average index error $\hat{E}_I$ introduced in Definition 2.11, we now define the average solution prediction error as the error function to be used for classification and solution prediction domains.

Definition 3.13 (Average Solution Prediction Error)
*Given a set of training queries $Q = \{q_1, \ldots, q_s\}$, corresponding training data $TD_u(Q) = \{TE_u(q_1), \ldots, TE_u(q_s)\}$ with $u \in \{s, p\}$ and a similarity measure Sim, the **average solution prediction error** is defined as*

$$\hat{E}_{SP}(TD_u(Q), Sim) = \frac{1}{s} \cdot \sum_{i=1}^{s} E_x(TE_u(q_i), Sim)$$

where x determines one of the error functions for one single query ($x \in \{I, IS, P, Cs\}$).

The average solution prediction error depicts a measure that is able of evaluating the quality of a similarity measure with respect to a number of training examples for corresponding training queries.

We employ the average solution prediction error as fitness function for FLSM's evolutionary algorithm as well. Here, it holds, too: The lower the average error, the fitter the corresponding individual and similarity measure, respectively.

3.4 Introspective Learning

After having defined appropriate error functions to measure the quality of a similarity measure and of the retrieval results it produces, we need to settle according to which strategy we want to exploit the information contained in the training data. For this purpose we follow the idea of a *leave-one-out test strategy* (Weiss and Kulikowski, 1991). This technique has also been described as *introspective learning* as it has a representation of its own learning model and of its performance quality, so that it may, for example, decide on itself when learning is needed or which of its elements needs adjustments (Zhang and Yang, 1999). Based on its introspective insights on its problem solving capabilities the optimisation process may be controlled. An exemplary application in the field of air traffic control using introspective learning is described by Bonzano et al. (1997).

We assume to be in possession of a given set $C = \{c_1, \ldots, c_l\}$ of cases including their respective solutions (e.g. correct classifications). Then, we first divide C into two disjunct subsets, CB and T, so that it holds $CB \cap T = \emptyset$ and $CB \cup T = C$. The first subset CB represents the case base and the training data set, respectively, and is used for learning. The second subset T is an independent test data set used for evaluating the learning results.

As follows, we want to present two algorithmic realisations by dint of which the average solution prediction error (cf. Definition 3.13), i.e. the quality measure assessing the similarity measure's performance with respect to the training data, is calculated. Both algorithms perform a leave-one-out test for the given case base CB – the first one employing ordinal/cardinal utility feedback that refers to the correctness of obtained retrieval results (cf. Sections 3.2 and 3.3.1), the second one using the CBR system's predictions regarding the solution attribute (see Sections 3.1 as well as 3.3.2 and 3.3.3).

Input : an error function selector $a \in \{IS, S\}$,
training examples $TE_s(q)$ for all $q \in CB$,
a similarity measure Sim

Output : $\hat{E}_{SP}(TD_s(CB), Sim)$

1. **for all** $c \in CB$ **do**
   a) $CB' := CB \setminus \{c\}$
   b) $K = \{r_1, \ldots, r_{|CB|}\}$:= retrieval result for query q from CB' based on Sim
   c) compute $E_a(TE_s(c), Sim)$
2. compute $e := \hat{E}_{SP}(TD_s(CB), Sim)$
3. **return** e

Algorithm 3.1: Error Calculation Based on Solution **Similarity** Training Examples TE_s

Apart from the indispensable training data and the similarity measure that has to be evaluated, the algorithm for the error calculation based on solution similarity training examples (cf. Definition 3.5) given in Algorithm 3.1 expects an error function selector as input parameter.

With the help of the latter it is decided whether the error function E_{IS} (index error from solution similarity) or E_S (cardinal retrieval error from solution similarity) is to be used. The main loop iterates over all cases c from case base CB. Within that loop first, the respective case is excluded from the case base (leave-one-out), and then the particular error E_{IS} or E_S is calculated. To do so, it may be necessary to accomplish an entire retrieval by which the similarities between c and all the cases from CB' are determined. However, the retrieval may be avoidable if $TE_s(c)$ contains utility feedback for a few cases from CB only. In that case it is of course sufficient to calculate the similarity between c and all those cases for which feedback is available. Finally, the average error $\hat{E}_{SP}$ is computed and returned.

When incorporating the CBR system's solution predictions into the error calculation, the parameter k for the nearest neighbour retrieval plays an important role. As announced in Section 3.1 we facilitate the learning of an optimal value for k as well, i.e. k represents a variable parameter in the learning process that is to be optimised as well.

Input : an error function selector $a \in \{P, C\}$,
training examples $TE_p(q)$ for all $q \in CB$,
a similarity measure Sim, k_{min} and k_{max}

Output : an error vector $(\hat{E}^{k_{min}}, \ldots, \hat{E}^{k_{max}})$ where $\hat{E}^k$ refers to $\hat{E}_{SP}(TE_p(CB), Sim)$ based on a k-NN retrieval

1. **for all** $c \in CB$ **do**
   a) $CB' := CB \setminus \{c\}$
   b) $K^\star = \{r_1, \ldots, r_{k_{max}}\}$:= retrieval result for k_{max}-NN retrieval for query q from CB' based on Sim
   c) **for** $k := k_{min}$ **to** k_{max} **do**
      i. **if** $a = P$ **then**
         A. $s^k := s^x_{Sim}(c)$ based on $K = \{r_1, \ldots, r_k\} \subseteq K^\star$
         B. $E^k_a(c) := E_P(TE_p(c), Sim)$ based on s^k
      ii. **if** $a = C$ **then**
         A. $\vec{p}^k$:= vector of class membership probabilities based on $K = \{r_1, \ldots, r_k\} \subseteq K^\star$
         B. $E^k_a(c) := E_C(TE_p(c), Sim)$ based on $\vec{p}^k$
2. **for** $k := k_{min}$ **to** k_{max} **do**
   a) $\hat{E}^k := \hat{E}_{SP}(TD_p(CB), Sim)$ based on $E^k_a(c)$ for all $c \in CB$
3. **return** $(\hat{E}^{k_{min}}, \ldots, \hat{E}^{k_{max}})$

Algorithm 3.2: Error Calculation Based on Solution **Prediction** Training Examples TE_p

Besides the parameters Algorithm 3.1 expects, the algorithm for error calculation based on solution *prediction* training examples (Algorithm 3.2) presumes the specification of an interval $[k_{min}; k_{max}]$ of potential values for k that is to be searched. The algorithm's main loop realises a

leave-one-out test iterating over all cases from CB as well. First, the current case c is excluded from the case base. Then, a k_{max}-nearest neighbour retrieval from the reduced case base CB' is carried out. The next step is represented by a loop that iterates over all possible values for k. For each possible choice of k the corresponding predicted solutions or corresponding class membership probabilities, respectively, are calculated. The results of these calculations are then used to determine the corresponding solution prediction error $E_P^k(c)$ or classification error $E_C^k(c)$ (depending on the error function selector input) for the current value of k and the actual case c. After having completed the main loop, the algorithm averages the collected error values for each choice of k separately. Hence, it computes a vector a vector of $k_{max} - k_{min}$ average solution prediction errors that is returned finally.

Concerning the fitness assigned to a similarity measure individual within an evolutionary algorithm, in Section 3.3.4 we have stated that the average solution prediction error $\hat{E}_{SP}$ shall be used as fitness value. The algorithms presented here compute the value of that error function for a particular similarity measure so that the values they return can be employed directly. However, the second algorithm for error calculation based on solution prediction training examples provides an entire vector of average error values. Consequently, a strategy is required by which the decision for one accurate fitness value is made. Though there is a number of potential amalgamation or choosing strategies, for our extensions to FLSM we decided to assign the fitness by choosing the minimal error value contained in the vector, i.e. it holds for the current similarity measure's fitness

$$f(Sim) := min_{k \in \{k_{min}, \ldots, k_{max}\}}(\hat{E}^k) \tag{3.2}$$

where $(\hat{E}^{k_{min}}, \ldots, \hat{E}^{k_{max}})$ refers to the result the error calculation algorithm returns.

Computational Effort

From the algorithms presented above it can be seen that the error calculation based on a leave-one-out test strategy requires the execution of $|CB|$ k-nearest neighbour retrievals (at least for Algorithm 3.2). Thus, when performing a leave-one-out test according to that algorithm (on the basis of a linear retrieval), it is necessary to compute the similarity between two cases for $|CB| \times (|CB| - 1)$ times. Hence, the computational effort – and so the time required for learning – grows quadratically with respect to the size of the training data.

As a matter of fact most of the basic arithmetic operations involved in the similarity computations that occur during the error/fitness calculation of a learning process are repeated for several times in exactly the same manner. Consequently, when storing several intermediate computation results externally and reusing them later, many dispensable calculations can be avoided and a lot of computational time can be saved. In Section 5.4 we show how we reduced the computational effort by more than 80% by introducing a "smart" fitness and error calculation.

4 Embracing Background Knowledge

The previous chapter has established the basis to apply FLSM in a wider range of application scenarios. With the current chapter we address the main goal of this work and highlight those of our extensions to that learning framework that go into another direction: By systematically incorporating different forms of background knowledge into the process of optimising similarity measures we aim at a noticeable enhancement of FLSM's learning capabilities.

First, we give a short motivation for our strategy to improve FLSM by means of additional knowledge. Then, we introduce a number of elaborated concepts on the basis of which knowledge can be used to actively bias the learning process. The endmost two sections of this chapter exhaustively give attention to certain forms of knowledge that will be employed in the scope of this work. In this regard the acquisition, utilisation and particular properties of those knowledge forms will be highlighted.

4.1 Motivation

It is true that prior knowledge has an considearble impact not only on the speed with which human beings learn a new concept, but even on the logical structure of the concepts learnt (Pazzani, 1991). Concerning the influence of background knowledge on the learning behaviour of humans many experiments have been carried out in the fields of Cognitive Science and Psychology. For example, for humans it is easier to learn conjunctive concepts than disjunctive ones, i.e. less training examples are required to learn a conjunctive concept (Dennis et al., 1973). Nakamura (1985) has investigated the role of background knowledge regarding classification accuracy. The author found that a human being's capabilities in learning a linearly separable concept are increased, when provided with a domain theory, i.e. when being in possession of background knowledge on the current application domain. And Wisniewski (1989) emphasises the importance of faultless prior knowledge: A nonlinearly separable concept that is consistent with a human's background knowledge is easier to learn than a linearly separable concept contradicting the subject's prior knowledge. Hence, one can say that the background knowledge a human learner is equipped with may be as influential as the information provided by the environment (training data).

This argument referred to human learners only. However, it lets emerge the question which of these insights from Cognitive Science can be transferred to the field of Machine Learning and to FLSM as well. Pazzani (1991) tries to answer that question by stating:

> "The ability of human learners to learn relatively quickly and accurately in a wide variety of circumstances is in sharp contrast to current machine learning algorithms. I hypothesise that this versatility comes from the ability to apply relevant background knowledge to the learning task and the ability to fall back on weaker methods in the absence of this background knowledge."

Of course, the advantage of enriching a machine learning system's capabilities by background knowledge has been recognised by many researchers. For example, Doan et al. (2001) allow

their system LSD to employ domain constraints – divided into soft and hard constraints – as an additional source of knowledge to improve the access to a multitude of data sources. Kopanas et al. (2002) criticised that the role of domain knowledge has often been paid too few attention to in many data mining and knowledge discovery projects and it is described what kind of particular knowledge may be employed during which phase of a KDD[1] project. Another successful utilisation of additional knowledge in a case-based application scenario where domain knowledge is extracted explicitly from the case base, is depicted by Li et al. (2001).

It must be stressed that in general learning an arbitrary complex concept, whose foundations are eventually poorly understood even by humans, represents a demanding and complicated task for any machine learning system. A learner performs the better, the more it knows about the actual learning goal – carrying it to extremes, we may argue that a learning system is only capable of learning what it almost already knows. In many cases sub-symbolic learning is accomplishable easier depending on the respective objects of learning. For example, learning a linear function is by far simpler than obtaining a more complex, highly parameterised function. Insofar our piecewise linear approximation of distance-based similarity functions via similarity function individuals (cf. Definition 2.12) already depicts a way to simplify the learning goal.

When reflecting on the question how background knowledge may be incorporated into the framework for learning similarity measures in order to improve its learning capabilities, it soon becomes clear that a proceeding, that – from an abstract point of view – is similar to the core idea from the work of Djoko et al. (1995), seems to be promising: We need background knowledge to input a bias toward certain types of similarity measures. Obviously, the form that bias can take should be dependent on the respective application domain as well as on additional knowledge that may perhaps be provided by a human expert. Let us approach that issue from a historical perspective focusing on the problem of overfitting for the time being.

4.1.1 Overfitting

William of Ockham (1285-1349), a medieval English philosopher and Franciscan monk idealising a life characterised by poverty, criticised the scholastic philosophy of those days, whose theories grew more and more complex without yielding a significant improvement in their predictive power, by stating: "Nunquam ponenda est pluralitas sin necessitate." Translating that statement, basically it means: "Plurality should not be assumed without necessity." In more recent times his quotation and the demand implied have gained popularity among researchers from the fields of learning theory and knowledge discovery[2]. Many authors refer to, discuss, form their own interpretations of and also criticise the so-called "Occam's Razor" — the basic principle that ought to be taken into consideration, when developing or applying a machine learning approach (for some details and examples see Blumer et al. (1987), Nordin and Banzhaf (1995) and Gamberger and Lavrac (1997)). In practice, the razor is usually interpreted in two different ways:

- Simplicity is a goal in itself.
- Simplicity leads to greater accuracy.

[1] Knowledge Discovery in Databases

[2] For example, in conjunction with the NIPS 2001 Conference (Neural Information Processing Systems) there was an entire workshop dedicated to the "Foundations of Occam's Razor and Parsimony in Learning".

Hence, Occam's Razor provides a justification for preferring simpler models over more complex ones. For instance, it is well-known that overfitting is caused by creating models that are overly complex, i.e. that fit "too" good to the training data provided, while showing poor performance and little ability in generalising on some independent test data set. Accordingly, Occam's Razor suggests to avoid that by establishing a preference for simpler models. However, Domingos (1999) shows that overfitting is not caused by the construction of complex models only. Instead, the author blames multiple testing as a key factor increasing the probability of overfitting: When attempting a large number of candidate models, the chance of finding one that (over)fits the training data (just by chance) very well is increased rapidly. Speaking about constrained knowledge discovery, Domingos raises the question: If we abandon the preference for simpler hypotheses (i.e. not following Occam's razor completely), in what other ways can we avoid overfitting, when still intending to learn very flexible models?

Giving an answer to that question represents a major task of the thesis at hand: The learning process – including the risk of overfitting – shall be optimised by incorporating domain or background knowledge and thus constraining the search. The employment of very diverse forms of background knowledge has been described in several works. Domingos (1999) gives a short, summarising overview on several forms, in which domain constraints can appear (e.g. restrictions on the signs of inequalities in rules, knowledge on cause and effect variables, combinations of domain constraints and a simplicity bias). In the work of Abu-Mostafa (1989) and Donoho and Rendell (1996) the authors speak about "hints" and several types of "fragmentary knowledge", respectively, that are given to the learner and about strategies how they are incorporated into the learning process. In the next section we want to build an overview on the various forms of fragmentary knowledge and hints that we intend to use in order to improve the learning behaviour of FLSM.

4.1.2 Overview

Within the context of the work at hand we identified two main sources of background knowledge that can be exploited (more or less) easily and utilised to improve the process of learning similarity measures in the scope of FLSM. In the following sections we describe our ideas on how to incorporate additional knowledge into the optimisation process, not only from a realisation-centred but also from a conceptual point of view. This bears the advantage that, in principle, those concepts may be applied to improve the behaviour of different learning algorithms employed by FLSM. However, as announced in the previous chapters, with our realisations and implementations we laid the focus on enhancing the capabilities of the evolution strategies used within FLSM (described in Section 2.4). In Figure 4.1 we give an overview of the two sources of additional knowledge and its various occurrences which we want to incorporate into the learning process.

Similarity Meta Knowledge The way of modelling similarity measures described in Section 2.2 allows us to define several forms of meta knowledge which determine general demands on the appearance of learnt similarity measures.

- Heuristics on the "typical" syntactical shape of local similarity measures represent a simpler form of knowledge. These refer to several basic properties of similarity measures and will be implemented as weak or strong constraints reducing the search space. A descriptive example concerns the reflexivity of similarity measures, for

instance: In most application domains a non-reflexive similarity measure would be unpropitious.

- The case knowledge container holds a certain amount of unexploited knowledge potential, too. Assumed that the case base contains a sufficient number of cases, the distribution of the cases throughout the entire space of possible cases and especially the distribution of their attribute values, may reveal interesting opportunities to improve the learning process. We intend to employ a statistical case base analysis ("mining" knowledge from the case base) to find out which regions of the space of similarity measures are really worth to be searched thoroughly.

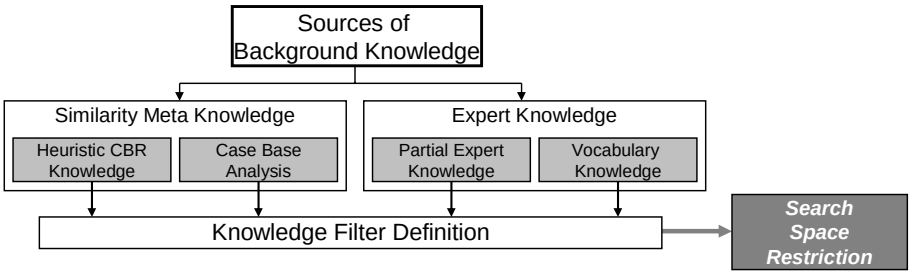


Figure 4.1: Sources of Background Knowledge

Expert Knowledge The aid of a knowledge engineer and the incorporation of his/her expert knowledge into the learning process may be unpayable. Furthermore, when building a CBR system, the expert settles the vocabulary knowledge, which in turn may include some implicit similarity knowledge to be exploited as well.

- Defining similarity measure bottom-up is a complex, time-consuming and probably error-prone task that is reliant to a human domain expert. Searching the whole space of possible similarity measures with the help of a learning algorithm can be time-consuming and is often susceptible to overfitting, if the amount of available training examples is rather small. So, why not meet in the middle? A domain expert has often some vague idea of the correct similarity $sim_A(x, y)$ between value x and y of a specific attribute A, although he/she finds it difficult to translate his/her understanding into an accurate numerical value within $[0; 1]$. For example, the human specialist might state: "I think the similarity between x and y is rather low, somewhere between 0.0 and 0.5. Maybe, it is around 0.3, but I am not really sure of this." Regarding feature weights, an expert may say: "I know, that w_1 is more important than w_2, while both are less influential than w_3, but I cannot express that in accurate numbers." Unfortunately, a CBR tool in general expects the user to enter an accurate value for $sim_A(x, y)$ as well as for each w_i. As a consequence, the expert is forced to make some kind of "sophisticated" guess about the correct local similarity value or feature weight – which is eventually imperfect. These arguments make clear that it may be advantageous to incorporate a domain expert's partial knowledge (if that is available) about the similarity measure to be learnt into the learning process. That form of knowledge is supposed to constrain the search space and thus to speed up the search while reducing the danger of overfitting at the same time.

- A further form of expert knowledge is represented by the vocabulary knowledge – this corresponds to one of the knowledge containers as introduced in Section 2.1. The vocabulary provides possibilities to constrain the search space by utilising the information that is contained within structured data types, such as symbolic types with an ordered or taxonomic value range.

This list of forms of additional knowledge does not assert a claim to be complete and may, of course, be extended. However, for the scope of this work we have tried to determine and utilise those sources of background knowledge that can be employed without major difficulties and whose application seemed to be promising. In the next section we introduce the concept of knowledge filters as objects that, on the one hand, bear the knowledge gathered from diverse sources and by means of which, on the other hand, the space of searchable similarity measures can be restricted and thus a bias can be exerted on the learning process.

The subsequent sections (4.3 and 4.4) deal with similarity meta knowledge and expert knowledge and investigate how those sources of background knowledge can be exploited to fill the knowledge filters mentioned, enabling them to actually conduct the search space restriction.

We want to point out that we also intend to utilise other, rather implementation-specific meta knowledge to ameliorate FLSM, in particular its run-time behaviour and its requirements regarding computational resources. Since these improvements, however, are conceptually quite different from the search space restriction we are promoting here, they are described within the subsequent implementation and optimisation chapter (Chapter 5).

4.2 Concepts

As mentioned in the previous section it is one of our aims to restrict the search space, i.e. the space of all representable similarity measures that are about to be searched, by exerting a bias on the respective learning algorithm so that it prefers certain regions of that space to other ones or completely avoids searching some subspaces of the entire search space. In other words, a bias towards certain types of similarity measures ought to be created.

One benefit of that proceeding is obvious: In general, the learner should be able to converge faster to an optimum since the search space is smaller. Another important matter is that certain optima may be held off from the search space. On the one hand, this is favourable, if those masked optima result from overfitting only: As the error functions we are dealing with and which we want to minimise (e.g. $\hat{E}_I$ or $\hat{E}_{SP}$, see Definitions 2.11 and 3.13) take a particular training data set TD as parameter, their shape throughout the entire search space is influenced by TD on a grand scale. Depending on the training data's size, quality and level of noise, additional optima may occur which are different from the "correct" global optimum, i.e. from the perfect similarity measure for the respective application domain. So, if it is possible to prevent the learner from running into those "overfit-optima" by restricting the search space, the probability of overfitting can be decreased. On the other hand, the confinement of the search will be disadvantageous, if it excludes the real optimum from the search space. However, that kind of trouble has to be blamed on erroneous background knowledge on the basis of which the search space restriction was carried out.

4.2.1 Knowledge-Based Optimisation Filters

In order to realise the restriction of the search space we introduce the concept of knowledge-based optimisation filters restricting the search space, which we briefly call *knowledge filters* in the following. With that term we refer to entities that, on the one hand, hold some knowledge concerning the learning of similarity measures that was gathered in several ways. On the other hand, they are meant to play an active role during the search for an optimal similarity measure insofar as they use their knowledge to explicitly direct the search.

The basic idea behind the definition of knowledge filters is to define "preferential regions" in the space of all representable similarity measures. Thus, from an abstract point of view, a knowledge filter covers – based on the knowledge it carries – the space with a probability function that determines which regions are considered to be more or less worthwhile searching.

There are a number of demands that should be made on an appropriate realisation of knowledge filters.

- They must be applicable to all elements of the representation mechanism for similarity measures that is used (feature weights, local measures and amalgamation function), at least to those that are actually optimised within FLSM's current implementation.

- Knowledge filters ought to be easily expandable so that other pieces of knowledge may be included in future.

- A third requirement concerns scalability: As similarity measures are composed of several elements (e.g. a local measure for each attribute of the chosen case representation), we should avoid using a single filter for the entire measure. Instead, the definition of a special knowledge filter for each attribute (i.e. each local similarity measure) as well as one additional filter for the feature weights is necessary. Hence, for a case representation consisting of n attributes we need $n + 1$ filters or $n + 2$, if the amalgamation function would be considered as well.

Returning to the abstract point of view mentioned above, a knowledge filter for one particular attribute (here we disregard feature weights) might be characterised as follows.

Definition 4.1 (Knowledge Filter Preference Function)
Let $\mathbb{S}$ be the set of all representable local similarity measures for attribute A. A knowledge filter K for A induces a preference function $pref_K : \mathbb{S} \rightarrow [0; 1]$ with $\int_{s \in \mathbb{S}} pref_K(s)ds = 1$.

Obviously, $pref_K$ represents a probability function on the space of local similarity measures for attribute A. So, for each $s \in \mathbb{S}$ the value of $pref_K(s)$ represents a probability and refers to the chance of testing s as a next candidate similarity measure, when performing a pure random search. However, we do not pursue the idea of restricting the search space with help of knowledge filters that are given via a complex, computationally intensive definition of a probability function. Instead, we want to realise filters as objects that include a number of constraints, preference relations or auxiliary information (which we will specify in more detail below). By using these information they are enabled to actively constrain the search space, while, on the other hand, the above-mentioned preference function is implicitly defined.

4.2.2 Intervening the Learning Process

An important question is in which way the knowledge filters interfere with the search process, in order to exert a bias toward certain types of similarity measures and thus to realise the preference function $pref_K$.

In general, there are two possibilities of intervening the learning process:

Quality Assessment Modification After the learner has generated a new hypothesis h – here, a new candidate similarity measure –, the respective knowledge filter K analyses h and determines to which extent that hypothesis contradicts the knowledge K includes. Depending on the degree of constraint violations, discrepancies with preference relations or mismatches with other knowledge pieces of K, the knowledge filter may assign an impairment on the quality of h in terms of an altered value of the objective function which is to be optimised. That way, the filter takes its active role during the evaluation of new hypotheses.

Hypotheses Creation Modification The alternative approach is to consider background knowledge already during the creation of new hypotheses. Here, the knowledge filter's task is to supervise and control the generation of new candidates in such a way that no (or as few as possible) contradictions to its prior knowledge occur, i.e. for example, hard constraints must always and soft ones should mostly be met. That way, the filter is given the chance to directly realise the preference function $pref_K$ that is defined by the knowledge it contains.

Working with evolutionary algorithms, the former strategy could be realised by imposing an additional fitness impairment to the respective individual's fitness, if the corresponding similarity measure is not consistent with the filter's background knowledge. In so doing the selection pressure would be increased: Individuals that actually yield a low retrieval error (eventually due to overfitting) might vanish from the evolutionary process rather soon, if the form of the similarity measure they represent does not accord to the demands given by the knowledge filter.

However, the definition of concrete numerical fitness impairment values turns out to be very complicated and troublesome. It is important to keep in mind that an individual's fitness and the strategy by which a fitness value is assigned to an individual has an extraordinary strong influence on the way a population develops. Accordingly, a wrong choice for fitness impairments might make the whole optimisation process unstable: Too small increments might be not influential enough to exert the desired bias towards certain types of similarity measures, while too large ones might outnumber the fitness assessments based on the retrieval error. Furthermore, the accurate quality impairments ought to be sufficiently granular: Some constraint violations should lead to higher impairments while other contradictions to the filter's knowledge should result in smaller ones. Besides that, the determination of concrete impairments should of course take the characteristics of the respective application and the absolute values of the retrieval error into account. These arguments make clear that a sophisticated realisation of the quality assessment modification strategy represents a highly complex optimisation problem in itself whose exploration goes beyond the scope of this work.

Apart from the described difficulties in realising that approach and in tuning its parameters, it brings about another crucial drawback: By incorporating knowledge into the learning process we also intend to speed up the optimisation. The most time-consuming part within FLSM's

algorithmic realisations, however, is the determination of the retrieval error induced by a certain similarity measure. Consequently, the number of necessary retrieval error computations should be minimised. Following the quality assessment modification strategy, the retrieval error has to be acquired for every single individual. If an individual or the corresponding similarity measure, respectively, does contradict to the knowledge filter's content for the most part, that individual would obtain a high fitness impairment and would probably be removed from the population immediately. And thus, the computational time required to calculate its retrieval error would have been wasted.

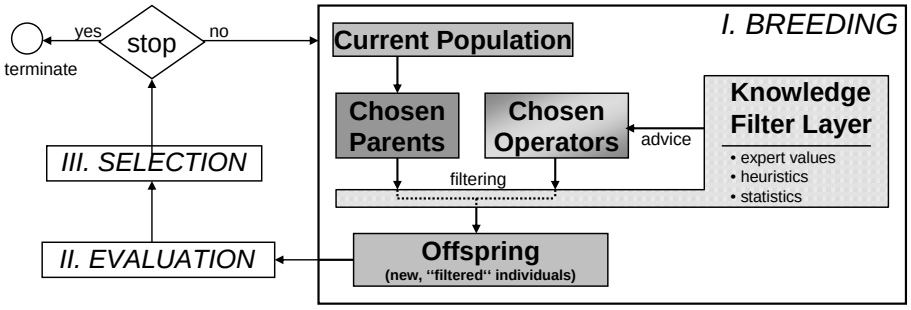


Figure 4.2: Intervention of Knowledge Filters to the Learning

For the reasons mentioned, we decided to follow the second approach – the hypotheses creation modification strategy – to incorporate the filters' knowledge into the learning process. Here, additional background knowledge influences the creation of new hypotheses, i.e. new candidate similarity measures, already. In the context of an evolution strategy this bias is supposed to be done during the generation of new individuals in the breeding stage (cf. Figure 2.7) of the evolutionary loop. At this, the knowledge filters we are employing exert their influence at the level of chromosomes: Based on the information they carry, knowledge filters may make some values for the genes of a new individual more probable or forbid other values, for instance. To do so, it is also allowed to adapt the behaviour of mutation and crossover operators (cf. Section 2.4.4). In Figure 4.2 we illustrate how knowledge filters can intervene the creation of offspring in general. The refined breeding phase of the evolutionary algorithm is extended by the layer of knowledge filters, in which the actual intentions (concerning offspring creation) of the evolution strategy are "filtered" so that they are in accordance to the filter's knowledge. Hence, a filter uses its heuristic, expert and statistical knowledge (cf. Section 4.1) to manipulate the creation of descendants. In the two conclusive sections (4.3 and 4.4) as well as (partly) in the next chapter we will explain in more detail how they perform that biasing. Before going into those details, we first need to settle on an appropriate knowledge filter composition.

4.2.3 Knowledge Filters and Their Elements

Let us concretise the structure of knowledge filters we propose, that shall enable them to realise the desired search space restriction. As already mentioned, each knowledge filter is responsible for one single element of the similarity measure representation only. Hence, when intending to confine the search space of the entire similarity measure, $n + 1$ appropriate knowledge filters are necessary (for n attributes plus one for the feature weights; the amalgamation function remains disregarded). Of course, our approach also works, if knowledge filters are defined for

some elements of the similarity measure representation only, e.g. just a single filter for the weights.

The contents of each knowledge filter is divided into three main parts as sketched in Figure 4.3: general knowledge, detail knowledge and management information.

a) General Knowledge

This part is specific for the respective element of the similarity measure representation the current filter is responsible for. So, it may for example contain general demands on one particular local similarity measure in form of constraints that this measure must fulfill. The general knowledge will be mainly influenced by heuristics on which we focus in Section 4.3.1.

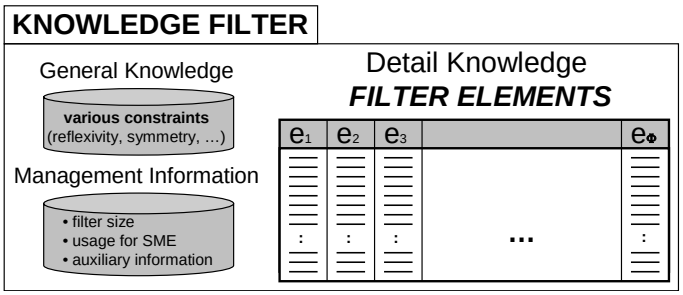


Figure 4.3: Composition of Knowledge Filters

b) Detail Knowledge (Knowledge Filter Elements)

The biggest part of the background knowledge with which we want to enhance the learning process, is available in a detailed, i.e. fine-grained, form. Depending on the respective element of the entire similarity measure representation to which this knowledge filter corresponds, a set $E = \{e_1, \ldots, e_\phi\}$ of so-called *knowledge filter elements* is employed. Each filter element is responsible for restricting the search space at some specific region of the entire space. For example, such a filter element e_j may demand the value for feature weight w_j to lie within $[0.4; 0.5]$, assumed that e_j belongs to a knowledge filter that is responsible for the feature weights.

If the knowledge filter at hand is used

- for the feature weights $(w_1, \ldots, w_n)$, then n knowledge filter elements are defined ($\phi = n$), one for each weight w_i.

- for a local similarity measure for a symbolic attribute A with value range $D_A = (d_1, d_2, \ldots, d_n)$, then we employ n^2 filter elements ($\phi = n^2$) corresponding to the entries in the similarity table that is used as local similarity measure for A.

- for a local similarity measure for a numeric attribute A, i.e. for a similarity function, then it is not as obvious how to define the filter elements. Here, we decided to make use of ϕ filter elements, where each one contains knowledge about a specific region of the distance-based similarity function's value range. That way its range $D_A = [A_{min}, A_{max}]$ is divided into ϕ disjoint parts for which a filter element is responsible in each case.

For reasons of easy interoperability with FLSM's evolution strategy we decided to set $\phi = s$, where s is the number of sampling points (cf. Section 2.4.2) a similarity function individual consists of.

Each of the knowledge filter elements introduced here is composed of several slots possessing specific pieces of information. The following section gives a short overview of their contents and tries to introduce a formalisation of knowledge filters and their elements.

c) Filter Management Information

Finally, each knowledge filter needs some general information about itself. Hence, this part summarises some information on the respective knowledge filter. Besides the number ϕ of knowledge filter elements included, called filter size, another important information is, for example, for which element of the similarity measure representation this filter shall be used (e.g. for a particular attribute or for the feature weights).

4.2.4 Knowledge Pieces and Their Formalisation

As mentioned before the main part of background knowledge, which shall be incorporated into the learning process, is contained in the knowledge filter elements, i.e. in the detail knowledge of a knowledge filter. The following list gives an overview of those knowledge pieces and points out from which of the described knowledge sources (see Section 4.1) they are derived.

The knowledge filters' elements we are proposing shall be filled with:

- **Relations** between the respective filter element and other ones: Those reflect greater-than or less-than relations between entries in a similarity table, between regions of a similarity function or between values of feature weights. In most cases these relations will have to be supplied by a human expert. However, sometimes they may also be inferred from other properties that are known about the respective element of the similarity measure representation.

- **Granularity Value**: Normally, each entry in a similarity table or the similarity value for a sampling point, for example, can be chosen arbitrarily from $[0; 1]$. When permitting to choose a limited number of values from that interval only, the search space can be confined enormously. The granularity value g determines how many values from the interval $[0; 1]$ are allowed to be chosen (that interval is divided into g equal parts): $g = 21$, for example, disapproves any value that is not a multiple of 0.05. Of course, granularity values might be specified by a human expert. But in Section 4.3.2 we show an approach how to gain sophisticated granularity values from a statistical case base analysis.

- **Expert Value and Confidence Level**: An expert may specify a potential value for the respective knowledge filter element and append how certain he/she is about his/her assumption. Depending on the expert's degree of confidence a more or less extensive search around the expert value should be conducted.

- **Lower and Upper Bound**: The definition of a lower and upper bound for the value the respective filter element is responsible for could also restrict the search space. The subinterval of $[0; 1]$ defined by these two values might be given by an expert, but can also be obtained from relations or from a confidence level.

- **Search Strategy**: This additional piece of information shall be used for choosing a specific strategy according to which the part of the search space, that this filter element is responsible for, is searched.

Based on these informal descriptions, we are now able to formally define knowledge filter elements. Note, that the following definition is only intended to introduce the required notation.

Definition 4.2 (Knowledge Filter Element)
*A **knowledge filter element** e_p, restricting the search space $\mathbb{S}$ at one single point p of the entire similarity measure representation, is a 7-tuple:*

$$e_p = (R, g, \mu_x, c, l, u, f)$$

Here, $R = \{r_1, \ldots, r_\rho\}$ is a set of relations, where each $r_i = (e_j, b)$ consists of a reference to another knowledge filter element e_j $(i \neq j)$ and $b \in \{<, \leq, =, \geq, >\}$ indicates the type of the relation. If v_i and v_j denote the respective similarity values for which e_i and e_j are responsible, then it must hold $v_i b v_j$. The filter element's granularity is determined by $g \in \mathbb{N}^+$, the expert value μ_x as well as the lower and upper bound l and u $(l \leq u)$ are within $[0; 1]$.

The confidence level c as well as the search strategy f are chosen from a set of possible values[3].

With help of Definition 4.2 we can also introduce a formalisation of the concept of knowledge filters. Note, that this definition disregards the filter management information, as a formal representation of this component is not needed in the following.

Definition 4.3 (Knowledge Filter)
Let $\mathcal{E}$ be an element of the similarity measure representation (weights or local measure) that, in its representation as an individual, is characterised by ϕ numerical similarity values. Further, be $\mathcal{C}$ a set of general demands by which $\mathcal{E}$ might be constrained.

*Then, a **knowledge filter** K for that element $\mathcal{E}$ is defined as a tuple*

$$K = (G_K, E_K)$$

where $E_K = \{e_1, \ldots, e_\phi\}$ is a set of knowledge filter elements that correspond to the ϕ similarity values, by which $\mathcal{E}$ is characterised, and G_K is a subset of $\mathcal{C}$.

After having introduced the concept of knowledge-based optimisation filters restricting the search space and clarified their compositional structure, we now, i.e. in the following sections, want to present several strategies how their contents, i.e. the actual background knowledge, can be acquired and how its incorporation into the optimisation process influences and changes the behaviour of FLSM's evolutionary learning algorithm and thus constrains the search space.

4.3 Similarity Meta Knowledge

In this section we want to investigate several approaches to employ meta knowledge about the similarity assessment with the goal of enhancing the learning process. Firstly, we define a set of basic constraints to local similarity measures whose fulfillment seems purposeful. Secondly, we intend to extract general knowledge from the available case data and introduce two different strategies how to exploit it as similarity meta knowledge with which to bias the learner.

[3] The elements contained in both sets are specified in the subsequent sections.

4.3.1 Heuristics on "Typical" Similarity Measures

Experience in the practical usage of CBR systems has shown that most of the similarity measures that are applied feature certain characteristics.

An awkward definition, like $sim_\delta(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{else} \end{cases}$ for an arbitrary symbolic attribute, would actually contradict to the CBR paradigm, according to which similar problem have similar solutions, and thus be very improbable to be used in practice. Nevertheless, peculiar measure like sim_δ are part of the search space $\mathbb{S}$ and, accordingly, have the chance of getting involved in the learning process.

By formulating a number of basic heuristic constraints, that should be fulfilled by local similarity measures, we intend to exclude unusual, highly improbable metrics from the search and thus to restrict the search space. Beforehand, we want to point out to two important issues: First, the considerations made in this section apply to local similarity measures only, not to feature weights. Second, the knowledge pieces identified within this section will be stored as general knowledge G_K in a knowledge filter K (see Figure 4.3) as it refers to an entire local similarity measure. Hence, in this section we want to identify a number of constraints whose fulfillment can be required from the candidate similarity measures that are being searched.

4.3.1.1 Normalisation Constraint

For the local-global principle (see Section 2.2.1) to function properly, we assume local similarity measures to be normalised, which basically means that the entire range $[0;1]$ of possible similarity values shall be exploited.

Definition 4.4 (Normalised Local Similarity Measure)
A local similarity measure sim for attribute A is called ***normalised****, if it holds:* $\exists x_1, y_1, x_2, y_2 \in D_A$ *with* $sim(x_1, y_1) = 0$ *and* $sim(x_2, y_2) = 1$.

When refraining from using normalised local similarity measure only, the effects of feature weights may be impaired and diluted. On the one hand, this bears the disadvantage that the search space is expanded artificially: Several combinations of non-normalised local similarity measures and feature weights can form the same overall similarity measure. An example illustrating that problem is shown in Figure 4.4. On the other hand, a learnt similarity measure is worse understandable for a human, when the effects of local similarities and weights are mingled.

The current implementation of FLSM's learning algorithms does not prohibit the generation of non-normalised local similarity measures. As their usage creates no benefit at all, we suggest to make use of a normalisation constraint in any case:

$$c_{norm} \in \mathcal{C} \text{ with}$$

$$c_{norm} \;=\; \ll \exists\, x_1, y_1, x_2, y_2 \in D_A \text{ with } sim_A(x_1, y_1) = 0 \text{ and } sim_A(x_2, y_2) = 1 \gg \qquad (4.1)$$

The algorithmic realisation of that normalisation constraint, when included in a knowledge filter element, must verify that the respective similarity measure fulfills c_{norm}. If not, i.e. if $sim(x,y) < 1\ \forall x, y \in D_A$ or $sim(x,y) > 0\ \forall x, y \in D_A$, the minimum $sim_{min} =$

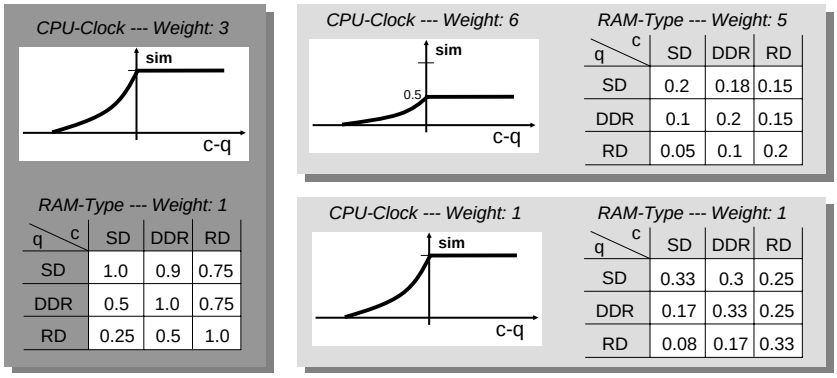


Figure 4.4: Example: Normalised Similarity Measure (left) and Two Examples (right) of Non-Normalised Variants with Identical Semantic

$\min_{x,y \in D_A}\{sim(x,y)\}$ and maximum $sim_{max} = \max_{x,y \in D_A}\{sim(x,y)\}$ must be determined and a normalisation according to

$$sim'(x,y) := \frac{sim(x,y) - sim_{min}}{sim_{max} - sim_{min}} \quad \forall x, y \in D_A$$

must be carried out. Here sim' represents the normalised local similarity measure that is to be used instead of sim, i.e. the knowledge filter is allowed to redefine the original similarity values of sim. In the following we assume all local similarity measures to be normalised.

4.3.1.2 Reflexivity Constraint

Most similarity measures employed in practice are supposed to be reflexive, as any entity can be considered as being maximally similar to itself.

Definition 4.5 (Reflexive Local Similarity Measure)
A local similarity measure sim for attribute A is called ***reflexive****, if it holds* $sim(x,x) = 1$ *for all* $x \in D_A$.

Of course, there are also examples of non-reflexive similarity measures that might be applied successfully. Imagine an e-commerce scenario including a case-based product retrieval, where the user has to specify a specific Price (among other attributes) for the respective query. A similarity function for that price attribute modelling human behaviour might reach its global maximum $sim(q,c) = 1.0$ (normalised measure) for $c - q < 0$ with $|c - q|$ small, whereas $sim(x,x)$ may be slightly less than one. This allows for the fact, that the average human customer is generally even more satisfied with a recommended product, if its price c falls short of the requested price q.

However, if there is no justified reason to make use of non-reflexive similarity measures, the constraint for reflexivity of a local similarity measures sim_A for attribute A

$$c_{refl} \in \mathcal{C} \text{ with } c_{refl} \;=\; \ll sim_A(x,x) = 1 \;\; \forall x \in D_A \gg \tag{4.2}$$

seems to be advantageous. The search space restriction induced by a constraint for reflexivity, can be realised as follows: When the reflexivity constraint is included in the knowledge

filter's general knowledge component, that filter may correct all points of reflexivity during the creation of a new individuals. That way it guarantees that at any point of time the current population includes individuals only for which it holds: $m^I_{ii} = 1$ for all i in the case of similarity matrix individuals and $v^I_{\lceil \frac{s}{2} \rceil} = 1$ in the case of similarity vector individuals.

4.3.1.3 Symmetry Constraint

The idea of introducing a symmetry constraint is inviting, since its application would approximately halve the search space. In the case of a symbolic attribute A_s with $D_{A_s} = n$, for example, the number of alterable entries in the respective similarity measure (individual) sinks from n^2 to $\frac{n^2+n}{2}$.

Definition 4.6 (Symmetric Local Similarity Measure)
*A local similarity measure for attribute A is **symmetric**, if it holds $sim(x, y) = sim(y, x)$ for all $x, y \in D_A$.*

Of course, in many application scenarios, asymmetric local similarity measures are indispensable. For instance, the capabilities of a case-based product recommendation system strongly relies on asymmetric measures. Consider a local measure for the attribute CPUClock illustrated in Figures 2.2 and 4.4. Here, the question whether a PC's clock rate is higher or lower than the user's demand plays a significant role, as a user will in general accept a PC that is faster than actually desired, whereas he/she would probably neglect a too slow one. Hence, only an asymmetric measure correctly captures the customer's preferences leading to a retrieval result that returns high-utility cases for the user's query.

The type of application domains about which we have spoken in Chapter 3, i.e. classification and solution prediction tasks, however, represents an example for a class of domains, where the use of symmetric similarity measures may – under certain circumstances – be sufficient. Assume a two-class problem and two cases c_1 and c_2 belonging to the same class, the first one being used as query. Accordingly, $sim(c_1, c_2)$ should be relatively high, so that c_2 may be retrieved as c_1's nearest neighbour. Using c_2 as query the other way round, the benefit of c_1's solution (class) for c_2 is the same as c_2's for c_1 and so c_1 is expected to be c_2's nearest neighbour as well. Therefore, a symmetric similarity measure calculating $sim(c_1, c_2) = sim(c_2, c_1)$ would perform very well. Generalising that example, we can say that a symmetric similarity measure is appropriate, if the respective domain's solution similarity measure is symmetric as well. Apart from that, an expert often finds it easy to determine for a specific application domain whether a certain local similarity measure should be symmetric or not.

To force the creation and handling of symmetric local similarity measures sim_A for a particular attribute A only, a symmetry constraint

$$c_{symm} \in \mathcal{C} \text{ with } c_{symm} \;=\; \ll sim_A(x, y) = sim_A(y, x) \;\; \forall x, y \in D_A \gg \tag{4.3}$$

can be added to the general knowledge G_K of the respective knowledge filter. Similarly as in the case of the reflexivity constraint, the presence of a symmetry constraint ought to change the behaviour of the offspring creation phase, so that only individuals that correspond to symmetric local similarity measures are generated.

In order to realise this, the knowledge filter (see knowledge filter layer in Figure 4.3) is allowed to overwrite some of the elements of an individual, that has been created by a genetic operator, to make the corresponding local similarity measure symmetric. To be exact, in the

case of similarity table individuals I the filter must guarantee that it holds $m_{ij}^I = m_{ji}^I$ for all $i, j \in \{1, \ldots, n\}$ and $v_i^I = v_{s+1-i}^I$ for all $i \in \{1, \ldots, s\}$ in the case of similarity function individuals I, respectively.

4.3.1.4 Monotony Constraints

According to common understanding, larger distances between the query's and case's value of an attribute make them more dissimilar, while smaller ones let them appear rather similar. Indeed, this heuristic is true for most application domains Case-Based Reasoning has been applied to. Based on that foundation, we now define what we understand under a monotonous distance-based similarity function.

Definition 4.7 (Monotonous Similarity Function)
Let sim_A be a distance-based similarity function for a numeric attribute A. If it holds

$$sim(x, y_1) \geq sim(x, y_2)\ \forall x, y_1, y_2 \in D_A \text{ with } 0 \leq y_1 - x \leq y_2 - x \text{ or } y_2 - x \leq y_1 - x \leq 0$$

we call sim_A ***monotonous****.*

The global maximum of a monotonous similarity function lies on the y-axis (i.e. distance is zero), for negative case-query differences it is increasing monotonously and for positive ones it is decreasing monotonously.

Nevertheless, one might also construct a scenario in which non-monotonous similarity functions are to be preferred. Assume a case base that basically consists of several natural numbers, $CB = \{(3, odd), (4, even), (7, odd), (8, even), (9, odd)\}$, where the second attribute is the class attribute and the task is to predict the class (even or odd) of a new number. Here, only a non-monotonous measure, like $sim(x, y) = \begin{cases} 1 & \text{if } (x - y)\ mod\ 2 = 0 \\ 0 & \text{else} \end{cases}$ would yield maximal classification accuracy. Given a query $q = (6, ?)$, sim would return $(4, even)$ and $(8, even)$ as nearest neighbour, whereas any monotonous function would have assigned the highest similarity to $(7, odd)$ and thus also included into the set of nearest neighbours.

Although this example is rather artificial, it shows that the demand for monotony is not advantageous at all times. The usage of non-monotonous distance-based similarity functions may be beneficial in domains with adaptation functionality where, for instance, the value of a particular attribute can be changed by a certain increment only (as a result of applying an adaptation rule).

However, because monotony has proved to be a reasonable heuristic for defining similarity functions in many application domains, we decided to allow any knowledge filter for a numeric attribute A to employ a monotony constraint:

$$c_{mon} \in \mathcal{C} \text{ with}$$

$$\begin{aligned} c_{mon} \ = \ \ll & sim_A(x, y_1) \geq sim_A(x, y_2)\ \ \forall x, y_1, y_2 \in D_A \\ & \text{with } 0 \leq y_1 - x \leq y_2 - x \text{ or } x - y_2 \leq x - y_1 \leq 0 \gg \end{aligned} \tag{4.4}$$

Assuring Monotony

To ascertain the monotony of a similarity function we employ the relations R that are designated to be part of each knowledge filter element. Hence, when the monotony constraint c_{mon} is included in the general knowledge G_K of a knowledge filter $K = (G_K, E_K)$, its presence ought to induce the creation of a set of relations R for each knowledge filter element in E_K, according to the rules specified in Algorithm 4.1.

Input : knowledge filter $K = (G_K, E_K)$ with $E_K = \{e_1, \ldots, e_\phi\}$, for a numeric attribute A and $\phi \in \{2n+1 | n \in \mathbb{N}^+\}$
Output : K with extended relation sets $e_i.R$

1. **for all** $i = 1$ **to** ϕ **do**

   a) **if** $i = 1$ **then** $e_i.R := e_i.R \cup \{(i+1, \leq)\}$
   else if $i = \phi$ **then** $e_i.R := e_i.R \cup \{(i-1, \leq)\}$
   else if $i = \frac{\phi+1}{2}$ **then** $e_i.R := e_i.R \cup \{(i-1, \geq), (i+1, \geq)\}$
   else if $i < \frac{\phi+1}{2}$ **then** $e_i.R := e_i.R \cup \{(i-1, \geq), (i+1, \leq)\}$
   else $e_i.R := e_i.R \cup \{(i-1, \leq), (i+1, \geq)\}$

2. **return** K

Algorithm 4.1: Relations to Realise the Monotony Constraint

The relations created by this algorithm guarantee that the similarity values at the sampling points of a similarity function form a monotonous measure. During offspring creation the knowledge filter must make sure that none of these relations are violated. To do so it is allowed to give advice to the involved genetic operators, i.e. to bias them. On the one hand, this task is not as trivial as in the case of the reflexivity or symmetry constraint. On the other hand, that kind of advice is not specific to the monotony constraint, but concerns and should be always given, if relations within the knowledge filter elements are to be regarded. For these reasons, we will describe the realisation of that advice and the extended genetic operators in detail in the Chapter 5.

Weak Monotony Constraint

When comparing the search space restriction introduced by the four types of constraints we have defined so far, the monotony constraint in general exerts the strongest bias. In order to reduce its restrictiveness we suggest the usage of an additional constraint, that we call *weak monotony constraint* and which actually represents an extension to the monotony constraint:

$$c_{monW} \in \mathcal{C} \text{ with}$$
$$c_{monW} = \ll sim_A(x, y_2) - sim_A(x, y_1) \leq \epsilon \quad \forall x, y_1, y_2 \in D_A$$
$$\text{with } 0 \leq y_1 - x \leq y_2 - x \text{ or } x - y_2 \leq x - y_1 \leq 0 \gg \tag{4.5}$$

Here, the parameter ϵ can be used to determine the strictness of the demand for monotony. For $\epsilon = 0$, of course, it holds $c_{monW} = c_{mon}$, for $\epsilon > 0$ small deviations from monotony are permitted.

4.3.2 Mining Knowledge from the Case Base

Considering a single local similarity measure sim, one can say that the similarity knowledge that is included in sim is distributed over several elements: in the case of similarity tables over its n^2 entries and in the case of similarity functions over a few parameters or sampling points describing that function.

Without any doubt, some of the measure's elements can be characterised to be more important than other ones, i.e. carrying "more valuable knowledge", than other ones, insofar as they are – during the practical usage of the CBR system – consulted more frequently. That means they are more frequently used to determine a query's and a case's similarity regarding the respective attribute. Therefore, we ought to aim that those, more frequently used parts of the similarity measure are outmost correct, since an erroneous similarity definition in these regions would have a higher negative impact on the CBR system's overall performance than in other ones. A typical example for such a region is – in the case of a distance-based similarity function sim_A that is defined over $[A_{min} - A_{max}, A_{max} - A_{min}]$ – the area around the y-axis, i.e. all $sim_A(x, y)$ with $|x - y|$ near zero. For a learning algorithm, this implies that it should focus more on such "high-importance regions", conducting a more thorough search there, while it should be permitted to spend less effort in other "low-importance regions".

But, which regions are of high importance? We intend to answer that question in the following by means of carrying out a statistical case base analysis. In so doing, we want to find out (specifically for single attributes and local similarity measures, respectively) which entries within a similarity table and which regions of a similarity function's value range are more frequently consulted and which can thus be considered to be of higher importance. We need to stress in prior, that all considerations we are doing here, presume a sufficiently substantial case base which in particular is also representative (in its attribute-value distributions for all attributes) for queries occurring typically in practice. A proceeding based on that assumption is especially trouble-free in classification and solution prediction domains where we apply introspective learning (see Chapter 3), i.e. where each $c \in CB$ is used as query during learning in the context of a leave-one-out test strategy so that the attribute-value distributions given by the case base are implicitly representative for the queries used during learning.

For other application domains a natural extension would be to confine the statistical analysis not only to the case base (as we do here), but to extend it to the training data sets (case order feedback) as well.

4.3.2.1 Statistical Case Base Analysis

Illustrating our idea of obtaining knowledge from analysing case data, we present a little example from the PC domain introduced in Section 2.3.3. Here, we focus on a numeric attribute GraphicMemory, short A_{GR}, whose value range is $D_{A_{GR}} = [4; 128]$. We assume a case base of 18 cases, $CB = \{c_1, \ldots, c_{18}\}$, whose attribute values for attribute GraphicMemory are set as follows: $c_i.A_{GR} = r$ with $r = 4$ for $i \in \{1, \ldots, 3\}$, $r = 8$ for $i \in \{4, \ldots, 6\}$, $r = 16$ for $i \in \{7, 8\}$, $r = 32$ for $i \in \{9, 10\}$, $r = 64$ for $i \in \{11, \ldots, 14\}$ and $r = 128$ for $i \in \{15, \ldots, 18\}$.

Of course, this tiny case base is to small to be considered representative. Nevertheless, we now presume it to be and proceed on the assumption that the distribution of all $c_i.A_{GR}$ given by the case base is representative and reflects the frequency with which certain values from $D_{A_{GR}}$ occur in practice.

Based on the attribute value distribution for A_{GR} (given by CB), we can sum up which

combinations of certain case values and query values occur how often for A_{GR}, if each $c \in CB$ is used once as query (leave-one-out test). Moreover, we are able to calculate the corresponding distances between case and query values. The results of these calculations for the example at hand are shown in Figure 4.5. The negative part of $sim_{A_{GR}}$'s value range $D_{A_{GR}} = [-124; 124]$ has been omitted in that figure as it is symmetric to the part right from the y-axis.

Obviously, some regions of $D_{A_{GR}}$ are used rather often for similarity computations, while other ones are not at all. This fact becomes even clearer, when dividing $D_{A_{GR}}$ into s subintervals (corresponding to the s sampling points that are used to interpolate $sim_{A_{GR}}$) and summing up the frequencies of case-query distances for each subinterval (negative half omitted as well).

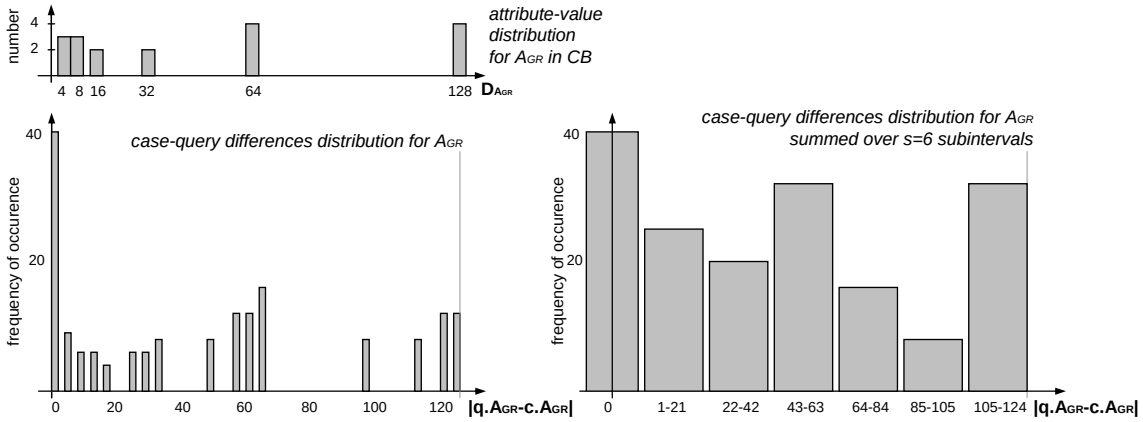


Figure 4.5: Examples for Case Base Analysis: Which Regions of $sim_{A_{GR}}$'s definition are of higher importance?

Although this example was designed for a numeric attribute, we want to point out that the analysis of occurrence frequency of certain case-query attribute values can be applied to symbolic attributes analogously.

Formalisation

Formalising the ideas presented with an example, we primarily need to introduce a measure that is able to reflect which parts of a local similarity measure definition are of higher importance due to being referred to rather often. In Figure 4.5 these regions may be identified easily through the height of the sketched bars. In order to define such a measure we first have to formalise the frequency with which certain values from an attribute's value range occur.

Definition 4.8 (Attribute-Specific Value Frequency)
Let A be an attribute with value range D_A and $CB = \{c_1, \ldots, c_m\}$ a case base of m cases. The ***attribute-specific value frequency*** *h_A is defined as*

$$\begin{aligned} h_A : D_A &\to \mathbb{N} \\ x &\mapsto \textstyle\sum_{i=1}^{m} \delta(c_i.A, x) \end{aligned}$$

$$\text{where } \delta(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{else} \end{cases}.$$

Based on the attribute-specific value frequency and on the above-mentioned presumption about the case base's representativeness, we may now specify a measure that expresses which regions of a local similarity measure are used how frequently.

Definition 4.9 (Similarity Function Consultation Frequency)
*Let A be a numeric attribute with value range $D_A = [A_{min}, A_{max}]$ and $CB = \{c_1, \ldots, c_m\}$ a case base of m cases. Moreover, let sim_A be a distance-based similarity function that is defined over $I_A = [A_{min} - A_{max}, A_{max} - A_{min}]$ and let h_A be the attribute-specific value frequency. Then, the **consultation frequency** of sim_A and I_A, respectively, is the auto correlation of h_A, which is defined as*

$$c_n : I_A \to \mathbb{N}$$
$$d \mapsto \begin{cases} \sum_{i=-\infty}^{\infty} h_A(i) \cdot h_A(i-d) & \text{if } d \neq 0 \\ \sum_{i=-\infty}^{\infty} h_A(i) \cdot (h_A(i) - 1) & \text{else} \end{cases}$$

where n denotes that A is a numeric attribute.

In that definition the semantic of the leave-one-out test strategy is taken into consideration by the special case for $d = 0$: If a certain case c_j is used as query during a leave-one-out test and, in so doing, excluded from CB, the number of cases c_i for which it holds $c_i.A = c_j.A$ must be decreased by one.

In analogy to Definition 4.9 we can define the consultation frequency of similarity tables.

Definition 4.10 (Similarity Table Consultation Frequency)
*Let A be a symbolic attribute with value range $D_A = \{d_1, \ldots, d_n\}$ and $CB = \{c_1, \ldots, c_m\}$ a case base of m cases. Moreover, let sim_A be a similarity table for A and let h_A be the attribute-specific value frequency. Then, the **consultation frequency** of sim_A, is the auto correlation of h_A, which is defined as*

$$c_s : D_A \times D_A \to \mathbb{N}$$
$$(x, y) \mapsto \begin{cases} h_A(x) \cdot h_A(y) & \text{if } x \neq y \\ h_A(x) \cdot (h_A(x) - 1) & \text{else} \end{cases}$$

where s denotes that A is a symbolic attribute.

After having introduced a way to estimate the importance of particular parts of a local similarity measure definition, we now have to find a way how to employ that knowledge in the scope of FLSM.

4.3.2.2 Exploitation of the Statistical Knowledge

The crucial issue, after having conducted a case base analysis, concerns the employment of the knowledge obtained. In the following we present two approaches by means of which the knowledge about consultation frequencies of certain regions of a local similarity measures' value range can be incorporated – as similarity meta knowledge – into the optimisation process in order to improve the learning. While the first approach may be employed for all kinds of local measures, the second one works for similarity functions only.

a) Granularity

Each knowledge filter element e is responsible for a similarity value v that can be chosen from $[0;1] \subset \mathbb{R}$. Accordingly, an infinite number of possible values may be taken into consideration for v. Hence, a self-evident strategy to efficiently restrict the search space is to introduce a "grid" for the respective similarity value v forcing it to be an element of $\{0, \frac{1}{g}, \frac{2}{g}, \cdots, \frac{g-1}{g}, 1\}$, where $g \in \mathbb{N}^+ \cup \{\infty\}$ determines the filter element's granularity[4]. That way, the filter element demands that v is a fractional number and allows to choose one out of $g+1$ values for it. A setting of $g = \infty$ shall be interpreted as allowing any real number for v, i.e. making no use of granularity restrictions. Without any doubt, a massive restriction of the search space can be realised with the usage of such a grid, especially if the granularity value g is small.

The results from a statistical case base analysis may be perfectly used to determine appropriate filter element granularities. On the one hand, the consultation frequency $c_s(x,y)$ of, for example, a specific entry in a similarity table may be very high. This indicates that this entry is supposed to be consulted very frequently and that it therefore should be adjusted as accurate as possible. Moreover, this reveals that the case base contains a lot of information about the combination "x as query value and y as case value". For that reason, one may conclude that learning of $sim_A(x,y)$ is less vulnerable to overfitting and that its value may be chosen on the basis of a higher granularity.

On the other hand, $c_s(x,y)$ may be a rather low value. Then it is clear, that, for example, the corresponding entry in A's similarity table will be consulted rarely. Consequently, a fine-grained definition for $sim_A(x,y)$ is, generally speaking, not necessary or, at least, not very important (regarding the CBR system's overall performance). Furthermore, the low value of $c_s(x,y)$ suggests that CB features little information about that case-query combination for attribute A only. Hence, learning $sim_A(x,y)$ suffers from a high risk of overfitting, so that the restriction of $sim_A(x,y)$ to a comparatively low number of possible values seems promising.

The following definition introduces our approach to restrict the search space via granularity levels, based on the knowledge gained from an analysis of the case base data.

Definition 4.11 (Granularity Values from Consultation Frequencies)
Let A be an attribute with value range D_A, CB a case base of m cases, K a knowledge filter that is responsible for A and $e \in E_K$ a knowledge filter element, which is responsible for

- *$sim_A(x,y)$ with $x,y \in D_A$, if A is symbolic*
- *sampling point $s_k \in [A_{min} - A_{max}, A_{max} - A_{min}]$ and thus for all $x,y \in D_A$ with $|x-y| = s_k$, where $k \in \{1,\ldots,s\}$, if A is numeric*

*Then, it holds for e's **granularity value** g, i.e. for e.g:*

- *if A is symbolic:* $g = 1 + \left\lceil \frac{c_s(x,y)}{m^\gamma} \right\rceil$
- *if A is numeric:* $g = 1 + \left\lceil \frac{\int_{s_k-\delta}^{s_k+\delta} w(x,s_k) \cdot c_n(x) dx}{m^\gamma} \right\rceil$ *with* $\delta = \frac{2 \cdot (A_{max} - A_{min})}{s-1}$ *and* $w(x,y) = 1 - \frac{|x-y|}{\delta}$

where γ is a parameter to scale the entire granularity assessment.

[4] In the following, we omit the leading "e.", as it is clear that g refers to the filter element e.

In the case of dealing with a symbolic attribute, the consultation frequencies can be employed directly to determine appropriate granularity values. As A's value range is continuous, if it is a numeric attribute, the computation of integrals, whose bounds are determined by the similarity function's sampling points and offset δ, is necessary. However, because the amount of available case data is finite, that calculation scales back to forming according finite sums instead of integrals. In practical tests we found that setting $\gamma = 1$ produces convincing results.

Concerning the question how those granularity values are actually employed to actively restrict the search space, we refer to the implementation chapter (Sections 5.2.3 and 5.3.3). Here, we only want to remark that it must be guaranteed that for each knowledge filter element and the belonging similarity value v it holds $v = \frac{n}{g}$. To illustrate the above-mentioned definition for g and the search space restriction induced, the left part of Figure 4.6 reverts to the introductory example from the beginning of this section and shows how the granularity values are used. Note, that due to the unrealistic small case base ($m = 18$) very small granularity values occur – to weaken that effect we have set $\gamma = \frac{2}{3}$.

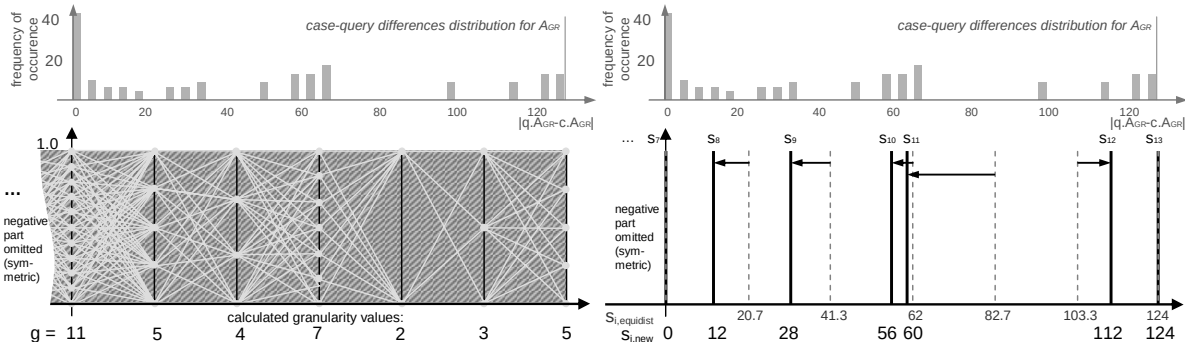


Figure 4.6: Examples for Granularity Value Determination (left) and Sampling Point Distribution (right) on the Basis of Consultation Frequencies for $sim_{A_{GR}}$

b) Sampling Point Distribution

Up to now we have presumed that the sampling points, that are used to represent a distance-based similarity function sim_A as an individual (similarity vector) in the context of an evolution strategy, are distributed uniformly over the value range $I_A = [A_{min} - A_{max}, A_{max} - A_{min}]$ of sim_A (cf. Figure 2.9). As Section 4.3.2.1 has taught, some regions of I_A may be of less importance for the similarity assessment than other ones – namely those regions with a rather low consultation frequency c_n. Consequently, in those low-importance regions – let us call them $R_- \subset I_A$ – we actually need not to interpolate the similarity function as elaborate as in other, high-importance regions R_+. Hence, less sampling points ought to be placed in R_-, while a higher number of sampling points and thus a better approximation of the similarity function seems suitable for R_+ regions. In short, an equidistant distribution of sampling points does not correspond to nuances in consultation frequency.

Algorithm 4.2 presents a sophisticated way to distribute the sampling points over I_A based on the knowledge gained from a case base analysis: Regions of I_A that are consulted more often are granted a higher sampling point density, whereas R_- regions are sparsely populated by sampling points.

Input : attribute A with value range $D_A = [A_{min}, A_{max}]$, odd number s of sampling points with $s \geq 3$ and a case base CB with $|CB| = m$

Output : vector of sampling points $S = (s_1, \ldots, s_s) \in [A_{min} - A_{max}, A_{max} - A_{min}]^s$ with $s_i < s_j$ for all $i < j$

1. compute integral of all consultation frequencies for positive case-query distances ($c.A - q.A > 0$):
$$\pi := \int_0^{A_{max}-A_{min}} c_n(x)dx \; - \; c_n(0)$$
2. **for** $i = 1$ **to** $\frac{s-1}{2}$ **do**
   a) determine maximal $t \in [0; A_{max} - A_{min}]$ so that
   $$\int_0^t c_n(x)dx \; - \; c_n(0) \; \leq \; \frac{i}{\frac{s-1}{2}} \cdot \pi$$
   b) **set** $s_{i+\frac{s+1}{2}} := t$
3. **for** $i = 1$ **to** $\frac{s-1}{2}$ **do** $s_i := s_{s+1-i}$
4. **set** $s_{\frac{s+1}{2}} := 0$
5. **return** $(s_1, \ldots, s_s)$

Algorithm 4.2:
Non-Equidistant Sampling Point Distribution on the Basis of a Case Base Analysis

As the computed consultation frequencies are symmetric with respect to I_A, the non-equidistant sampling point distribution calculated by this algorithm is symmetric as well. For the example from above (similarity function $sim_{A_{GR}}$), π is determined to 133 (Step 1). The resulting sampling point distribution for positive case-query differences, as calculated in Step 2, is sketched in the right part of Figure 4.6. Step 3 mirrors the distribution of sampling points calculated so far to the negative part of I_A, and Step 4 sets the intermediate sampling point (with index $\frac{s+1}{2}$) to $x = 0$.

4.4 Expert Knowledge

If no learning functionality is provided, a CBR system's capabilities rely (apart from the other knowledge containers' contents) on the similarity measures defined by a knowledge engineer only. On the other hand, if the knowledge engineer is unable to specify an appropriate similarity measure, the system's only chance to obtain an adequate similarity measure is by means of a machine learning technique (provided that training data is available).

In practice, however, in many cases none of these extremes occur. Instead, the expert usually possesses at least some partial knowledge about the needed similarity measure. And often some training data, that may help learning a similarity measure, is available, too. So a combination of "the best of both worlds" might permit the following advantages:

- The knowledge acquisition effort is decreased. The expert first issues his/her (partial)

knowledge about the respective measure, then presses a button and the remaining parts of the similarity measure are learnt automatically.

- Since the expert is not urged to specify the similarity measure completely, he/she is not forced to make educated "guesses" about elements of the measure he/she actually does not know much about.
- Due to the additional partial knowledge given by the expert, the learner might less likely tend to overfit its learning results to the training data.
- As the search space is restricted by the expert knowledge, not only the danger of overfitting, but also the time required for learning might be decreased.

Apart from these benefits, one should not forget that the knowledge engineer is fully responsible for the definition of the CBR system's vocabulary knowledge as well. Knowledge about the vocabulary, in particular about structured data types a CBR system makes use of, can be employed to a-priori restrict the search space. Therefore, we also aim at taxonomic and ordered symbolic data types whose structured value ranges provide information about regions of the search space that ought to be excluded from the search.

The remainder of this section is divided into four parts. First, we focus on feature weights and examine how expert knowledge can support learning them. Second, we identify several strategies how a knowledge engineer might bring in his/her more or less uncertain and partial knowledge and we prospect how that uncertain knowledge can be employed to guide the optimisation process. The two conclusive parts of this section address the utilisation of vocabulary knowledge, as specified by the knowledge engineer, and in so doing examine the exploitability of taxonomic and ordered symbolic data types to restrict the search space, when learning local similarity measures for corresponding attributes.

4.4.1 Attribute and Weight Preferences

As emphasised in Chapter 2, feature weights have a crucial influence on the entire similarity calculation. So, for example, a wrong choice of feature weights can distort the similarity assessment, even if a lot of effort has been put into the tuning of local measures. In practice, however, knowledge engineers find it often difficult to translate their implicit knowledge about the relative importance of attributes to explicit numerical ratios. Experts usually do not care or cannot say, whether A_1 is two or three times as important as A_2. Yet, the answer to questions like that may have an enormous impact on the quality of the entire similarity measure.

We argue that it is much easier for an expert to formulate a number of preference relations by which he/she can determine a partial order of weights, ordered with respect to the relevance of the corresponding attributes. The task of assigning concrete numerical values to the weights can then be left to a machine learning algorithm.

Assume a case representation consisting of six attributes. An expert may, for example, utter that he/she considers A_1 to be less important than A_3, A_2 to be more important than A_3 and A_3 to be more important than A_4, while having no idea about A_5's importance and knowing that A_3 and A_6 are equally important. On the one hand, these simple relations induce an attribute preference graph as depicted in Figure 4.7. On the other hand, we can wrap this expert knowledge into the sets of relations R that are part of each knowledge filter element: From knowing that it must hold $w_1 < w_3$, $w_4 < w_3$, $w_3 < w_2$ and $w_3 = w_6$, a total of fourteen

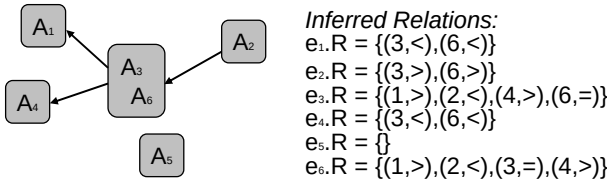


Figure 4.7: Example for an Expert's Attribute Preferences

relations can be added to the six knowledge filter elements ($e_1, \ldots, e_6$; one for each weight) that are involved here. In doing so the number of allowed weight combinations for $w_1, \ldots, w_6$ – and thus the number of possible feature weight individuals that may be generated – is reduced. Of course, the degree of restriction depends on the number of preference relations the expert is able to provide. Here, the optimal, i.e. maximally restrictive, case is present, if the knowledge engineer has specified a total order concerning attribute relevance. Then, the search space restriction implied is comparable to the restriction introduced by the monotony constraint for similarity functions.

Our extension to FLSM assures the compliance with those relations during the generation of new feature weight individuals, i.e. in the breeding phase of the evolution strategy – just as in the case of the relations that are incorporated, when using the monotony constraint (see Section 4.3.1.4) for distance-based local similarity measures. Concerning implementation-specific details on the advice for and extension of genetic operators to produce offspring that is consistent with the relations mentioned here, we refer to Sections 5.2 and 5.3.

4.4.2 Expert Estimations

In the previous section we have spoken about the difficulties a knowledge engineer encounters, when he/she is asked to assign accurate feature weights. The situation for local similarity measures, however, is even worse, because here a large number of concrete parameter or similarity values has to be devised for each attribute. Moreover, the effects of changing a single value (e.g. one single entry within a similarity table) are not as obvious as in the case of feature weights. As a consequence, many of the values that have to be determined during a bottom-up definition of a similarity measure are left to the expert's intuitiveness and thus mostly represent estimations of the correct value only. Furthermore, the numeric similarity values returned by a similarity measure convey the impression of originating from a maximally accurate measure, so that the CBR system's user will in general not doubt the measure's correctness.

On the other hand, the larger search space for local similarity measures (compared to the space of feature weights, for instance) implies an increased risk of overfitting for a learning algorithm. Hence, expert estimations on a similarity value somewhere in the search space represent an excellent knowledge source to restrict that search space.

In this section we examine two further ways for a knowledge engineer to incorporate his/her knowledge into the learning process. Apart from that, we introduce several strategies according to which a learner can exploit that knowledge. Note, that the considerations made in the following may be applied not only to local similarity measures, but to feature weights as well.

Bound Specifications

An obvious possibility enabling an expert to directly cut off parts of the search space is by allowing him/her to define lower and upper bounds for specific similarity values returned by the similarity measure. For instance, he/she may decide that for a symbolic attribute's values d_1 and d_2 it holds: $sim(d_1, d_2) \in [0.6; 0.8]$. Consequently, in this scenario the learning algorithm does not have to take the whole interval $[0; 1]$ into consideration for $sim(d_1, d_2)$, but only that subinterval. Of course, the degree of restriction achieved here depends on the expert's subinterval specification and thus on his/her experience.

The utilisation of this *bounds approach* is provided already within the knowledge filter elements (see Section 4.2.4): Each element e includes an entry l for the lower and u for the upper bound, which by default should be set to 0.0 and 1.0, respectively. Evidently, the behaviour of the crossover operators employed by FLSM (cf. Section 2.4.4) does not interfere with these kinds of constraints, i.e. given two parent individuals, which are consistent with lower and upper bound specifications, the application of simple, arbitrary as well as arithmetic crossover will not cause any bound violation. Sure enough, this is not true for mutation operators; for this reason in Section 5.2 we focus on the advice given from filters to mutation operators in order to enable them to handle bound restrictions appropriately.

Specificational Effort and Confidence

Although the way of restricting the search space with help of specifying subintervals of $[0; 1]$ is straightforward and effective, it bears a crucial drawback: When intending to define a lower and an upper bound for each single knowledge filter element involved, the effort for the knowledge engineer accomplishing the specifications is extremely high – probably more time-consuming than a manual bottom-up definition of the entire measure. Instead of n^2 entries in a similarity table for a symbolic attribute, for instance, he/she would now be required to specify $2n^2$ bound values. As a consequence, this proceeding seems to be reasonably applicable for a limited number of knowledge filter elements only, e.g. for those concerning whose actual value the expert is rather sure and thus able to specify a small subinterval.

In response to the problems mentioned we also developed a more human-centred, *confidence-based approach*. Here, the expert may (eventually only partially) define the similarity measure in the usual bottom-up manner. In the context of knowledge filters we consider his/her specifications as expert values μ_x, that are stored within the knowledge filter elements. Moreover, the expert is allowed to add an assertion about the level of confidence $c \in C$ regarding his/her specification.

Definition 4.12 (Confidence Levels)
The set C of allowed ***confidence levels*** *is defined as*

$$C = \{uncertain, low, average, high, certain\}$$

For example, he/she may state $e_{1,2}.\mu_x = 0.7$ and $e_{1,2}.c = high$ (where $e_{1,2}$ denotes the knowledge filter element that is responsible for $sim(d_1, d_2)$ from the example above) to express that he/she is quite certain, that it holds $sim(d_1, d_2) \approx 0.7$. That type of knowledge specification implies the following advantages:

- The knowledge engineer is not obliged to input μ_x and c for all knowledge filter elements (this is true for bound specifications via $[l, u]$ as well). Hence, he/she can provide the

learner with additional knowledge for certain search space regions while refraining from doing so for other regions.

- Since confidence values do not necessarily have to be defined for each knowledge filter elements separately, but may be for a knowledge filter as a whole (hence, for the entire weight vector, similarity function or similarity table) or for certain parts of the filter at once (e.g. for those filter elements that refer to a single or a number of rows or lines in a similarity table), the specificational effort is less than in the case of employing the bounds approach.
- The semantic of the confidence levels from C may be interpreted differently by the learner, depending, for example, on the respective application domain or on the knowledge engineer's experience. This means, the search space restriction induced by confidence levels is not as strict and inflexible as the one induced by bound specifications.

Filter Element Search Strategy

Let us for now assume that an expert value μ_x for a filter element e together with a belonging confidence level c are mapped (according to some strategy) to an interval $I_c = [l_c, u_c]$ so that it holds $l_c \leq \mu_x \leq u_c$.

Then, the question remaining is, how the learning algorithm searches through I_c, i.e. which regions of I_c it favours. Naturally, the region around the expert's explicit suggestion μ_x should be favoured over, for instance, the limits of I_c.

As follows, we present three different approaches regarding that issue: Each one introduces a probability function for a knowledge filter element, which is defined over $\mathbb{R}$ with respect to I_c and which can be understood as a "contributor" to the overall knowledge filter preference function $pref_K$ realised by the knowledge filter K (cf. Definition 4.1).

In contrast to the usage of expert-specified bounds l and u, whose usage is intended to simply ensure that the similarity value for which e is responsible lies within $[l, u]$, the probability functions we are employing here allow a more systematic biasing of the search process.

Definition 4.13 (Filter Element Search Strategies)
Let e be a knowledge filter element for which an expert value $\mu_x \in [0;1]$ as well as a confidence level $c \in C$ are given. Furthermore, c determines a probable lower bound l_c and a probable upper bound u_c (according to some strategy) so that it holds $l_c \leq \mu_x \leq u_c$. The set Q of search strategy filter qualifiers is defined as $Q = \{unif, lin, gauss\}$.

A filter element search strategy is a probability function $f = f_q$ (with $q \in Q$), i.e. $f : \mathbb{R} \to [0;1]$ and $\int_s f_q(s)ds = 1$, that is defined as follows:

- ***uniform filter element search strategy*** *(q=unif):*

$$f_{unif} : \mathbb{R} \to [0;1] \text{ with } x \mapsto \begin{cases} \frac{1}{u_c - l_c} & \text{if } l_c \leq x \leq u_c \\ 0 & \text{else} \end{cases}$$

- ***linear filter element search strategy*** *(q=lin):*

$$f_{lin} : \mathbb{R} \to [0;1] \text{ with } x \mapsto \begin{cases} \frac{2(x - l_c)}{(u_c - l_c)(\mu_x - l_c)} & \text{if } l_c \leq x \leq \mu_x \\ \frac{2(u_c - x)}{(u_c - l_c)(u_c - \mu_x)} & \text{if } \mu_x \leq x \leq u_c \\ 0 & \text{else} \end{cases}$$

- ***Gaussian filter element search strategy*** *(q=gauss):*

$$f_{gauss} : \mathbb{R} \to [0;1] \text{ with } x \mapsto G(\mu_x, \sigma_c, x) + corr$$

Here, it holds $G(\mu_x, \sigma_c, x) = \frac{1}{\sigma_c\sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_c^2}}$*, where we assume that* $\sigma_c = u_c - \mu_x = \mu_x - l_c$*. The correction addend is defined as* $corr = \int_{-\infty}^{0} G(\mu_x, \sigma_c, s)ds + \int_{1}^{\infty} G(\mu_x, \sigma_c, s)ds$*.*

The uniform filter element search strategy employs the information given by a lower and an upper bound only: Each value from within $[l_c, u_c]$ may be chosen with equal probability. Hence, the filter simply ensures that no bound violations occur, so that this strategy is essentially the same as the bounds approach (expert-specified bound) described before.

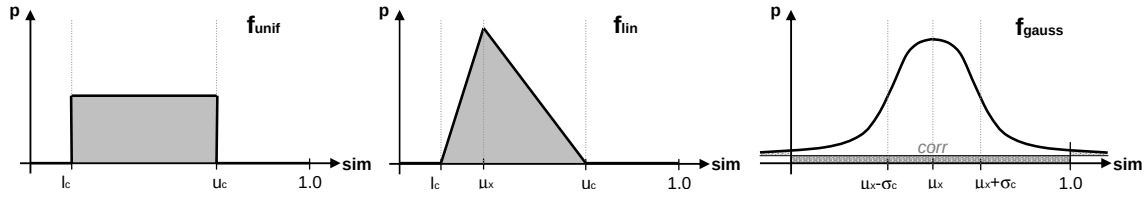


Figure 4.8: Filter Element Search Strategies (Examples)

The linear search strategy, on the other hand, does not only guarantee that the respective similarity value lies within $[l_c, u_c]$, but also focuses the search around the expert value μ_x. It does so by choosing values around μ_x with a higher probability than values nearer to l_c or u_c (see also Figure 4.8). This is also true for the Gaussian filter element search strategy (which presumes $u_c - \mu_x = \mu_x - l_c$). Here, μ_x is interpreted as a Gaussian distribution's expected value and $u_c - \mu_x = \mu_x - l_c$ as its standard deviation. Accordingly, the interval $[l_c, u_c]$ is interpreted in a more relaxed way, i.e. also values $x \in [0;1]$ with $x \notin [l_c, u_c]$ have a (though relatively low) chance of being taken into consideration. Note, that the correction addend *corr* in the definition of f_{gauss} is needed to ensure that $\int_0^1 f_{gauss}(x)dx = 1$.

The decision which search strategy to use for a specific knowledge filter element e may be made by the knowledge engineer according to his/her demands and self-estimation of his/her own knowledge. If no further information is specified, we suggest to employ the linear filter element search strategy by default.

4.4.3 Taxonomic Symbolic Attributes

The elements of a symbolic data type for a specific attribute may be structurable in a taxonomy. Of course, that taxonomic ordering effects the belonging local similarity measure and the way the similarity between a query and a case is computed.

Definition 4.14 (Taxonomic Symbolic Attribute)
Let A be a symbolic attribute with value range $D_A = \{d_1, \ldots, d_n\}$. A is called a ***taxonomic attribute****, if there exists an accentuated root element $r \in D_A$ for the taxonomy. Furthermore, the remaining elements $d \in D_A \setminus r$ must be arranged in a tree structure $\mathcal{T}_A$ with r as root of the tree.*

The predicate $isChild(d_i, d_j)$ is true, if d_i is a (possibly indirect) successor of d_j within $\mathcal{T}_A$.

Bergmann (1998) describes very detailed how taxonomies can be used for representing case features and also addresses the resulting implications for appropriate local similarity measures. Particularly, the author focuses on the possible semantics for leaves and inner nodes as well as on the respective similarity assessments. Here, we employ a simplified approach to calculate the similarity for taxonomic attributes.

Definition 4.15 (Taxonomic Similarity Measure)
Let A be a taxonomic attribute with its taxonomic value range $D_A = \{d_1, \ldots, d_n\}$ arranged in $\mathcal{T}_A$. Moreover, let each node d of $\mathcal{T}_A$ be annotated with similarity value $s_d \in [0;1]$ so that it holds:

$$s_i \leq s_j \quad \textit{iff} \quad dist(i,r) \leq dist(j,r) \tag{4.6}$$

where dist measures the distance (number of edges) between two tree nodes and r denotes $\mathcal{T}_A$'s root element.

*Then, the **taxonomic local similarity measure** for A is defined as*

$$\begin{aligned} sim_A : D_A \times D_A &\rightarrow [0;1] \\ (q,c) &\mapsto s_{NCP(q,c)} \end{aligned}$$

where $NCP(q,c)$ determines the nearest common parent node for q and c.

Sometimes, it can be advantageous to allow the leaves of $\mathcal{T}_A$ to be elements of D_A only, i.e. it holds $D_A \subsetneq nodes(\mathcal{T}_A)$. Then, the notation introduced above is still appropriate, if we consider D_A to be extended by $\mathcal{T}_A$'s inner nodes.

Representing Taxonomic Individuals

When intending to learn similarity measures for taxonomic attributes, we need to determine an appropriate representation of such measures as individuals. We propose to employ trees as individuals that correspond to the taxonomically ordered value range of the respective attribute.

Definition 4.16 (Similarity Tree Individual)
*An individual I representing a taxonomic local similarity measure sim_A for the taxonomic symbolic attribute A is coded as a tree T_A^I whose structure is identical to $\mathcal{T}_A$. The nodes of that **similarity tree** T_A^I are real numbers whose values are determined by the node annotations s_i ($i \in \{1, \ldots, n\}$) that are made to the nodes in $\mathcal{T}_A$.*

When realising similarity tree individuals according to Definition 4.16, essentially the condition given by Equation 4.6 must be fulfilled. For that purpose it is advisable to employ the knowledge filter elements' relations introduced in Section 4.2.4. We have utilised their functionality already to accomplish the monotony constrained for similarity functions (cf. Section 4.3.1.4). Here, we may use those relations accordingly in order to ensure that the similarity values in a similarity tree are increasing from the root towards the leaves of the tree. Algorithm 4.3 shows how those relations can be set.

Input : knowledge filter $K = (G_K, E_K)$ with $E_K = \{e_1, \ldots, e_\phi\}$, for a taxonomic attribute A with value range $D_A = \{d_1, \ldots, d_\phi\}$ that is ordered taxonomically in a tree T_A
Output : K with extended relation sets $e_i.R$

1. **for** $i = 1$ **to** ϕ **do**

   for $j = 1$ **to** ϕ **do**

   if $isChild(i, j)$ **then** $e_j.R := e_j.R \cup \{\geq, i\}$

2. **return** K

Algorithm 4.3: Relations to Realise a Similarity Tree Individual

Genetic Operators

The application of mutation operators (as introduced in Section 2.4.4) is straightforward, when using the similarity values of a tree node's predecessor node (i.e., s_p) and the maximum of its m child nodes ($max\{s_{c_1}, \cdots, s_{c_m}\}$) as lower and upper bound, respectively, for the determination of the adapted (mutated) value. Arithmetic crossover's application is trouble-free as well, since an averaged similarity tree individual created from two parent individuals, that fulfill all relations, is consistent with those relations, too.

Crossover variants that make use of splitting the parental genome and composing the offspring by genome parts from its parents (e.g. simple crossover) are also applicable to similarity tree individuals. On the one hand, the similarity tree may be mapped to a flat vector of similarity values, so that the operators introduced for similarity function and similarity table individuals can be used without change. However, in addition to the crossover operators defined in Section 2.4.4 we also suggest another specialised genetic operator:

- *Sub-tree crossover:* Given a taxonomic attribute A whose value range is arranged in a tree $\mathcal{T}_A$ and two parent similarity tree individuals $T_A^{I_1}$ and $T_A^{I_2}$, sub-tree crossover randomly picks a node d from $\mathcal{T}_A$. Then, it creates the offspring $T_A^{I_{new}}$ according to
$$t_i^{I_{new}} = \begin{cases} t_i^{I_1} & \text{if } d \rhd i \\ t_i^{I_2} & \text{else} \end{cases} \quad \text{for all } i \in \mathcal{T}_A$$
where the operator $\rhd$ denotes that i is a node within the sub-tree of $\mathcal{T}_A$ having root d.

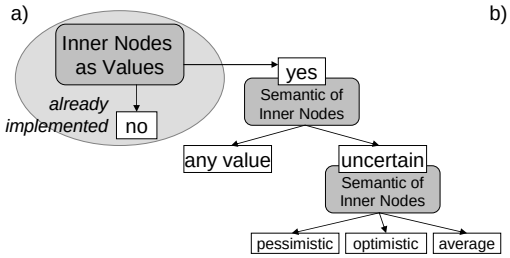
Concerning the difficulties in handling constraints/relations included in the knowledge filter for the respective attribute A, when applying any kind of split point-based crossover, we refer to Section 5.3.1 in the implementation-specific chapter.

Search Space Restriction

Analysing the restriction of the search space that can be achieved, when utilising information about an attribute's taxonomic value range, we ought to consider the worst case scenario in which the minimal restriction is reached.

First of all, we need to stress that our current realisation does not yet allow to use inner tree nodes as values for queries or cases, i.e. the values from D_A are to be found in the leaves exclusively. This entails the following implications:

- Differentiated semantics for inner nodes, as depicted by Bergmann (1998) and illustrated in Figure 4.9a, cannot be employed. As asymmetries in the respective similarity measure result from different interpretations of inner nodes when used as query or case, only symmetric taxonomic similarity measures can be learnt up to now.
- Since we allow values from D_A ($|D_A| = \phi$) in the leaves of a similarity tree individual only, additional nodes (including a root element) are necessary to form the tree. Thus, in the worst case (binary tree) $\phi - 1$ additional nodes are needed, i.e. a similarity tree individual maximally consists of a total of $2\phi - 1$ similarity values.



b)

Φ	Free Parameters K_{empty}	Free Parameters K_{taxo}
1	1	1
2	3	3
3	6	5
4	10	7
5	15	9
6	21	11
7	28	13
…	…	…

Figure 4.9: Usage of Inner Nodes (a) and Feasible Regions for Similarity Matrices vs. Similarity Tree Individuals (b)

Here, we ought to mention that for a similarity tree individual I for attribute A with $D_A = \{d_1, \ldots, d_\phi\}$ exactly $2\phi - 1$ independent similarity values have to be optimised. If we had treated A as a (non-taxonomic) symbolic attribute and used similarity matrices instead (assuming symmetric measures only), there would have been $\frac{\phi^2+\phi}{2}$ free parameters to be tuned. Hence, the utilisation of vocabulary, here taxonomic, knowledge in fact results in a search space restriction as it holds $2\phi - 1 \leq \frac{\phi^2+\phi}{2}$ for all $\phi \in \mathbb{N}$ (see also Figure 4.9b). Note, that the actual gain is in fact higher than maybe presumed from the numbers in that figure: Due to the taxonomic structuring of D_A and due to the relations associated with all nodes (cf. Algorithm 4.3) within the tree, most of the $2\phi - 1$ similarity values cannot be chosen from $[0;1]$ but from a smaller subinterval of $[0;1]$ only.

The mentioned gain may be even increased, when the current restrictions listed above, particularly the consideration of different semantics for inner nodes and thus the allowance of asymmetric measures, are overcome.

4.4.4 Ordered Symbolic Attributes

A simpler ordering of a symbolic attribute's allowed values $d_1, \ldots, d_n$ (simpler in comparison to a taxonomic ordering) is given, when the elements d_i are ordered totally.

Definition 4.17 (Ordered Symbolic Attribute)
*Let A be a symbolic attribute with value range $D_A = \{d_1, \ldots, d_n\}$. A is called an **ordered symbolic attribute**, if each d_i is associated with a numeric scaling value $o_i \in \mathbb{R}$, so that it holds $o_i < o_j$ for all $i < j$.*

The mentioned associations o_i may then be employed as representative for the actual elements d_i in order to realise an ordered symbolic similarity measure on the basis of a similarity function (cf. Definition 2.3).

Definition 4.18 (Ordered Symbolic Similarity Measure)
Let A be an ordered symbolic attribute with its ordered value range $D_A = \{d_1, \ldots, d_n\}$ and belonging numeric associations $o_1, \ldots, o_n$. Then, the ***ordered symbolic local similarity measure*** *is defined as*

$$\begin{aligned} sim_A : D_A \times D_A &\rightarrow [0;1] \\ (q, c) &\mapsto sim_{A_0(q,c)} \end{aligned}$$

where sim_{A_0} is a similarity function that is defined on $I_d = [o_1 - o_n, o_n - o_1]$.

Representing Ordered Symbolic Individuals

An ordered symbolic similarity measure sim_A shall be represented by an individual that consists of two real-valued vectors. The first one is to define the association o_i mentioned above and in so doing affects the scaling of A's value range. Thus, it basically determines the x-axis on which the second vector with its similarity values operates: Figure 4.10 illustrates how we represent individuals for local similarity measures with an ordered value range, giving an example for an ordered symbolic attribute RainbowColour.

Definition 4.19 (Ordered Symbolic Individuals)
Let A be an ordered symbolic attribute with value range $D_A = \{d_1, \ldots, d_n\}$ and belonging associations $o_1, \ldots, o_n$. An ***ordered symbolic individual*** *representing an ordered symbolic similarity measure sim_A is coded as a tuple of two vectors: $I = (O_A^I, V_A^I)$. Here, $O_A^I = (o_1^I, \ldots, o_n^I)$, called* ***scaling vector****, represents a normalisation of the associations $o_1, \ldots, o_n$ according to*

$$o_i^I = \frac{o_i - o_1}{o_n - o_1} \qquad \text{for all } i \in \{1, \ldots, n\}$$

so that it holds $o_i^I \leq o_j^I$ for all $i < j$ as well as $o_1^I = 0$ and $o_n^I = 1$.

$V_A^I = (v_1^I, \ldots, v_s^I)$, called ***similarity vector****, corresponds to a similarity function individual (cf. Definition 2.12), that is defined over $[o_1^I - o_n^I, o_n^I - o_1^I] = [-1; 1]$, where s denotes the number of sampling points used.*

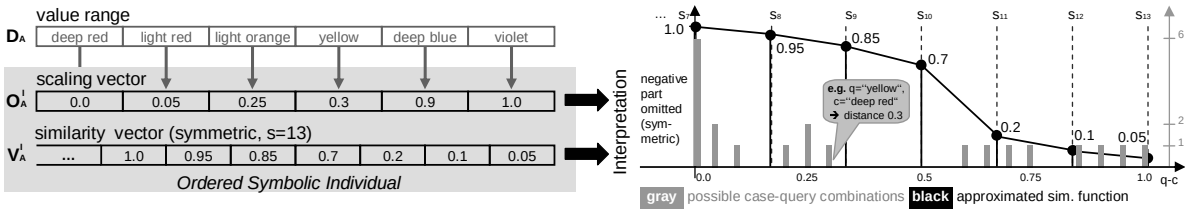


Figure 4.10: Representation of Ordered Symbolic Similarity Measures as Individuals, Example for an Ordered Symbolic Attribute RainbowColour

During optimisation, when ordered symbolic individuals are used in the scope of an evolution strategy applied by FLSM, both vectors are learnt simultaneously. For the constraints related

to the scaling vector O_A^I, the functionality provided by knowledge filters can be utilised. Here, we mainly need to meet relational constraints ($o_i^I < o_j^I$ for all $i < j$) as well as predefined values ($o_1^I = 0$ and $o_n^I = 1$). For the former ones we employ the knowledge filter elements' relations – very similar to our realisation of the monotony constraint (see Section 4.3.1.4). For the latter ones we may use knowledge filters' expert values together with the confidence level "certain" (see Section 4.4.2).

Genetic Operators

Regarding genetic operators the situation for ordered symbolic individuals (in particular for its first component, the scaling vector) is not much different from the situation for taxonomic ones (see Section 4.4.3). In addition to the previous operators, we introduce a further specialised, heuristic crossover operator that is responsive to the constraints that must be fulfilled by the respective individual's scaling vector:

- *Information exchange crossover:* Given two parent scaling vectors $O_A^{I_1}, O_A^{I_2} \in [0;1]^n$, this operator mingles both vectors' elements and sets
  $$z = (z_1, \ldots, z_{2n}) \text{ with } z_i \leq z_j \text{ for all } i < j \text{ and } \forall i \exists j \text{ so that } z_i = o_j^{I_1} \text{ or } z_i = o_j^{I_2}$$
  The offspring is formed according to
  $$O_A^{I_{new}} = (z_1, z_3, \ldots, z_{2n-1}) \text{ or } O_A^{I_{new}} = (z_2, z_4, \ldots, z_{2n}).$$

Note, that this new operator might also be applied to create offspring of similarity functions that have to fulfill the monotony constraint.

Search Space Restriction

Let us again assume an ordered symbolic attribute A with an ordered value range $D_A = \{d_1, \ldots, d_\phi\}$ (note, that $\phi = n$). Treating that attribute as a "normal" symbolic one and modelling its similarity measure with a similarity table – using, so to say, an empty knowledge filter K_{sym} –, there would be ϕ^2 independent entries to be adjusted. Making use of the total ordering of D_A's value range, however, the search space becomes restricted: Doing so, the modelling of ordered symbolic local similarity measures requires the scaling vector, which consists of ϕ elements, and the similarity vector, which represents a similarity function and thus consists of s sampled similarity values. Consequently, (using a knowledge filter K_{ord} that contains the above-mentioned constraints concerning the scaling vector) learning on the basis of ordered symbolic similarity measures requires the optimisation of $\phi + s$ similarity values only.

Having a look at the example from above ($\phi = n = 6$) a search space restriction occurs, when choosing $s < 30$. Since a reasonable choice for s is, for instance, $s = 13$ (as depicted in Figure 4.9), the achieved restriction is in this case almost 50%. Note, that the actual search space restriction is, however, even higher. The total ordering of the scaling vector's elements causes that those values cannot be chosen arbitrarily from $[0;1]$, but from a subinterval of $[0;1]$ instead. Moreover, for $o_1^I = 0$ and $o_n^I = 1$ the values are even fixed.

5 Implementation and Optimisation

In this chapter we want to present some implementation-specific details which illustrate how the concept of knowledge-based optimisation filters is realised and integrated into the framework for learning similarity measures. In particular, we focus on how the filters give their knowledge-based advice to the respective genetic operators employed by FLSM's evolution strategy. Moreover, it is our goal to improve FLSM's run-time behaviour, i.e. to speed up the optimisation process. In order to do so, we analyse the current implementation, figure out its shortcomings and develop strategies to overcome the deficiencies.

We start with a short description of the architecture of our extensions to FLSM. The following two sections focus in detail on both groups of genetic operators we are applying, mutation and crossover operators. There, we discuss how the gathered knowledge, that is stored in knowledge filters, is employed to actively exert a bias on the learning process, i.e. how filters can convey some form of advice to genetic operators.

The subsequent section deals with a smart optimisation of the fitness calculation, employing some knowledge about the mechanisms according to which retrievals and similarity assessments are carried out. We need to stress that all the experimental evaluations conducted in the scope of this work (cf. Chapter 6) would not have been feasible without that optimisation (due to the enormous computational effort involved). In the last section we highlight a further enhancement of FLSM covering a guiding of the optimisation process into regions worth searching and, hence, an intelligent dispatching of computation time.

5.1 Filter Realisation

The realisation of FLSM represents an extension to the commercial CBR tool CBR-Works (Schulz, 1999). As the task of the thesis at hand is to enhance the capabilities of FLSM, our implementations provide an extension to CBR-Works as well.

The mixed UML and block symbol chart in Figure 5.1 shows some of the classes whose functionality is part of this work. The functionality offered by knowledge filters resides mainly in two classes: `LSMKnowledgeFilter` and `LSMKnowledgeFilterElement`.

The learning controller includes the control algorithm (cf. Sections 2.4.6 and 5.5), which supervises the learning process. Hence, it is responsible for the creation and maintenance of appropriate populations of `LSMIndividuals` (for feature weights and local measures). Moreover, it may – with the help of user or expert knowledge supplied – create knowledge-based optimisation filters which are associated with the respective populations.

The next two sections focus on the question how the knowledge filters interfere with the evolutionary process, i.e. how they exert the bias on the learning process.

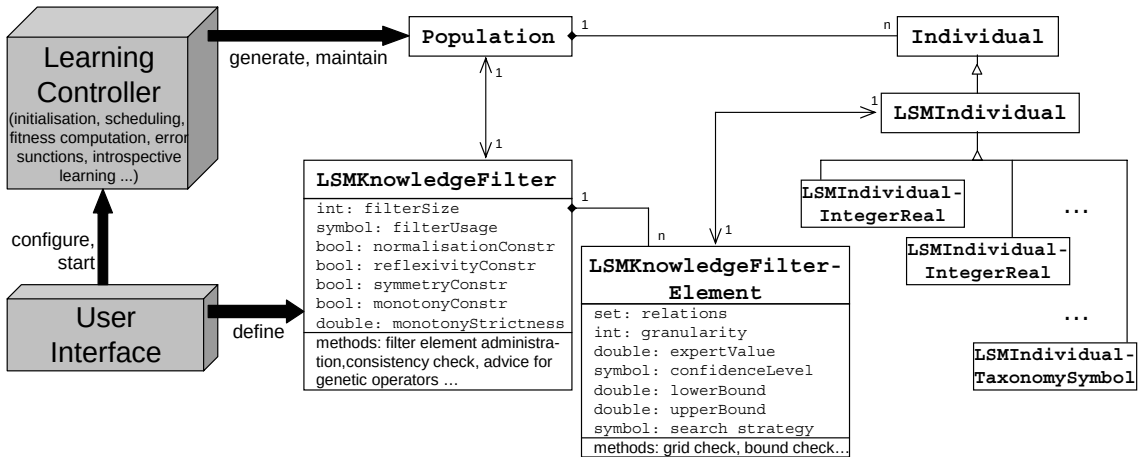


Figure 5.1: Excerpt of the Overall Class Structure Employed by FLSM and Enhanced in the Scope of this Thesis

5.2 Advising Mutation Operators

In the following, we describe how our mutation operators handle the integration of three main forms of knowledge, that may be contained in a knowledge filter, into the offspring creation process: relational knowledge (via sets R of relations between several knowledge filter elements), expert knowledge (in the form of lower and upper bound specifications l and u as well as expert values μ_x with confidence levels c) and knowledge about a used grid (via granularity values g).

5.2.1 Relational Constraints

According to our depiction in Chapter 4 relational knowledge may become involved,

- when using the monotony constraint for a certain local similarity measure,
- when the respective attribute is taxonomic or ordered symbolic and the individuals are modelled as similarity trees or ordered symbolic individuals, respectively (cf. Sections 4.4.3 and 4.4.4)
- or when the expert defines weight preferences (cf. Section 4.4.1).

Moreover, an expert may specify relations to be fulfilled between entries in similarity tables or between several of a similarity vector's sampling points and in doing so employ sets of relations to be included in the particular knowledge filter.

The most straightforward strategy to guarantee the compliance with all relations would be to simply reset the concerned similarity value, if its mutation has made it violate a relation. Let, for example, $v_1^I = 0.1$, $v_2^I = 0.3$ and $v_3^I = 0.4$ be the values of some sampling points, contained in a similarity vector. The corresponding knowledge filter's relation sets be $e_1.R = \{(2, \leq)\}$, $e_2.R = \{(1, \geq), (3, \leq)\}$ and $e_3.R = \{(2, \geq), (4, \leq)\}$ and the application of a mutation operator changed v_2^I from 0.3 to 0.5. In that case, a relation induced by $e_2.R$ is not met – and K might simply reset v_2^I to 0.4 so that it holds $v_1^I \leq v_2^I \leq v_3^I$.

Unfortunately, that kind of "error correction" would not be utilisable for crossover operators in such a simple way. For that reason we made the design decision – aiming at the development of a uniform concept to integrate the filter knowledge into the processing of genetic operators – that the knowledge filter should be allowed to give some kind of "advice" to genetic operators (both, mutation and crossover operators), enabling them to construct offspring that fulfills all constraints.

On account of these arguments, a knowledge filter analyses all the relations it includes, that way determines the minimal ($v^I_{i,min}$) and maximal ($v^I_{i,max}$) permitted value for v^I_i (for all i) and hands these limits to the respective mutation operator. Then, the mutation operator interprets $v^I_{i,min}$ and $v^I_{i,max}$ as the lower and upper bound (l and u), as specifiable by an expert, and calculates v^I_i's new value from interval $[v^I_{i,min}, v^I_{i,max}]$ as in the case of handling expert knowledge (see below) employing the uniform filter element search strategy ($f = f_{unif}$).

5.2.2 Bound Specifications

For reasons of simplicity, we assume that the expert's constraints on similarity values are always symmetric, i.e. if the expert specifies a lower bound l, an upper bound u *and* an expert value μ_x, then it holds $u - \mu_x = \mu_x - l$. In practical usage, however, it is likely that an expert is about to express his/her ideas about the respective measure via confidence levels c (cf. Definition 4.12) in combination with the expert value μ_x for each filter element. Note, that we map those confidence levels to lower/upper bounds (see Table 5.1).

Confidence Level $c \in C$	Mapped to	
	Lower Bound l	Upper Bound u
uncertain	$\mu_x - 0.5$	$\mu_x + 0.5$
low	$\mu_x - 0.3$	$\mu_x + 0.3$
average	$\mu_x - 0.15$	$\mu_x + 0.15$
high	$\mu_x - 0.05$	$\mu_x + 0.05$
certain	$\mu_x - 0.5$	$\mu_x + 0.5$

Table 5.1: Numerical Realisation of Confidence Levels

Depending on the expert-specified search strategy f_q ($q \in \{unif, lin, gauss\}$, cf. Definition 4.13) the behaviour of each of our mutation operators – simple, multivariate non-uniform and in-/decreasing mutation – is biased by l, u and μ_x. Algorithm 5.1 gives an overview how their behaviour changes in comparison to their standard behaviour which was introduced in Section 2.4.4.

The Gaussian filter element search strategy is in need of a Gaussian random number generator in order to implement f_{gauss} (cf. Definition 4.13). Typical programming environments, however, provide uniform (pseudo) random number generators only.

In Algorithm 5.1 the desired Gaussian random number generator is abbreviated by $\mathcal{G}$ taking the expected value as first and the standard deviation as second parameter. For the concrete realisation one might apply the *Box-Muller Method* (Box and Muller, 1958) which calculates two normally distributed random numbers n_0 and n_1 from two uniform ones ($u_0, u_1 \in [0; 1]$) according to

$$n_i := \sqrt{-2 \cdot log(u_i)} \cdot cos(2\pi \cdot u_{1-i}) \tag{5.1}$$

Input : filter element e (including μ_x, l and u with $c = u - \mu_x = \mu_x - l$, current age of the respective population $t = age(P_i)$, current value v of the similarity value to be mutated, filter element search strategy f, increment ι to in-/decreasing mutation, Gaussian random number generator G

Output : $v' \in [0;1]$ as mutated value of v

1. **set** allowed value range for v' due to search strategy

$$min := \begin{cases} l & \text{if } f = f_{unif} \text{ or } f = f_{lin} \\ 0 & \text{else} \end{cases}, \quad max := \begin{cases} u & \text{if } f = f_{unif} \text{ or } f = f_{lin} \\ 1 & \text{else} \end{cases}$$

2. **do**
   a) **set** $r := rnd(0,1)$ `// random number from [0;1]`
   b) calculate v' according to

Mutation Operator	**Filter Element Search Strategy**		
	uniform	linear	gaussian
simple	$v' := rnd(l,u)$	$v' := \begin{cases} c\sqrt{2r}+l & \text{if } r > 0.5 \\ -c\sqrt{2-2r}+u & \text{else} \end{cases}$	$v' := \mathcal{G}(\mu_x, c)$
non-uniform	$\delta := c \cdot (1 - r^{(1-\frac{t}{T})^2})$		
	$v' := v \pm \delta$	$p_{down} := \frac{v-l}{2c}$, $x := rnd(0,1)$ $v' := \begin{cases} v-\delta & \text{if } x \le p_{down} \\ v+\delta & \text{else} \end{cases}$	$d := \begin{cases} 1 & \text{if } \mathcal{G}(\mu_x, c) > v \\ -1 & \text{else} \end{cases}$ $v' := v + d \cdot \delta$
in-/de-creasing	$\delta := \iota \cdot c$		
	$v' := v \pm \delta$	$p_{down} := \frac{v-l}{2c}$, $x := rnd(0,1)$ $v' := \begin{cases} v-2\delta & \text{if } x \le p_{down} \\ v+2\delta & \text{else} \end{cases}$	$d := \begin{cases} 1 & \text{if } \mathcal{G}(\mu_x, c) > v \\ -1 & \text{else} \end{cases}$ $v' := v + d \cdot \iota$

3. **return** v'

Algorithm 5.1: Exploitation of a Filter Element's Knowledge by Mutation Operators

for $i \in \{0,1\}$.

However, since the calculation of sine or cosine is computationally very intensive, we use the *polar method* which is a variant of the Box-Muller method. It is substantially faster than the Box-Muller method since it avoids the calculation of the trigonometric functions and it has perfect accuracy (Marsaglia and Bray, 1964). Our implementation calculates two (μ, σ)-distributed random numbers g_0 and g_1 from two uniform ones according to Algorithm 5.2.

5.2.3 Granularity Values

Each knowledge filter element e for a similarity value v may contain a granularity value g ($e.g$) that determines how many values from $[0;1]$ are allowed to be assigned to v. Since the application of a mutation operator in general determines a mutated value v' which is not

Input : expected value μ,
standard deviation σ
Output : Gaussian random variables g_0 and g_1

1. **do**
   set $u_0 := rnd(-1, 1)$ // $= 2 \cdot rnd(0, 1) + 1$
   set $u_1 := rnd(-1, 1)$
   set $r := u_0^2 + u_1^2$
   while $r \notin (0; 1)$
2. **set** $g_0 := u_0 \cdot \sqrt{-2 \cdot \frac{ln(r)}{r}} \cdot \sigma + \mu$
3. **set** $g_1 := u_1 \cdot \sqrt{-2 \cdot \frac{ln(r)}{r}} \cdot \sigma + \mu$
4. **return** (g_0, g_1)

Algorithm 5.2: Polar Method to Compute Gaussian Random Variables

consistent with the grid induced by g, the respective filter must be allowed to redefine v' according to

$$v' := \frac{round(v' \cdot g)}{g} \tag{5.2}$$

This means that the mutated value v' is aligned with the nearest grid "line". Note, that the usage of granularities may, under certain circumstances, interfere with other demands made on the respective similarity measure due to filter knowledge. By forcing all similarity values to be consistent with the grid, it may, for example, happen that the monotony – which eventually should be preserved, supplied that the filter's monotony constraint is active – may go lost.

Our current implementation intentionally assigns higher priority to the consideration of granularity values, which may accidently result in the creation of individuals that in part are slightly inconsistent with the corresponding filter's knowledge. A more elaborated solution to that problem, i.e. a more sophisticated strategy to solve conflicts of that kind, is left for future work.

5.3 Advising Crossover Operators

Making the same distinction into three forms of knowledge included in knowledge filters, we now intend to describe how our crossover operators exploit that knowledge to create offspring that does not violate any of the filter's constraints.

5.3.1 Relational Constraints

Let us consider the situation for the monotony constraint as a representative for the utilisation of relational background knowledge (for relation sets with other semantics, e.g. for taxonomic similarity trees, the following arguments hold as well).

The arithmetical crossover operator (see Section 2.4.4) can be characterised as being convex with respect to fulfilling the monotony constraint. This means, when creating a new individual from two monotonous parents by means of arithmetical crossover, the descendant is obviously monotonous as well. Information exchange crossover, introduced especially for handling ordered symbolic attributes, is convex as well.

Regrettably, this is not true for the remaining crossover operators (e.g. for simple, arbitrary crossover or line/row crossover). Hence, for the offspring to be consistent with the constraint for monotony, these operators need some advise from the knowledge filter likewise. Our approach to handle that problem requires an extension of the functioning of simple and arbitrary crossover. Provided with two parents $I_A = (i_1^{I_A}, \cdots, i_s^{I_A})$ and $I_B = (i_1^{I_B}, \cdots, i_s^{I_B})$ and a split point $\varsigma \in \{1, \cdots, s\}$, that is chosen randomly, simple crossover composes the child as $I_C = (i_1^{I_A}, \cdots, i_\varsigma^{I_A}, i_{\varsigma+1}^{I_B}, \cdots, i_s^{I_B})$. In doing so, I_C will probably violate the monotony constraint, i.e. one or several relations, that are defined in the knowledge filter element for the split point.

To avoid that we suggest to broaden the individual composition rule according to which simple crossover works (the extension is analogous for arbitrary and line/row crossover):

$$I_C = (i_1^{I_A}, \cdots, i_\varsigma^{I_A}, (1-\alpha) \cdot i_{\varsigma+1}^{I_A} + \alpha \cdot i_{\varsigma+1}^{I_B}, \cdots, (1-\alpha) \cdot i_s^{I_A} + \alpha \cdot i_s^{I_B}) \tag{5.3}$$

Input : individuals $I_A = (i_1^{I_A}, \cdots, i_s^{I_A})$ and $I_B = (i_1^{I_B}, \cdots, i_s^{I_B})$
split point ς, knowledge filter $K = (G_K, E_K)$ with sets of relations $e.R$ in at least one $e \in E$,
binary search accuracy $\varepsilon \in (0; 1]$

Output : parameter $\alpha \in [0; 1]$ as advise to the crossover operator

1. **set** $\alpha_{low} := 0$, $\alpha_{upp} := 2$, $\alpha := \alpha_{upp}$
2. **do**
   a) **set** $\alpha := \frac{\alpha_{low} + \alpha_{upp}}{2}$
   b) **set**
   $I_\alpha := (i_1^{I_A}, \cdots, i_\varsigma^{I_A}, (1-\alpha) \cdot i_{\varsigma+1}^{I_A} + \alpha \cdot i_{\varsigma+1}^{I_B}, \cdots (1-\alpha) \cdot i_s^{I_A} + \alpha \cdot i_s^{I_B})$
   c) **if** I_α is consistent with all relations in $e.R$ for all $e \in E$
   then set $\alpha_{low} := \alpha$
   else set $\alpha_{upp} := \alpha$

   while $\alpha_{low} < 1$ **and** $\alpha_{upp} - \alpha_{low} > \varepsilon$
3. **return** α_{low}

Algorithm 5.3: Filtered Individual Composition via Simple Crossover

Note, that this rule ($\alpha \in [0; 1]$) represents an extension of the original one as it contains the behaviour of simple crossover as a special case for $\alpha = 1$. Furthermore, it is obvious that there is always a largest $\alpha \geq 0$ for which I_C is monotonous. The difficulty for the knowledge filter is to determine the very largest $\alpha \leq 1$ for which no constraint violations occur – of course we are interested in finding the largest one as this will result in maximal information exchange between both parents forming the successor.

For that purpose we apply a simple *binary search* according to Algorithm 5.3. The maximal α calculated by that algorithm depicts the advice the knowledge filter hands to the respective crossover operator.

5.3.2 Bound Specifications

The presence of expert knowledge in form of bound specifications, expert-estimated values and confidence levels does not make any demands to the crossover operators. In other words, all crossover operators we employ are convex with respect to the forms of knowledge mentioned because those operators refer to single similarity values only. Consequently, our operators do not need further extensions to handle that knowledge.

5.3.3 Granularity Values

If a granularity value is given in a knowledge filter element, we can presume that the parent individuals are consistent with the grid induced that way. Accordingly, simple, arbitrary, line/row or subtree crossover – which do not change any similarity values, but concentrate on composing offspring by parts of the parents' genomes – work precisely with granularity values. As far as arithmetic crossover – indeed assigning new similarity values via average calculation – is concerned, we proceed in the same manner as in the case of handling granularity values for mutation operators (see Section 5.2.3).

5.4 Retrieval Meta Knowledge

Retrieval meta knowledge, viz meta knowledge about the mechanisms according to which retrievals and involved similarity assessments and calculations are carried out, may be exploited to optimise the learning process as well.

When having a look at the contents of the retrieval container in the knowledge container model, one can distinguish between two kinds of knowledge included: first, similarity measures themselves and, second, the meta knowledge mentioned before. In the following, we intend to employ the latter to improve the learning of the former kind of retrieval knowledge – we will do so by introducing a smart fitness calculation.

5.4.1 Foundations

The most time-consuming part of the entire optimisation process is represented by the evaluation stage and fitness assessment (see Figure 2.7) of newly created individuals, respectively. As described in Section 2.4.5 that process involves the conduction of a complete CBR retrieval for each query contained in the set of available training examples, which requires plenty of computation time. Since the similarity measures we are dealing with are composed according to the local-global principle (cf. Section 2.2.1) a single computation of the similarity between a particular case and query requires the calculation of all belonging local similarities based on corresponding local similarity measures as well as an appropriate amalgamation of those using feature weights.

It is our goal to significantly speed up the process of assigning fitness values to individuals by improving the way in which the necessary similarities and thus retrieval results, which provide the basis for the fitness assessment, are calculated.

Let us for now consider the optimisation of the local similarity measure for a particular attribute A_j with a corresponding population P_j of individuals representing local similarity measure for A_j. In the breeding stage λ successor individuals $I_{A_j}^{off_1}, \ldots, I_{A_j}^{off_\lambda}$ are generated that all have to be assigned a fitness value.

To assess their fitness, however, we must regard the currently available individuals in the remaining populations P_i as well, which provide us with the mandatory local measures (and maybe feature weights either) for the other attributes A_i ($i \in \{1, \ldots, n\}, i \neq j$). We of course pick the currently fittest individual $I_{A_i}^{fit}$ from each population P_i ($i \neq j$). Those top-fit individuals – translated into a corresponding local similarity measure $sim_{A_i}^{fit}$ – together with one of the offspring $I_{A_j}^{off_k}$ form an entire similarity measure Sim^k based on which a retrieval may be carried out.

During the evaluation of all $I_{A_j}^{off_k}$ ($k = 1, \ldots, \lambda$) the remaining fittest individuals and hence local similarity measure for all other attributes A_i ($i \neq j$) remain unchanged. This fact provides the basis for our intended improvement of the individual evaluation procedure.

Input : individuals $I_{A_j}^{off_1}, \ldots, I_{A_j}^{off_\lambda}$ and fittest individuals $I_{A_i}^{fit}$ from P_i ($i \in \{1, \ldots, n\} \setminus \{j\}$), normalised weights $w_1, \ldots, w_n$, case base CB, training data $TD(Q)$ based on query set Q

Output : fitness $f_1, \ldots, f_\lambda$ values for $I_{A_j}^{off_1}, \ldots, I_{A_j}^{off_\lambda}$

1. **define** $LSimStore$ as $|Q| \times |CB|$-matrix
2. **forall** $q \in Q$ **do**
   forall $c \in CB$ **do**
   $LSimStore_{q,c} := \sum_{i=1, i \neq j}^{n} w_i \cdot sim_{A_i}^{fit}(q.A_i, c.A_i)$
3. **for** $r = 1$ **to** λ **do**
   a) **forall** $q \in Q$ **do**
      i. **forall** $c \in CB$ **do**
         $sim_j := sim_{A_j}^{off_r}(q, c)$
         $Sim_q^r(c) := w_j \cdot sim_j + LSimStore_{q,c}$
      ii. **set** $retrResult^r(q) := (Sim_q^r(c_1), \ldots, Sim_q^r(c_{|CB|}))$
         with $c_s \in CB$ for all $s \in \{1, \cdots, |CB|\}$ and
         $Sim_q^r(c_s) \geq Sim_q^r(c_{s+1})$ for all $s \in \{1, \cdots, |CB| - 1\}$
   b) **set** $f_r := \hat{E}_x(Sim^r, TD(Q))$
      on the basis of $retrResult^r(q)$ for all $q \in Q$
4. **return** $(f_1, \cdots, f_\lambda)$

Algorithm 5.4: Smart Fitness Calculation

5.4.2 Brute Force vs. Smart Fitness Calculation

By default, the computation of the retrieval error $\hat{E}_x(Sim^k, TD(Q))$, which equals the fitness, would compose Sim^k from $sim^{fit}_{A_i}$ (for all $i \neq j$) and a specific $sim^{off_k}_{A_j}$ (with $k \in \{1, \ldots, \lambda\}$) and start a retrieval for each query $q \in Q$ in the training data[1]. That would involve the calculation of all local similarities $sim^{fit}_{A_i}(q.A_i, c.A_i)$ for all cases $c \in CB$ and all queries $q \in Q$.

When proceeding to the evaluation of offspring $I^{off_{k+1}}_{A_j}$, this *brute force fitness calculation* would again calculate all $sim^{fit}_{A_i}(q.A_i, c.A_i)$ for all c and q – which in fact represents an enormous overhead and waste of computational resources. Instead, we suggest to store as many intermediary computation results, i.e. local similarities, as possible and to reuse them during the subsequent fitness evaluations for $I^{off_l}_{A_j}$ (with $l > k$).

Obviously, the *smart fitness calculation* (Algorithm 5.4) first stores all local similarities that may be reused to evaluate further individuals of a specific population in the matrix $LSimStore$. As a consequence, the actual CBR retrievals (Step 3.a)) done for each $I^{off_r}_{A_j}$ ($r = 1, \ldots, \lambda$) involve only the calculation of one single local similarity ($sim^{off_r}_{A_j}$), a table lookup to $LSimStore$, one addition and one multiplication operation and finally the ordering of global similarities $Sim^r_q(c)$.

The construction of $LSimStore$ is accomplished beforehand (Step 2) and has to be done only once prior to the actual evaluation of the λ offspring individuals.

5.4.3 Achieved Improvement

Before representing an overview of the savings of computational time that can be obtained, when using the smart fitness calculation, we want to emphasise that its benefit can be even increased: In Section 2.4.6 we point that we optimise all attributes of the respective case representation in a pseudo-parallel way using a round-robin processing. This means, that each population P_i is optimised for ϱ (round robin size) generations ($t \rightarrow t+1$, $t+1 \rightarrow t+2, \cdots$, $t+\varrho-1 \rightarrow t+\varrho$) before proceeding with P_{i+1}. When taking that optimisation technique into consideration, it becomes clear that it is sufficient to construct $LSimStore$ only once – at the beginning of generation t – and to leave it unchanged and use it for the subsequent ϱ evolutionary cycles.

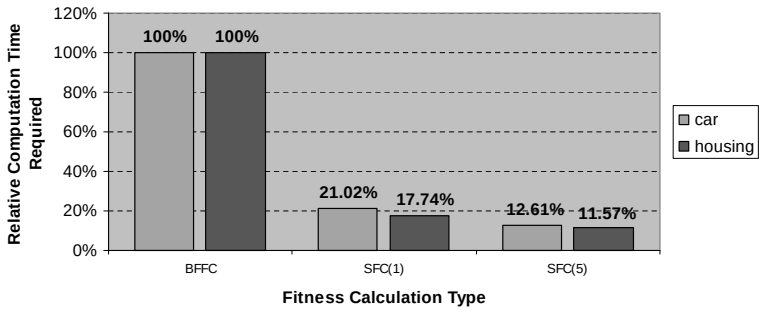


Figure 5.2: Performance of Smart Fitness Calculation Compared to Brute Force Fitness Calculation

[1]Note, that $sim^x_{A_i}$ denotes the local similarity measure the individual $I^x_{A_i}$ stands for.

Figure 5.2 summarises the achieved improvements regarding saved computation time exemplarily for two application domains. Since the accurate computation times depend on the used hardware, we decided to show the relative improvement in comparison to the brutal force fitness calculation ($BFFC \mathrel{\widehat{=}} 100\%$). The smart fitness calculation as described above is denoted as $SFC(1)$, this case corresponds to using a round robin size $\varrho = 1$. Furthermore, we exemplarily show the further improvement that can be yielded when the smart fitness calculation is employed during a certain number of evolutionary generations in the scope of a round robin optimisation ($SFC(5)$ for $\varrho = 5$ is diagrammed).

From Algorithm 5.4 it is clear that the benefits of SFC depend linearly on the number λ of offspring generated within one generation (we used $\lambda = 10$), on the round robin size ϱ as well as on the number of attributes (car: $n = 6$, housing: $n = 12$) that are used in the respective application domain.

5.5 Intelligent Control of the Learning Process

In our previous work (Stahl and Gabel, 2003) we employed FLSM to improve a similarity measure to be used for a product recommendation and configuration system for personal computers. A closer look at the progress of the optimisation process as conducted by FLSM in that domain reveals that a big part of the learning procedure proceeds without significant improvements (with respect to the retrieval error function).

5.5.1 Initial Situation

There are several reasons why computational resources may be wasted during learning:

- The local similarity measures for each attribute are treated equally. This means, each population P_i representing candidate measures for attribute A_i is optimised for the same number of evolutionary cycles, regardless of the respective feature weight. Thereby, the fact is completely disregarded, that local measures for attributes with high weights contribute more to the overall quality of the similarity measure. Hence, due to their higher influence on the similarity assessment, they might hold more potential to yield learning improvements.

- The optimisation of a local similarity measure for one particular attribute A_1 may proceed faster than for another attribute A_2, approaching the respective optimum (in terms of individual fitness and retrieval error, respectively) at an earlier point of time, i.e. after a smaller number of evolutionary generations $t_{opt_1} < t_{opt_2}$. This behaviour depends on how "learnable" a specific attribute's similarity measure is and it is also influenced by the number of similarity values by which the respective local measure is prescribed: Without any doubt, it should be possible to find a 3×3 similarity table's (genome of 9 variable similarity values) optimum faster than to reach the learning goal for a 10×10 table (100 values to be adjusted).

These arguments make clear that a uniform dispatching of computation time, as done by the round-robin processing (see Section 2.4.6), seems disadvantageous. In particular, it does not make sense to further optimise the local similarity measure for a certain attribute after t_{opt} has been transcended.

The description of a personal computer in the scope of the PCCustomisationDomain comprises 11 attributes, for which the local similarity measures were optimised for 400 evolutionary generations via round-robin processing with round-robin size $\varrho = 5$, while the feature weights were learnt prior (Stahl and Gabel, 2003).

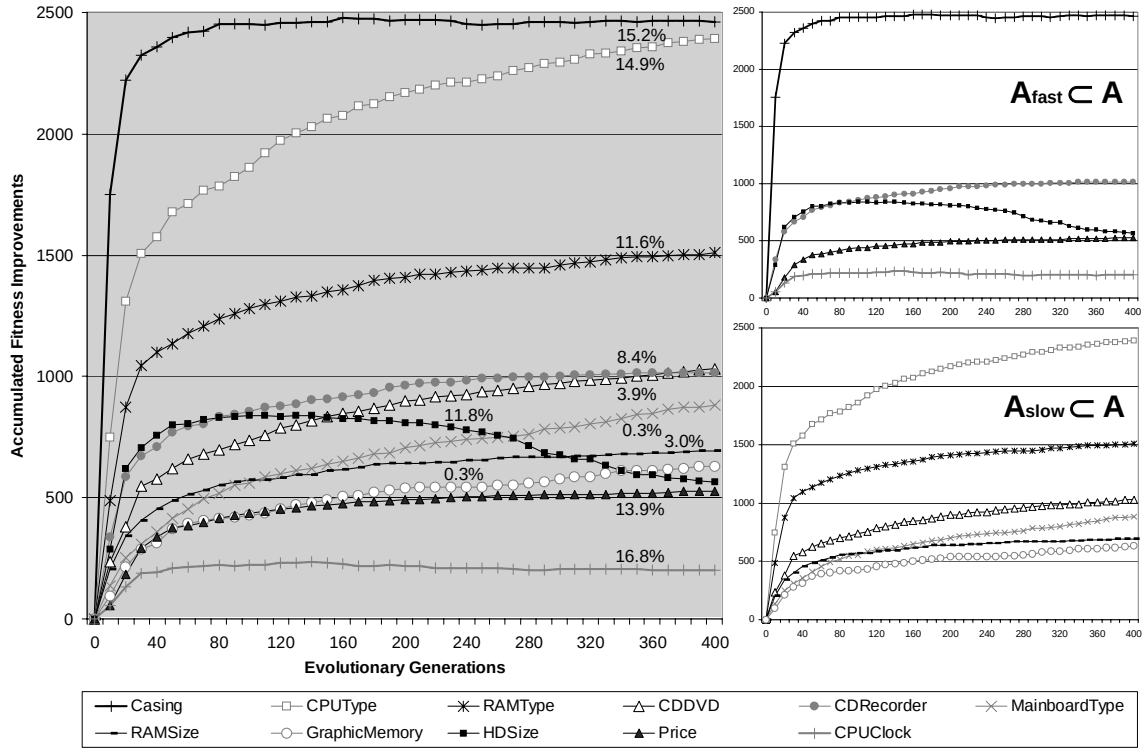


Figure 5.3: Attribute-Specific Analysis of the Learning Progress Obtained in the PCCustomisationDomain: Average Relative Improvements (left) and Attribute Splitting Concerning Fast and Slowly Learnable Local Measures (right)

The total retrieval error (which equals fitness) improvement for the given training data ($\hat{E}(Sim_{t=0}, TD) - \hat{E}(Sim_{t=400}, TD) = 11956$) can be decomposed into partial improvement contributions provided by the eleven attributes whose local measures were learnt pseudo-simultaneously. Figure 5.3 (left part) illustrates that decomposition averaged for 10 repetitions of a learning process with 200 training examples. The attributes Casing and RAMType, for instance, produce comparatively high accumulated fitness improvements during learning, whereas CPUClock or GraphicMemory contribute the lowest ones. Note, that this behaviour can only in part be explained by the different feature weights that are assigned to these attributes. In Figure 5.3 the respective relative feature weights (in percent) are denoted in the right part of each chart. It becomes obvious that, in general, highly-weighted attributes yield higher learning improvements (e.g. Casing with more than a fifth of the overall improvements), but that there are also some exceptions (e.g. the Price attribute with a comparatively high weight and rather little contributions to the optimisation process).

The underlying reason for the attributes' differences in absolute learning improvement contributions and in the shape of the respective learning curves, however, goes back to the specifics of the PCCustomisationDomain. The domain modelling includes a number of adaptation rules

to customise retrieved personal computers with respect to the user's demands. As described in Section 2.3.3, the learning goal can be characterised as finding a similarity measure that implicitly, i.e. already during the retrieval phase, regards the possibilities of case adaptation. So, the creation of training examples, i.e. of case order feedback, involves the adaptation of all cases in the case base with respect to the respective query and a corresponding utility assessment to determine that case order. Consequently, those attributes to which the adaptation rules refer rather often, will presumably be able to contribute more to the learning's progress.

We need to emphasise two properties of the observations we made: On the one hand, the mentioned characteristics are not specific to a single run of FLSM or to the training data set used, but rather domain-specific. As specified before, we determined the averaged accumulated retrieval error improvements for 10 repetitions of the learning process for newly generated training data each time: The results, as summarised in the left part of Figure 5.3, show that the described behaviour appears not by chance but rather systematically. For reasons of the charts' readability we omit to also plot the corresponding standard deviations.

On the other hand, the set of attributes characterising a personal computer can be divided into two subsets – those whose local similarity measures are learnable comparatively fast, reaching the optimum (w.r.t. the training data) before $t = 400$, and those for which the optimisation takes a longer time, creating further fitness improvements even at later times of the evolutionary process. That heuristic division into $A_{fast} \subset A$ and $A_{slow} \subset A$ is manifested in the right part of Figure 5.3. The upper chart shows the attributes belonging to the former subset (e.g. Casing or HDSize), while the bottom part contains the attributes that can be classified as belonging to the latter one (e.g. RAMType or MainboardType).

Those arguments support our initial claim that FLSM's optimisation process ought to avoid a uniform dispatching of computation time via round-robin processing and should instead have a model of its own to realise an intelligent learning scheduling. In the following, we want to present an approach realising such a scheduler that utilises dynamically acquired knowledge about the previous progress of the optimisation process.

5.5.2 Intelligent Learning Scheduler

We keep on pursuing the idea of optimising each population P_i for a certain number of successive evolutionary cycles and denote that number with ϱ further on. As shown in Section 5.4.3 that proceeding results in an enormous reduction of computation time. Nevertheless, we intend to prescind from a strict round-robin processing – instead, a learning scheduler shall decide which population to evolve at which point of time, fulfilling the following requirements:

- When deciding which population, i.e. measures for which attribute, to evolve next, the previous progress of the contribution to learning improvements from the respective attribute/population should be taken into consideration. Those populations ought to be favoured, for which it is likely that their evolution will soon result in further fitness improvements.
- A periodic, deterministic evolution of feature weights is indispensable because of their superior influence on the similarity assessment.
- As feature weights may change from time to time, the current weights for a specific attribute must also exert influence on the decision with which population to continue.

In order to accomplish those demands, the learner needs a reflective model of the preceded learning process.

Definition 5.1 (Reflective Optimisation History)
*Given n attributes $A_1, \cdots, A_n$ whose local similarity measures sim_{A_i} are to be optimised and let P_i be the corresponding populations. Further, let $\hat{t} = age(P_{weights}) + \sum_{i=1}^{n} age(P_i)$ be called **total generation counter**, where $age(P_i)$ denotes the age of the respective population.*

*Then, the **reflective optimisation history** H is defined as a vector*

$$H = (h_1, \cdots, h_{\hat{t}}) \text{ with } h_i =< \mathcal{P}, \tau, w, \hat{E} > \text{ for all } i \in \{1, \cdots, \hat{t}\}$$

Here, $\mathcal{P} \in \{P_1, \cdots, P_n\}$ denotes the population which was evolved for one cycle at total time index i, w corresponds to the feature weight currently assigned to A_j ($P_j = \mathcal{P}$) at time i, τ stands for the age of $\mathcal{P}$, i.e. $\tau = age(\mathcal{P})$, and $\hat{E}$ is the fitness of the currently fittest individual within $\mathcal{P}$.

On the basis of the inner representation of its learning process, that the learner obtains via the reflective optimisation history, we can define a dynamic scheduling algorithm which computes and assigns *evolvement probabilities* p_i^{ev} to each attribute A_i and population P_i, respectively. That probability is then used to determine for which population the subsequent ϱ generations are calculated next. As a consequence, some populations are preferred and are thus allowed to evolve for a longer time than others – which basically means that the learner exerts a bias towards certain regions of the search space, i.e. lays its focus on local measures for certain attributes. For example, it may happen that, when presuming a total generation counter $\hat{t} = 100$, the individuals in P_1 were allowed to evolve for 50 generations, the measures for attribute A_2 in population P_2 evolved for 30 cycles and that P_3 reached an age of $age(P_3) = 20$ only.

The crucial element of Algorithm 5.5 is the computation of evolvement probabilities p_i^{ev} in Step 2b)i. Before we can introduce an accurate definition for them, we need to specify a measure ($PGain$, see below) for an attribute's previous contribution to the retrieval error improvement on the training data. Roughly speaking, $PGain$ employs, first, a coarse approximation of the accumulated fitness improvements obtained during previous learning of a local similarity measure for the respective attribute and, second, an appropriate weighting scheme.

Definition 5.2 (Attribute-Specific Retrieval Error Improvement Contribution)
*Given the learner's reflective optimisation history $H = (h_1, \cdots, h_{\hat{t}})$, the round-robin size ϱ, an attribute $A_i \in \{A_1, \cdots, A_n\}$ with belonging current weight w_i, the **attribute-specific retrieval error improvement contribution** is defined as*

$$PGain_i(\hat{t}) = w_i \cdot \sum_{j=1}^{\nu} \frac{h_{find(i,j)}.\hat{E} \; - \; h_{find(i,j)+\varrho}.\hat{E}}{h_{find(i,j)}.w} \tag{5.4}$$

where $find(i, j)$ searches the index of the history entry that corresponds to the point of time at which P_i was j generations younger than presently ($age(P_i)$), i.e.

$$\begin{array}{lcll} find(i,j) & = & t & \text{so that} \quad h_t.\mathcal{P} \; = \; P_i \\ & & & \text{and} \quad\;\; h_t.\tau \; = \; age(P_i) - j \cdot \varrho \end{array}$$

*The value of parameter $\nu \in \mathbb{N}$ determines the **backward reach** covered by function $PGain$.*

Input : populations $P_1, \cdots, P_n$ and $P_{weights}$ for local similarity measures and feature weights, respectively, round-robin size ϱ

Output : optimised similarity measure Sim^{opt} consisting of optimised local measures and feature weights

1. **define** reflective optimisation history H
2. **while** stop criterium is false **do**
   a) **let** $P_{weights}$ evolve for ϱ generations
   b) **for** $dummy = 1$ **to** n
      i. **for** $i = 1$ **to** n
         set p_i^{ev} according to Equation 5.4 in Def. 5.2
      ii. **for** $i = 1$ **to** n
         set $\hat{p}_i^{ev} := \sum_{j=1}^{i} p_j^{ev}$
      iii. **set** $r :=$ random number from $[0; 1]$
      iv. **set** $m := min\{\ j \mid j \in \{1, \cdots, n\},\ r \leq \hat{p}_j^{ev}\}$
      v. **let** P_m evolve for ϱ generations and update H appropriately
3. **return** $Sim^{opt} = (sim_{A_1}^{fit}, \cdots, sim_{A_n}^{fit}, (w_1, \cdots, w_n)^{fit})$

Algorithm 5.5: Control Algorithm Based on an Intelligent Learning Scheduler

The definition of $PGain$ also allows for a more intelligent realisation of the stop criterium. Instead of terminating the evolution process after a certain number of generations has been processed, the stop criterium can also consider the current value of $PGain$. For example, the stop predicate might become true, if $\sum_{i=1}^{n} PGain_i(\hat{t})$ has fallen below a particular threshold for a certain number of consecutive cycles.

Note, that $PGain_i(\hat{t})$ may also be negative; this is, for instance, true for attribute HDSize in the experiment summarised in Figure 5.3, at least at later stages of the evolutionary process. Based on Definition 5.2 we may now introduce the calculation rule for the mentioned evolvement probabilities utilised by the intelligent learning scheduler.

Definition 5.3 (Evolvement Probabilities)
Given attributes $A_1, \cdots, A_n$, the round robin size ϱ and the learner's reflective optimisation history $H = (h_1, \cdots, h_{\hat{t}})$, the evolvement probability p_i^{ev} for each attribute A_i ($i \in \{1, \cdots, n\}$) is calculated after

$$p_i^{ev} = \begin{cases} \frac{1}{n} & \text{if } \hat{t} < n \cdot \varrho \text{ or } \sum_{j=1}^{n} PGain_j(\hat{t}) \leq 0 \\ \frac{norm(PGain_i(\hat{t}))}{\nu \cdot \sum_{j=1}^{n} norm(PGain_j(\hat{t}))} + \frac{1}{\vartheta \cdot n} & \text{else} \end{cases} \tag{5.5}$$

where $\vartheta > 1$ and $norm(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$.

The "normalisation" of $PGain_i(\hat{t})$ via $norm(x)$ ensures that the sum of all P_i^{ev} is 1. The parameter ϑ, which pragmatically should have a value larger 10, guarantees that a minimal evolvement probability is assigned to any population/attribute, even if there were no improvements achieved previously (i.e. $PGain_i(\hat{t}) \leq 0$). Thus, for example, $\vartheta = 20$ causes that it holds $PGain_i(\hat{t}) \geq 0.01$ for all $i \in \{1, \cdots, n\}$, when assuming $n = 5$. The first case in Equation 5.5 results in a statistically uniform scheduling which assures that the local similarity measures for all attributes are optimised with equal chance at the beginning of the learning process.

6 Experimental Evaluation

This chapter presents experimental results gained from an evaluation of the extensions to FLSM introduced in the scope of this work. The first section provides a proof of the principle learning capabilities of FLSM when applied to classification and solution prediction tasks. Afterwards, we focus on the learning improvements that can be gained by means of additional background knowledge that is incorporated into the optimisation process. The last section within this chapter examines the benefits of employing a learning scheduler to dynamically guide the search process into promising regions of the search space and compares the results to the standard round-robin proceeding.

6.1 Performance Evaluation for Predictive Application Domains

In this section we want to evaluate the extensions to FLSM that we have introduced throughout Chapter 3. First, we carry out an analysis of the error functions discussed above, as being applied to a smaller number of classification and solution prediction domains. On the basis of the corresponding experimental results we decide for specific parameter settings and then apply the introspective learning of similarity measures to various application domains. The results obtained from that second set of experiments shall also be considered with respect to the issue of overfitting.

The application domains and hence the belonging training (case) data sets used in the following are taken from the UCI Machine Learning Repository[1] of the University of Irvine, California (Blake and Merz, 1998).

6.1.1 Comparison of Error Functions

The choice of an appropriate error/fitness function is of crucial importance for the proper functioning of the evolutionary optimisation techniques we are employing. In order to make a decision for a specific error function, that shall be used throughout the remainder of this thesis, we compare FLSM's learning capabilities, when making use of different error functions as introduced in Chapter 3. Doing so, we distinguish between the following types of tasks and domains, respectively:

Classification Tasks: determining the best value for $\omega \in [0;1]$ (as parameter to E_C) in combination with the optimal class membership prediction strategy (linear, exponentiated with $\beta = 2$, position-weighted with $\alpha = 5$)

Numerical Solution Prediction Tasks: deciding on the best error function (E_{IS}, E_S, E_{P_s}, E_{P_d}) in combination with the optimal solution prediction strategy (linear, exponentiated, position-weighted)

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html

For the former tasks we created a benchmark on the basis of the learning results obtained in *three* classification domains (car, hayesroth and testdomain) for a relatively high number of training examples ($|TD| = 200$) and a sufficiently high number of evolutionary generations ($t = 200$). For the latter ones we confined ourselves on *two* domains only (autompg and servo) due to the higher computational effort, while keeping the other basic conditions unmodified.

In order to achieve a better comparability of results, we set $k = 10$ for all experiments conducted, i.e. a query's 10 nearest neighbours are taken into account, when predicting its class/solution. Of course, an online optimisation of k might have resulted in even better learning results.

The quality of a learnt similarity measure is expressed in terms of

- classification accuracy on an independent test data set in the case of classification tasks
- accuracy in predicting the (numerical) solution on independent training examples from a test data set in the case of solution prediction tasks.

Figure 6.1 summarises the average errors over all benchmark domains and over 10 repetitions of the entire experiment as well as the corresponding standard deviations.

Regarding solution prediction tasks, whose results are summarised in the left part of that figure, we need to mention that the *relative* prediction errors (relative to the error induced by the default similarity measure) are shown. That kind of normalisation is necessary and helpful since the absolute differences between correct and predicted solutions in the two involved application domains lie in different orders of magnitude, so that a simple average calculation would have distorted the results.

Apparently, the error functions based on ordinal/cardinal feedback about the correctness of the retrieved case order, E_{IS} and E_S, perform worse than the two error functions that are based on outcome prediction results. In particular, the employment of the index error from solution similarity does not produce a learnt similarity measure that yields better retrieval performance than the default similarity measure. On the other hand, both solution prediction error functions E_{P_s} and E_{P_d} lead to similarity measures that have retrieval errors that are on average more than 30% less than the error induced by the default measure. In the following, we make use of the combination $\alpha = 5/E_{P_s}$, i.e. the solution prediction similarity error with position-weighted ($\alpha = 5$) solution prediction strategy, which has yielded top results in the scope of the benchmark at hand.

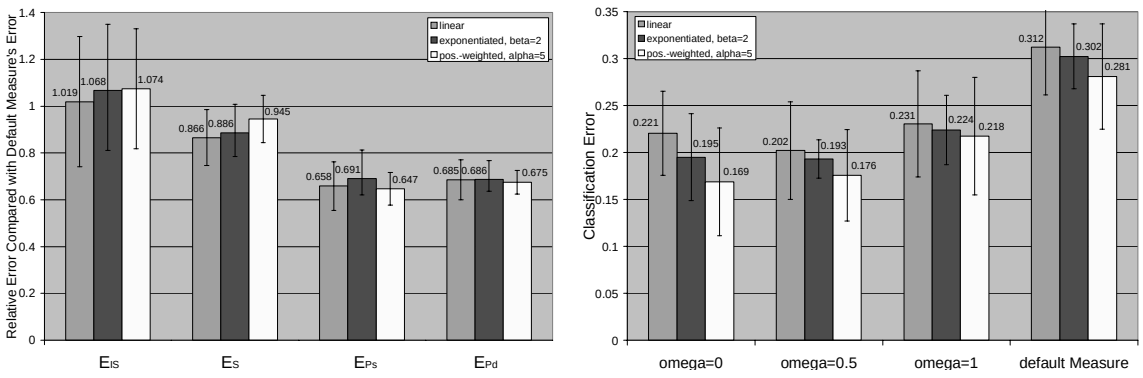


Figure 6.1: Comparison of Error Functions

Concerning classification tasks Figure 6.1 (right part of that illustration) makes clear that the differences induced by the usage of different error functions are not as tremendous. However, a significant reduction of the average classification error – compared to the error induced when using the default similarity measure – can be perceived. We decide to employ the combination $\beta=2/\omega=0.5$ in the following, meaning to use the exponentiated class membership prediction strategy with $\beta = 2$ and setting $\omega = 0.5$. That setting for ω obviously yields pretty good results no matter according to which strategy the class membership probabilities are calculated. Although the position-weighted class membership prediction strategy ($\alpha = 5$) narrowly outperforms our choice in the scope of this benchmark, we actually do not want to employ it for both types of application domains – since it is also our interest to experiment with different settings – and thus settle on using the exponentiated strategy ($\beta = 2$) for classification tasks.

6.1.2 Comparison of Application Domains

In this section we apply FLSM's learning capabilities to numerous application domains, proving that learning of similarity measures is beneficial in most application scenarios (learnt measures clearly outperform knowledge-poor default measures) and that, on the other hand, there are significant differences between achieved performance improvements in different domains. Setting off, we introduce a simple measure to formalise a learning algorithm's overfitting behaviour under certain parameter settings. As mentioned earlier, we use the default similarity measure $Sim_{default}$ (cf. Definition 2.6) as a basing point, i.e. we compare the quality of learnt measures with the quality of the default similarity measure.

The start of a single optimisation process is represented by forming and evaluating an initial population P_0 of random similarity measures (cf. Section 2.4.1). When the stop criterion t has become true, the learning algorithm has created an improved population of similarity measures with I_{opt} at its top. Thus, for the corresponding similarity measure Sim_{opt} it (usually) holds: $\hat{E}_x(TD_{train}, Sim_{default}) \geq \hat{E}_x(TD_{train}, Sim_{opt})$. However, due to the effects of overfitting (see Section 4.1) in many cases the improvement yielded during optimisation does not pass on to the test data, i.e. the improvements on the test data are usually smaller than on the training data. In other words, the model generated, here a similarity measure, is too specialised to the training data set.

Definition 6.1 (Optimisation Ratios)
*Given the default similarity measure $Sim_{default}$ as well as an optimised one Sim_{opt}, a training data set TD_{train} and an independent test data set TD_{test}, the **training optimisation ratio** is defined according to*

$$\zeta = \frac{\hat{E}_x(TD_{train}, Sim_{default})}{\hat{E}_x(TD_{train}, Sim_{opt})}$$

*and the **test optimisation ratio** as*

$$\xi = \frac{\hat{E}_x(TD_{test}, Sim_{default})}{\hat{E}_x(TD_{test}, Sim_{opt})}$$

where x denotes the error function and may be chosen from $\{IS, S, P_s, P_d, C\}$.

*Moreover, the fraction $\varphi = \frac{\zeta-1}{\xi-1}$ is called the **overfit ratio**.*

Using that notation, overfitting can be described as $\zeta \gg \xi$. More generally, one can say: The higher the difference between ζ and ξ, the higher the impact of underfitting ($\frac{\zeta}{\xi} < 1$) or overfitting ($\frac{\zeta}{\xi} > 1$).

In practice, the performance of a learning algorithm is good, if it keeps the quotient $\frac{\zeta}{\xi}$ as small as possible, preferably near 1, while maximising ξ.

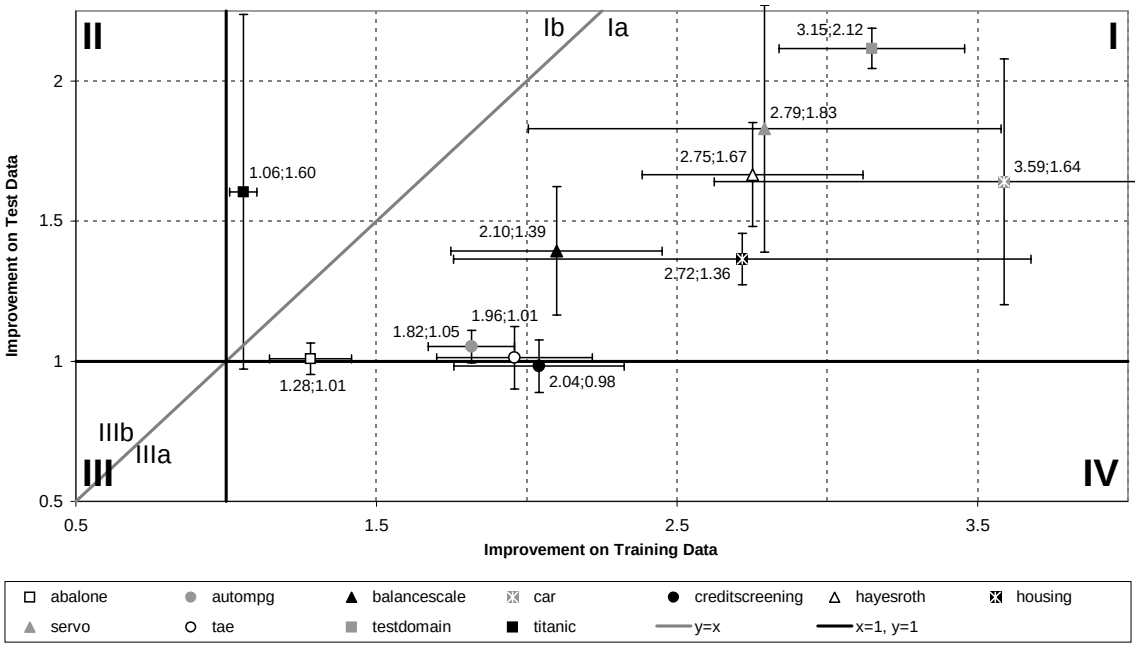


Figure 6.2: Optimisation Ratios in Various Application Domains and the Problem of Overfitting

Figure 6.2 opposes the training (x-axis) and test (y-axis) optimisation ratios for several application domains. It shows the average relative improvements for each domain together with the standard deviations along the training and test data dimension. All experiments are based on 200 training examples (except for hayesroth, servo and tae with $|TD| = 75$ due to a restricted amount of case data), on 200 evolutionary cycles and on 10 repetitions.

The diagram is divided into four quadrants, where quadrants II, III and IV determine regions in which no improvement on the training data ($\zeta < 1$), no improvement on the test data ($\xi < 1$) or even no improvement on both data sets is reached. Fortunately, all considered domains are to be found in the first quadrant, which itself is subdivided into two parts by the function $y = x$, creating an overfitting region (Ia) and an underfitting region (Ib). Reading the chart, one must keep in mind the following: On the one hand, the effects of overfitting are marginal, if the optimisation ratios for a particular domain lie near or above $y = x$. This is, at least, true for the titanic, testdomain and servo domains where $\varphi < 2.5$. On the other hand, a domain's ξ-value (y-axis) must be paid special attention to, as it refers to the actual improvement the similarity measure yields. We can distinguish three classes of domains: For some a clear improvement of more than 50% ($\xi > 1.5$) can be reached (testdomain, servo, hayesroth, car, titanic), for some an average improvement of less than 50% (balancescale, housing) and for the remaining ones only a very small improvement with a test optimisation ratio of less than 1.1.

The default similarity measure $Sim_{default}$ represents a benchmark to which we relate the results obtained using optimised measures. For all considered application domains the determination of the test optimisation ratio ξ (including the calculation of the average retrieval error on the basis of using $Sim_{default}$) reveals how beneficial the utilisation of FLSM is. Figure 6.3 depicts that improvement in percent ($\xi \cdot 100\%$) for each domain ($100\% \mathrel{\widehat{=}}$ quality achieved with default measure), which is achieved, when using the optimised, learnt similarity measure Sim_{opt}.

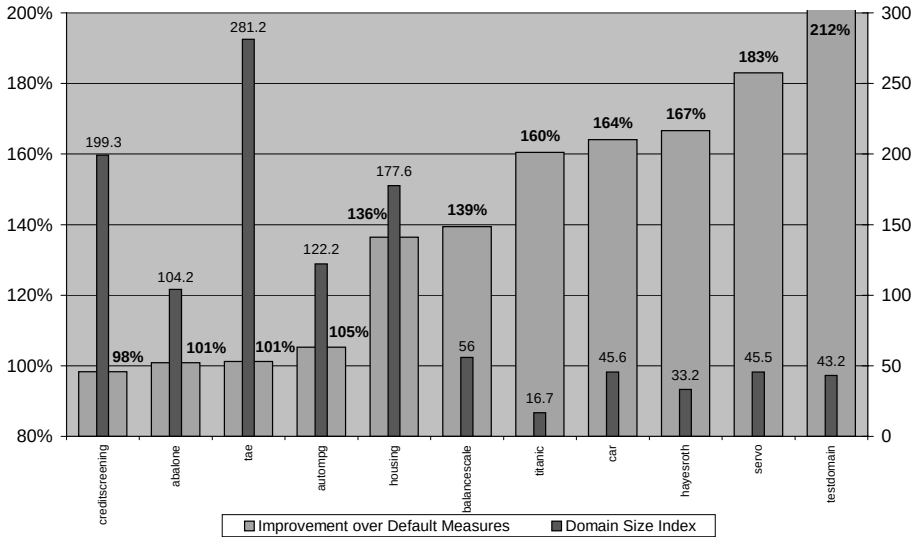


Figure 6.3: Relative Retrieval Improvements Compared to Default Measures

The domains in Figure 6.3 are ordered from left to right with respect to increasing performance yielded, i.e. with respect to increasing improvement ξ over the default measure. Obviously, there are considerable differences in FLSM's ability to ameliorate the similarity measures in the various domains (from no improvement to 112%). We wondered by what these differences are caused and found a correlation with the complexity of the respective domain.

Definition 6.2 (Domain Size Index)
Let $\mathbb{D}$ be an application domain with n attributes $A_1, \ldots, A_n$ of which the attributes $A_1, \ldots, A_\sigma$ ($\sigma \leq n$) are symbolic. Furthermore, let the value range D_{A_i} of each symbolic attribute A_i consist of m_i elements.

Then the domain size index is defined as

$$\delta_{\mathbb{D}} = n + s \cdot (n - \sigma) + \sum_{i=1}^{\sigma} m_i^{2\theta}$$

where s denotes the predetermined number of sampling points used to interpolate similarity functions and $\theta \in [0.5; 1]$ may be used to take into consideration that some regions of similarity tables are of minor relevance.

The domain size index $\delta_{\mathbb{D}}$ represents a very coarse heuristic to measure the complexity of a specific application domain. For $\theta = 1$ it equals the number similarity values that can be

altered during learning, i.e. the sum of all feature weights, sampling points and entries in similarity tables. The second (dark gray) data series in Figure 6.3 gives an impression of the domains' sizes we used during our experiments (we set $\theta = 0.75$).

Apart from one exception, FLSM performed bad ($\xi < 1.20$) on "big" domains ($\delta_{\mathbb{D}} > 100$) and well on "small" domains. At first glance, one might suspect that this peculiarity can be blamed on overfitting. This is partly true, as three of the four bad domains have an overfit ratio of $\varphi \geq 15$, whereas the remaining domains obtain an overfit ratio of less than 5. Anyway, the more underlying explanation goes back to the complexities of the domains as shown in the chart above.

However, when having a closer look at Figure 6.2, one can find that FLSM did (in most cases) achieve only very little improvements for the bigger domains on the *training* data as well. Originally, we expected that especially the increased domain size and thus the increased search space would give way for an evident optimisation success on the training data (of course, mainly due to overfitting), i.e. causing the entries for abalone, creditscreening or tae to feature a higher training optimisation ratio and hence a higher x-value in Figure 6.2. Here, it remains uncertain whether that effect would have happened to occur, if we had increased the number of evolutionary cycles performed during the experiments. One may presume that a more dynamic stop criterion, which does not dispatch the same amount of computational resources to each learning process (e.g. a fixed number of evolutionary generations) but which also takes the respective domain size and thus the complexity of the optimisation process into consideration, would have led to different results.

6.2 Filter Experiments

The previous section has stressed the general learning capabilities of FLSM when applied to classification and solution prediction tasks. On the basis of our observations and determination of relevant parameter settings we now want to examine the possible benefits, i.e. learning improvements, that can be obtained when certain forms of background knowledge are incorporated into the learning process.

Without any doubt, the number of different strategies and techniques to improve the optimisation of similarity measures, that we have introduced, is numerous. Accordingly, the number of definable knowledge filters is amazingly high. Furthermore, the framework for learning similarity measures allows for plenty of parameter settings, so that the number of experiments, that actually ought to be conducted in order to obtain a really thorough evaluation, is unfeasibly high as well. Moreover, the learning process in general is – despite all the optimisations we have developed – still a complex task, computationally very demanding and hence extremely time-consuming. For these reasons it is impossible to realise an exhaustive experimental analysis of the whole parameter space, knowledge filter space and space of application domains in the scope of this work. As a consequence, some of the approaches to incorporate knowledge into the learning will not be evaluated explicitly, that task is left for future work.

Nevertheless, we try to design our experimental evaluation as comprehensive as possible by selecting parameter settings for the learning algorithm that – at least so far – have proved to result in convincing and stable results (e.g. the fitness functions determined in the previous section) and by constructing a little number of classes of filters. For each of those classes, one representative will be employed during learning. Finally, we picked six application domains with rather different characteristics so that a good overview, though no statistically significant

generalisations, of the achieved improvements over filterless learning can be obtained.

6.2.1 Filter Definition and Experiment Planning

As already mentioned, it is infeasible to define, apply and evaluate each possible knowledge filter. Instead we identify a number of classes of filters and exemplarily employ one representative from each class in each considered domain. Defining *filter classes*, we pursue the distinction regarding the types of knowledge that are used to enhance the learning, namely similarity meta knowledge (cf. Section 4.3) and expert knowledge (cf. Section 4.4) as well as a combination of both.

m-Filters contain similarity meta knowledge only. This means, they may make use of the reflexivity, symmetry or monotony constraint, for example, or of an arbitrary combination of them, depending on the purposefulness of constraint usage in the respective application domain. Moreover, they are allowed to employ the knowledge mined from the case base, e.g. by introducing a grid all similarity values have to fit in. It is important to emphasise, that m-Filters are of special interest since the knowledge acquisition effort to define them is only marginal.

e-Filters are enhanced via specific expert knowledge. Hence, those knowledge-based optimisation filters can include bound specifications, expert estimations, confidence levels etc. Although e-Filters are also permitted to exploit the advantages of structured (taxonomic and ordered symbolic) attributes, none of our experiments covers these opportunities.

me-Filters represent the combination of the former ones, incorporating both kinds of additional knowledge to guide the optimisation process.

The main focus of our experiments is laid upon a confrontation of unfiltered and filtered learning. We compare the learning results obtained, when using no filter at all, an m-Filter, an e-Filter and an me-Filter, respectively. Moreover, we distinguish between different amounts of training examples used for learning and illustrate the improvement, called *filter gain*, that is created due to the usage of filters, i.e. relative to unfiltered learning. That proceeding is repeated for each combination of filter type and training data size used.

We conducted the experiments in six application domains already known from the previous section (balancescale, car, hayesroth, housing, servo, testdomain) and determined average results as well as standard deviations for 10 experiment repetitions in each domain. We have to admit that we favoured "smaller" domains (with one exception: housing), i.e. those with a domain size index of less than 100 (cf. Figure 6.3) due to the enormous computational effort required to accomplish all experimental repetitions[2]. However, we presume that at least similar results are to be expected in the more complex domains.

Note, that the gradations concerning training data size are not equal for all domains, which is due to a limited amount of available case data in some domains. The number of evolutionary cycles, however, granted to the optimisation of each attribute's local similarity measures as well as to the learning of feature weights was held constant throughout all experiments ($t = 200$). This implies that we did not employ the intelligent learning scheduler (cf. Section 5.5) and instead proceeded according to standard round-robin processing.

[2] 6 domains × 10 repetitions × 4 filter types × 4 or 5 training data sizes

Another relevant question is from which source we obtained the necessary expert knowledge to be incorporated into the respective e-Filter. The answer is: By different means. The testdomain was created artificially on the basis of known optimal (target) similarity measures whose usage results in an error-free retrieval. Consequently, it is easy to simulate the corresponding expert knowledge. In Figure 6.4 we exemplarily show how an e-Filter for one of the testdomain's attributes looks like. Furthermore, the granularity values for the corresponding m-Filter are shown.

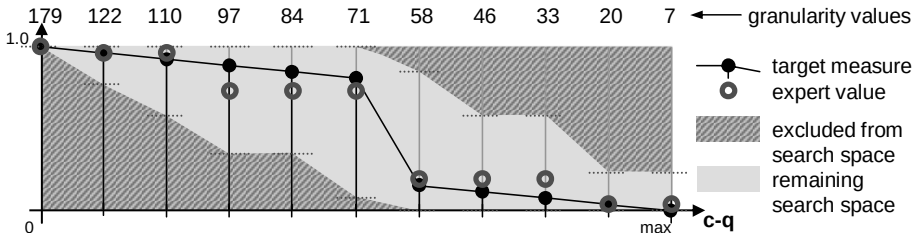


Figure 6.4: Example for an e-Filter in the testdomain and Granularity Values for the Corresponding m-Filter

For three further domains (balancescale, car and hayesroth) there is a domain theory available (Blake and Merz, 1998) which gives substantial hints concerning appropriate similarity measures. For the housing and servo domain we conducted a large-scale similarity measure learning, i.e. on the basis of a maximally high amount of training examples. The average learning results after a number of (very time-consuming) repetitions of that learning gave us an indication how adequate expert knowledge might look like. Regarding the e-Filter's definition, we paid attention that the search space restriction induced by the respective filter is approximately the same for each domain so that the results are better comparable.

Because of its increased complexity we omitted the conduction of an me-Filter evaluation in the housing domain. Furthermore, the m-Filter in that domain, which actually is intended to predict the prices of housings in the Boston area, makes use of the monotony constraint only, so that the corresponding experiment represents an evaluation of that constraint's benefits.

6.2.2 Results

Before going into a detailed domain-specific presentation of results, we want to start with some general remarks on the conducted experiments and thus consider the utilisation of additional background knowledge via filters as a whole.

We are aware that our results cannot be regarded as being statistically significant as restrictions in available computing power allowed us to repeat each experiment only 10 times (in most cases) and calculate average learning results and standard deviations. Despite that small number of iterations, we think that our evaluations show the general usability and purposefulness of knowledge filters as a mean to enhance the optimisation process.

All in all, the employment of knowledge-based optimisation filters led to improved learning results (apart from minor exceptions which probably represent outliers). The magnitudes of achieved improvements, however, differed enormously over the various application domains. Moreover, no clear tendency emerged concerning which filter type yields the highest gains for which training data sizes. In many cases, the me-Filter produced learning improvements that are approximately the sum of the m-Filter's and the e-Filter's gain. This effect is plausible

since me-Filters represent a combination of the other ones and hence incorporate the most knowledge.

Furthermore, it became obvious that the incorporation of expert knowledge in general generates higher gains than the usage of similarity meta knowledge only. This is not too surprising insofar as expert knowledge can be described as more exhaustive and substantial than similarity meta knowledge. That kind of outperforming, however, must be paid with the higher knowledge acquisition effort that has to be invested when employing expert knowledge.

The left result charts shown in Figures 6.5 and 6.6 summarise the achieved error reductions for increasing training data sizes (x-axis) with respect to the employment of default similarity measures, learnt similarity measures without filter and learnt similarity measures with three different types of knowledge filters (m-, e- and me-Filter). The charts in the right sketch the filter gain in percent ($100\% \cdot (1 - \frac{error_{filtered}}{error_{unfiltered}})$), i.e. the additional learning improvement that is achieved due to the utilisation of knowledge filters (compared to filterless learning).

For reasons of clarity, we try to group the obtained results into three fractions.

Group A Learning improvements could be obtained especially for small training data sizes, regardless of the used filter types (though e-Filters slightly outperform m-Filters). When learning on the basis of a higher number of training examples ($|TD| > 50$), the impact of filter usage is in decline.

Group B No matter how many training examples were given as input to the repeated optimisation processes, the usage of knowledge filters always causes improved learning results. Tendentially, the gains for larger amounts of training data are even higher than for smaller ones. Here, the improvements reached via m-Filters are comparatively low.

Group C In this group, a filter-specific distinction must be introduced. This means, while one filter type is suitable for rather small training data sizes, another one creates more learning improvements for higher amounts of training data.

In the following, we present and discuss the obtained results with respect to that division.

Discussion

The car, housing and servo domain can be classified as belonging to Group A. The filter gains achieved by the [m]e-Filters are indeed notable, varying between 15% and 35% for small training data sizes. In the housing domain all attributes are numeric and the m-Filter contained the monotony constraint exclusively. Obviously, that "simple" heuristic's impact on the learning success is impressive (around 25% gain).

When learning on the basis of a higher amount of training examples, the benefit of incorporating knowledge into the optimisation process shrinks. That effect can be explained by a better informedness of the learner. For example, one may say that 75 training examples are sufficient for the learning algorithm to produce optimal results – additional knowledge does not create further improvements. Yet, we may also claim that the quality of the additional expert knowledge did not suffice to yield those extra improvements.

That mentioned observation is less distinct in the housing domain since that domain's complexity is much higher. Note, that we have omitted the housing experiments for $|TD| = 200$, as one single learning process with that amount of training examples would have lasted approximately 30 hours on a P-IV 1.8 GHz machine.

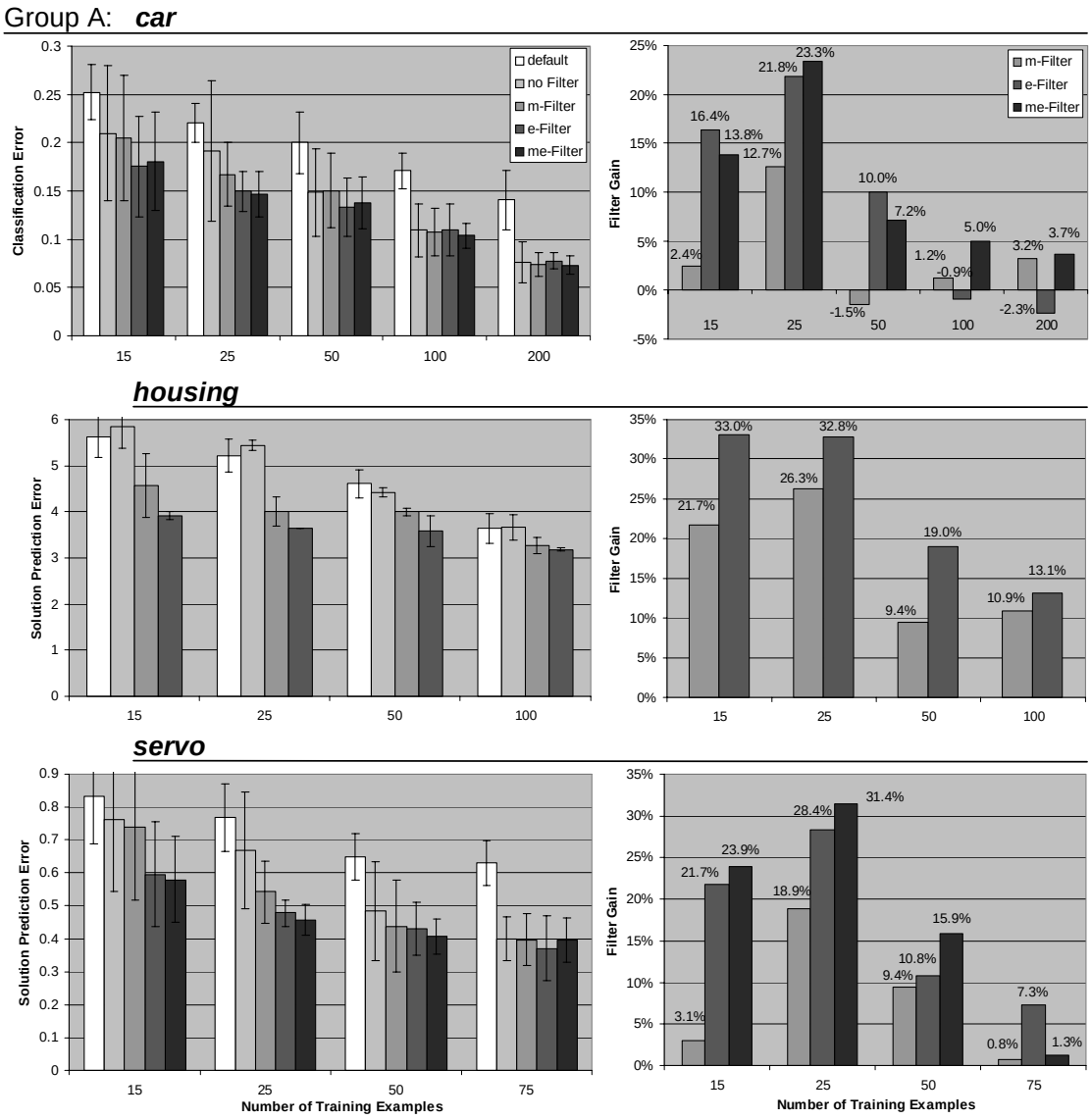


Figure 6.5: Filter Experiments – Results for Group A

Group B includes the balancescale and hayesroth domain. Similarity meta knowledge, to be exact the usage of the symmetry constraint[3] and appropriate granularity values, resulted in minor filter gains only (in general not much more than 10%). The incorporation of expert knowledge via e-/me-Filters however, has shown a substantial influence. This may be due to the fact that in both domains the expert knowledge for defining the corresponding e-Filter, partly stemmed from an available domain theory. Consequently, that knowledge can be considered as being of high quality – compared to the rather vague expert knowledge for the domains in Group A – and therefore responsible for quite high filter gains even for training data sizes $|TD| \geq 75$.

[3]The monotony constraint was not employed here, as all of both domains' attributes are symbolic.

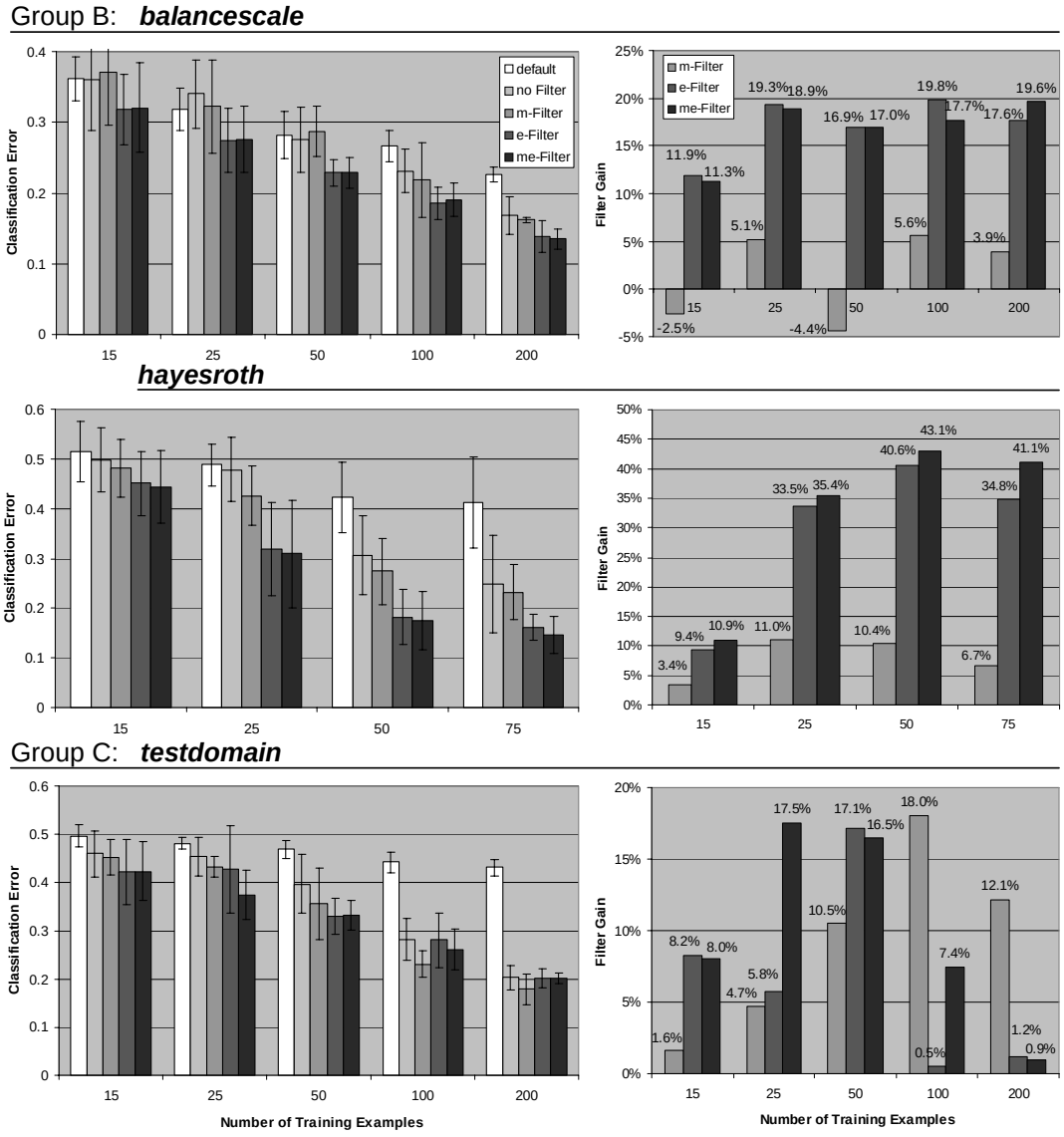


Figure 6.6: Filter Experiments – Results for Groups B and C

The testdomain, the only domain in result Group C, represents a special case. This artificially created domain was designed to produce relatively poor results, when classifying on the basis of default similarity measures (classification error > 0.4). Accordingly, the application of FLSM is in general supposed to optimise the similarity measures to be used on a grand scale. As a consequence, the room for further improvements by means of utilising background knowledge is rather small.

Concerning m-Filters we found that their benefit grows with the amount of training data used for learning. The contrary observation could be made for expert knowledge: Using an e-Filter, the filter gains are larger for smaller amounts of training data. Of course, we would have been able to define a "perfect" e-Filter as we know the "target" similarity measures which produce no classification error at all in this domain. However, as mentioned above, the search

space restriction induced by the testdomain's e-Filter is approximately the same as in other domains.

6.2.3 Extended Overfit Analysis

The bias which the knowledge contained in our filters exerts on the learning algorithm, i.e. on the evolution strategy, is supposed to suppress the learner's tendency to adapt the learnt similarity measures too much to the training data. In other words, it should increase the learner's ability to generalise. This effect ought to manifest itself in a reduced overfit ratio φ, as well.

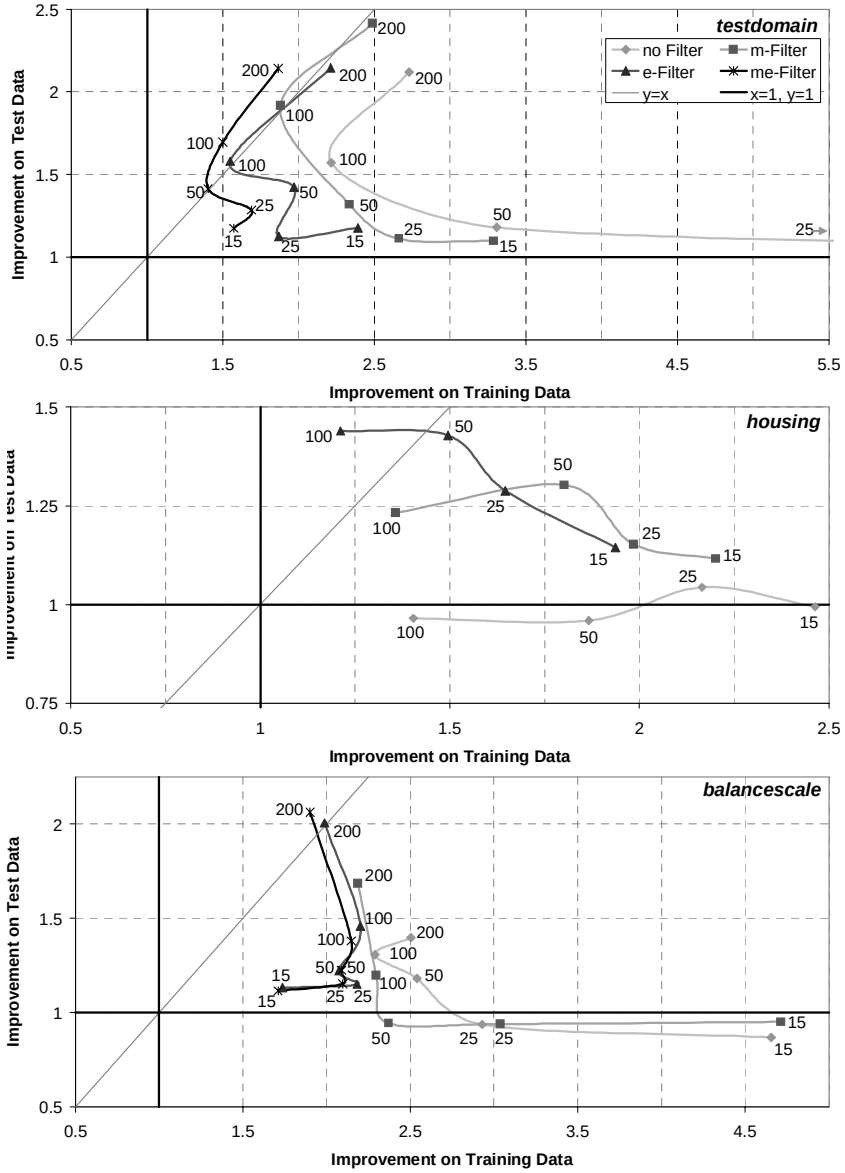


Figure 6.7: Extended Overfit Analysis for Three Application Domains: Train vs. Test Optimisation Ratios with Respect to Filter Usage and Used Training Data Size

Returning to the illustration of overfitting as given in Figure 6.2 in Section 6.1, we expected that the employment of knowledge-based optimisation filters results in a "shift" of the data points into the upper-left direction, i.e. towards quadrant Ib. In fact, that kind of move was – at least in principle – observable for some domains.

In Figure 6.7 three charts are shown that oppose the train and test optimisation ratios[4] for increasing training data sizes used for learning (data points on a single curve) and with respect to the usage of knowledge filters (different curves). Obviously, the usage of knowledge filters indeed causes an upper-left shift of the respective curves.

The first chart in Figure 6.7 reflects the improvements for the testdomain. Here, apparently, the utilisation of additional knowledge via knowledge filters causes an notable reduction of the train optimisation ratios while increasing the test optimisation ratios marginally. Furthermore, the general runs of the curves are interesting. They reveal the dependency on the amount of training data used for learning. Obviously, with increasing training data sizes the curves approach $y = x$.

In the housing domain (second chart) filterless learning reaches almost no learning improvements on the test data, at least for $|TD| \leq 200$. For $|TD| = 200$ there is an improvement of 36% (see Figure 6.3), but as already mentioned we did not conduct filter experiments for that training data size. For filtered learning, however, the upper-left shift is obvious, though it takes a different shape than in the other two domains.

The curves for the balancescale domain in the third chart suggest the following: No matter which type of filter is employed, up to a certain amount of training data the learner yields none or only very little improvements on the test data (y-value less than 1.2) compared to the default similarity measures. When the number of training examples used for learning exceeds a certain number, however, the learning success grows almost linearly with the amount of training data. The application of knowledge filters here mainly causes smaller train optimisation ratios (in particular expert knowledge) and slightly increased test optimisation ratios for $|TD| \leq 100$. For higher amounts of training data ($|TD| > 100$), however, additional knowledge obviously boosts the test optimisation ratio.

We are convinced that the properties and appearance of the resulting graphs in some domains are influenced by the fact that two-dimensional plottings are given here. Accordingly, both, the respective x-value and the y-value, being characterised by relatively high standard deviations, feature some element of randomness which collectively leads (for some data points) to unbalanced values and thus to a maybe slightly peculiar illustration. Presumably, the quality of the respective charts would have increased, if we had made more than only 10 repetitions for each data point.

6.3 Scheduler Analysis

The implementation- and optimisation-specific details presented in Chapter 5 covered two techniques to improve the run-time behaviour of FLSM. On the one hand, we suggested a smart fitness calculation. Due to its enormous impact on the overall performance of FLSM (it speeds up the learning process by more than 5 times), the smart fitness calculation has been applied during all experiments whose results are presented in the scope of this work.

On the other hand, we developed an intelligent learning scheduler whose intention is to avoid a uniform dispatching of computational resources to all attributes a specific case rep-

[4] Mind the differently scaled x- and y-axes in these charts.

resentation consists of. Instead it aims at approving those attributes of more computation time whose similarity measures' optimisation will presumably result in early and high learning improvements. Note, that throughout the experiments discussed in the current chapter we have not made use of that scheduling algorithm. Consequently, a thorough evaluation of that tool depicts a future task. Nevertheless, we have conducted first practical tests with the intelligent learning scheduler and applied it to the PCCustomisationDomain (cf. Section 2.3.3). An attribute-specific analysis of all attributes' fitness improvement contributions (cf. Figure 5.3) in that domain gave the impetus to the development of the intelligent learning scheduler.

Figure 6.8 summarises the result of our tests. It shows the accumulated fitness improvements with respect to the number of evolutionary cycles that has been processed (averaged over 5 repetitions). Obviously, the learning with using our scheduler causes a steeper learning curve and yields in creating fitter individuals, i.e. better similarity measures, at earlier points of time. At later stages of the evolutionary process both curves slightly approach towards each other.

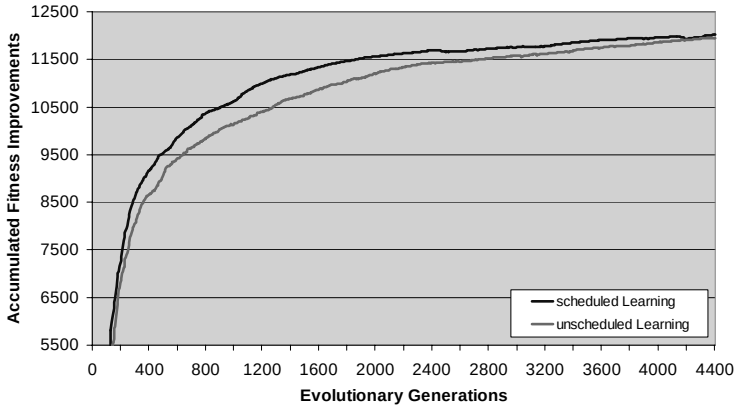


Figure 6.8: A First Analysis of the Benefits of Using the Intelligent Learning Scheduler

Note, that the actual difference between both curves lies in the sequence according to which the local similarity measures for the involved attributes are optimised. While unscheduled learning treats all attributes equally (round-robin processing), scheduled learning favours the optimisation of some attributes' measures over the optimisation of other ones. As a consequence, the former strategy grants a constant number of evolutionary generation ($t = 400$) to the evolution of each attribute's similarity measure. The latter strategy, however, optimised the population for attributes HDSize and Casing for only 158 and 243 cycles (on average), respectively, whereas letting CPUType's and MainboardType's population evolve for 661 and 934 generations, respectively.

A more exhaustive analysis of the learning scheduler's benefits as well as a fine-tuning of its relevant parameters is left for future work. Moreover, another striking question will have to be addressed: The usage of the intelligent learning scheduler apparently results in an improved learning as far as the training data is concerned (fitness improvements refer to the quality of the corresponding similarity measures when applied to the training data). But, at this point it remains uncertain, how much of those improvements can be conveyed to some independent test data set. In other words, it has to be examined whether the employment of a learning scheduler does not impair the learner's ability to generalise and whether it really causes a faster detection of high-quality similarity measures.

7 Conclusions and Future Work

The knowledge container model for CBR systems speaks of four containers of knowledge by which a specific system is characterised (Richter, 1995). The focus of the thesis at hand was laid upon similarity measures, i.e. on the retrieval knowledge container.

7.1 Summary

The work of Stahl (2003) introduces a framework for learning similarity measures (FLSM). Its main idea is to employ information about the correctness of retrieval results (called case order feedback) in order to adjust and optimise a CBR system's similarity measures. Here, the term "optimise" means to enable the similarity measures to assess the cases' utility for the respective query more accurately.

Our work directly builds upon that learning framework and aims at its enhancement into various dimensions. To do so, we started with giving a brief overview of that framework, an outline of some of Case-Based Reasoning's basics and with an identification of those of FLSM's characteristics whose improvement should be tackled within the scope of this work (Chapter 2). Moreover, we also introduced some of the learning framework's properties in more detail, e.g. the utilisation of evolutionary algorithms, in cases in which those details were needed for the subsequent chapters.

A first achievement of this thesis was the extension of the learning framework's applicability, to be exact, to make it applicable in classification and solution prediction domains, too. Therefore, we have introduced several new possibilities to calculate the retrieval error (and therewith to assess a similarity measure's quality) as well as a form of introspective learning that optimises the CBR system's similarity measures from a collection of, for instance, pre-classified cases.

On the one hand, that extension results in a broader applicability of FLSM. On the other hand, the opportunity to learn similarity measures in classification and solution prediction domains is beneficial insofar as it has given us access to a testbed of various application domains for which a sufficient amount of cases is available. The latter issue has been of special importance for the experimental evaluation of our work.

The main goal of this work concerned the development of strategies to incorporate additional background knowledge into the optimisation process, therewith improving it. We have identified a number of different sources of background knowledge and divided them into two groups: retrieval meta knowledge and expert knowledge. Retrieval meta knowledge denotes simple heuristic constraints to guide the search into regions of the search space where similarity measures can be found that are considered to be of "realistic shape", i.e. whose appearance is probable in practice. Moreover, an essential source of knowledge is represented by the available case data: A closer analysis of the interrelations within the case base may reveal certain characteristics of the respective application domain that can be utilised to bias the search (semi-automatically derivable). Expert knowledge, on the other hand, denotes substantial

additional knowledge that may be provided by a knowledge engineer, if he/she is capable of giving a partial definition of an appropriate similarity measure, thus effectively restricting the search space. All those knowledge sources aim at a restriction of the search space making some regions of that space "preferable" over other ones. The actual restriction is accomplished via so-called knowledge-based optimisation filters – entities into which all the gathered knowledge is included and which are allowed to actively intervene and thus bias the search.

Another objective of the thesis at hand has addressed the amelioration of the optimisation process itself, in particular aiming at speeding up the rather time-consuming evolutionary optimisation techniques. Concerning that, we have proposed the employment of a smart fitness calculation (smart retrieval error calculation), storing as many intermediary computation results as possible and reusing them at later points of processing time. Finally, we have suggested a reflective control of the entire learning process which at run-time dynamically dispatches computation time to the search in those regions of the search space in which quick learning improvements are highly presumable.

Demonstrating the general capabilities of the concepts introduced throughout this thesis, we have conducted numerous experimental examinations. On the one hand, we have highlighted the learning results that could be obtained for classification and solution prediction scenarios. Though FLSM was able to learn similarity measures that clearly outperform default measures, it was obvious that there was a lot of room left for further improvements (especially, when using small amounts of training data). So, on the other hand, we have also examined the possible benefits that could be obtained when additional background knowledge was incorporated into the learning process via knowledge filters. The results, as presented in Chapter 6, depict the convincing performance of the presented approach: In almost all considered application scenarios the utilisation of background knowledge has caused clear learning improvements. Furthermore, we have shown that the impact of overfitting (which usually shows up for smaller training data sizes used for learning) could be reduced due to knowledge filter usage.

7.2 Perspective

Nowadays the amount of data, information and knowledge to be processed grows steadily. As described in the Introduction, CBR represents an established technology tackling that problem. However, as practical tasks are becoming more and more complex, the effort to build up a case-based system increases evermore. Hence, one logical consequence is to automate some of the steps required, when developing a CBR system. A comprehensive approach to realise that promising idea has been introduced with FLSM (Stahl, 2003) – and our work has shown a number of possibilities to enhance and ameliorate the capabilities of that framework.

The current implementations of both the learning framework and our extensions are of prototypic character and are not suited for practical use. Regarding the definition of knowledge filters, one of the next steps ought to comprise the design and development of ergonomic, human-centred graphical user interfaces which allow for a simple acquisition and incorporation of background knowledge, in particular of expert knowledge.

The sources of knowledge we have considered in the scope of this work have revealed a considerable potential to yield improved learning results. When identifying and using further forms of additional knowledge those benefits may be even increased – the architecture of our implementation facilitates the easy incorporation of new types of knowledge disregarded so far.

No doubt, a more extensive experimental evaluation would require a higher number of experimental repetitions (i.e. more than 10) and should produce statistically significant results. Due to the tremendous computational effort, we leave that task for future work. Furthermore, a more thorough analysis of different parameter settings might generate interesting insights. Apart from that, it was unfeasible to conduct an exhaustive examination of all definable knowledge filters. Instead we have confined ourselves to three classes of filters and in so doing omitted to analyse the effects of incorporating all possible forms of knowledge into the learning process. So, we have not evaluated the utilisation of vocabulary knowledge (a special form of expert knowledge, cf. Sections 4.4.3 and 4.4.4) or the benefits that may be achieved via a non-uniform sampling point distribution for numeric attributes (cf. Section 4.3.2). An experimental analysis of those knowledge sources represents another future task.

Finally, we suggest to also compare the learning results (e.g. achieved classification accuracies) obtained with optimised similarity measures – via filterless as well as knowledge-enhanced learning – with other symbolic or sub-symbolic machine learning techniques. This may prove or eventually disprove that the application of Case-Based Reasoning is superior to non-CBR systems in the respective application domain. No matter, whether it is or whether it turns out to be not, our analyses have clearly shown that the optimisation of similarity measures in general leads to greater accuracy in problem-solving, when employed in various application domains, and that the incorporation of additional knowledge even boosts that effect.

Bibliography

A. Aamodt and E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994. 2.2.4

Y.S. Abu-Mostafa. Learning from Hints in Neural Networks. *Journal of Complexity*, 6:192–198, 1989. 4.1.1

K.-D. Althoff. *Evaluating Case-Based Reasoning Systems: The Inreca Case Study*. Springer Verlag, Professorial Dissertation, University of Kaiserslautern, 1997. 2.2.3

R. Bergmann. On the Use of Taxonomies for Representing Case Features and Local Similarity Measures. In L. Gierl and M. Lenz, editors, *Proceedings of the Sixth German Workshop on Case-Based Reasoning (GWCBR)*, 1998. 4.4.3, 4.4.3

C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html`. 6.1, 6.2.1

A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Occam's Razor. *Information Processing Letters*, 24:377–380, April 1987. 4.1.1

A. Bonzano, P. Cunningham, and B. Smyth. Using Introspective Learning to Improve Retrieval in CBR: A Case Study in Air Traffic Control. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 291–302, Providence, Rhode Island, July 1997. Springer. 3.4

G.E.P. Box and M.E. Muller. A Note on the Generation of Random Normal Deviates. *Annals of Mathematical Statistics*, V(29):610–611, 1958. 5.2.2

L.K. Branting. Acquiring Customer Preferences from Return-Set Selections. In *Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR01)*, pages 59–73, Vancouver, British Columbia, Canada, July/August 2001. Springer. 3

C. Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. 2.4.1, 2.5

I. Dennis, J. Hampton, and S. Lea. A New Problem in Concept Formation. *Nature*, 243: 101–102, 1973. 4.1

S. Djoko, D. Cook, and L. Holder. Analyzing the Benefits of Domain Knowledge in Substructure Discovery. In *Proceedings of the KDD-95: First International Conference on Knowledge Discovery and Data Mining*, pages 75–80, Menlo Park, USA, 1995. IEEE Computer Society. 4.1

A. Doan, P. Domingos, and A. Helevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD-2001)*, pages 509–520, Menlo Park, 2001. ACM Press. 4.1

P. Domingos. The Role of Occam's Razor in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999. 4.1.1

S. Donoho and L. Rendell. Constructive Induction Using Fragmentary Knowledge. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 113–121, Bari, Italy, 1996. Morgan Kaufmann. 4.1.1

T. Gabel and M. Veloso. Selecting Heterogeneous Team Players by Case-Based Reasoning: A Case Study in Robotic Soccer Simulation. Technical Report CMU-CS-01-165, Carnegie Mellon University, December 2001. 3, 3.2, 3.2

D. Gamberger and N. Lavrac. Conditions for Occam's Razor Applicability and Noise Elimination. In M.v. Someren and G. Widmer, editors, *Proceedings of the Ninth European Conference on Machine Learning*, pages 108–123. Springer, 1997. 4.1.1

J. Heitkoetter and D. Beasley. The Hitch-Hiker's Guide to Evolutionary Computation. Issue 9.1, published as FAQ for comp.ai.genetic, available from `http://surf.de.uu.net/encore/www`, April 2001. 2.4.1

J.H Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975. 1.2, 2.4.1

J. Jarmulak and S. Craw. Genetic Algorithms for Featute Selection and Weighting. In S. Anand, A. Aamodt, and D.W. Aha, editors, *IJCAI-99 Workshop on Automating the Construction of Case-Based Reasoners*, pages 28–33, 1999. 3, 3.1

J. Jarmulak, S. Craw, and R. Rowe. Self-Optimising CBR Retrieval. In *Proceedings of the Twelfth IEEE International Conference on Tools with Artificial Intelligence*, pages 376–383, Vancouver, Canada, 2000. IEEE Computer Society. 3.1

J.D. Kelly and L. Davis. A Hybrid Genetic Algorithm for Classification. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 645–650, Sydney, Australia, August 1991. Springer. 3.1

I. Kopanas, N.M. Avouris, and S. Daskalaki. *Methods and Applications of Artificial Intelligence*, chapter The Role of Domain Knowledge in a Large Scale Data Mining Project, pages 288–299. Number 2308 in Lecture Notes in AI. Springer, Berlin, 2002. 4.1

F. Kursawe. Evolution Strategies – Simple Models of Natural Processes? *Revue Intl. de Systemique*, 7:627–642, 1993. 2.4.1

D.B. Leake. *Case-Based Reasoning - Experiences, Lessons & Future Directions*. AAAI Press, 1996a. 1.2

D.B. Leake. *Case-Based Reasoning - Experiences, Lessons & Future Directions*, chapter CBR in Context: The Present and Future, pages 3–30. AAAI Press, 1996b. 2

D.B. Lenat and R.V. Guha. *Building Large Scale Knowledge-Based Systems.* Addison-Wesley, Readinf, Massachusetts, 1990. 1.1

M. Lenz, B. Bartsch-Spoerl, H.-D. Burkhard, and S. Wess. *Case Based Reasoning Technology – from Foundations to Applications.* Lecture Notes in Artificial Intelligence 1400. Springer, 1998. 1.2

T. Li, S. Zhu, and M. Ogihara. Mining Patterns from Case Base Analysis. In Workshop on Integrating Data Mining and Knowledge Management Organised in Conjunction with ICDM'01: The 2001 IEEE International Conference on Data Mining, San Jose, USA, November 2001. 4.1

G. Marsaglia and T.A. Bray. A Convenient Method for Generating Normal Variables. *SIAM Review*, 6:260–264, 1964. 5.2.2

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, 1996. 2.4.1, 2.4.3

D. Michie, D. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994. 3.1

G. Nakamura. Knowledge-Based Classification of Ill-Defined Categories. *Memory & Cognition*, 13:377–384, 1985. 4.1

P. Nordin and W. Banzhaf. Complexity Compression and Evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, Pittsburgh, USA, 1995. Morgan-Kaufmann. 4.1.1

M.J. Pazzani. Influence of Prior Knowledge on Concept Acquisition: Experimental and Computational Results. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(3):416–432, 1991. 4.1

F. Ricci and P. Avesani. Learning an Asymmetric and Anistropic Similarity Metric for Case-Based Reasoning. Technical Report #9505-04, IRST, April 1995. 2.2.1

M.M. Richter. The Knowledge Contained in Similarity Measures. Invited Talk, The First International Conference on Case-Based Reasoning, Sesimbra, Portugal, 1995. 2.1, 7

M.M. Richter. *Lernende Systeme.* Kaiserslautern University of Technology, 2001. Lecture Notes on University Lecture 89-181. 2.2.1, 3.1

S. Schulz. CBR-Works – A State-of-the-Art Shell for Case-Based Applications. In *Proceddings of the Seventh German Workshop on Case-Based Reasoning, (GWCBR99)*, Wuerzburg, Germany, March 1999. 5.1

H.P. Schwefel. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. *Interdisciplinary Systems Research*, 26:319–354, 1977. 2.4.1

H.P. Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, 1981. 1.2, 2.4.3

H.P. Schwefel and T. Baeck. *Genetic Algorithms and Evolutionary Strategies in Engineering and Computer Science*, chapter Artificial Evolution: How and Why?, pages 1–19. Wiley, Chichester, 1998. 2.4.1, 2.4.3, 2.4.3

D.B. Skalak. Using a Genetic Algorithm to Learn Prototypes for Case Retrieval and Classification. In *Proceedings of the AAAI-93 Case-Based Reasoning Workshop*, pages 64–69, Washington, D.C., July 1993. American Association for Artificial Intelligence, Melo Park, CA. 3.1

A. Stahl. Learning Feature Weights from Case Order Feedback. In *Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR01)*, Vancouver, British Columbia, Canada, July/August 2001. Springer. 2, 2.5

A. Stahl. Defining Similarity Measures: Top-Down vs. Bottom-Up. In *Proceedings of the Sixth European Conference on Case-Based Reasoning (ICCBR 2002)*, Aberdeen, Scotland, UK, September 2002. Springer. 2.2.4, 2.3.3, 2.5

A. Stahl. *A Framework for Learning Similarity Measures in Case-Based Reasoning*. PhD thesis, University of Kaiserslautern, Kaiserslautern, 2003. To appear. 1, 1.1, 1.2, 2, 2.2.4, 3, 2.3.3, 2.4.6, 2.5, 7.1, 7.2

A. Stahl and T. Gabel. Using Evolution Programs to Learn Similarity Measures. In *Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCBR03)*, Trondheim, Norway, June 2003. Springer. 2.1, 5.5, 5.5.1

A. Stahl and S. Schmitt. Optimizing Retrieval in CBR by Introducing Solution Similarity. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'02)*, Las Vegas, USA, June 2002. CSREA Press. 2.3.2, 3.2

Z. Michalewicz T. Baeck, D.B. Fogel. *Handbook of Evolutionary Computation*. Oxford University Press, New York, 1997. 2.4.3

S.M. Weiss and C.A. Kulikowski. *Computer Systems that Learn – Classification and Prediction Methods from Statistics, Neural Nets, and Expert Systems*. Morgan Kaufmann, 1991. 3.1, 3.4

D. Wettschereck and D.W. Aha. Weighting Features. In M. Veloso and A. Aamodt, editors, *Case-Based Reasoning, Research and Development, First International Conference*, pages 347–358, Berlin, 1995. Springer. 3.1

W. Wilke and R. Bergmann. Considering Decision Cost During Learning of Feature Weights. In *Proceedings of Advances in Case-Based Reasoning, Third European Workshop (EWCBR-96)*, pages 460–472, Lausanne, Switzerland, 1996. Springer. 3.1, 3.1

E. Wisniewski. Learning from Examples: The Effect of Different Conceptual Roles. In *The Eleventh Annual Conference of the Cognitive Science Society*, pages 980–986, Ann Arbor, 1989. MI: Lawrence Erlbaum Associated, Inc. 4.1

J. Yang and V. Honavar. Feature Subset Selection Using A Genetic Algorithm. Technical Report TR #97-02a, Iowa State University, May 1997. 3

Z. Zhang and Q. Yang. Dynamic Refinement of Feature Weights Using Quantitative Introspective Learning. In *International Joint Conference on Artificial Intelligence*, pages 228–233. Morgan Kaufmann, August 1999. 3.4