Available online at www.sciencedirect.com
SCIENCE DIRECT
Engineering Applications of Artificial Intelligence 19 (2006) 381–393
Engineering Applications of ARTIFICIAL INTELLIGENCE
www.elsevier.com/locate/engappai

Case-base maintenance by conceptual clustering of graphs

Petra Perner

Institute of Computer Vision and Applied Computer Sciences, August-Bebel-Str. 16-20, 04275 Leipzig

Received 16 January 2006; accepted 21 January 2006
Available online 3 April 2006

Abstract

Case-base maintenance typically involves the addition, removal or revision of cases, but can also include changes to the retrieval knowledge. In this paper, we consider the learning of the retrieval knowledge (organization) as well as the prototypes and the cases as case-based maintenance. We address this problem based on cases that have a structural case representation. Such representations are common in computer vision and image interpretation, building design, timetabling or gene-nets. In this paper we propose a similarity measure for an attributed structural representation and an algorithm that incrementally learns the organizational structure of a case base. This organization schema is based on a hierarchy and can be updated incrementally as soon as new cases are available. The tentative underlying conceptual structure of the case base is visually presented to the user. We describe two approaches for organizing the case base. Both are based on approximate graph subsumption. The first approach is based on a divide-and-conquer strategy whereas the second one is based on a split-and-merge strategy which better allows to fit the hierarchy to the actual structure of the application but takes more complex operations.

Keywords: Conceptual clustering; Case-based reasoning; Graph clustering

1. Introduction

Case-based reasoning is used when general knowledge is lacking, but a set of cases is available which can be made available for reasoning immediately. Such a system contains different knowledge containers which may change during the life time of a CBR system and by doing so the performance of the CBR system will improve. These knowledge containers (Richter, 1998) are: the vocabulary knowledge (used to describe the cases and the problem domain), the retrieval knowledge (including indexing and similarity knowledge), and the adaptation knowledge. The process of changing the knowledge containers is called case-base maintenance.

Case-base maintenance typically involves the addition, removal or revision of cases, but can also include changes to the retrieval knowledge (Leake and Wilson, 1998; Craw et al., 2001). In this paper, we consider the learning of the retrieval knowledge (organization) as well as the prototypes and the cases as case-based maintenance. We address this problem based on cases that have a structural case representation. Such representations are common in computer vision and image interpretation (Bischof and Caelli, 2000; Perner, 1993), building design (Gebhardt et al., 1997), timetabling (Burke et al., 2001) or gene-nets.

Determination of similarity between structural cases is time-consuming and requires a good organization of the case base. Most applications have attribute assignments to the components of the structure and attributes that describe the relationships between the components. This requires a similarity measure which can describe both structural similarity and the proximity of the attribute labels. The similarity measure should have the flexibility to view the similarity from these different perspectives.

In the paper we propose a similarity measure for an attributed structural representation. We use an image-related application to show how it can be used for matching.

We are usually not concerned with the problem of having a large case base in an image-related application. On the contrary, we are usually concerned with the problem that only a few cases are available at the time when the system is designed. These few cases can be made available for

E-mail address: ibaiperner@aol.com.

0952-1976/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2006.01.014

reasoning immediately and during the use of the system new cases should be learned, so that the case base will grow and cover more problems. Therefore, we propose an algorithm that incrementally learns the organizational structure. This organization scheme is based on a hierarchy and can be updated incrementally as soon as new cases are available. The tentative underlying conceptual structure of the case base is visually presented to the user. We describe two approaches for organizing the case base. Both are based on approximate graph subsumption. The first approach is based on a divide-and-conquer strategy whereas the second is based on a split-and-merge strategy which better allows to fit the hierarchy to the actual structure of the application, but requires more complex operations. The first approach uses a fixed threshold for the similarity values. The second approach uses for the grouping of the cases an evaluation function.

The paper is organized as follow: In Section 2 we will describe the concepts for CBR indexing and learning. The definition of a structural case, the similarity measure as well as matching are presented in Section 3. Our two approaches for case-base organization and learning are described in Section 4. We compare our methods to related work in Section 5. Finally, conclusions are given in Section 6.

2. Concepts for CBR indexing and learning

2.1. Case-base organization and retrieval

Cases can be organized into a flat case base or in a hierarchical fashion. In a flat organization, we have to calculate similarity between the problem case and each case in memory. It is clear that this will take a considerable amount of time even when the case base is very large.

To speed up the retrieval process, a more sophisticated organization of the case base is necessary. This organization should allow separating the set of similar cases from those cases not similar to the recent problem at the earliest stage of the retrieval process. Therefore, we need to find a relation p that allows us to order our case base:

Definition 1. A relation p on a set CB is called a partial order on CB if it is reflexive, antisymmetric, and transitive. In this case, the pair $\langle \mathrm{CB}, p \rangle$ is called a partial ordered set or poset.

The relation can be chosen depending on the application. One common approach is to order the case base, based on the similarity value. The set of cases can be reduced by the similarity measure to a set of similarity values. The relation $\leqslant$ over these similarity values gives us a partial order over these cases. The derived hierarchy consists of nodes and edges. Each node in this hierarchy contains a set of cases that do not exceed a specified similarity value. The edges show the similarity relation between the nodes. The relation between two successor nodes can be expressed as follows: Let z be a node and x and y are two successor nodes of z, then x subsumes z and y subsumes z. By tracing down the hierarchy, the space gets smaller and smaller until finally a node will not have any successor. This node will contain a set of similar cases. Among these cases has to be found the most similar case to the query case. Although we still have to carry out matching, the number of matches will have decreased through the hierarchical ordering. The nodes can be represented by the prototypes of the set of cases assigned to the node. When the hierarchy is used to process a query, the query is only matched with the prototype. Depending on the outcome of the matching process, the query branches right or left of the node.

Such hierarchy can be created by hierarchical or conceptual clustering (Perner, 1998a), k–d trees (Wess et al., 1993) or decision trees (McSherry, 2001). There are also set membership-based organizations known, such as semantic nets (Grimnes and Aamodt, 1996) and object-oriented representations (Bergmann and Stahl, 1998).

The problem is to determine the right relation p that allows to organize the case base, a procedure for learning prototypes and case classes, and a similarity measure.

2.2. Learning in a CBR system

CBR maintenance is closely related to learning. It aims to improve the performance of the system.

Let X be a set of cases collected in a case-base CB. The relation between each case in a case base can be expressed by the similarity value sim. The case base can be partitioned into n case classes $C : \mathrm{CB} = \cup_{i=1}^{n} C_i$ such that the intracase class similarity is high and the intercase class similarity low. The set of cases in each class C can be represented by a case that generally describes the cluster. This representative can be the prototype, the median, or an a priori selected case. Whereas the prototype implies that the representative is the mean of the cluster which can easily be calculated from numerical data, the median is the case whose sum of all distances to all other cases in a cluster is minimal. The relation between the different case classes C can be expressed by higher order constructs expressed, e.g. as super classes that gives us a hierarchical structure over the case base.

There are different learning strategies that can take place in a CBR system:

1. Learning takes place if a new case x has to be stored into the case base such that:

   $$\mathrm{CB}_{n+1} = \mathrm{CB}_n \cup (x).$$

2. It may incrementally learn the case classes and/or the prototypes representing the class.
3. The relationship between the different cases or case classes may be updated according to the new case classes.
4. The system may learn the similarity measure.

2.2.1. Learning new cases and forgetting old cases

Learning new cases means just adding cases into the case base upon some notification. Closely related to case adding

is case deletion or forgetting cases which have shown low utility. This should control the size of the case base. There are approaches that keep the size of the case-base constant and delete cases that have not shown good utility within a fixed time window (Portinale et al., 1999). The failure rate is used as utility criterion. Given a period of observation of N cases, when the CBR component exhibits M failures in such a period, we define the failure rate as $f_r = M/N$. Other approaches try to estimate the "coverage" of each case in memory and use this estimate to guide the case memory revision process (Smyth and McKenna, 1998).

The adaptability to the dynamics of the changing environment that requires storing new cases in spite of a finite case-base size is addressed in Surma and Tyburcy (1998). Based on intraclass similarity it is decided whether a case is to be removed from or to be stored in a cluster. This requires to define a priori the thresholds for inter- and intracase class similarity.

Another approach for dynamic partitioning of the space of uncovered cases is proposed by McSherry (2000).

In this paper, we do not consider case deletion, but focus instead on the addition of new cases to the case base.

2.2.2. Learning of prototypes

Learning of prototypes has been described in Surma and Tyburcy (1998) for the flat organization of a case base. The prototype or the representative of a case class is the most general representation of a case class. A class of cases is a set of cases sharing similar properties. It is a set of cases that do not exceed a boundary for the intraclass dissimilarity. Cases that are on the boundary of this hyperball have a maximal dissimilarity value. A prototype can be selected a priori by the domain user. This approach is preferable when the domain expert knows for sure the properties of the prototype. The prototype can be calculated by averaging over all cases in a case class, or the median of the cases is chosen. When only a few cases are available in a class and subsequently new cases are stored in the class, then it is preferable to incrementally update the prototype according to the new cases.

2.2.3. Learning of higher order constructs

The ordering of the different case classes gives an understanding of how these case classes are related to each other. For two case classes which are connected by an edge, similarity relation holds. Case classes that are located at a higher position in the hierarchy apply to a wider range of problems than those located near the leaves of the hierarchy. By learning how these case classes are related to each other, higher order constructs are learned (Perner and Paetzold, 1995).

2.2.4. Learning of similarity

For generality, we will also consider here that feature weights can be learned.

By introducing feature weights, we can put special emphasis on some features for the similarity calculation. It is possible to introduce local and global feature weights. A feature weight for a specific attribute is called a local feature weight. A feature weight that averages over all local feature weights for a case is called a global feature weight. This can improve the accuracy of the CBR system. By updating these feature weights we can learn similarity (Wettscherek et al., 1997; Bonzano and Cunningham, 1997). Learning feature weights in a structural representation is computationally complex and time consuming. Therefore, we will only propose a similarity measure for structural representations which can handle feature weights, but we will not consider how these feature weights can be learned. Instead, we will assume that the user can specify this feature weight a priori.

3. Structural representation and structural similarity measure

Before we can describe our approach to maintaining the retrieval knowledge in detail, we introduce the basic definitions and notation that will be used in this paper.

3.1. Definition of a case

The structural representation of a case can be described as a graph. If we assign attributes to the nodes and the edges, then we have an attributed graph defined as follow:

Definition 2.

W…set of attribute values
A…set of all attributes
b: $A \rightarrow W$ partial mapping, called attribute assignments
B…set of all attribute assignments over A and W.

A graph $G = (N, p, q)$ consists of
N…finite set of nodes
p: $N \rightarrow B$ mapping of nodes to attribute assignment
q: $E \rightarrow B$ mapping of nodes to attribute assignment,
where $E = (N \cdot N) - I_N$ and
I_N is the Identity relation in N.

This representation allows us to consider only objects, the spatial relation between objects, or the whole graph (see Section 3.2).

For generality, we will not consider specific attribute assignments to the nodes and the edges. To give the reader an idea what these attributes could be, we will consider the cases to be images such as the ultrasonic image shown in Figs. 1a–d. The images show defects such as cracks in a metal component taken by an ultrasonic image acquisition system called SAFT. A defect comprises of several reflection points which are in a certain spatial relation to each other. Here the nodes could be the objects (reflection points) in the image and the edges are the spatial relation between these objects (e.g. right-of, behind, etc.). Each object has attributes (e.g. size, mean gray level value, etc.)

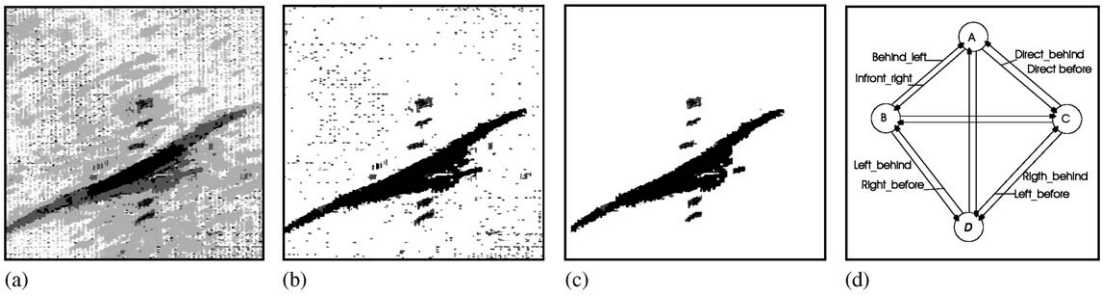


Fig. 1. (a) Original image, (b) binary image (c) result, and (d) representation.

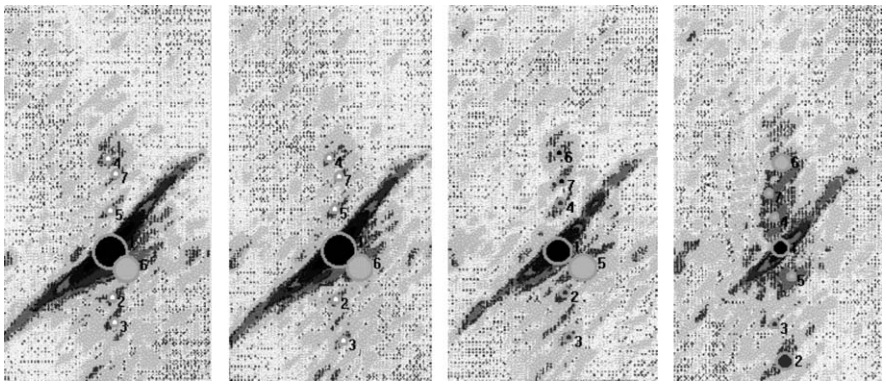


Fig. 2. Cases 1, 2, 3, and 4 (in the order of their appearance).

which are associated to the corresponding node within the graph. We will not describe the application in this paper, nor will we describe how these objects are extracted from the images, combined into a graph and labeled by symbolic terms. For these details we refer to Perner (1993, 1998b, 1996).

The cases used for this study are shown in Figs. 2a–d.

3.2. Similarity measure

We may define our problem of similarity as to find structural identity between two structures. However, structural identity is a very strong requirement. An alternative approach is to require only part isomorphism.

Definition 3. Two graphs $G_1 = (N_1, p_1, q_1)$ and $G_2 = (N_2, p_2, q_2)$ are in the relation $G_1 \leqslant G_2$ iff there exists a one-to-one mapping f: $N_1 \rightarrow N_2$ with

(1) $p_1(x) = p_2(f(x))$ for all $x \in N_1$ and
(2) $q_1(x, y) = q_2(f(x), f(y))$ for all $x, y \in N_1,\ x \neq y$.

Definitions 3 and 4 require identity in the attribute assignments of the nodes and the edges. To allow proximity in the attributes labels, we introduce the following way to handle similarity.

In Definitions 3 and 4 we may relax the required correspondence of attribute assignment of nodes and edges to that we introduce ranges of tolerance:

If $a \in A$ is an attribute and $W_a \subseteq W$ is the set of all attribute values which can be assigned to a, then we can determine for each attribute a a mapping:

$$\mathrm{dist}_a :\ W_a \rightarrow [0, 1].$$

The normalization to a real interval is not absolutely necessary but advantageous for the comparison of attribute assignments.

For example, let a be the attribute spatial_relation as it is in, e.g. in the ultrasonic image the relation between the nodes and

$$W_a = \{\text{behind}, \text{right}, \text{behind\_left}, \text{in\_front\_right}, \ldots\},$$

then we could define:

$$\mathrm{dist}_a(\text{behind\_right}, \text{behind\_right}) = 0,$$

$$\mathrm{dist}_a(\text{behind\_right}, \text{inf ront\_right}) = 0.25,$$

$$\mathrm{dist}_a(\text{behind\_right}, \text{behind\_left}) = 0.75.$$

Based on such a distance measure for attributes, we can define different variants of distance measure as mapping:

$$\mathrm{dist} :\ B^2 \rightarrow R^+$$

(R^+ is the set of positive real numbers) in the following way:

$$\mathrm{dist}(x,y) = \frac{1}{D}\sum_{a\in D} \mathrm{dist}_a(x(a), y(a)),$$

with $D = \mathrm{domain}(x) \cap \mathrm{domain}(y)$.

Usually, in the comparison of graphs not all attributes have the same priority. Thus it is good to determine a weight factor w_a and then define the distance as follows:

$$\mathrm{dist}(x,y) = \frac{1}{D}\sum_{a\in D} w_a\, \mathrm{dist}_a(x(a), y(a)).$$

For definition of part isomorphism, we get the following variant:

Definition 4. Two graphs $G_1 = (N_1, p_1, q_1)$ and $G_2 = (N_2, p_2, q_2)$ are in the relation $G_1 \leqslant G_2$ iff there exists a one-to-one mapping f: $N_1 \to N_2$ and thresholds C_1, C_2 with

(1) $\mathrm{dist}(p_1(x), p_2(f(x)) \leqslant C_1$ for all $x \in N_1$,
(2) $\mathrm{dist}(q_1(x,y), q_2(f(x), f(y)) \leqslant C_2$ for all $x, y \in N_1$, $x \neq y$.

Obviously, it is possible to introduce a separate constant for each attribute. Depending on the application, the similarity may be sharpened by a global threshold:

If it is possible to establish a correspondence g according to the requirements mentioned above, then an additional condition should be fulfilled:

$$\sum_{(x,y)\in g} \mathrm{dist}(x,y) \leqslant C_3,$$

where C_3 is the global threshold.

3.3. Mean of a graph

The mean graph can be computed as follows:

Definition 5. A graph $G_p = (N_p, p_p, q_p)$ is a prototype of a class of cases $C_i = \{G_1, G_2, \ldots, G_t(N_t, p_t, q_t)\}$ iff $G_p \in C_i$ and if there is a one-to-one mapping f: $N_p \to N_i$ with

(1) $p_p(x_i) = \frac{1}{n}\sum_{n=1}^{t} p_n(f(x_i))$ for all $x_i \in N$ and
(2) $q_p(x_i, y_i) = \frac{1}{n}\sum_{n=1}^{t} q_n(f(x_i), f(y_i))$ for all $x_i, y_i \in N$.

3.4. Graph subsumption

On the basis of the part isomorphism, we can introduce a partial order over the set of graphs. If a graph G_1 is included in another graph G_2 than the two graphs are in the relation $G_1 \leqslant G_2$ and the number of nodes of G_1 is not higher than the number of nodes of G_2. We can also say G_1 subsumes G_2 and we can write $G_1 \succ G_2$.

If we have three graphs G_1, G_2 and G_3 with $G_3 \succ G_1$ and $G_3 \succ G_2$, then G_3 is the most specific common substructure *mscs* of the two graphs G_1 and G_2. We can also say that G_3 is the generalization or the general concept of the two graphs G_1 and G_2.

We can use these relations to organize our case base and on the other hand it allows us to discover the underlying concept of the domain.

3.5. Matching

Now consider an algorithm for determining the part isomorphism of two graphs. The main approach is to find a superset of all possible correspondences f and then exclude nonpromising cases. In the following, we assume that the number of nodes of G_1 is not greater than the number of nodes of G_2.

A technical aid is to assign to each node n a temporary attribute list $K(n)$ of all attribute assignments of all the connected edges:

$$K(n) = (a:\ q(n,m) = a, m = N - \{n\}) \text{ where } (n \in N).$$

The order of list elements has no meaning. Because all edges exist in a graph the length of $K(n)$ is equal to $2(|N| - 1)$.

For demonstration purposes, consider the examples in Figs. 3a–b. The result would be

$$K(X) = (\mathrm{bl}, \mathrm{bl}, \mathrm{br}),$$

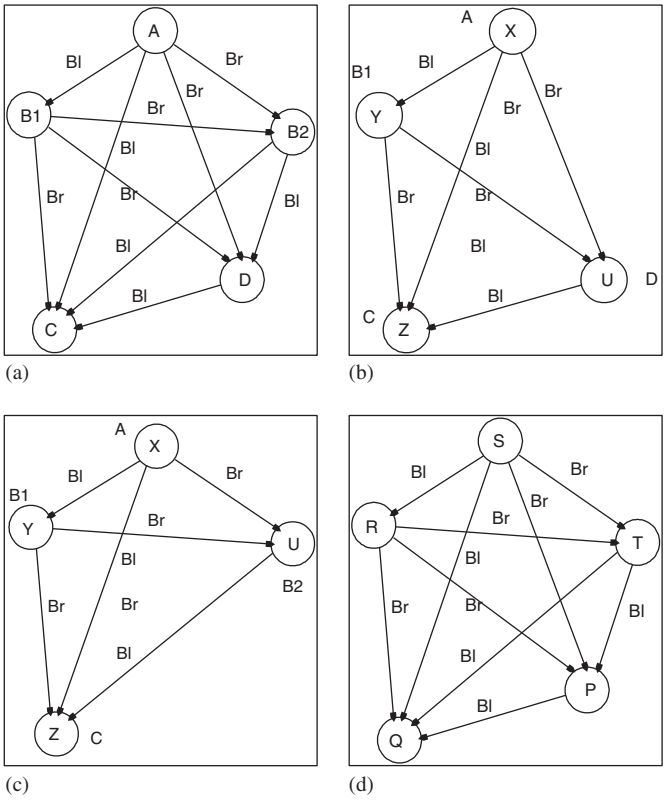


Fig. 3. (a) graph_1, (b) graph_2 and result of subgraph isomorphism to graph_1, (c) second result of subgraph isomorphism of graph_1 and graph_2, and (d) graph_3 and results for subgraph isomorphism to graph_2.

$K(Y) = (\mathrm{br}, \mathrm{br}, \mathrm{bl})$.

In the worst case, the complexity of the algorithm is $O(|N|^3)$.

In the next step we assign to each node of G_1 all nodes of G_2 that could be assigned by a mapping f. That means we calculate the following sets:

$$L(n) = \{m : \ m \in N_2, p_1(n) = p_2(m), K(n) \subseteq K(m)\}.$$

The inclusion $K(n) \subseteq K(m)$ shows that the list $K(m)$ contains the list $K(n)$ without considering the order of the elements. When the list $K(n)$ contains multiple times an attribute assignment, then the list $K(m)$ also contains multiple times this attribute assignment.

For the example in Figs. 3a and b we get the following L-sets:

$$L(X) = \{A\},$$

$$L(Y) = \{B_1\},$$

$$L(Z) = \{C\},$$

$$L(U) = \{D, B_2\}.$$

We did not consider in this example the attribute assignments of the nodes.

Now, the construction of the mapping f is prepared and if there exists any mapping then the following condition must hold:

$$f(n) \in L(n) \rightarrow (n \in N_1).$$

The first condition for the mapping f regarding the attribute assignments of nodes holds because of the construction procedure of the L-sets. In case that one set $L(n)$ is empty, there is no partial isomorphism.

Also, if there are nonempty sets, in a third step it has to be checked if the attribute assignments of the edges match.

If there is no match, then the corresponding L-set should be reduced to

for all nodes n_1 of G_1
  for all nodes n_2 of $L(n_1)$
    for all edges (n_1, m_1) of G_1
    if for all nodes m_2 of $L(m_1)$
      $p_1(n_1,m_1) \neq p_2(\mathrm{n}_2,m_2)$
    then $L(n_1) := L(n_1) \backslash \{n_2\}$

If the L-set of a node has changed during this procedure, then the examinations already carried out should be repeated. That means that this procedure should be repeated until none of the L-sets has changed.

If the result of the third step is an empty L-set, then there is also no partial isomorphism. If all L-sets are nonempty, then some mappings f from N_1 to N_2 have been determined. If each L-set contains exactly only one element, then there is only one mapping. In a final step, all mappings should be excluded that are not of the one-to-one type.

Table 1
Results for the L-sets for matching graph_1 with graph_2 and graph_3

N_1	f_1	f_2
X	A	A
Y	B_1	B_1
Z	C	C
U	D	B_2

For example, let us compare graph_1 in Fig. 3a and graph_2 in Fig. 3b. In the third step, the L-set of graph_1 will not be reduced and we get two solutions, shown in Figs. 3b and c and for the L-sets in Table 1.

When we compare the representation of graph_1 and graph_3 in Fig. 3d, the L-set of graph_1 also contains two elements:

$$L(U) = \{T, P\}.$$

However, in the third step the element T will be excluded if the attribute assignments of the edges (U, Y) and (T, R) do not match when node U is examined.

If the L-set of a node has changed during the third step, then the examinations already carried out should be repeated. That means that step 3 is to be repeated until there is no change in any L-set.

This algorithm has a total complexity of the order $O(|N_2|^3, |N_1^3| \cdot |M|^3)$. $|M|$ represents the maximal number of elements in any L-set ($|M| \leqslant |N_2|$).

The similarity in the attribute labels can be handled by the way the L-sets are defined and particularly the inclusion of K-lists:

Given C a real constant, $n \in N_1$ and $n \in N_2$. $K(n) \subseteq {}_cK(m)$ is true iff for each attribute assignment b_1 of the list $K(n)$ attribute assignment b_2 of $K(m)$ exists, such that $\mathrm{dist}(b_1, b_2) \leqslant C$. Each element of $K(m)$ is to be assigned to a different element in list $K(n)$.

Obviously, it is possible to introduce a separate constant for each attribute. Depending on the application, the inclusion of the K-lists may be sharpened by a global threshold:

If it is possible to establish a correspondence g according to the requirements mentioned above, then an additional condition should be fulfilled:

$$\sum_{(x,y)\in g} \mathrm{dist}(x, y) \leqslant C_3 \ (\mathrm{C}_3 \text{ threshold constant}).$$

Then we get the following definition for the L-set:

Definition 6.

$$L(n) = \{m : \ m \in N_2, \mathrm{dist}(p_1(n), p_2(m)) \leqslant C_1, K(n) \subseteq_c K(m)\}.$$

In step 3 of the algorithm for the determination of one-to-one mapping, we should also consider the defined distance function for the comparison of the attribute assignments of the edges. This new calculation increases the total amount of effort, but the complexity of the algorithm is not changed.

4. Case-base organization and learning

We propose a hierarchical organization schema of the case base that can be updated incrementally. Two approaches are described in the following section. Both are based on approximate graph subsumption. The first one is based on a divide-and-conquer strategy whereas the second is based on an evaluation function and a split-and-merge strategy.

4.1. Approach I

4.1.1. Index structure

The initial case base may be built up by existing cases. Therefore, a nonincremental learning procedure is required in order to build the index structure. When using the system, new cases may be stored into the case base. They should be integrated into the already existing case base. Therefore, we need an incremental learning procedure (Perner and Paetzold, 1995).

Cases in the case base are representations between graphs. As an important relation between structural cases, we have considered similarity based on part isomorphism. Because of this characteristic it is possible to organize the case base as a directed graph.

In terminological logics (Nebel, 1990) the extension of the concept, with respect to a domain of interpretation, is the set of all things in the domain which satisfy the concept description. As we have shown before, the subsumption is a subset relation between the extensions of concepts. Subsumption allows us to observe the conceptual knowledge of the domain of interpretation. Therefore, we introduce a new attribute a which is called concept_description (in short: cd). To this attribute are assigned the names for the various case classes. They are not known before they should be learned during usage of the system. Thus the attribute gets the temporary value "unknown". We assume that the attribute values can be specified by the user of the system when a new case class is found by the incremental learning procedure and this name is assigned to the attribute.

In the following, we will define the index structure of the case base as a graph that contains the graphs described above in the nodes:

Definition 7. H is given, the set of all graphs.

A index graph is a tupel $\mathrm{IB} = (N, E, p)$, with

(1) $N \subseteq H$ set of nodes and
(2) $E \subseteq N^2$ set of edges. This set should show the partial isomorphism in the set of nodes, meaning it should be valid $x \leqslant y \Rightarrow (x, y) \in E$ for all $x, y \in N$.
(3) $p: N \to B$ mapping of case names to the index graph (also the attribute values for the case class).

Because of the transitivity of part isomorphism, certain edges can be directly derived from other edges and do not need to be separately stored. A relaxation of point 2 in Definition 5 can be reduced storage capacity.

In the nodes of the index graph are stored the names of the case and not the case itself.

Note that case identifiers are stored into the nodes of the directed graph, not the actual cases. The root note is the dummy node.

It may happen that by matching two graphs we will find two solutions, as it is shown in Figs. 3a–d. This will result in an index structure as shown in Fig. 4. The hypergraph will branch into two paths for one solution. That means at $n = 4$ we have to match twice, once for the structure $\{A, B_1, C, D, B_2\}$ and the other time for the structure $\{A, B_1, C, B_2, D\}$. Both will result in the same solution, but by doing so the matching time will double.

The solution to this problem could be that in the index structure a link to point p will advise the matcher to follow this path. But this will only be the right solution if the increase in the number of nodes from one step to another is one and not more. Otherwise, more than one solution will still be possible and should be considered during the construction and update of the index structure.

4.1.2. Incremental learning of the index structure

Now the task is to build up the graphs of IB into a supergraph by a learning environment.

Input:
Supergraph $\mathrm{IB} = (N, E, p)$ and
Graph $x \in H$.

Output:
Modified Supergraph $\mathrm{IB}' = (N', E', p')$
With $N' \subseteq N \cup \{x\}$, $E \subseteq E'$, $p \subseteq p'$

At the beginning of the learning process or the process of construction of index graph N can be an empty set.

The attribute assignment function p' gives the values $(p'(x), (dd))$ as an output. This is an answer to the question: What is the name of the image name that is mirrored in the image graph x?

The inclusion

$$N' \subseteq N \cup \{x\}$$

says that the graph x may be isomorphic to one graph y contained in the case base, so $x \leqslant y$ and also $y \leqslant x$ hold.

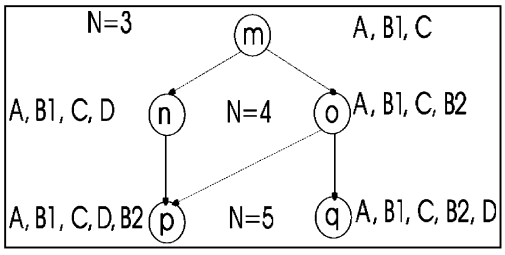


Fig. 4. Hypergraph and the problem of two solutions.

Then no new node is created, which means the case base is not increased.

The algorithm for the construction of the modified index structure IB′ can also use the circumstance that no graph is part isomorphic to another graph if it has more nodes than the second one.

As a technical aid for the algorithm, we introduced a set N_i. N_i contains all graphs of the case-base IB with exactly i nodes. If the maximal number of nodes of the graph contained in the case base is k, then:

$$N = \bigcup_{i=k}^{k} N_i.$$

The graph which has to be included in the case base has l nodes ($l>0$). By comparison of the current graph with all graphs contained in the case base, we can make use of transitivity of part isomorphism for the reduction of the nodes that have to be compared. The full algorithm for the construction of the hierarchy is shown in Fig. 5.

If we use the approach described in Section 3.2 for uncertainty handling, then we can use the algorithm presented in Section 3.5 without any changes. But we should notice that for each group of graphs that is approximately isomorphic, the image graph that occurred first is stored in the case base. Therefore, it is better to calculate for every instance and each new instance of a group a prototype and store it in the index structure of the case base.

Fig. 5 illustrates this index hierarchy. Suppose we have given a set of structural cases, where the supergraph is the empty graph. Then we open the first node in the supergraph for this case at level n which refers to the number of nodes this structural case has, and make a link to the root node which is the dummy node. Then a new case is given to the supergraph. It is first classified by traversal of the tentative supergraph. If it does not match with a case stored in case base then a new node is opened at the level which refers to the number of nodes this structural case has and a link to the root node is made. If the node matches with the case in the supergraph and if the case is in the relation $G_1 < G_2$, then a new node is opened at the level k, the link between the root node and the recent node is removed and a link between the new node and the old node is inserted and another link from the new node to the root is installed. This procedure is repeated until all cases are inserted into the supergraph.

Algorithm

E' := E;
Z := N;
for all y ∈ N_l
if x ≤ y **then** [IB' := IB; return];
N' := N ∪ {x};
for all i with 0 < i < l;
for all y ∈ N_i \ Z;
for all y ≤ x **then** [Z := Z \ {u | u ≤ y, u ∈ Z};
E' := E' ∪ { (y,x)}];
for all i with l < i ≤ k
for all y ∈ N_i \ Z
if x ≤ y **then** [Z := Z \ {u _ y ≤ u, u ∈ Z };
E' := E' ∪ { (x,y)}];
p' := p ∪ { (x, (dd : unknown))};

Fig. 5. Algorithm of approach I.

4.1.3. Retrieval and result

Retrieval is done by classifying the current case through the index hierarchy until a node represented by a prototype having the same number of nodes as the query is reached. Then the most similar case in this case class is determined.

The output of the system are all graphs y in the image database which are in relation to the query x as follows:

$$(x \leqslant y \vee x \geqslant y) \leqslant C$$

where C is a constant which can be chosen by the user of the system.

Fig. 6 shows an example of a learned index structure. Cases that are grouped together in one case class can be viewed by the user by clicking at the node which opens a new window showing all cases that belong to the case class. The node itself is labeled with the name of the graphs in this example but can be also labeled with a user given name.

The visualization component allows the user to view the organization of his case base. It shows him the similarity relation between the cases and case classes and by doing this gives him a good understanding of his domain.

4.2. Approach II

Whereas in Section 4.1 the index structure is built based on a divide-and-conquer technique, in this approach we use a strategy which is more flexible to fit the hierarchy dynamically to the cases. It does not only allow to incorporate new cases into the hierarchy and open new nodes by splitting the leaf nodes into two child nodes. It also allows to merge existing nodes and to split existing nodes at every position in the hierarchy.

4.2.1. Index structure

A concept hierarchy is a directed graph in which the root node represents the set of all input instances and the terminal nodes represent individual instances. Internal nodes stand for sets of instances attached to those nodes and represent a super concept. The super concept can be represented by a generalized representation of this set of instances such as the prototype, the median or a user-selected instance. Therefore a concept C, called a class, in the concept hierarchy is represented by an abstract concept description and a list of pointers to each child concept $M(C) = \{C_1, C_2, \ldots, C_i, \ldots, C_n\}$, where C_i is the child concept, called a subclass of concept C.

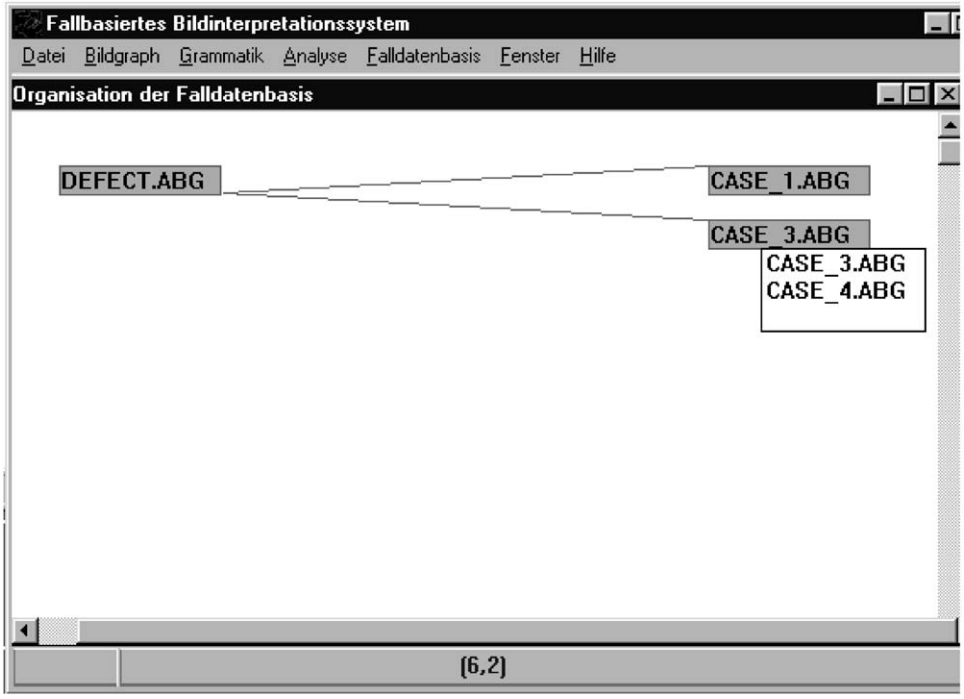


Fig. 6. Example of case-base organization.

4.2.2. Learning

4.2.2.1. Utility function. When several distinct partitions are incrementally generated over the case base, a heuristic is used to evaluate the partitions. This function evaluates the global quality of a single partition and favors partitions that maximize potential for inferring information. In doing this, it attempts to minimize intracase class variances and to maximize intercase class variances. The employment of an evaluation function avoids the problem of defining a threshold for class similarity and interclass similarity. The threshold is to determine domain-dependent and it is not easy to define them a priori properly. However, the resulting hierarchy depends on a properly chosen threshold. We will see later on by example what influence it has on the hierarchy.

Given a partition $\{C_1, C_2, \ldots, C_m\}$, the partition which maximizes the difference between the case class variance s_B and the within-case class variance s_W is chosen as the right partition:

$$\text{SCORE} = \frac{1}{m}\left|s_B^{*2} - s_W^{*2}\right| \Rightarrow \text{MAX!} \tag{1}$$

The normalization to m (m—the number of partitions) is necessary to compare different partitions.

If G_{pj} is the prototype of the jth case class in the hierarchy at level k, $\bar{G}$ the mean graph of all cases in level k, and G_{vj}^2 is the variance of the graphs in the partition j, then:

$$\text{SCORE} = \frac{1}{m}\left|\sum_{j=1}^{m} p_j(G_{pj} - \overline{G})^2 - \sum_{j=1}^{m} p_j G_{vj}^2\right|, \tag{2}$$

where p_j is the relative frequency of cases in the partition j.

4.2.2.2. Learning of new cases and case classes. It might happen that the reasoning process results in the answer: "There is no similar case in the case base". This indicates that such a case has not been seen before. The case needs to be incorporated into the case base in order to close the gap in the case base. This is done by classifying the case according to the case base and a new node is opened in the hierarchy of the case base at that position where the similarity relation holds. The new node represents a new case class and the present case is taken as case representative.

The evidence of new case classes increases when new cases, where the evaluation measure holds, are incorporated into the node. This is considered as learning of case classes.

4.2.2.3. Prototype learning. When learning a class of cases, cases where the evaluation measure holds are grouped together. The case that appeared first would be the representative of the class of these cases and each new case is compared to this case. Obviously, the first case to appear might not always be a good case. Therefore, it is better to compute a prototype for the class of cases as described in Section 3.3.

In the same manner we can calculate the variance of the graphs in one case class. The resulting prototype is not a case that exists in reality. Another strategy for calculating a prototype is to calculate the median of the case in a case class.

4.2.2.4. Refinement and abstraction of case classes. The constructed case classes and the hierarchy are sensitive to

the order of case presentation, creating different hierarchies from different orders of the same cases. We have already included one operation to avoid this problem by learning of prototypes. Two additional operations (Gennari et al., 1989) should help it recover from such nonoptimal hierarchies. At each level of the classification process, the system considers merging the two nodes that best classify the new instance. If the resulting case class is better according to the evaluation function described in Section 4.2.2.1 than the original, the operation combines the two nodes into a single, more abstract case class. This transforms a hierarchy of N nodes into one having $N+1$ nodes, see Fig. 7. This process is equivalent to the application of a "climb-generalization tree" operator (Michalski, 1983), with the property that the generalization tree is itself built and maintained by the system. Other merge strategies are described in Manuela Veloso (1997).

The inverse operation is splitting of one node into two nodes as illustrated in Fig. 8. This is also known as refinement. At each level, the learning algorithm decides to classify an instance as a member of an existing case class, it also considers removing this case class and elevating its children.

If this leads to an improved hierarchy, the algorithm changes the structure of the case-base hierarchy accordingly.

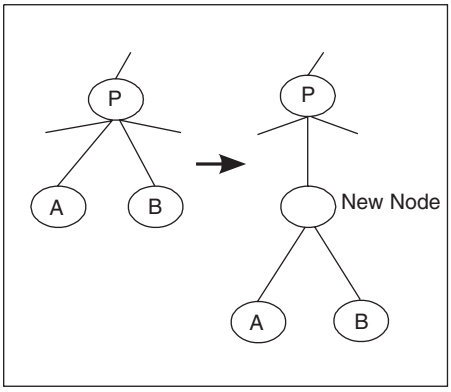


Fig. 7. Node merging.

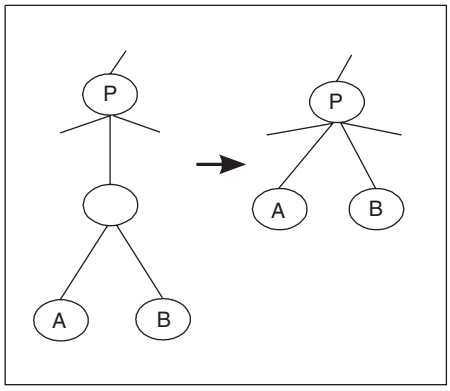


Fig. 8. Node splitting.

4.2.3. Algorithm

Now, that we have defined our evaluation function and the different learning levels, we can describe our learning algorithm. We adapt the notion of Fisher (Fisher, 1987) for concept learning to our problem. If a new case has to be entered into the case base, the case is tentatively placed into the hierarchy by applying all the different learning operators described before. The operation which gives us the highest evaluation score is chosen. The new case is entered into the case base and the case-base hierarchy is reorganized according to the selected learning operation. Fig. 9 shows the CBR learning algorithm in detail.

4.2.4. The learning algorithm in operation: an example

The learning process is illustrated in Figs. 10a–d for the four cases shown in Figs. 2a–d that were used to construct the hierarchy in Fig. 6.

The learning algorithm first tests where the case should be placed in the hierarchy. Fig. 10a shows how Case 1 is incorporated into the case base. At this stage the case base is empty, and so the only possible operation is to *create a new node* for Case 1. Next, Case 2 is inserted into the case base (see Fig. 10b). First, it is placed in the existing node and the score for this operation is calculated. Then the *create a new node* operation is performed and its score is calculated. The last operation which can be applied in this case is *node merging*. The score for this operation is also calculated. Since all three scores have the same value, the first operation, *insert into existing node*, is selected on this occasion.

Fig. 10c shows how Case 3 is incorporated into the case base. Any of the operations *insert to existing node*, *create a new node*, and *node merging* can be applied to the existing hierarchy. The highest score is obtained for the operation *create a new node*. This operation is therefore used to update the hierarchy. Case 4 can be inserted into either of the two existing nodes (see Fig. 10d). The operations *create a new node* and *node merging* can then be applied in sequence. The highest score is obtained for the operation *insert to node_2*. This is the final hierarchy resulting from the application of our learning algorithm to the four cases. Note that the *node-splitting* operation was not used in our example because the resulting hierarchy did not allow for this operation. The learning algorithm can be seen to have achieved meaningful groupings of cases in the example.

4.3. Discussion of the two approaches

The results in Approach I shown in Fig. 6 are two nodes containing a set of cases each. The second node is highlighted in Fig. 5 and shows that the two cases (Case 3 and Case 4) belong to the same case class. The first node shows only the name of Case 1 but also contains Case 2. The partition {1,2} and {3,4} is reached by a threshold of 0.0025 for the similarity. Matching has to be performed three times for this structure. First, matching the prototypes is performed and afterwards the closest case in the

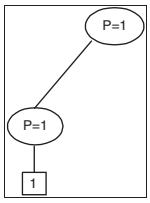
```
Input:          Case Base Supergraph CB
                An unclassified case G
Output:         Modified Case Base CB´

Top-level call: Case base (top-node, G)
Variables:      A, B, C, and D are nodes in the hierarchy
                K, L, M, and T are partition scores

Case base (N, G)

IF N is a terminal node,
        THEN Create-new-terminals (N, G)
             Incorporate (N, G)
        ELSE For each child A of node N,
                  Compute the score for placing G in A.
                  Compute the scores for all other action with G
                  Let B be the node with the highest score K.
                  Let D be the node with the second highest score.
                  Let L be the score for placing I in a new node C.
                  Let M be the score for merging B and D into one node.
                  Let T be the score for splitting D into its children.

                  IF K is the best score
                       THEN Case base (P, G) (place G in case class B).
                       ELSE IF L is the best score,
                            THEN Input a new node C
                                 Case base (N, G)
                            Else IF M is the best score,
                                 Then let  O be merged (B, D, N)
                                      Case base (N, G)
                                 Else IF T is the best score,
                                      Then Split (B, N)
                                           Case base (N, G)

Operations over Case base

Variables:  X, O, B, and D are nodes in the hierarchy.
            G is the new case

Incorporate (N, G)
    Update the prototype and the variance of case class N

Create new terminals (N, G)
    Create a new child W of node N.
    Initialize prototype and variance

Merge (B, D, N)
    Make O a new child of N
    Remove B and D as children of N
    Add the instances of P and R and all children of B and D to the node O
    Compute prototype and variance from the instances of B and D

Split (B, N)
    Divide Instances of Node B into two subsets according to evaluation criteria
    Add children D and E to node N
    Insert the two subsets of instances to the corresponding nodes D and E
    Compute new prototype and variance for node D and E
    Add children to node D if subgraph of children is similar to subgraph of node D
    Add children to node E if subgraph of children is similar to subgraph of node E
```

Fig. 9. Algorithm II.

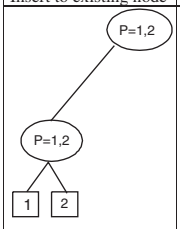
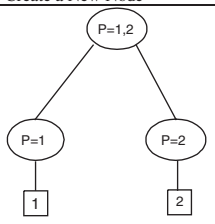
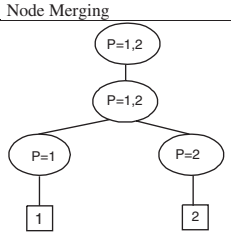
case class of the prototype with the highest similarity is determined. If we increase the threshold to 0.02, then we obtain the partition {1,2,4}{3}. This hierarchy still has two nodes but the time for matching is increased since now in one node there are three cases that should be matched.

The four cases are placed into four separate nodes each when using a threshold less than 0.001. Thus, for this hierarchy matching has to be performed four times.

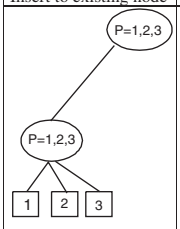
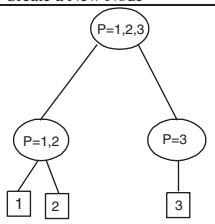
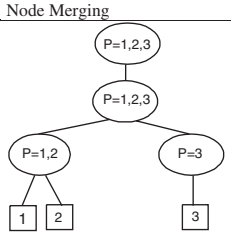
This example shows how the threshold affects the resulting hierarchy.



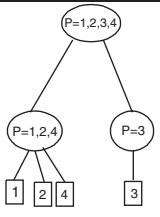
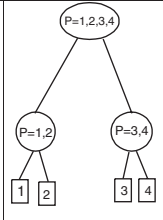
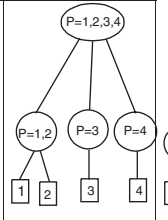
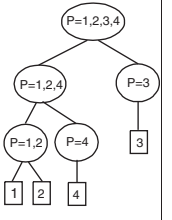
(a)

Insert Case 2		
Insert to existing node	Create a New Node	Node Merging
		
SB = 0	SB = 0.00018172	SB = 0.00018172
SW = 0.00018172	SW = 0	SW = 0
SCORE= 0.00018172	SCORE = 0.00018172	SCORE = 0.00018172
Resulting Case Base		

(b)

Insert Case 3		
Insert to existing node	Create a New Node	Node Merging
		
SB = 0	SB = 0.0255671	SB = 0.0255671
SW = 0.0156513	SW = 0.0001211	SW = 0.0001212
SCORE = 0.0156513	SCORE = 0.0254459	SCORE = 0.0254459
	** **Resulting Case Base** **	

(c)

Insert Case 4			
Insert to existing node_1	Insert to existing node_2	Create a New Node	Node Merging
			
SB = 0.0159367	SB = 0.0250232	SB = 0.0218856	SB = 0.0204
SW = 0.0120498	SW = 0.0008960	SW = 0.0000795	SW = 0.00009086
SCORE = 0.0038869	SCORE = 0.024127	SCORE= 0.021805	SCORE = 0.0203091
	Resulting Case Base		

(d)

Fig. 10. (a) Starting point, (b) insert case 2 into the case base, (c) insert case 3 into case base, and (d) insert case 4 into case and final case base.

Algorithm II automatically selects the best partitions {1,2} and {3,4}. The predetermination of a threshold is not necessary. This algorithm protects the user from a try-and-test procedure in order to figure out the best threshold for the similarity of his application and besides that it guarantees that the hierarchy will not grow too deep or too broad. The evaluation measure as well as the allowed operations over the hierarchy keep the hierarchy in balance according to the observed cases.

5. Related work

Bunke and Messmer (1994) used a network of model graphs for matching graphs which is created based on the subgraph isomorphism relation. They calculated off-line each possible permutation from each model graph and used them to construct the network. This network was then used for matching the graphs. In their approach they did not consider approximate graph matching, nor can their approach be used in an incremental fashion. The resulting structure is a network of model graphs not a directed graph. It also requires that the network is complete. In contrast, our approach can tolerate incompleteness. The approach of Bunke et al. is also used by Burke et al. (2001).

An approach for tree structured cases has recently been described by Ricci and Senter (1998). It assumes that cases are in a tree-structured representation or can be transformed into this structure which is not always possible.

The two approaches described in this paper are based on approximate graph subsumption and can be used in an incremental fashion. The first approach is based on a divide-and-conquer strategy and requires a fixed threshold for the similarity value. A similar approach has been used by Börner et al. (1996) for an application in building design. Their approach is not incremental and groups cases together which have different numbers of nodes. The importance of a concept is described by its probabilities and they assume a large enough case base.

An approach which uses such a flexible strategy as our approach II is to our knowledge not used by anyone for building the index structure of a CBR system.

6. Conclusions

We consider a case to be a structural representation such as an attributed graph. Such representations are useful to describe multimedia objects such as images, text documents, log-files or even in building design, software engineering, and timetabling. The similarity between these objects must be determined based on their structural relations as well as on the similarity of their attributes.

The retrieval and matching of structural representation are time-consuming processes and often NP complex. To reduce the time complexity, efficient organization of the case base is essential. We propose two approaches which enable cases to be organized in a hierarchical fashion so that we can discover the underlying concepts of the domain over time. The index hierarchy can be reorganized as new cases are learned. Cases can be incrementally stored in the retrieval hierarchy based on their similarity to other cases. Thus the index structure will change over time as new cases are learned. We consider this process as case-base maintenance and learning.

Our two approaches for maintaining and incrementally learning the index hierarchy are based on approximate graph subsumption. The first approach requires an off-line defined threshold for the similarity value. This as well as the order of cases presented to the system strongly influence the resulting hierarchy. Therefore, not the first stored case in a case class is taken as the class representative instead of a prototype incrementally calculated. However, this approach can only construct new case classes. Once a case class has been established, the structure cannot be reversed. This is only possible in the second approach since it can merge and split nodes at any position in the hierarchy. Another advantage of the second approach is that it does not rely on a fixed threshold for similarity. Instead, an evaluation function is used to select the right partition. This gives the flexibility needed when incrementally building the index.

The performance of the two approaches was evaluated on a data set of ultrasonic images from nondestructive testing. The image description of the reflection points of the defect in the image is represented as an attributed graph. The learning approach for the index structure was developed for real-time collection of ultrasonic images into the case base. It allows to update the case-base organization at any time a new case is collected.

Acknowledgement

Part of this work has been financed by the German Science Foundation under the grant Pe-456/1. This funding is gratefully acknowledged.

References

Bergmann, R., Stahl, A., 1998. Similarity measures for object-oriented case representations. In: Smith, B., Cunningham, P. (Eds.), Proceedings of the Advances in Case-Based Reasoning, LNAI 1488. Springer, New York, pp. 25–36.

Bischof, W., Caelli, T., 2000. Learning spatio-temporal relational structures. Applied Artificial Intelligence 15 (8), 707–722.

Bonzano, A., Cunningham, P., 1997. Learning feature weights for CBR: global versus Local. Advances in Artificial Intelligence 97, 417–426.

Börner, K., Pippig, E., Tammer, E.-Ch., Coulon, C.H., 1996. Structural similarity and adaptation. In: Smith, I., Faltings, B. (Eds.), Advances in Case-Based Reasoning. Spinger, New York, pp. 58–75.

Bunke, H., Messmer, B., 1994. Similarity measures for structured representations. In: Wess, S., Althoff, K.-D., Richter, M.M. (Eds.), Topics in Case-Based Reasoning. Springer, New York, pp. 106–118.

Burke, E.K., MacCarthy, B., Petrovic, S., Qu, R., 2001. Case-based reasoning in course timetabling: an attributed graph approach. In: Aha, D.W., Watson, I. (Eds.), Case-Based Reasoning Research and Development, LNAI 2080π. Springer, New York, pp. 90–103.

Craw, S., Jarmulak, J., Rowe, R., 2001. Maintaining retrieval knowledge in a case-based reasoning system. Computational Intelligence 17 (2), 346–363.

Fisher, D.H., 1987. Knowledge acquisition via incremental clustering. Machine Learning 2, 139–172.

Gebhardt, F., Voss, A., Gräther, W., Schmidt-Belz, B., 1997. Reasoning with Complex Cases. Kluwer, Dordrecht.

Gennari, J.H., Langley, P., Fisher, D., 1989. Models of incremental concept formation. Artificial Intelligence 40, 11–61.

Grimnes, M., Aamodt, A., 1996. A two layer case-based reasoning architecture for medical image understanding. In: Smith, I., Faltings, B. (Eds.), Advances in Case-Based Reasoning, LNAI 1168. Springer, New York, pp. 164–178.

Leake, D.B., Wilson, D.C., 1998. Categorizing case-based maintenance: dimensions and directions. In: Smyth, B., Cunningham, P. (Eds.), Advances in Case-Based Reasoning, LNAI 1488. Springer, New York, pp. 196–207.

McSherry, D., 2000. Intelligent case-authoring support in casemaker-2. In: Blanzieri, E., Portinale, L. (Eds.), Advance in Case-Based Reasoning, LNAI 1898. Springer, New York, pp. 198–209.

McSherry, D., 2001. Precision and recall in interactive case-based reasoning. In: Aha, D.W., Watson, I. (Eds.), Case-Based Reasoning Research and Development, LNAI 2080. Springer, New York, pp. 392–406.

Michalski, R.S., 1983. A theory and methodology of inductive learning. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (Eds.), Machine Learning: Artificial Intelligence Approach. Morgan Kaufmann, Los Altos, CA.

Nebel, B., 1990. Reasoning and Revision in Hybrid Representation Systems. Springer, New York.

Perner, P., 1993. Case-based reasoning for image interpretation in non-destructive testing. In: Richter, M. (Ed.), First European Workshop on Case-Based Reasoning, Otzenhausen, November 1993. Proceedings of SFB 314, vol. II, University of Kaiserslautern, pp. 403–410.

Perner, P., 1996. Ultrasonic image interpretation, In: IAPR Workshop on Machine Vision and Applications MVA'96, Tokyo, Japan, pp. 552–554.

Perner, P., 1998a. Different learning strategies in a case-based reasoning system for image interpretation. In: Smith, B., Cunningham, P. (Eds.), Advances in Case-Based Reasoning, LNAI 1488. Springer, New York, pp. 251–261.

Perner, P., 1998b. Using CBR learning for the low-level and high-level unit of a image interpretation system. In: Singh, Sameer (Ed.), Advances in Pattern Recognition. Springer, New York, pp. 45–54.

Perner, P., Paetzold, W., 1995. An incremental learning system for interpretation of images. In: Dori, D., Bruckstein, A. (Eds.), Shape, Structure, and Pattern Recognition. World Scientific Publishing Co., Singapore, pp. 311–323.

Portinale, L., Torasso, P., Tavano, P., 1999. Speed-up, quality and competence in multi-modal case-based reasoning. In: Althoff, K.-D., Bergmann, R., Branting, L.K. (Eds.), Case-Based Reasoning Research and Development, LNAI 1650. Springer, New York, pp. 303–317.

Ricci, F., Senter, L., 1998. Structured cases, trees and efficient retrieval. In: Smyth, B., Cunningham, P. (Eds.), Proceedings of the Advances in Case-Based Reasoning. Springer, New York, pp. 88–99.

Richter, M.M., 1998. Introduction. In: Lenz, M., Bartsch-Spörl, B., Burkhardt, H.-D., Wess, S. (Eds.), Case Based Reasoning Technology: From Foundations to Applications, LNAI 1400. Springer, Berlin.

Smyth, B., McKenna, E., 1998. Modelling the competence of case-bases. In: Smyth, B., Cunningham, P. (Eds.), Advances in Case-Based Reasoning, LNAI 1488. Springer, New York, pp. 208–220.

Surma, J., Tyburcy, J., 1998. A study of competence-preserving case replacing strategies in case-based reasoning. In: Smyth, B., Cunningham, P. (Eds.), Advances in Case-Based Reasoning, LNAI 1488. Springer, New York, pp. 233–238.

Veloso, M.M., 1997. Merge strategies for multiple case plan replay. In: Proceedings of ICCBR-97.

Wess, S., Althoff, K.-D., Derwand, G., 1993. Using *k–d* trees to improve the retrieval step in case-based reasoning. In: Wess, S., Althoff, K.-D., Richter, M.M. (Eds.), Topics in Case-based Reasoning. Springer, New York, pp. 167–182.

Wettscherek, D., Aha, D.W., Mohri, T., 1997. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artificial Intelligence Review 11 (1–5), 273–314.