On the Use of Taxonomies for Representing Case Features and Local Similarity Measures

Ralph Bergmann

University of Kaiserslautern
Centre for Learning Systems and Applications (LSA)
PO-Box 3049
D-67653 Kaiserslautern, Germany
bergmann@informatik.uni-kl.de

Abstract

For defining attribute types to be used in the case representation, *taxonomies* occur quite often. The symbolic values at any node of the taxonomy tree are used as attribute values in a case or a query. A taxonomy type represents a relationship between the symbols through their position within the taxonomy-tree which expresses knowledge about the similarity between the symbols. This paper analyzes several situations in which taxonomies are used in different ways and proposes a systematic way of specifying local similarity measures for taxonomy types. The proposed similarity measures have a clear semantics and are easy to compute at run-time.

1. Introduction

In current academic and commercial CBR systems, cases are often represented in an object-oriented fashion. Cases are collections of objects, each of which is described by a set of attribute-value pairs. The structure of an object is described by an object class that defines the set of attributes together with a type (set of possible values) for each attribute. Usually, the similarity between a query and a case from the case base is computed in a bottom up fashion: for each attribute, a local similarity measure determines the similarity between two attribute values and for each object (and the case) a global similarity measure determines the similarity between two objects (or between the case and the query) based on the local similarities of the belonging attributes (Wess, 95).

For defining attribute types (sets of possible attribute values), taxonomies are widely used. A taxonomy is an n-ary tree in which the nodes represent symbolic values. The symbols at any node of the tree can be used as attribute values in a case or a query. Unlike a plain symbol type, which only lists possible attribute values, a taxonomy represents an additional relationship between the symbols through their position within the taxonomy-tree. This relationship expresses

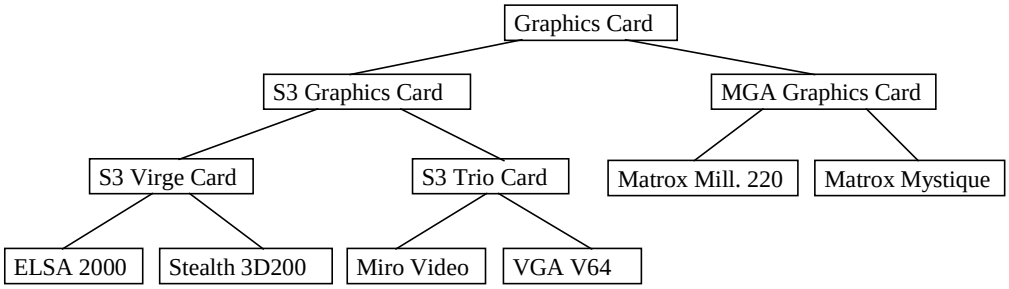

Fig. 1. Taxonomy of Graphics Cards.

knowledge about the similarity of the symbols in the taxonomy. The case representation language CASUEL (Manago, Bergmann, et al. 1994) includes such a taxonomy type, which is very often used for modeling a CBR application. For example, the "classification of marine sponges" application contains a taxonomy of body forms of sponges and the PC configuration domain contains a taxonomy of PC components".

Although taxonomies are widely used, there is currently no clear picture of what knowledge about local similarities is captured in a taxonomy. The impression that similarity measures are usually constructed in an ad hoc manner (Osborne & Bridge, 1996) also holds for local similarity measures for taxonomy type attributes. This paper analyzes several situations in which taxonomies are used in different ways and proposes a systematic way of specifying local similarity measures for taxonomy types. The proposed similarity measures have a clear semantics and are easy to compute at run-time.

2. Examples for Different use of Taxonomies

We now describe four examples in which the taxonomy shown in Figure 1 is used.

Example 1a. Consider a CBR system for the sales support of Personal Computers. A Case represents an available PC from the stock. The case representation contains an attribute „graphics card", and the taxonomy from Figure 1 represents the set of possible values. Consider a case c_1 with the *ELSA 2000* card and a case c_2 with *Matrox Mystique* card. If we assume that a customer enters a query to our hypothetical CBR system in which she/he specifies that she/he wants a *Miro Video* graphics card, then c_1 is certainly closer than c_2, because *Miro Video* and *Elsa 2000* have more in common (e.g. the S3 chip) than the *Miro Video* and the *Matrox Mystique*. In general, we could use a similarity measure that assesses similarity based on the distance between case and the query value in the taxonomy tree.

Example 1b. Imagine, the customer states in the query a request for a *S3 Graphics Card*. Then, any of the graphics cards in the S3 sub-tree are perfectly suited.

Hence, we expect the local similarity value between this query and case c_1 (from example 1a) to be 1. From this consideration we can conclude that whenever the query value is located above the case value, the similarity measure should be 1.

Example 2a. Consider a trouble-shooting CBR system for PCs in which cases encode diagnostic situations and faults that have occurred previously. The domain expert describes a fault that can occur with any S3 card. Therefore, the respective case contains the attribute value *S3 Graphics Card*. Assume now, a PC user has a problem and she/he states that there is an *Elsa 2000* card in the PC, than the local similarity for the graphics card attribute should be equal to 1 because the case matches exactly w.r.t. this attribute. From this consideration we can conclude that whenever the case value is located above the query value the similarity measure should be 1.

Example 2b. For the same trouble-shooting example, imagine now a different query in which the user does not exactly know what kind of graphics card is installed in the PC, but she/he knows that it is a *S3 Trio* card. She/He therefore enters *S3 Trio* as attribute value in the query. Again, the case about *S3 cards* mentioned in Example 2a matches exactly because, whatever graphics card the user has, we known it is an *S3 card* and the situation described in the case applies. However, if we consider a different case that describes a problem with the *Miro Video* card, then this case does not match exactly. Since we don't know what graphics card the user has (it can be a *Miro Video* but it can also be a *VGA V64*) we expect a local similarity value less than 1. Therefore we cannot conclude that whenever the query value is located above the case value the similarity measure should be 1.

Although we have used the same taxonomy in all four examples, it is obvious that they have to be treated differently for the similarity computation. In the query and cases from example 1a, only leaf nodes from the taxonomy are used. The examples 1b to 2b make use of inner nodes of the taxonomy, but in each example the semantics of the inner nodes is different which lead to different similarity measures.

3. Knowledge Contained in Taxonomies

We now analyze the knowledge that is contained in taxonomies. We will show that a taxonomy contains two different kinds of knowledge:

1. Knowledge about classes of objects[1] (represented by inner nodes).
2. Knowledge about the similarity between leaf nodes.

[1] Here, the word object is not meant in the sense of the object-oriented paradigm.

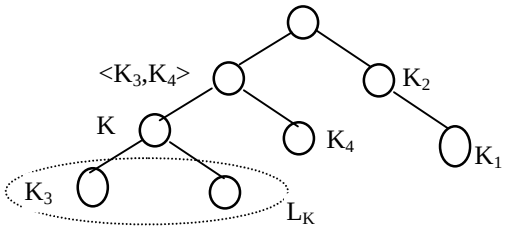

Fig. 2. Illustration of basic notions

3.1 Basic Notions

We briefly introduce a few notions (see Fig. 2) that will be further used in the paper. Let *K* be an inner node of the taxonomy, then L_K denotes the set of all leaf notes from the sub-tree starting at *K*. Further, $K_1 < K_2$ denotes that K_1 is a successor node of K_2, i.e., K_2 is on a path from K_1 to the root node. Moreover, $<K_3,K_4>$ stands for the node that is the nearest common predecessor of K_3 and K_4, i.e., $<K_3,K_4> \geq K_3$ and $<K_3,K_4> \geq K_4$ and it does not exist a node $K' < <K_3,K_4>$ such that $K' \geq K_3$ and $K' \geq K_4$ holds.

3.2 Semantic of Taxonomy Nodes

In a taxonomy, we must distinguish between leaf nodes and inner nodes. Leaf nodes represent *concrete objects*[1] of the real world, e.g., existing graphics cards. Inner nodes, however, describe *classes* of real world objects. An inner node *K* represents a class with certain properties that all of the concrete objects from the leaf nodes L_K have in common. Unlike classes that occur in the object-oriented paradigm, the classes that are represented by the inner nodes of a taxonomy are not described intentionally by a set of properties, but extensionally through the set of concrete objects L_K that belong to the class. Therefore, an inner node K stands for the set L_K of real world objects.

In the taxonomy shown in Figure 1, the leaf nodes represent existing graphics cards and the inner nodes represent classes of graphics cards. For instance, *S3 Virge* stands for all graphics cards with the S3 Virge chip on them, i.e, for the set of cards {*Elsa 2000, Stealth 3D 2000*}.

When a CBR application developer builds a taxonomy, she/he should introduce useful sets of real-world objects, i.e., sets that are likely to occur in a case or a query. The taxonomy defines unique names (like *S3 Virge*) for these sets which are then used as abbreviations. Since the sets that are represented by these inner nodes are defined by the taxonomy itself, they are the same in all of the examples shown above, e.g., *S3 Virge* always stands for {*Elsa 2000, Stealth 3D 2000*}. However, the meaning of this set is quite different in the examples as we will discuss in detail in section 3.5.

3.3 Similarity Between Leaf Nodes

Besides the definition of classes of objects, a taxonomy also encodes some knowledge about the similarity of the real-world objects, i.e., knowledge about the similarity of the leaf nodes of the taxonomy. The inner nodes cluster real-world objects that have some properties in common. The deeper we decent in the taxonomy, the more features do the objects, that the inner node represents, have in common. For example, all real-world objects (leaf nodes) from the hierarchy in Fig. 1 have in common that they are all graphics cards. The objects below the *S3 graphics card* node have in common that the all use some kind of S3 chip, and the objects below the *S3 Trio* node have in common that they all use the specific S3 Trio chip. We can now define local similarity as a measure of how many features the compared objects have in common. The more features are shared, the higher is the similarity. For example, the similarity between *Elsa 2000* and *Stealth 3D200* is higher than the similarity between *Elsa 2000* and *VGA V64*.

This consideration leads to the following general constraint for defining the local similarity measure for the leaf nodes of a taxonomy:

(1) $sim(K, K_1) \leq sim(K, K_2)$ IF $\langle K, K_1 \rangle > \langle K, K_2 \rangle$

It states that the similarity between the leaf node K and K_1 is smaller than the similarity between the leaf node K and K_2 if the nearest common predecessor of K and K_1 is located below the nearest common predecessor of K and K_2. It does not state anything about the relationship between $sim(K_1,K_2)$ and $sim(K_3,K_4)$ unless $K_1=K_3$. Please note that this constraint defines an ordinal similarity measure, i.e., if the value K is given in the query, a partial order of all cases is induced. A similar approach can be also found in (Osborne and Bridge, 1996).

3.4 Assigning Similarity Values for Leaf Nodes

The taxonomy only represents the constraint shown above, but does not define numeric values for the similarity between two leaf node objects. However, several models of similarity computation require a numeric value (e.g. from the interval [0..1]) to express the local similarity, because this value is further used in the computation of a global similarity. For this purpose, we have to add additional knowledge to the taxonomy. Basically, there can be different ways of doing this in a way, that the resulting cardinal similarity measure is compatible with the constraint. We now present a new approach which is quite simple and easy to use, but nevertheless very powerful.

Every inner node K_i of the taxonomy is annotated with a similarity value $S_i \in [0..1]$, such that the following condition holds: if $K_1 > K_2$ then $S_1 \leq S_2$. The deeper the nodes are located in the hierarchy, the larger the similarity value can become. The semantic of the similarity value is as follows:

The value S_i represents a lower bound for the similarity of two arbitrary objects from the set L_{Ki}, or written formally: $\forall x, y \in L_{K_i}\ sim(x, y) \geq S_i$

Any two objects from L_{Ki} are at least similar to each other with the value S_i, but their similarity can be higher. The similarity value that is assigned to a node should be justified by the features that all of the objects that belong to this inner node (class) have in common. The fact that the objects belong to this class and have common properties justifies that we can state a lower bound for their similarity. However, objects belonging to one class can of course also belong to a more narrow class further down in the taxonomy, which means that these objects share even more properties and therefore possibly have a higher similarity. We therefore define the similarity between to objects as follows:

$$(2)\quad sim(K_1, K_2) = \begin{cases} 1 & \text{if } K_1 = K_2 \\ S_{\langle K_1, K_2 \rangle} & \text{otherwise} \end{cases}$$

where $S_{<K1,K2>}$ denotes the similarity value assigned to the node $<K_1,K_2>$, i.e., the nearest common predecessor of K_1 and K_2. It can be shown that this similarity definition preserves constraint (1).

3.5 Semantic and Similarity of Inner Nodes

If we now recall again the examples that we have presented in section 2, it is obvious that the "graphics card" attribute must be treated differently in the different examples, although they all use the same taxonomy. From that it becomes clear, that some additional knowledge which we have not yet discussed, plays a role during similarity assessment. However, this knowledge is not contained in the taxonomy itself.

The knowledge that we are looking for is the knowledge about the semantic of the inner nodes, i.e., the semantic of the set of concrete objects that they represent. In our example, the question is: what does it mean when the case or query contains the statement: "graphics card: S3 Graphics Card"?

In fact, there are different interpretations of this statement that are now discussed.

Any value in the query. The user specifies the value K in the query. This means that she/he is looking for a case that contains one of the values from the set L_K. In the example 1b, the user wants an S3 graphics card, but he does not care whether it is a *Elsa 2000, Stealth 3D 200, Miro Video,* or an *VGA V64.* It is clear that the local similarity between this query and any of these four leaf nodes is equal to 1. But what about the similarity to any other leaf node? To answer this question more systematically, we can define the required retrieval result indirectly as follows: Instead of submitting a single query to the system that contains an inner node *K*, the user could alternatively submit several queries to the system, one for each concrete value from L_K and merge the retrieval results, i.e., select the case

with the highest similarity. The result of using the query with the inner node *K* should yield the same case with the same similarity as the merging of the multiple retrievals. To achieve this, the similarity measure for the inner node must select the maximum similarity that arises from each of the leaf nodes.

Any value in case. The case contains an inner node K, which describes a situation in which the case is valid for all attribute values of the set L_K. This leads to a kind of generalized case. The generalized case (in which the attribute value *K* is used) stands for the set of cases that results by replacing *K* by all of the members of the set L_K. In Example 2a, the case representing a fault for any *S3 graphics card* stands for a set of four cases, each of which represents a fault for the *Elsa 2000*, *Stealth 3D 200*, *Miro Video*, and the *VGA V64*, respectively. Here, the inner node is used to keep the number of cases in the case base small. However, the retrieval result should of course not be affected. Therefore, the result of having a case in the case-base that contains an inner node K should be the same as having all cases in the case base, one for each concrete value from L_K. Since we are looking for the most similar case, we again have to assess the similarity for the inner node by selecting the maximum similarity that arises from each of the leaf nodes.

Uncertainty. This situation differs significantly from the previous two. Here, the use of an inner node K means that we don't know the exact value for this attribute, but we know that it is one from the set L_K. In Example 2b, we know that the user has a *S3 Trio* card which means it can be one from the set {*Miro Video, VGA V64}*. This kind of uncertainty can occur in queries as well as in cases. The user can think of this uncertainty in different ways: treating it optimistically, pessimistically, or as an average case.

We can now define the local similarity SIM(Q,C) between a query value Q and a case value C each of which can be either a leaf node, an inner node with the "any value" interpretation or an inner node with the "uncertainty" interpretation. This leads to 9 possible combinations shown in Table 1. Seven of the 9 combinations in the table are marked with a roman number that is further used to reference the formulas for computing the similarity. These are the ones that occur most likely. However, the following considerations can easily be extend also to the two missing combinations.

Query \ Case	**Leaf Node**	**Any Value**	**Uncertainty**
Leaf Node	I	II	V
Any Value	III	IV	
Uncertainty	VI		VII

Tab. 1. *Combinations of different semantics for taxonomy values in query and case*

In the following, *sim(q,c)* denotes the similarity between two leaf nodes, q from the query and c from the case. It can be computed as shown in section 3.4.

I: Only the similarity between leaf nodes is computed and hence *SIM(Q,C)* =

sim(Q,C) holds.

II: The query contains a leaf node and the case contains an inner node representing a set of values each of which is a correct value for the case. Therefore, the use of this set in the attribute is a shortcut for the use of several cases, one for each value in the set. Since we are looking for the most similar case in the cases base, the similarity between the query and our case containing the inner node is equal to the highest similarity between the query and one of the values from the set. Hence:

$$SIM(q,C) = \max\{sim(q,c) | c \in L_C\} = \begin{cases} 1 & \text{if } q < C \\ S_{\langle q,C \rangle} & \text{otherwise} \end{cases}$$

holds. This definition ensures, that the similarity is the same as the similarity that arises when each of the cases with leaf node values would have been stored in the case base. This measure is appropriate for example 2a.

III: Here, the specification of this inner node can be viewed as a shortcut for posing several queries to the system, one for each of the values from the set that the node represents. Since we are again interested in the most similar case, we can again select the most similar attribute value from the set. Hence:

$$SIM(Q,c) = \max\{sim(q,c) | q \in L_Q\} = \begin{cases} 1 & \text{if } c < Q \\ S_{\langle Q,c \rangle} & \text{otherwise} \end{cases}$$

holds. This measure is appropriate for example 1b.

IV: This is a combination of II and III. We are looking for the highest possible similarity between two objects from the two sets since both, the query and the case, represent alternatives that are suited equally well. Hence,

$$SIM(Q,C) = \max\{sim(q,c) | q \in L_Q, c \in L_C\} = \begin{cases} 1 & \text{if } C < Q \text{ or } Q < C \\ S_{\langle Q,C \rangle} & \text{otherwise} \end{cases}$$ holds.

V: The case contains an inner node which represents a set of values from which only one value is actually correct for the case, but we don't know which one. Therefore, our similarity measure has to reflect this lack of information. There are three possible approaches: we can assess the similarity in a pessimistic or optimistic fashion, or we can follow an averaging approach:

Pessimistic approach: We assess the similarity between the known object (in the query) and the partially unknown object (in the case) by computing the lower bound for the similarity as follows: $SIM(q,C) = \min\{sim(q,c) | c \in L_C\} = S_{\langle q,C \rangle}$.

Optimistic approach: We assess the similarity between the known object (in the query) and the partially unknown object (in the case) by computing the upper bound for the similarity, which results in the same formula that was already shown in III.

Average approach: We assess the similarity between the known object (in the query) and the partially unknown object (in the case) by computing the expected value of the similarity as follows: $SIM(q,C) = \sum_{c \in L_C} P(c) \cdot sim(q,c)$, where *P(c)* is the probability that the value of the attribute under consideration has the value c given the fact that we know that $c \in L_C$ and given the known information about the current case. Since P(c) is hard to determine, we can, for example, estimate P(c) by $1/|L_C|$, assuming that all attribute values are equally distributed and that all attributes are independent.

VI: The uncertain information is contained in the query; the information in the case is certain. This case is quite similar to the previous case V, i.e., we can again use a pessimistic, an optimistic, or an average approach. The only change in the formulas for similarity computation is the fact that the minimum, maximum, and sum operations are now performed using the elements from the query L_Q and not the elements form the case.

VII: The uncertain information is contained in the query and in the case. The similarity is computed as follows:

Pessimistic approach: $SIM(Q,C) = \min\{sim(q,c) | c \in L_c, q \in L_Q\} = S_{\langle Q,C \rangle}$

Optimistic approach: $SIM(Q,C) = \max\{sim(q,c) | q \in L_Q, c \in L_c\} = \begin{cases} 1 & \text{if } C < Q \text{ or } Q < C \\ S_{\langle Q,C \rangle} & \text{otherwise} \end{cases}$

Average approach: $SIM(Q,C) = \sum_{c \in L_C, q \in L_Q} P(c) \cdot P(q) \cdot sim(q,c)$.

We see that in all of these cases (except for the average approach to uncertainty), similarity between inner nodes can be computed very easily by determining the position of the query and the case value in the taxonomy and by looking up the similarity value at the appropriate taxonomy node. This enables the use of taxonomies in CBR.

4. Conclusion

We have shown that taxonomies represent two kinds of knowledge: first, knowledge about classes of objects and second, knowledge about the similarity between leaf nodes which represent real-world objects. We have presented a new approach for defining a numeric similarity-value between leaf nodes by assigning similarity values to the inner nodes of the taxonomy. Moreover, we have shown that additional knowledge is required to decide how the similarity between inner nodes of the taxonomy can be computed. This knowledge states how the classes (set of real-world objects) have to be interpreted: as any value from the set or as a kind of uncertainty. However, independent on the kind of interpretation, there is a quite simple way of computing the similarity between two inner nodes, if the

proposed approach to determine the similarity between leaf nodes is used.

From these considerations we can see that a taxonomy can be used (and should be used because of the simple computation of similarities) if

- an attribute shall contain a set of values in the query and/or in the case *and*
- these sets represent either uncertainty *or* a list of equally well suited objects *and*
- we can define in advance a hierarchy of disjoint sets of similar objects that can occur in the query or the case.

These three rules of thumb can be used as guidelines within a similarity definition method of a case-based reasoning methodology (Bergmann et al., 1997).

In our discussion, we restricted ourselves to taxonomies of basic objects which don't have an internal structure. However, our considerations also apply to the generalization/specialization hierarchy of the object classes in an object-oriented data (or case) model. This inheritance hierarchy is of the same nature than the taxonomies we have just discussed. The only difference is that the objects, which are instances of classes, have an additional internal structure, i.e., each object is described by a set of attributes. Therefore, we can extend the approach presented here to a similarity framework for comparing cases using an object-oriented case representation. This issue will be the topic of a forthcoming paper.

Acknowledgment

Funding for this work has been provided by the Commission of the European Union (INRECA-II: Information and Knowledge Reengineering for Reasoning from Cases; Esprit contract no. 22196) to which the author is greatly indebted. The partners of INRECA-II are: AcknoSoft (prime contractor, France), Daimler Benz (Germany), TECINNO (Germany), Irish Multimedia Systems (Ireland), and the University of Kaiserslautern (Germany). Further thanks to Frank Heister, Armin Stahl, Stefan Wess, Wolfgang Wilke, Max Wolf, and Andreas Zimmermann for several discussions about the topic of the paper and special thanks to Wolfgang Wilke for additional comments on the final version.

References

Bergmann, R., Wilke, W., & Schumacher, J. (1997). Using software process modeling for building a case-based reasoning methodology: Basic approach and case study. In: D. Leake & E. Plaza (eds.) Case-Based Reasoning Research and Development (ICCBR'97). Lecture Notes in AI. Springer, pp. 509-518.

Manago, M. Bergmann, R. et al. (1994). CASUEL: A common case representation language. Deliverable D1 of the INRECA Esprit Project.

Osborne H. and Bridge,D. (1996). A case base similarity framework. In Smith & Faltings (Eds.) *Advances in Case-Based Reasoning.* Proceedings of EWCBR'96. Springer.

Wess, S. (1995). *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungs-unterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen*. Dissertation, Universität Kaiserslautern, Infix-Verlag.