Approximation of Utility Functions by Learning Similarity Measures

Armin Stahl

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

Abstract. Expert systems are often considered to be logical systems producing outputs that can only be correct or incorrect. However, in many application domains results cannot simply be distinguished in this restrictive form. Instead to classify a result as correct or incorrect, here results might be more or less *useful* for solving a given problem or for satisfying given user demands, respectively. In such a situation, an expert system should be able to estimate the *utility* of possible outputs a-priori in order to produce reasonable results. In Case-Based Reasoning this is done by using similarity measures which can be seen as an approximation of the domain specific, but a-priori unknown *utility function*. In this article we present an approach how this approximation of utility functions can be facilitated by employing machine learning techniques.

1 Introduction

When developing knowledge-based systems one must be aware of the fact that the output of such a system often cannot simply be judged as correct or incorrect. Instead, the output may be more or less *useful* for solving a given problem or for satisfying the users' demands. Of course, one is interested in maximizing the *utility* of the output even if the underlying *utility function* is (partially) unknown. Therefore, one must at least be able to estimate the a-priori unknown utility of possible outputs, for example, by employing heuristics.

Typical examples for knowledge-based systems where we have to deal with unknown utility functions are *Information Retrieval (IR)* [12] and *Case-Based Reasoning (CBR)* [1, 9] systems. In IR systems one is interested in finding textual documents that contain information that is useful for satisfying the information needs of the users. Since CBR systems are used for various application tasks, here, we have to distinguish different situations. On the one hand, for traditional application fields like classification, the external output of a CBR system — here the predicted class membership of some entity — usually can only be correct or incorrect, of course. However, due to the problem solving paradigm of CBR, internally a case-based classification system relies on an appropriate estimation

of the utility of *cases*[1] which are used to infer the class prediction. On the other hand, in more recently addressed application fields, also the external outputs of CBR systems underlie utility functions. A typical example are product recommendation systems used in e-Commerce [4, 6, 19]. Here, CBR systems are used to select products (or services) — represented as cases — that fulfill the demands and wishes of customers as good as possible. Hence, a recommended product does not represent a correct or incorrect solution, but a more or less suitable alternative.

In CBR the utility of cases is estimated according the following heuristics: *"Similar problems have similar solutions"*. Here, the assumption is that a case consists of a problem description and the description of a corresponding, already known solution. When being confronted with a new problem for which a solution is required, the description of this problem — also called *query* — has to be compared with the problem descriptions contained in available cases. If two problems are *similar* enough, the probability that also similar solutions can be applied to both problems should be high. This means, the similarity between a given problem situation and the problem described in a case can be seen as a heuristics to estimate the utility of the corresponding known solution for solving the current problem (see Figure 1). If the found solution cannot directly be reused to solve the given problem, it has to be *adapted* so that it fits the changed requirements.

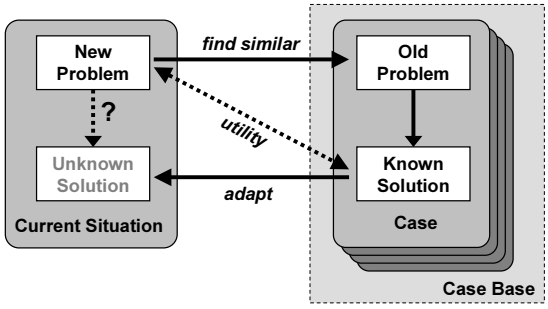

Fig. 1. Approximating Utility through Similarity in CBR

In order to be able to calculate the similarity of two problem descriptions, so-called *similarity measures* are employed. Basically, a similarity measure can be characterized as an approximation of an a-priori unknown utility function [3]. The quality of this approximation strongly depends on the amount of domain specific knowledge one is able to encode into the similarity measure. However, a manual definition of *knowledge-intensive similarity measures* leads to the well known *knowledge acquisition bottleneck* when developing knowledge-based sys-

[1] In CBR cases typically represent experiences about already solved problems or other information that can be reused to solve a given problem.

tems. In this article we show that machine learning approaches can be applied to learn similarity measures from a special kind of (user) feedback, leading to better approximations of the underlying utility functions. A more detailed description of the presented approach is given by [17].

In Section 2 we first introduce the foundations of similarity measures in general and we show how they are represented in practice. In Section 3 we present our approach to learning similarity measures which is based on a special kind of feedback and corresponding learning algorithms. To demonstrate the capabilities of our approach, in Section 5 we describe some evaluation experiments and discuss the corresponding results. Finally, we close with a short summary and conclusion.

2 Similarity Measures

Basically, the task of a similarity measure is to compare two cases[2], or at least particular parts of two cases, and to compute a numeric value which represents the degree of similarity. In traditional CBR systems the cases' parts to be compared are usually descriptions of problem situations, however, in more recent application domains a clear distinction between problems and solutions is often not given. Hence, in the following we assume that a similarity measure compares two *case characterizations* describing the aspects that are relevant to decide whether a case might be useful for a given query or not:

Definition 1 (Similarity Measure, General Definition). *A **similarity measure** is a function* $Sim : \mathbb{D} \times \mathbb{D} \longrightarrow [0, 1]$*, where* $\mathbb{D}$ *denotes the space of case characterizations.*

Depending on the application tasks, case characterizations might be represented by using very different formalisms. Of course, the used formalism strongly influences the manner how corresponding similarity measures have to be represented. Therefore, we first introduce the kind of case representation that we presume in the following.

2.1 Attribute-Value Based Case Representation

In most application domains *attribute-value based case representations* are sufficient to represent all information required to reuse cases efficiently. Such a representation consists of a set of attributes $A_1, A_2, \ldots, A_n$ where each attribute A_i is a tuple (A_{name}, A_{range}). Here, A_{name} represents the name of the attribute and A_{range} defines the set of valid values that can be assigned to that attribute. Depending on the value type (numeric or symbolic) of the attribute, A_{range} may be defined in form of an interval $[v_{min}, v_{max}]$ or in form of an enumeration $\{v_1, v_2, \ldots, v_s\}$. In principle the set of attributes might be divided into two subsets, one for the case characterization and one for the *lesson* part,

[2] A query to a CBR system can also be seen as a (partially known) case.

which describes the known solution in traditional application tasks. Further, the set of all attribute definitions, i.e. the names and the ranges, is called the *case model*. A case c is then a vector of attribute values $(a_1, a_2, \ldots, a_n)$ with $a_i \in A_i.A_{range} \cup \{undefined\}$.

An illustration of the described representation formalism is shown in Figure 2. Here, cases represent descriptions of personal computers, for example, to be used in a product recommendation system. The lesson part plays only a minor role, since a clear distinction between problems and solutions is not given in this application scenario.

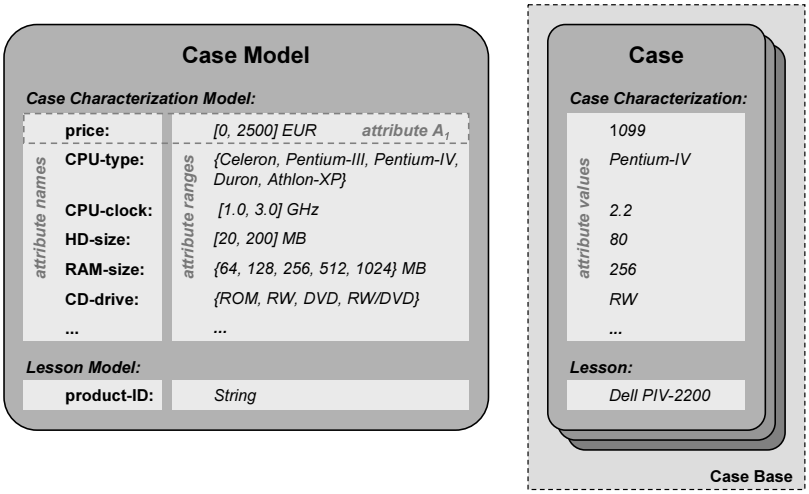

Fig. 2. Example: Attribute-Value Based Case Representation

2.2 Foundations

Although a similarity measure computes a numerical value (cf. Definition 1), usually one is not really interested in these absolute similarity values, because their interpretation is often quite difficult. One is rather interested in the partial order these values induce on the set of cases. Such a partial order can be seen as a preference relation, i.e. cases that are considered to be more similar to the query are preferred to being reused during the further processing steps.

Definition 2 (Preference Relation Induced by Similarity Measure). *Given a case characterisation $d \in \mathbb{D}$, a similarity measure Sim induces a* ***preference relation*** $\preceq_d^{Sim}$ *on the case space $\mathbb{C}$ by $c_i \preceq_d^{Sim} c_j$ iff $Sim(d, c_i) \leq Sim(d, c_j)$.*

In general, we do not assume that similarity measures necessarily have to fulfill general properties beyond Definition 1. Nevertheless, in the following we introduce two basic properties that are often fulfilled.

Definition 3 (Reflexivity). *A similarity measure is called* ***reflexive*** *if* $Sim(x, x) = 1$ *holds for all* x*. If it holds additionally* $Sim(x, y) = 1 \rightarrow x = y$*,* Sim *is called* ***strong reflexive****.*

On the one hand, reflexivity is a very common property of similarity measures. It states that a case characterisation is maximal similar to itself. From the utility point of view, this means, a case is maximal useful with respect to its own case characterisation. On the other hand, similarity measures are usually not strong reflexive, i.e. different cases may be maximal useful regarding a given query. For example, different solution alternatives contained in different cases might be equally accurate to solve a given problem.

Definition 4 (Symmetry). *A similarity measure is called* ***symmetric****, if it holds* $Sim(x, y) = Sim(y, x)$ *for all* x, y*. Otherwise it is called* ***asymmetric****.*

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

2.3 The Local-Global Principle

Definition 1 still does not define how to represent a similarity measure in practice. To reduce the complexity, here one usually applies the so-called *local-global principle*. According to this principle it is possible to decompose the entire similarity computation in a local part only considering *local similarities* between single attribute values, and a global part that computes the *global similarity* for whole cases based on the local similarity assessments. Such a decomposition simplifies the modelling of similarity measures significantly and allows to define well-structured measures even for very complex case representations consisting of numerous attributes with different value types.

This approach requires the definition of a case representation consisting of attributes that are independent from each other with respect to the utility judgements. In the case that the utility depends on relations between attributes one has to introduce additional attributes — so-called *virtual attributes* — making these relations explicit. Consider the example that we want to decide whether a given rectangle is a quadrat or not by applying case-based reasoning. Here, obviously the ratio between the length and the width of the rectangle is crucial. Hence, if the original cases are described by these two attributes, an additional

attribute "length-width-ratio" has to be introduced which represents this important relationship between the two attributes.

Given such a case representation, a similarity measure can be represented by the following elements:

1. *Attribute weights* define the importance of each attribute with respect to the similarity judgement,
2. *Local similarity measures.* calculate local similarity values for single attributes.
3. An *aggregation function* calculates the *global similarity* based on the attribute weights and the computed local similarity values.

With respect to the aggregation function mostly a simple weighted sum is sufficient. This leads to the following formula for calculating the global similarity Sim between a query q and a case c:

$$Sim(q, c) := \sum_{i=1}^{n} w_i \cdot sim_i(q_i, c_i)$$

Here, sim_i represents the local similarity measure for attribute a_i and q_i,c_i are the corresponding attribute values of the query and the case. In the following, we describe how local similarity measures can be represented.

2.4 Local Similarity Measures

In general, the representation of a local similarity measure strongly depends on the value type of the underlying attribute. Basically, we can distinguish between similarity measures for unordered and ordered data types. The former ones are typical for symbolic attributes while the latter ones are typical for numeric attributes. Nevertheless, also symbolic values may be associated with an order.

Similarity Tables. When dealing with unordered data types, the only feasible approach to represent local similarity measures is an explicit enumeration of all similarity values for each possible value combination. The result is a *similarity table* as illustrated in Figure 3 for the attribute 'casing" in the PC domain.

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

Fig. 3. Similarity Table

Since similarity measures are usually reflexive (cf. Definition 3), the values of the main diagonal in such a table are set to 1. When dealing with a symmetric

similarity measure the upper and the lower triangular matrices are symmetric, which reduces the modelling effort. The shown table represents an asymmetric measure. For example, the similarity between the casing types "mini-tower" and "midi-tower" is different depending on which value represents the query and which the case value.

Difference-Based Similarity Functions. For ordered types which are often also even infinite, similarity tables are not feasible, of course. Here, similarity values can be calculated based on the difference between the two values to be compared. This can be realized with *difference-based similarity functions* as illustrated in Figure 4. Here, the x-axis represents the difference between the query and the case value. For numeric types this difference can be calculated directly, for example with $\delta(q, c) = c - q$. For other ordered non-numeric types (e.g., ordered symbolic types), the distance may be inferred from the position of the values within the underlying order, i.e. one might assign an integer value to each symbolic value according its index. A difference-based similarity function then assigns every possible distance value a corresponding similarity value by considering the domain specific requirements. The function shown in Figure 4, for example, might represent the local utility function for an attribute like "price" typically occurring in product recommendation systems. The semantics of this function is that lower prices than the demanded price are acceptable and therefore lead to a similarity of 1. On the other hand, larger prices reduce the utility of a product and therefore lead to decreased similarities.

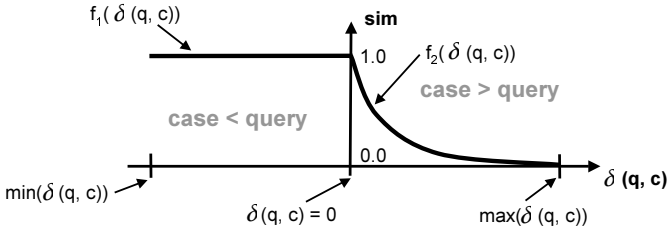

Fig. 4. Difference-Based Similarity Functions

2.5 Defining Similarity Measures

Although today available CBR tools provide comfortable graphical user interfaces for defining similarity measures, modelling similarity measures manually leads to some problems.

The similarity definition process commonly applied nowadays can be characterized as a *bottom-up* procedure. This means, the entire similarity assessment is based on the acquisition of numerous single knowledge entities about the influences on the utility function. These knowledge-entities have to be encoded

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

- The procedure is very time-consuming. For example, consider a symbolic attribute with 10 allowed values. This will require the definition of a similarity table with 100 entries!
- In some application domains a sufficient amount of the described low-level knowledge might be not available. Possible reasons are, for example, a poorly understood domain, or the fact that an experienced domain expert who could provide the knowledge is not available or too expensive.
- Even if an experienced domain expert is available, s/he is usually not familiar with the similarity representation formalisms of the CBR system. So, the provided knowledge may only be available in natural language. This informal knowledge then has to be translated into the formal representation formalisms by an experienced knowledge engineer who possesses the required skills which leads to additional costs.
- Due to the effort of the representation, even experienced knowledge engineers often make definition failures by mistake. Unfortunately, the recognition of such failures is very difficult.
- The bottom-up procedure does not consider the utility of whole cases directly. Instead, the final utility estimation is completely based on the ensemble of the individual low-level knowledge entities. Nowadays, the overall quality of the completely defined similarity measure is mostly 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. So, one often blindly trusts the correctness of the global similarity values computed by the defined measure.
- Due to the complexity of the bottom-up procedure its application is usually restricted to the development phase of the CBR application. This means, all similarity knowledge is acquired during the development phase and is assumed to be valid during the entire lifetime of the application. However, in many domains changing requirements and/or changing environments require not only maintenance of case knowledge, but also maintenance of general knowledge [13].
- The knowledge about the actual utility of cases might not be available at all during the development phase. For example, when applying similarity measures in an e-Commerce or knowledge management scenario, the knowledge

often can only be provided by the users themselves during the usage of the system. However, here the bottom-up procedure is not feasible.
- Sometimes the required knowledge about the cases' utility might already be available in a formal but quite different representation form. For example, when supporting case adaptation, the utility of cases strongly depends on the provided adaptation possibilities. Hence, to obtain an accurate similarity measure one has to transfer adaptation knowledge into the similarity measure. When using the bottom-up procedure this is a very time-consuming task. The adaptation knowledge has to be analysed manually and the knowledge considered to be relevant then has to be encoded into the similarity measure.

In the following we present an alternative approach for modelling similarity measures which tries to avoid the mentioned problems by applying machine learning techniques.

3 Learning Similarity Measures From Utility Feedback

As described in the previous section, one problem of the manual definition of similarity measures is the necessity to analyse the underlying utility functions in detail in order to determine the influences on them. Only if the different influences are known, one is able to consider them in form of appropriate weights and local similarity measures.

3.1 Utility Feedback

We have proposed an alternative approach to acquire knowledge about the only partially or informally known utility function [15, 17]. The basic idea of this approach is the capability of some *similarity teacher* to give feedback about the utility of given cases with respect to concrete problem situations or queries, respectively. Here, the similarity teacher must not be able to explain the reasons why cases are more or less useful, but he has only to compare cases according their utility. This means, the utility of a case must not be expressed absolutely, but only relatively to other cases, for example, by giving statements like "case c_2 is more useful than case c_1". The similarity teacher first might analyze the result of a similarity-based retrieval, i.e. a given partial order of cases (see Figure 5). By reordering the cases according to their actual utility for a given query one obtains an additional partial order which can be characterized as a corrected retrieval result, also called *training example*.

In principle, such *utility feedback* might be provided by different types of similarity teachers depending on the concrete application scenario:

Human Domain Expert: The most obvious possibility is a *human domain expert* who posses implicit knowledge about the unknown domain specific utility function to be approximated. Due to his/her experiences, a domain expert should be able to decide which cases are more useful than others for a given problem situation.

System Users: In application scenarios where the utility of cases strongly depends on the preferences and expectations of the system's users (e.g. e-Commerce applications), also the *users* might play the role of the similarity teacher.

Software Agents: Generally, the similarity teacher has not necessarily be represented by a human being. In some application scenarios also *software agents* which are able to evaluate retrieval results automatically might be applied.

Application Environment: If the output of a CBR system is directly applied in some *application environment*, also feedback about the cases' application success or failure might lead to the required utility feedback.

In the following we assume the existence of some arbitrary similarity teacher who is able to provide utility feedback containing implicit knowledge about the unknown utility function. The objective of our approach is to extract this knowledge and to encode it in an explicit form by defining appropriate similarity measures. This can be achieved by applying machine learning techniques as described in the following.

3.2 Evaluating Similarity Measures

The foundation of our learning approach is the definition of a special *error function* E which compares retrieval results computed according to a given similarity measure with utility feedback provided by the similarity teacher (see Figure 5). This means, the basic element of the error function E has to be a measure for the distinction between two partial orders, namely a retrieval result and utility feedback with respect to a given query q. The requirement on this measure is that it should compute an error value of zero, if and only if the two partial orders are equal. Otherwise it should compute an error value greater zero representing the degree of distinction. Such an error function then can be used to evaluate the quality of a given similarity measure, since it is our goal to find a similarity measure which produces partial orders as defined by the similarity teacher. This means if we are able to find a similarity measure leading to an error value of zero, we have found an *ideal similarity measure* with respect to the given utility feedback.

In the following we introduce a possible definition of such an error function E. For better understanding, here, we introduce a simplified version of E. A more sophisticated one can be found in [17]. Before being able to compare entire partial orders, we need the following function:

Definition 5 (Elementary Feedback Function). *Let q be a query, c_i and c_j be two cases, and let Sim be a similarity measure. The function ef defined as*

$$ef(Sim, (c_i, c_j)) := \begin{cases} 1 & if \;\; sgn(u(q, c_i) - u(q, c_j)) \neq sgn(Sim(q, c_i) - Sim(q, c_j)) \\ 0 & otherwise \end{cases}$$

*is called **elementary feedback function** where u represents the informal utility feedback provided by some arbitrary similarity teacher.*

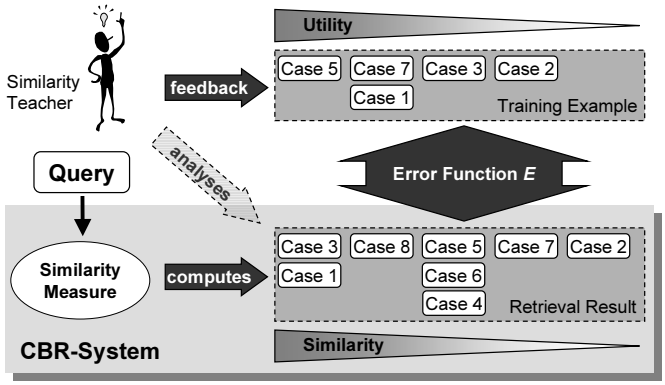

Fig. 5. Utility Feedback

The objective of this elementary feedback function is the evaluation of a given similarity measure Sim regarding the correct ranking for a particular case pair (c_i, c_j). This function can now be used to define the mentioned measure for evaluating retrieval results:

Definition 6 (Index Error). *Consider a similarity measure Sim, a query q and utility feedback for a set of cases $\{c_1, \ldots, c_n\}$ with respect to q, also called **training example** $TE(q)$. We define the **index error** induced by Sim w.r.t. to $TE(q)$ as*

$$E_I(TE(q), Sim) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} ef(Sim, (c_i, c_j))$$

.

The index error can be seen as a measure for the quality of a given similarity measure regarding a particular query and the corresponding training example. However, we are interested in similarity measures that supply reasonable case rankings for arbitrary queries or at least for a certain set of queries. This leads to the following extension of the index error allowing the evaluation of similarity measures with respect to a set of training examples:

Definition 7 (Average Index Error). *Consider a set of training queries $Q = \{q_1, q_2, \ldots, q_m\}$, corresponding training data $TD_u = \{TE(q_1), TE(q_2), \ldots, TE(q_m)\}$ consisting of m training examples, and a similarity measure Sim. We define the **average index error** induced by Sim w.r.t. to Q as*

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

3.3 The Learning Task

With the previously introduced error function we are now able to implement a procedure for learning similarity measures from utility feedback. This learning

procedure can also be characterized as an optimization process controlled by the error function. In principle, we are interested in finding an *optimal similarity measure* leading to a minimal error value, i.e. we want to find a global minimum of the error function $\hat{E}_I$ (see Figure 6). Unfortunately, in general it cannot be guaranteed that we are able to find actually a global minimum, nevertheless we are interested to minimize the error value as far as possible. When starting with some initial similarity measure $Sim_{initial}$ coupled with a corresponding error value $e_{initial}$, it should at least be possible to find a measure coupled with an error value smaller than $e_{initial}$, e.g. Sim_{supopt}. This measure then hopefully represents a better approximation of the unknown utility function.

It must be pointed out that the search space, i.e. the set of representable similarity measures, usually does not contain an ideal similarity measure. Hence, even an optimal similarity measure is mostly also coupled with an error value greater zero.

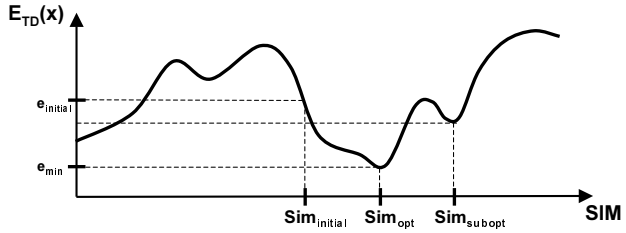

Fig. 6. Finding Minima of the Error Function

For implementing the described learning or optimization task, respectively, different methods, for example, gradient descent approaches, simulated annealing [2] or evolutionary algorithms [8, 10], have already been developed. In the following section we present an approach particularly suited to learn local similarity measures that is based on evolutionary programs [10]. In contrast to traditional genetic algorithms where the entities to be optimized are encoded by using bit strings, our evolutionary algorithm operates on more sophisticated representations. We restrict the description of our approach on an overview of the most important aspects, namely the representation of individuals and the definition of appropriate genetic operators. For more details about the functionality of our learning algorithm we refer to [18, 17].

4 A Genetic Algorithm for Learning Local Similarity Measures

When employing a genetic algorithm, the most important issues are the definition of an appropriate *fitness function*, an adequate representation of the entities to be optimized and the determination of corresponding genetic operators. The

average index error $\hat{E}_I$ introduced in Definition 7 already represents the required fitness function. In this section we show how local similarity measures can be represented so that a genetic algorithm is able to handle them easily. Further, we introduce corresponding genetic operators needed to realize an evolutionary process.

4.1 Representation of Individuals

Concerning the representation of local similarity measures as individuals of an evolutionary process we presume the representation formalisms introduced in Section 2.4, i.e. difference-based similarity functions and similarity tables.

Representing Difference-Based Similarity Functions. Consider some difference-based similarity function Sim_A used as local similarity measure for a numeric attribute A. Since Sim_A may be continuous in its value range $[min(A_{range}) - max(A_{range}), max(A_{range}) - min(A_{range})]$ it is generally difficult to describe it exactly with a fixed set of parameters. Thus, we employ an approximation based on a number of sampling points to describe arbitrary functions:

Definition 8 (Similarity Function Individual, Similarity Vector). *An individual I representing a similarity function Sim_A for the numeric attribute A is coded as a vector V_A^I of fixed size s. The elements of that* ***similarity vector*** *are interpreted as sampling points of Sim_A, between which the similarity function is linearly interpolated. Accordingly, it holds for all $i \in \{1, \ldots, s\}$: $v_i^I = (V_A^I)_i \in [0, 1]$.*

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

Uniform Sampling: The simplest strategy is to distribute sampling points equidistantly over the entire value range (see Figure 7a).

Center-Focused Sampling: However, when analyzing the structure of difference-based similarity functions in more detail, it becomes clear that different inputs of the functions will usually occur with different probabilities. While the maximal and minimal inputs, i.e. the values $d_{min} = (min(A_{range}) - max(A_{range}))$ and $d_{max} = (max(A_{range}) - min(A_{range}))$, can only occur for one combination of query and case values, inputs corresponding to small differences can be generated by various of such combinations. Thus, case data and corresponding training data usually provides much more information about the influences of small differences, compared with the information

available about extreme differences. Another aspect is that changes in similarity are usually more important for small value differences, since greater differences usually correspond to very small similarity values. In order to consider these facts during learning, it might be useful to use more sampling points around the "center" of the difference-based similarity function like illustrated in Figure 7b.

Dynamic Sampling: While the center-focused approach is more a heuristics, it is also possible to analyse the training data in order to determine an optimal distribution for the sampling points [7]. Then, areas where a lot of training information is available might be covered with more sampling points than areas for which no respective information is contained in the training data.

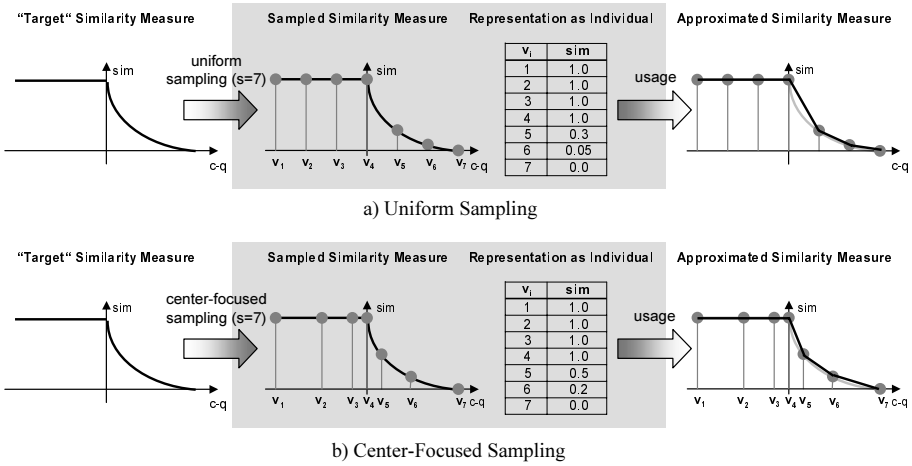

Fig. 7. Representing Similarity Functions as Individuals

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

Definition 9 (Similarity Table Individual, Similarity Matrix). *An individual I representing a similarity table for a symbolic attribute A with a list of allowed values $A_{range} = (d_1, d_2, \ldots, d_n)$ is a $n \times n$-matrix M_A^I with entries $m_{ij}^I = (M_A^I)_{ij} \in [0, 1]$ for all $i, j \in \{1, \ldots, n\}$.*

This definition corresponds to the representation of similarity tables, i.e. the original representation of this type of local similarity measures is directly used by the genetic algorithm. So, the definition presented here is only required for introducing the necessary notation.

4.2 Genetic Operators

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

The operators we use for learning of local similarity measures are different from classical ones since they operate on a different genome representation. However, because of underlying similarities, we divide them also into the two standard groups: mutation and crossover operators.

Crossover Operators for Similarity Vectors and Matrices. Applying crossover operators on the data structures used for representing local similarity measures, 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 [10], we rely on the traditional approach using exactly two parental individuals, I_1 and I_2.

- *Simple crossover* is defined in the traditional way as used for bit string representations: A split point for the particular similarity vector or matrix is chosen. The new individual is assembled by using the first part of parent I_1's similarity vector or matrix and the second part of parent I_2's.
- *Arbitrary crossover* represents a kind of multi-split-point crossover with a random number of split points. Here, for each component of the offspring individual it is decided randomly whether to use the corresponding vector or matrix element from parent I_1 or I_2.
- *Arithmetical crossover* is defined as the linear combination of both parent similarity vectors or matrices. In the case of similarity matrices the offspring is generated according to: $(M_A^{I_{new}})_{ij} = m_{ij}^{I_{new}}$ with $m_{ij}^{I_{new}} = \frac{1}{2}m_{ij}^{I_1} + \frac{1}{2}m_{ij}^{I_2}$ for all $i, j \in \{1, \ldots, d\}$.
- *Line/column crossover* is employed for similarity tables, i.e. for symbolic attributes, only. Lines and columns in a similarity matrix contain coherent information, since their similarity entries refer to the same query or case value, respectively. Therefore, cutting a line/column by simple or arbitrary crossover may lead to less valuable lines/columns for the offspring individual. We define line crossover as follows: For each line $i \in \{1, \ldots, n\}$ we randomly determine individual I_1 or I_2 to be the parent individual I_P for that line. Then it holds $m_{ij}^{I_{new}} = m_{ij}^{I_P}$ for all $j \in \{1, \ldots, n\}$. Column crossover is defined accordingly.

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

Mutation Operators for Similarity Vectors and Matrices. Operators of this class are the same for both kinds of local similarity measures we are dealing with. They change one or more values of a similarity vector V_A^I or matrix M_A^I according to the respective mutation rule. Doing so, the constraint that every new value has to lie within the interval $[0, 1]$ is met. The second constraint that needs to be considered concerns the reflexivity of local similarity measures (cf. Definition 3). As a consequence, the medial sampling point of a similarity vector should be 1.0 as well as the elements m_{ii}^I of a similarity matrix for all $i \in \{1, \ldots, n\}$. Since any matrix can be understood as a vector, we describe the functionality of our mutation operators for similarity vectors only:

- *Simple mutation*: If $V_A^I = (v_1^I, \ldots, v_s^I)$ is a similarity vector individual, then each element v_i^I has the same probability of undergoing a mutation. The result of a single application of this operator is a changed similarity vector $(v_1^I, \ldots, \hat{v}_j^I, \ldots, v_s)$, with $1 \leq j \leq s$ and $\hat{v}_j^I$ chosen randomly from $[0, 1]$.
- *Multivariate non-uniform mutation* applies the simple mutation to several elements of V_a^I. Moreover, the alterations introduced to an element of that vector, become smaller as the age of the population is increasing. The new value for v_j^I is computed 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. The sign $\pm$ indicates, that the alteration is either additive or subtractive. The decision about that is made randomly as well.
- *In-/decreasing mutation* represents a specialisation of the previous operator. Sometimes it is helpful to modify a number of neighbouring sampling points uniformly. The operator for in-/decreasing mutation randomly picks two sampling points v_j^I and v_k^I and increases or decreases the values for all v_i^I with $j \leq i \leq k$ by a fixed increment.

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

With the described representation and genetic operators together with the error function introduced in Definition 7 we are now able to implement a genetic algorithm for learning local similarity measures. For more details about the general functionality of genetic algorithms see [8, 10].

For the other important part of a global similarity measure, namely the attribute weights, it is also possible to define a corresponding genetic algorithm. However, here other learning strategies can usually be applied more efficiently, for example, gradient descent algorithms [15–17].

5 Experimental Evalutation

In this section we give a short summary of an experimental evaluation that demonstrates the capabilities of our learning approach in two different application scenarios. A more detailed description is given by [17]. In both scenarios a

similarity measure is required to approximate an a-priori unknown utility function. However, the two scenarios clearly differ in the aspects that determine the utility of cases.

5.1 Learning Customer Preferences

In our first evaluation scenario we consider a CBR system used to recommend appropriate used cars with respect to a given requirement specification represented by the query. Here, we assume that the cars cannot be customized, i.e. no case adaptation is performed after the similarity-based retrieval. The case representation we used for our experiment consists of 8 attributes (4 symbolic, 4 numeric) describing important properties of the cars, like "price" or "engine power". Concerning the similarity measure, this results in 4 similarity tables, 4 difference-based similarity functions and 8 attribute weights to be optimized by applying our learning approach.

Since no real customers were available, we have applied a simulation approach in order to obtain the required utility feedback. The foundation of this simulation is an additional similarity measure representing virtual preferences of some class of customers, so to speak the target measure to be learnt by the learning algorithm. With this additional similarity measure Sim_U we were able to generate utility feedback like shown in Figure 8. In order to obtain a single training example, the following steps are performed automatically:

1. Generating a random query q.
2. Retrieving the 10 most similar cases with respect to q by using an initial similarity measure Sim_I.
3. Recalculating the similarity of the 10 retrieved cases by using Sim_U.
4. Selecting the 3 most similar cases with respect to Sim_U, however by incorporating some noise.

The idea of step four is a realistic simulation of the behaviour of customers. One the one hand, customers will usually not be willed to give feedback about the entire retrieval result, however to get feedback about only three cases should be realistic. Further we have introduced noise in order to simulate inconsistent behaviour of customers. Hence, the finally obtained training example does not always come up to the target measure Sim_U, but may contain minor or major deviations according to some probability values ρ_i.

By repeating the described procedure we were able to generate an arbitrary number of training examples to be used by the learning algorithm. In our experiment we have used a genetic algorithm to learn attribute weights and local similarity measures (cf. Section 4).

In order to be able to measure the quality of the learned similarity measure we have generated 200 additional noise free test examples. For a given learned similarity measure Sim_L we then counted the percentage of retrievals based on the 200 queries of the test examples where

- the most similar case was also the most useful one (*1-in-1*)

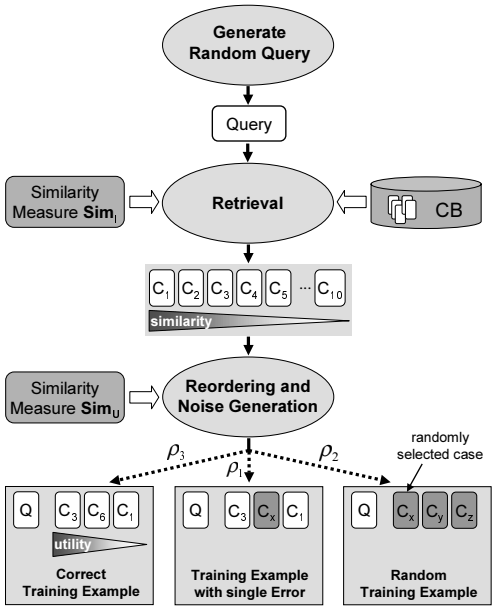

Fig. 8. Learning Customer Preferences: Generation of Training Examples

– the most similar case was at least among the ten most useful ones (*1-in-10*)

according to the noise free utility feedback of the test examples. In order to get an impression of the amount of training data required to get meaningful results, we have started the learning algorithm for an increasing number of training examples. Further, we have repeated the entire experiment at least 3 times in order to achieve average values, even though the number of repetitions is to small for obtaining statistically significant results, of course.

The results of the described experiment are illustrated in Figure 9. Generally, one observes clear improvements in both quality measures at least when providing more then 100 training examples. When using less examples we notice a decrease of retrieval quality which can be explained by overfitting the provided training data, so that the quality of the learned similarity measure is bad with respect to independent test examples. Further, we see that noise seems to have a significant impact on learning only when providing less then 250 training examples.

5.2 Learning to Retrieve Adaptable Cases

For our second experiment we suppose again a product recommendation system, this time used for recommending optimal configurations of personal computers. Since PCs can easily be customized by adding or replacing components, here we assume a CBR system that provides adaptation functionality. This means, the

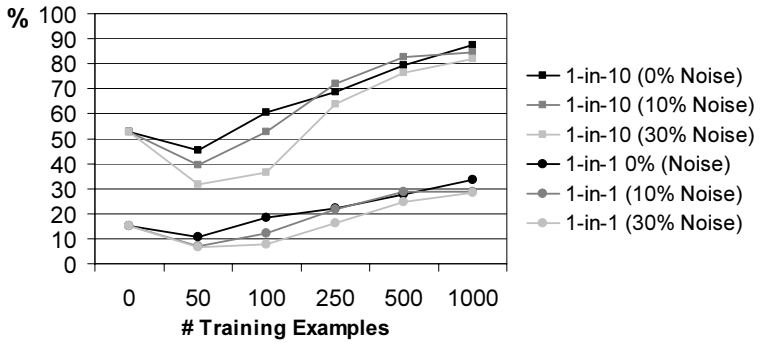

Fig. 9. Learning Customer Preferences: Results

case base contains a set of base PC configurations[3] which can be modified by applying certain adaptation rules [5]. In our example domain PCs are described by 11 attributes (5 numeric and 6 symbolic) and can be adapted by applying 15 more and less complex adaptation rules.

When providing adaptation functionality, optimal retrieval results can only be achieved if the similarity measure considers the adaptation possibilities [16, 14, 11]. Since the utility of a case might change clearly after being adapted, it is not sufficient to estimate the direct utility of cases for a given query. However, to define a similarity measure that estimates the utility of a case that can be achieved by adaptation, one has to analyze the available adaptation knowledge in detail. Further, the knowledge has to be transferred into the similarity measure by using the given knowledge representations. Because this is a very complex and time consuming process, we propose to automate it by applying our learning framework.

Therefore, we assume that a similarity measure Sim_U which estimates the direct utility of cases without considering adaptation possibilities is already given. Usually such a measure can be defined much more easily, or it might also be learned, e.g. like described in Section 5.1. Then, we are able to generate utility feedback like illustrated in Figure 10:

1. Generating a random query q.
2. Retrieving the 10 most similar cases with respect to q by using the similarity measure Sim_U or some other initial measure.
3. Adapting the retrieved cases by applying the adaptation rules.
4. Recalculating the similarity of the 10 adapted cases by using Sim_U.
5. Constructing a training example by reordering the 10 original cases.

This feedback can be used to learn a new similarity measure Sim_A which can be seen as an optimized version of Sim_U since it tries to approximate the utility of cases under consideration of adaptation possibilities. The achieved results

[3] The case base used for the experiment contained 15 base configurations

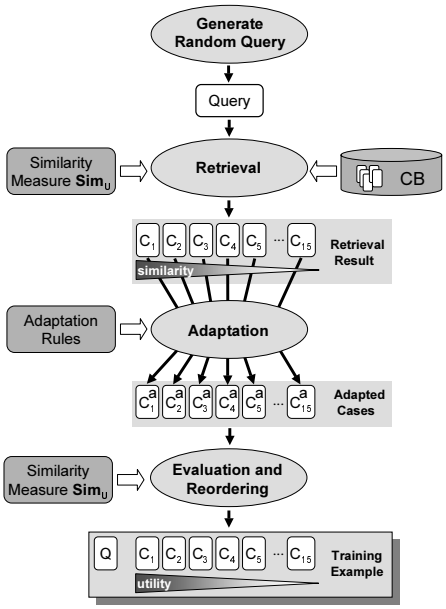

Fig. 10. Learning to retrieve Adaptable Cases: Generation of Training Examples

are illustrated in Figure 11. The results shown here represent average values obtained during 10 repetitions of the experiment. The experimental settings were similar to the experiment described in Section 5.1. Again we have applied our learning algorithm by using increasing number of training examples. To measure the quality of the learned similarity measures we have determined the corresponding retrieval results for 200 independent test queries. Instead of the *1-in-10* quality measure, here, we have chosen the analogous *1-in-3* version. The idea of this measure is the assumption that it is computational feasible to adapt the 3 most similar cases after retrieval in order to select the best resulting case.

For both selected quality measures we notice a clear improvement of the retrieval quality achieved with the optimized similarity measure when using more then 20-35 training examples. In this experiment less training examples were necessary to avoid overfitting because each training example contained more knowledge (feedback about 10 cases instead 3 cases in the previous scenario).

6 Conclusion

We have discussed that the output of knowledge-based systems cannot always simply be judged as correct or incorrect as one would expect from a problem-solving system. In many application domains the output of such systems rather has to be interpreted by considering certain utility functions. This means the output might be more or less useful for solving a problem or for satisfying the

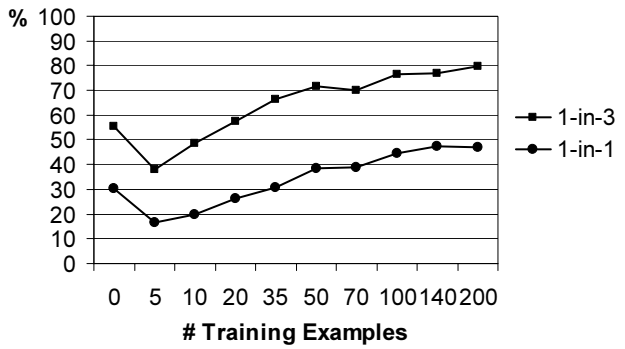

Fig. 11. Learning to retrieve Adaptable Cases: Results

users' demands. In order to produce maximal useful outputs, a knowledge-based system should be able to estimate the utility of a possible output a-priori.

In CBR systems, for example, the utility of available knowledge is approximated by employing similarity measures. The more domain specific knowledge one is able to encode into a similarity measure the higher should be the probability that useful knowledge is selected. According to the local-global principle, particular local similarity measures can be used to encode much domain specific knowledge about the utility function to be approximated. However, the definition of such knowledge-intensive similarity measures is a complex and time consuming process.

In this article we have proposed to facilitate the definition of knowledge-intensive similarity measures by applying a machine learning approach. The approach is based on feedback about the actual utility of cases provided by some similarity teacher. This feedback enables us to evaluate the quality of given similarity measures and can be used to guide an search process with the goal to find an optimal similarity measure. We have described a genetic algorithm for realising this search or optimization process, respectively. In order to show the capabilities of our learning approach, finally we have presented the results of two different evaluation experiments. Here, we have seen that the quality of an initially given similarity measure can be improved significantly, at least if reasonable amount of training data is available. However, in both presented scenarios this should be possible in practice. On the one hand, a product recommendation system is usually used by numerous customers, which may provide feedback explicitly or implicitly, e.g. by their buying behaviour. On the other hand, in our adaptation scenario, the required feedback even can be generated automatically. Here our learning approach allows an optimization of an initial similarity measure "on a mouse click".

In order to reduce the risk of overfitting the training data, and thus reduce the amount of necessary data, the presented learning approach may be extended. Several possibilities to improve the learning process by incorporating additional background knowledge are presented in [7].

References

1. A. Aamodt and E. Plaza. Case-based reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994.
2. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, 1989.
3. R. Bergmann, M. Michael 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.
4. R. Bergmann, S. Schmitt, and A. Stahl. *E-Commerce and Intelligent Methods*, chapter Intelligent Customer Support for Product Selection with Case-Based Reasoning. Physica-Verlag, 2002.
5. R. Bergmann, W. Wilke, I. Vollrath, and S. Wess. Integrating General Knowledge with Object-Oriented Case Representation and Reasoning. In *Proceedings of the 4th German Workshop on Case-Based Reasoning (GWCBR'96)*, 1996.
6. R. Burke. The Wasabi Personal Shopper: A Case-Based Recommender System. In *Proceedings of the 11th International Conference on Innovative Applications of Artificial Intelligence (IAAI'99)*, 1999.
7. T. Gabel. Learning Similarity Measures: Strategies to Enhance the Optimisation Process. Master thesis, Kaiserslautern University of Technology, 2003.
8. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.
9. M. Lenz, B. Bartsch-Spörl, H.D. Burkhard, and S. Wess, editors. *Case-Based Reasoning Technology: From Foundations to Applications*. LNAI: State of the Art. Springer, 1998.
10. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.
11. M.M. Richter. Learning Similarities for Informally Defined Objects. In R. Kühn, R. Menzel, W. Menzel, U. Ratsch, M.M. Richter, and I.-O. Stamatescu, editors, *Adaptivity and Learning*. Springer, 2003.
12. C. J. van Rijsbergen. *Information Retrieval*. Butterworths & Co, 1975.
13. T. Roth-Berghofer. *Knowledge Maintenance of Case-Based Reasoning Systems*. Ph.D. Thesis, University of Kaiserslautern, 2002.
14. 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 (EWCBR'93)*. Springer, 1993.
15. A. Stahl. Learning Feature Weights from Case Order Feedback. In *Proceedings of the 4th International Conference on Case-Based Reasoning (ICCBR'2001)*. Springer, 2001.
16. A. Stahl. Defining Similarity Measures: Top-Down vs. Bottom-Up. In *Proceedings of the 6th European Conference on Case-Based Reasoning (ECCBR'2002)*. Springer, 2002.
17. A. Stahl. *Learning of Knowledge-Intensive Similarity Measures in Case-Based Reasoning*. Ph.D. thesis, Technical University of Kaiserslautern, 2003.
18. A. Stahl and T. Gabel. Using Evolution Programs to Learn Local Similarity Measures. In *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR'2003)*. Springer, 2003.
19. W. Wilke, M. Lenz, and S. Wess. *Case-Based Reasoning Technology: From Foundations to Applications*, chapter Case-Based Reasoning and Electronic Commerce. Lecture Notes on AI: State of the Art. Springer, 1998.