Using Evolution Programs to Learn Local Similarity Measures

Armin Stahl, Thomas Gabel

Kaiserslautern University of Technology, Computer Science Department
Artificial Intelligence - Knowledge-Based Systems Group
67653 Kaiserslautern, Germany
`stahl@informatik.uni-kl.de, tgabel@rhrk.uni-kl.de`

Abstract. The definition of similarity measures is one of the most crucial aspects when developing case-based applications. In particular, when employing similarity measures that contain a lot of specific knowledge about the addressed application domain, modelling similarity measures is a complex and time-consuming task. One common element of the similarity representation are *local similarity measures* used to compute similarities between the values of single attributes. In this paper an approach to learn local similarity measures by employing an evolution program — a special form of a genetic algorithm — is presented. The goal of the approach is to learn similarity measures that sufficiently approximate the *utility of cases* for given problem situations in order to obtain reasonable retrieval results.

1 Introduction

When developing case-based applications, the definition of accurate similarity measures is one of the most important tasks. Even if a reasonable amount of high-quality case knowledge is available, the overall problem-solving capability of a CBR system strongly depends on the similarity measure employed. Only if it is possible to retrieve cases that are really useful for the current problem-solving situation, the full potential of the available case knowledge can be exploited.

According to the traditional CBR paradigm, namely "*similar problems have similar solutions*", the concept of *similarity* can be characterised as a heuristic used to estimate the *utility* of cases for particular problem-solving situations. However, the specific semantic of the term "similarity" generally depends on the employed similarity measures. In traditional CBR applications similarity often was interpreted as a kind of *similar look*. Similarity measures that are defined according to this semantic only consider the syntactical differences between the entities to be compared. Popular examples are the Hamming distance, the simple matching coefficient, the Euclidean distance and other simple distance metrics. In what follows, we call such measures *knowledge-poor similarity measures (kpSM)*. While such measures can easily be defined, the drawback of them is that they do not consider the particular coherences of the addressed application domain.

This often leads to bad retrieval results due to an insufficient approximation of the cases' utility.

Hence, in many recent CBR applications the term similarity is interpreted differently. Instead of only measuring syntactical differences, here, the *utility* of cases is approximated by considering the specific semantic of the case knowledge. This means, particular knowledge about the domain has to be encoded into similarity measures in order to obtain a well-founded utility approximation [1]. A very simple form of such *knowledge-intensive similarity measures (kiSM)* are specific feature weights that are used to express the different importance of the single case features.

Unfortunately, the use of kiSM is generally coupled with a significant drawback. To be able to define such measures, at least a partial understanding of the underlying application domain is required. However, CBR is often applied in domains where this understanding is nearly missing completely. And even if the domain is partially understood, the acquisition and formal representation of the required domain knowledge usually leads to additional development effort. This additional effort may prevent the use of kiSM, although they might significantly increase the efficiency and/or competence of the CBR system to be developed.

One possibility to avoid the mentioned drawback is the application of machine learning techniques to simplify the acquisition of similarity knowledge. Unfortunately, existing approaches to learn similarity measures are mostly restricted to classification domains only and cannot be employed directly in other application domains like e-commerce or general knowledge management scenarios. Further, existing approaches focus mainly on learning feature weights. However, when employing more sophisticated kiSM, major portions of the domain knowledge are usually encoded in complex *local similarity measures* used to compute similarities between single attribute values.

In this paper an approach to learn local similarity measures by using an evolution program — a special form of a genetic algorithm — is presented. The described approach bases on our general framework for learning similarity measures that has already been presented in [11, 12]. First we will discuss the basic difficulties that arise when defining kiSM manually and we review our general learning framework. After that we describe a new learning algorithm used to extend the framework on learning local similarity measures. To show the general applicability of the algorithm the results of a first experimental evaluation are presented. Finally, we close with a discussion of related work and an outlook on future research issues.

2 Defining Knowledge-Intensive Similarity Measures

In contrast to simple syntactical measures, the definition of accurate kiSM requires a difficult knowledge acquisition and engineering process. To guarantee reasonable retrieval results, knowledge about the utility of cases for particular problem situations has to be acquired and formalised. Basically, this knowledge strongly depends on the underlying application domain and the concrete application scenario of the CBR system to be developed. For example, the following aspects might influence how to estimate the utility of cases:

- Consider a traditional CBR scenario, like a classification or diagnosis task. Even if the domain is not completely understood, domain experts often possess *partial knowledge about the relationship between problems and solutions.* So, they know, for example, upper bounds for tolerable differences between the query and the cases' attribute values to ensure that a case is still relevant for a given problem situation.
- A different situation occurs in e-commerce or general knowledge management scenarios. Here, the utility of cases is not only determined by the domain, but also by the *individual preferences of the customers/users.* To be able to provide personalised recommendation functionality, the CBR system has to employ similarity measures that consider the preferences of individual customers/users or at least the preferences of certain customer/user classes.
- When providing case adaptation functionality, the semantic of the similarity measure is different to that in a retrieval-only system. Here, it is not appropriate to estimate the utility of a case for the given query directly. Instead, the similarity measure must approximate the utility of a case with respect to the available adaptation possibilities, i.e. it has to *guarantee the retrieval of easily adaptable cases* [10, 7].

When defining kiSM, one is often confronted with major problems that complicate or even avoid a good approximation of cases' utility. The kind of problems arising depends on the particular application scenario, for example:

- In traditional classification/diagnosis scenarios, *insufficient explicit domain knowledge* might be available. Possible reasons are, for example, a poorly understood domain, or the fact that an experienced domain expert is not available or too expensive.
- Even if an experienced domain expert is available, s/he is *usually not familiar with the similarity representation formalisms* of the CBR system. So, the formalisation of the provided knowledge might be very expensive because it might only be available in natural language.
- The knowledge about the real utility of cases might be not available at all during the development phase of the CBR system. When applying personalised similarity measures in an e-commerce or knowledge management scenario, the *knowledge can only be provided by the users themselves* during the use of the system.
- The *knowledge might only be available in another representation form.* For example, to estimate the adaptability of cases, one has to transfer adaptation knowledge into the similarity measure.

2.1 Representing Similarity Measures

In the following we assume an attribute-value based case representation and a commonly used structure to represent similarity measures consisting of

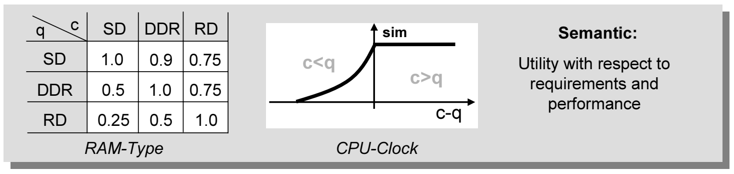
1. *local similarity measures* used to compute similarities between values of individual attributes,
2. *attribute weights* used to express the importance of individual attributes with respect to the utility approximation,
3. an *amalgamation function* used to compute the final similarity value for a given query and a case, based on local similarities and attribute weights.

Because this paper deals with local similarity measures we focus now on this part of the entire similarity representation. Generally, the representation of local measures strongly depends on the data type of the corresponding attribute. The two mostly used kind of types are numeric types, like Integer or Real, and different symbolic types, like unordered symbols, ordered symbols, and taxonomies. In this paper, we focus on two common representation formalisms that can be used to represent local similarity measures for numeric and symbolic data types. To reduce the dimensionality of similarity measures, commonly used commercial CBR tools derive similarity for numeric attributes from the difference of the values to be compared, leading to following definition:

Definition 1 (Similarity Function). *Let a be a numeric attribute with a defined value range of $D_a = [d_{min}, d_{max}]$. Under a **similarity function** for D_a, we understand a function $sim_a : [(d_{min} - d_{max}), (d_{max} - d_{min})] \longrightarrow [0, 1]$ that computes a similarity value out of the interval $[0, 1]$ based on the difference between a case value $c = d_x$ and a query value $q = d_y$ with $d_x, d_y \in D_a$.*

Definition 2 (Similarity Table). *Let a be a symbolic attribute with a defined list of allowed values $D_a = (d_1, d_2, \ldots, d_n)$. A $n \times n$-matrix with entries $x_{i,j} \in [0, 1]$ representing the similarity between the query value $q = d_i$ and the case value $c = d_j$ is called **similarity table** for D_a.*

Figure 1 shows two examples of local similarity measures that might be used in a product recommendation system used to recommend personal computers. The first measure is a similarity table for a symbolic attribute RAM-Type and the second measure represents a similarity function for a numeric attribute CPU-Clock. The semantic of these measures is an estimation of the utility of the technical properties of a given PC with respect to the given customer requirements. For example, if the customer wants to buy a PC with DDR-RAM and gets a PC with SD-RAM this leads to a significantly decreased similarity (0.5) because



q \ c	SD	DDR	RD
SD	1.0	0.9	0.75
DDR	0.5	1.0	0.75
RD	0.25	0.5	1.0

Fig. 1. Examples of Local Similarity Measures

the performance of SD-RAM is poor compared with the performance of DDR-RAM. On the other hand, the similarity with respect to the CPU-Clock is only decreased if the case value is smaller than the query value. A higher value always leads to a similarity of 1.0 because the customer will probably be satisfied if he gets an even faster PC, provided that other properties still match his demands (e.g. the price).

2.2 Bottom-Up Procedure

To allow a comfortable definition of local similarity measures, common CBR tools[1] provide powerful graphical modelling tools. Nevertheless, manually defining accurate local similarity measures is still a complicated and time-consuming task. It requires an analysis of each attribute to identify its influence on the utility of cases. Further, the estimated influence has to be encoded into an appropriate similarity measure by using the modelling facilities of the employed CBR tool. However, this bottom-up procedure has some general drawbacks:

- The procedure is very time-consuming. For example, consider a symbolic attribute with 10 allowed values. This requires the definition of a similarity table with 100 entries!
- Sometimes its not possible to estimate the influence of each attribute sufficiently due to the lack of reasonable domain knowledge. However, domain experts are often able to estimate the utility of whole cases for particular problem situations.
- Due to the complexity of the representation, users often make definition failures by mistake. Unfortunately, the identification of such failures is very difficult.
- Usually the quality of the completely defined similarity measure is not validated in a systematic way. Existing approaches (e.g. leave-one-out tests and measuring classification accuracy) only measure the overall performance of the CBR system, that is, of course, also influenced by other aspects, for example, the quality of the case data.

2.3 Alternative Strategy: Learning

In [12] we proposed an alternative strategy for defining kiSM that can be characterised as a top-down approach compared with the bottom-up procedure described in the previous section. The basic idea of this approach is to acquire only high-level knowledge about the utility of cases for some set of given problem situations. The necessary low-level knowledge required to compute the utility of cases for new problem situations is then extracted from the acquired high-level knowledge by employing machine learning techniques.

Figure 2 illustrates the basic idea of this approach. Generally, it requires some kind of *similarity teacher* that is able to provide the mandatory training

[1] For example, the commercial CBR shells CBR-WORKS and ORENGE of empolis knowledge management GmbH, formerly tecinno GmbH.

data [12]. This training data can be described as a set of corrected retrieval results called *case order feedback*. This means training queries are used to perform retrievals based on some initial similarity measure. The task of the similarity teacher is then the analysis of the obtained retrieval results with respect to the actual utility of the retrieved cases for the given queries. Obvious deficiencies have to be corrected by reordering the cases. Note, that the approach does not require feedback for all retrieved cases. Even information about the utility of a single case might be useful, for example, in an e-commerce scenario where the customer does not buy the most similar product but another one contained in the retrieval result. Of course, then a greater number of training queries is required to obtain a reasonable training data set. To be able to compare the obtained retrieval results with the case order feedback provided by the similarity teacher, a special error function has to be defined that measures the "distance" between the two given partial orders. Finally, the task of the learning algorithm is to minimise this error function by modifying the initial similarity measure.

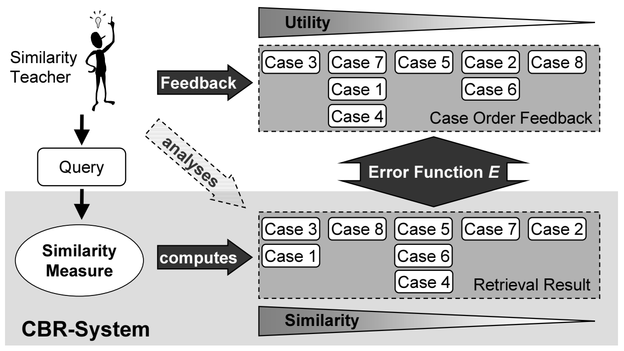


Fig. 2. Learning through Utility Feedback

Generally, the actual learning task, i.e. the minimisation of the error function, can be performed by any accurate learning algorithm. For example, in [11, 12] we used a gradient descent algorithm to learn attribute weights. Below we present an approach that employs an evolution program to learn local similarity measures.

3 Learning Local Similarity Measures

3.1 Evolution Programs

Evolution programs and genetic algorithms, respectively, are search algorithms based on the mechanics of natural selection, natural genetics, and the principle of the "survival of the fittest". In every generation a new set of artificial creatures (individuals) is created using pieces (genes) of the genome of individuals of the previous generation (crossover). Occasionally, mutations are introduced, sometimes leading to improved fitness. For the foundations of and more details on EPs and GAs the reader is referred to [8, 5].

For the following reasons we decided to employ an evolutionary program for the purpose of optimising local similarity measures:

– Evolutionary strategies have proved to provide powerful and robust mechanisms for the search for an optimum in complex search spaces.
– Optimisation techniques that try to minimise an error function's value with the help of its derivation, such as gradient descent methods, are difficult to apply here. Since local similarity measures depict complex entities that are characterised by several parameters, it is not feasible to find an adequate error function that can be derived with respect to these parameters.
– Local similarity measures can adequately be represented as individuals 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 in incorporating constraints on the solution that a single individual stands for. Moreover, the calculation of the similarity between a query and a case (as depicted in Section 2.1) is mainly based on real-valued numbers, which can only be extracted from bit strings in a round about way. For these reasons we developed an evolutionary algorithm that makes use of "richer" data structures and applies appropriate genetic operators, while still being similar to a standard genetic algorithm.

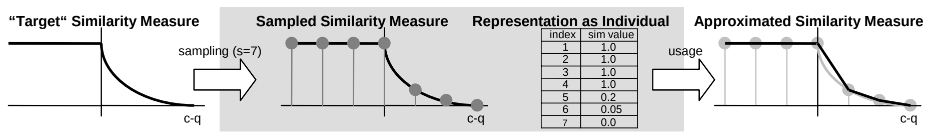
3.2 Representation of Individuals

In order to be able to learn local similarity measures as defined in Definition 1 and 2, i.e. similarity functions and tables, we have to settle on how to represent the respective individual. Further, we need a formalism that maps those individuals, as used in the evolutionary algorithm, to a local similarity measure.

Assume an arbitrary similarity function sim_a according to Definition 1. 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 based on a number of sampling points:

Definition 3 (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. Hence, it holds for all $i \in \{1, \dots, s\}$: $v_i^I = (V_a^I)_i \in [0, 1]$.*

The sampling points are distributed equidistantly over the value range D_a of attribute a. Figure 3 illustrates how a local similarity measure for a numeric attribute is modelled. The length s of the similarity vector may be chosen due


"Target" Similarity Measure
sampling (s=7)
Sampled Similarity Measure
Representation as Individual
usage
Approximated Similarity Measure
c-q
c-q
c-q

index	sim value
1	1.0
2	1.0
3	1.0
4	1.0
5	0.2
6	0.05
7	0.0

Fig. 3. Representation of Similarity Vectors as Individuals

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 $[0, 1]$. The definition following is in the main no different to Definition 2, however, we need it due to the notation it introduces:

Definition 4 (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 $i, j \in \{1, \ldots, n\}$.*

3.3 Specialised Genetic Operators

Genetic operators are responsible for the creation of new individuals and have a major influence on the way a population develops. The operators we use 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 both kinds of local similarity measures we are dealing with. They change one or more values of a similarity vector V_a^I or matrix M_a^I according to the respective mutation rule. Doing so, the constraint that every new value has to lie within the interval $[0, 1]$ is met. The second constraint that needs to be attended concerns the reflexivity of local similarity measures. The similarity between a query q and a case c should always be 1.0, if it holds $q = c$. As a consequence, the medial sampling point of a similarity vector should be 1.0 as well as the elements m_{ii}^I of a similarity matrix for all $i \in \{1, \ldots, n\}$. Since any matrix can be understood as a vector, we describe the functionality of our mutation operators for similarity vectors only:

- *Simple mutation*: If $V_a^I = (v_1^I, \ldots, v_s^I)$ is a similarity vector individual, then each element v_i^I has the same 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 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 $[0, 1]$. 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.

– *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 CPU-Clock (see Section 2.1) 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 left 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, 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. 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 $i, j \in \{1, \ldots, d\}$.
– *Row/column crossover* is employed for similarity tables, i.e. for symbolic attributes, only. Rows and columns in a similarity matrix contain coherent information, since their similarity entries refer to the same query or case value, respectively. Therefore, cutting a row/column by simple or arbitrary crossover may lead to less valuable rows/columns for the offspring individual. We define row crossover as follows: For each row $i \in \{1, \ldots, n\}$ we randomly determine individual I_A or I_B to be the individual for that row. Then it holds $m_{ij}^{I_{new}} = m_{ij}^{I_P}$ for all $j \in \{1, \ldots, n\}$. Column crossover is defined accordingly.

3.4 Controlling the Learning Procedure

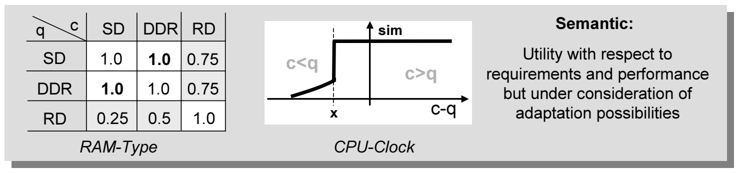
Given a particular attribute a and the mandatory training data (case order feedback, see 2.2), our algorithm settles how to represent local similarity measures for that type of individuals. It then proceeds through evolutionary techniques (each of the genetic operators described above is employed with a specified probability), creates new similarity measures while abolishing old ones, in order to search for the *fittest individual*, whose corresponding local similarity measure yields the minimal value of the error function for the training data. The control structure of our implementation is similar to a standard genetic algorithm [5] and proceeds at a high level as follows:

1. Generate an initial population P_a of random local similarity measures for the respective attribute.
2. Evaluate each $I \in P_a$ by assigning a fitness value and a lifetime value. For the computation of a local similarity measure's fitness we use a slight modification of the index error (see [11]), a distance measure that regards the number of realignments that are necessary to make the retrieval results (for a specific set of queries), to which that measure leads, match with the case order feedback given by the similarity teacher. Of course individuals with smaller fitness values are considered to be fitter, since the mentioned error function has to be minimised. Moreover, we assign a lifetime value to each individual that corresponds to its fitness value, telling for how many generations the individual is maximally allowed to remain in the population.
3. Randomly select mating partner from P.
4. Create a set P_o of new individuals by applying crossover operators and reproducing mutation to the selected mating partners.
5. Mutate the individuals in P_o (adaptive mutation).
6. Increase age for each $I \in P_a$, evaluate each $I \in P_o$, and form $P_a := P_a \cup P_o$.
7. Remove dead (lifetime below 0) and the most unfit individuals from P_a.
8. Repeat step 3 and following.

The algorithm is also capable of learning several local similarity measures simultaneously. Given a set of attributes $\{a_1, \ldots, a_m\}$, for which local similarity measures are to be learnt, a population P_{a_i} is generated for each attribute a_i. Thus, it is possible to optimise the local similarity measures for all attributes of the case representation simultaneously.

4 Experimental Evaluation

To evaluate the presented learning algorithm we have chosen a similar experiment as already described in [11]. The idea of this experiment is to learn a similarity measure that considers the provided adaptation possibilities during the retrieval of useful cases. When providing adaptation functionality, it is generally insufficient to determine how well a case matches the current problem situation in order to obtain an accurate approximation of the utility of cases. Instead, the similarity measure has also to pay attention to the *adaptability of cases* [10, 7]. For example, consider again a product recommendation system for the PC domain. In Figure 1 we have already shown local similarity measures that might be used to estimate the utility of PCs with respect to the customers' requirements. However, these measures have to be modified if the used CBR system also provides possibilities to adapt retrieved PCs. Then, the measures shown in Figure 4 might be more accurate. Here, the similarity between SD-RAM and DDR-RAM (bold) is always considered to be 1.0 due the compatibility of these RAM types. The similarities concerning the RD-RAM (grey shadowed) are not changed because this RAM type is incompatible with the other types and therefore adaptation is impossible. Further, the similarity function for the CPU-Clock is also modified. The assumption here is that it is possible to replace slow CPUs against faster ones as long as the difference between the clock rates does not exceed a critical value[2] x.



q \ c	SD	DDR	RD
SD	1.0	**1.0**	0.75
DDR	**1.0**	1.0	0.75
RD	0.25	0.5	1.0

RAM-Type

sim
c<q
c>q
x
c-q
CPU-Clock
Semantic:
Utility with respect to requirements and performance but under consideration of adaptation possibilities

Fig. 4. Considering Adaptability in Local Similarity Measures

4.1 The Experiment

The case model of our PC domain consists of 11 attributes $a_1, \ldots, a_{11}$ (5 numeric and 6 symbolic attributes) that describe the technical properties of a PC. For each attribute we defined a local similarity measure used to estimate the utility of particular PCs for a given query. However, these similarity measures do not consider the customisation possibilities provided by a set of adaptation rules.

To obtain the required training data S we have generated case order feedback by creating 600 queries randomly, performing adaptation of all retrieved cases (in our experiment 20) and reordering the (adapted) cases by measuring their utility with help of the initially defined similarity measure. For our experiments (see Figure 5) we divided S at the ratio of $1 : 2$ into a subset S_{train} of training examples and S_{test} of test examples used for evaluation. Prior to the utilisation of our evolutionary algorithm we learnt the 11 attributes' feature weights w_i with the help of the approach introduced in [11]. When learning the involved local measures sim_{a_i}, we make use of these optimised feature weights and do not modify them any further. Our learning algorithm creates and manages a population of appropriate individuals for each attribute (5 populations of similarity vectors and 6 of similarity matrices). In a kind of round-robin optimisation the delineated evolutionary techniques are applied to these 11 populations and thus the PC domain's local similarity measures are optimised simultaneously.

Evaluating the learning results, we determine a new retrieval result (based on the learnt similarity measure) for each query contained in the training examples

[2] Motherboards usually only support a limited CPU clock range.

of S_{test}. Regarding the quality of these case orders we revert to the quality measure $CR1_i$ and $CR3_i$ as defined in [12]. These measures express the percentage (in relation to $|S_{test}|$) of those retrievals that return the optimal case as most similar one and, respectively, among the 3 most similar ones. The optimal case for a specific query q is determined by S_{test} and denotes that case which yields the highest utility for q after having been adapted.

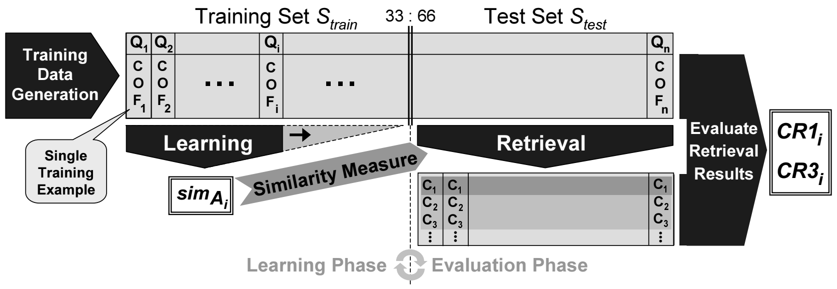


Fig. 5. The Experiment

To achieve an understanding of the amount of training data needed we have repeated the procedure of similarity measure optimisation for incrementally increasing subsets of S_{train}. Further, we have focused on our learning algorithm's convergence behaviour and analysed the quality measures' development by time (by number of evolutionary generations). In order to obtain statistically significant results we have repeated the described learning process 10 times, using newly generated training and test data, and calculated the worst, average, and best values for the mentioned quality measures.

4.2 Results

Figure 6a) shows the gradual optimisation of the global similarity calculation due to the number of evolutionary generations, making use of all 200 training examples from S_{train} (evaluated on S_{test}). Here, in the left part of the chart the initial improvement resulting from feature weight learning is to be seen. The right part illustrates the further improvement of the quality measures as yielded by the optimisation of local similarity measures. After about 300 generations the evolutionary algorithm converges to an optimum, on average enhancing the value of quality measure $CR1_{200}$ from 36.3% to 45.6%, and $CR3_{200}$ from 66.3% to 75.3%[3]. Figure 6b) illustrates the influence of the size of the training set S_{train} on the quality of the learning results. Here, at each case the worst, average, and

[3] Note, that a single run of the optimisation process in the presented application domain for $|S_{train}| = 200$ and for 200 evolutionary generations takes about 7 hours on a P-IV machine with 1.8 GHz.

best value (reached after 300 generations of the evolutionary algorithm) of $CR1_i$ and $CR3_i$ are shown.

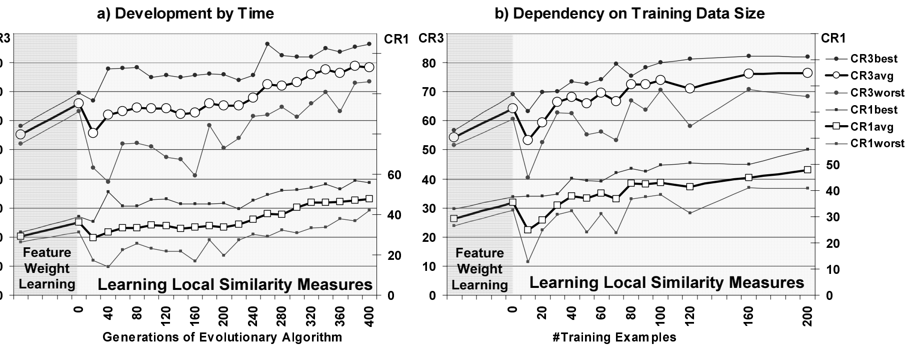


Fig. 6. Evaluation Results

The results presented here indicate the fundamental capabilities of our approach to learn local similarity measures with the help of an evolutionary algorithm. In a domain where case adaptation is applied, our algorithm clearly improves the quality of a similarity measure so that it takes the possibilities of case adaptation into consideration. With respect to the required amount of training data, we expected to need more training examples compared with the learning of attribute weights to achieve reasonable results due to the much more complex search space. And indeed, in our experiment about 100 training examples are required to ensure stable learning results. Using smaller training data sets, the algorithm sometimes tends to over-fit the presented data leading to poor results on the independent test data. Nevertheless, assumed that enough training data is available, the experimental results show that our approach is able to improve the retrieval quality clearly. At least in the presented application scenario the amount of training data is not a crucial problem because it is generated automatically and so the risk of over-fitting is marginally low.

When omitting prior feature weight optimisation and starting the local optimisation for the fixed weights of the given utility measure, the learnt local similarity measures result in an even more noticeable improvement of $CR1$ and $CR3$. The reason is that it is obviously possible to simulate the effect of feature weights partially with accurate local similarity measures.

5 Related Work

When talking about learning in CBR, traditionally the focus lays on the acquisition of new case knowledge. Concerning the learning of similarity measures, existing approaches are restricted to learning attribute weights. Several techniques have been developed to improve the accuracy of case-based classification systems [13, 2, 14, 9, 15]. However, these techniques are difficult to apply in domains without classified case data, like e-commerce and general knowledge management domains. An alternative approach to learn customer preferences in e-commerce is presented in [3]. The training data used here, called *return-set selections*, is comparable with the case order feedback presumed by our approach. However, it also focuses only on attribute weights to represent preferences of customers.

Another research field that deals with the retrieval of information that may have different utility for the user is Information Retrieval (IR). To ensure the retrieval of documents that have a high utility for the particular user, some systems acquire *relevance feedback* about the retrieved documents from the user. This feedback is used to learn the *actual user context* represented by the user's query extended with additional key terms to obtain a more specific query [4].

Work that applies a genetic algorithm to learn the retrieval index and feature weights in a design domain, namely tablet formulation, is presented in [6]. Here, the learning is driven through the available case data by defining a fitness function that measures the retrieval quality of a special leave-n-out test.

6 Conclusion and Outlook

In this paper we have presented an approach that can be seen as a first step towards the learning of knowledge-intensive local similarity measures. Such measures are commonly used in many current application domains, in particular when employing powerful commercial CBR shells that provide graphical modelling tools. The advantage of our learning approach is that it avoids some major problems that arise when defining similarity measures manually. On the one hand, it may simplify the definition process clearly, on the other hand, it may also help to define similarity measures that represent a better approximation of the underlying utility function. In our point of view, the major strength of the learning approach is its more goal directed fashion compared with the manual procedure. Instead of tuning numerous single representation elements of the entire similarity measure, like weights and local similarity measures, the similarity definition is directly based on the expected outcome of the similarity computation, namely the utility of cases for particular problem situations. However, like other machine learning techniques, the success of the approach strongly depends on the available training data. Only if acquiring the required training data requires less effort than a manual definition of the similarity measure, the learning approach might be useful. A typical example are domains that require case adaptation. Here, the training data might be generated automatically like described in Section 4. However, in [11] we have already discussed other domains and application scenarios where our learning framework might be applied successfully.

An interesting issue for future research is the consideration of additional, easy to acquire background knowledge during the learning process (see also [11]). This might decrease the risk of over-fitting small training data sets and therefore could also decrease the amount of training data required to obtain accurate learning results. Further, additional experiments with real-world application domains are necessary to show the general applicability of our approach.

References

1. R. Bergmann, M. M. Richter, S. Schmitt, A. Stahl, and I. Vollrath. Utility-oriented matching: A new research direction for Case-Based Reasoning. In *Professionelles Wissensmanagement: Erfahrungen und Visionen. Proceedings of the 1st Conference on Professional Knowledge Management*. Shaker, 2001.
2. 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 2nd International Conference on Case-Based Reasoning*. Springer, 1997.
3. K. Branting. Acquiring customer preferences from return-set selections. In *Proceedings of the 4th International Conference on Case-Based Reasoning*. Springer, 2001.
4. A. Göker. Capturing information need by learning user context. In *Working Notes of the Workshop on Learning about Users, 16th International Joint Conference in Artificial Intelligence*, 1999.
5. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
6. J. Jarmulak, S. Craw, and R. Rowe. Genetic algorithms to optimise CBR retrieval. In *Proceedings of the 5th European Workshop on Case-Based Reasoning*. Springer, 2000.
7. D. Leake, A. Kinley, and D. Wilson. Linking adaptation and similarity learning. In *Proceedings of the 18th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum, 1996.
8. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.
9. F. Ricci and P. Avesani. Learning a local similarity metric for case-based reasoning. In *Proceeding of the 1st International Conference on Case-Based Reasoning*, pages 301–312. Springer, 1995.
10. B. Smyth and M. T. Keane. Retrieving adaptable cases: The role of adaptation knowledge in case retrieval. In *Proceedings of the 1st European Workshop on Case-Based Reasoning*. Springer, 1993.
11. A. Stahl. Learning feature weights from case order feedback. In *Proceedings of the 4th International Conference on Case-Based Reasoning*. Springer, 2001.
12. A. Stahl. Defining similarity measures: Top-down vs. bottom-up. In *Proceedings of the 6th European Conference on Case-Based Reasoning*. Springer, 2002.
13. D. Wettschereck and D. W. Aha. Weighting features. In *Proceeding of the 1st International Conference on Case-Based Reasoning*. Springer, 1995.
14. W. Wilke and R. Bergmann. Considering decision cost during learning of feature weights. In *Proceedings of the 3rd European Workshop on Case-Based Reasoning*. Springer, 1996.
15. Z. Zhang and Q. Yang. Dynamic refiniement of feature weights using quantitative introspective learning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1999.