Improving KD-Tree Based Retrieval for Attribute Dependent Generalized Cases

Ralph Bergmann
University of Trier
54286 Trier, Germany
www.wi2.uni-trier.de
Email: bergmann@uni-trier.de

Alexander Tartakovski
Piterion GmbH
Hanns-Klemm-Str. 5
71034 Böblingen, Germany
Email: alexander.tartakovski@piterion.com

Abstract

Generalized cases are cases that cover a subspace rather than a point in the problem-solution space. Attribute dependent generalized cases are a subclass of generalized cases, which cause a high computational complexity during similarity assessment. We present a new approach for an efficient index-based retrieval of such generalized cases by an improved kd-tree approach. The experimental evaluation demonstrates a significant improvement in retrieval efficiency compared to previous methods.

Introduction

The design of a case-based reasoning (CBR) application is strongly influenced by the nature, complexity, and number of the cases. CBR applications in which cases can be represented adequately in a structured manner by a set of independent features are quite well understood. Commercial CBR tools (e.g. e:IAS by empolis) enable efficient similiarity-based retrieval of huge case bases with millions of cases.

Several applications, however, require more sophisticated case representations and thereby introduce a particular complexity into similarity assessment, retrieval, and adaptation. This paper deals with one such class of case representations, namely *attribute dependent generalized cases* (Maximini, Maximini, & Bergmann 2003). These are cases which are *generalized* in the sense that they cover a (possibly infinite) subspace rather than a point in the problem-solution space. They are *attribute dependent* in the sense that all or some attributes of an individual case depend on each other. The kind of dependency may differ from case to case. A representation of a generalized case therefore requires the use of constraints, i.e. a case is described by a set of attributes together with a set of constraints over these attributes. The used constraints and the attributes which are constraint may differ from case to case.

Attribute dependent generalized cases occur, for example, in CBR applications for the recommendation of parameterized or configurable products within electronic commerce or brokerage services. A single generalized case represents the different variants of a product and the dependencies in the generalized case reflect the dependency of the product features. Examples of such applications in our previous work are brokerage of electronic chip designs (Mougouie & Bergmann 2002) and sales support for life insurance policies (Tartakovski, Schaaf, & Bergmann 2005).

In our previous research we investigated representation (Maximini, Maximini, & Bergmann 2003; Bergmann & Vollrath 1999), similiarity assessment (Mougouie & Bergmann 2002) and retrieval (Tartakovski *et al.* 2004) of generalized cases. With prototypically implemented CBR applications in the two above mentioned domains we could demonstrate a sound approach for similarity assessment by converting the similarity assessment problem into an optimization problem (in the worst case a mixed integer non-linear optimization problem). However, this approach introduces a significant computational complexity for each similarity assessment instance. Therefore, efficient retrieval requires appropriate index structures that reduce the number of similarity assessments requited to find the most similar cases. We proposed the use of kd-trees (Wess, Althoff, & Derwand 1993) as such index structures (Tartakovski *et al.* 2004) and demonstrated in principle their ability to improve the retrieval (Tartakovski, Schaaf, & Bergmann 2005) by reducing the number of required similarity assessments. However, the advantage of this kd-tree based retrieval is quite limited, and particularly when the number of attributes increases, its ability to speed up the retrieval nearly vanishes.

This paper continues this line of research by proposing a new, significantly improved kd-tree based retrieval approach for attribute dependent generalized cases. The proposed extensions of standard kd-trees by introducing *priority queues* and *dynamic bounds* during retrieval is original and the experimental evaluation shows significant performance improvements.

Representation and Similarity of Generalized cases Revisited

We now briefly summarize the foundations of representation and similarity assessment for attribute dependent generalized cases (Mougouie & Bergmann 2002; Maximini, Maximini, & Bergmann 2003; Tartakovski *et al.* 2004) as prerequisite for the subsequent description of the new retrieval approach.

Representation

Let $\mathcal{A}$ be the (possibly infinite) representation space for cases which is structured into n attributes with data types $T_1 \ldots T_n$, i.e., $\mathcal{A} = T_1 \times \ldots \times T_n$

A traditional case (also called point case) c is a point in the representation space, i. e., $c \in \mathcal{A}$. A *generalized case* gc, however, is a subset of the representation space, i. e., $gc \subseteq \mathcal{A}$. Hence, a generalized case stands for a possibly infinite set of point cases. Depending on the structure of the set, two sub-types can be distinguished. The simpler type is the attribute independent generalized case (AIGC) defined as the Cartesian product of some sets $\mathbb{S}_1 \subseteq T_1, \ldots, \mathbb{S}_n \subseteq T_n$, each of which describes an independent set of values for an attribute. Hence, $AIGC = \mathbb{S}_1 \times \ldots \times \mathbb{S}_n$.

The second sub-type is the attribute dependent generalized case (ADGC). This is the class of generalized cases that cannot be represented as attribute independent generalized cases. Hence, a generalized case is attribute dependent, if the subspace it represents cannot be decomposed into independent subsets for each attribute. Dependencies among attributes must be represented by constraints with arity ≥ 2. Applying constraints in CBR is not new. Particularly in design tasks, similar ideas can be found. In our work, we use constraints for representing dependencies between attributes. The vocabulary for representing generalized cases consists of the representation space $\mathcal{A} = T_1 \times \ldots \times T_n$ and of a set of variables $V = \{v_1, \ldots, v_n\}$, one variable for each attribute, such that v_i holds values from T_i. A generalized case is represented by a set of constraints:

$$\mathbb{GC} = \{C_1, \ldots, C_l\}$$

such that $V_i := \text{Var}(C_i) \subseteq V$.[1] Such a generalized case represents the set of point cases whose attribute values fulfill all constraints, i.e.,

$$gc = \{(v_1, \ldots, v_n) \,|\, \forall_i C_i \text{ is fulfilled by } v_1, \ldots, v_n\}.$$

Similarity Assessment

For retrieving generalized cases, the similarity between a query and a generalized case must be determined. As in traditional CBR, we assume that the query is a point in the representation space that may be only partially described. We further assume that a traditional similarity measure $\text{sim}(q, c)$ is given which assesses the similarity between a query q and a point case c. Such a similarity measure can be extended in a canonical way to assess the similarity $\text{sim}^*(q, gc)$ between a query q and a generalized case gc: The similarity measure

$$\text{sim}^*(q, gc) := sup\{\text{sim}(q, c) \,|\, c \in gc\}$$

is called the *canonical extension* of the similarity measure sim to generalized cases.

Applying sim* ensures that those generalized cases are retrieved that contain the point cases which are most similar to the query. Rephrased more formally: given a case base CB of point cases and a similarity measure sim, then for any case base CB^* of generalized cases such that $CB = \bigcup_{gc \in CB^*} gc$ holds: if c_{ret} is a case from CB which is most similar to a query q w. r. t. sim then there is a generalized case gc_{ret} from CB^* which is most similar to q w. r. t. sim* such that $c_{ret} \in gc_{ret}$.

[1]Var(C) denotes the set of variables of the constraint C; V_i is an abbreviation for the set of variables that occur in C_i.

This states that if we introduce generalized cases together with sim*, the same cases are retrieved during problem solving, independent from the clustering of point cases into generalized cases. This provides a clear semantics of generalized cases defined in terms of traditional point cases and similarity measures.

Similarity Assessment as Optimization Problem

Bergmann and Vollrath (1999) have shown that this similarity assessment problem for attribute dependent generalized cases can be viewed as a specific optimization problem. An optimization problem is the maximization or the minimization of some objective function, often under restrictions given through equalities and inequalities. In general, optimization problems are defined as follows:

$$\begin{array}{ll} \max\limits_{x} & f(x) \\ s.t. & x \in F \end{array}$$

The function f is called an *objective function* and the set of all feasible values F a *feasible set*, implicitly defined through constraints. By defining an objective function $f(x) := sim(q, x)$ and the feasible set $F := GC$ the similarity assessment problem is transformed into an equivalent optimization problem.

In mathematical optimization several classes of optimization problems are known. They differ in computational complexity, problem solution methods and problem formulation. Therefore, it is important to find out the proper class and a respective formulation as optimization problem. In the domains we have investigated so far, cases are represented with both symbolic (discrete) and numeric (integer and real values) attributes. The resulting optimization problems therefore belong to the class of mixed integer nonlinear optimization problems (MINLP). Tartakovski (2008) developed general framework for transforming a large class of typically occurring constraints and similarity functions into an equivalent MINLP formulation. Thereby, the similarity assessment problem can in principle be solved at least approximately, e.g. by using a commercial solver for MINLP problems, such as GAMS/Baron, Xpress-SLP, and MINLP.

Retrieval of Generalized Cases

Because of the high computational complexity of the similarity assessment problem for generalized cases by solving the respective MINLP problem (see e.g. (Tartakovski, Schaaf, & Bergmann 2005) for experimental results), it is very important to develop index-based retrieval approaches. The overall strategy is to build an index structure in advance, which helps finding the most similar generalized case without the need to sequentially assess the similarity between the query and each generalized case in the case base. The less similarity assessment computations are needed, the faster the retrieval is.

Buidling kd-Tree for Generalized Cases

We now introduce a retrieval approach which applies a kd-tree (Bentley 1975; Friedman, Bentley, & Finkel 1977) as an index structure for generalized cases. The application of kd-trees for similarity-based retrieval in CBR was first proposed by Wess et al. (1993); Tartakovski et al. (2004) sketched an initial proposal of how it could be applied for generalized cases.

The basic idea behind a standard kd-tree is to partition an n-dimensional attribute space into some simple subspaces, e.g. hypercubes having faces parallel to the coordinate planes, and to search for nearest neighbors in that subspaces excluding several of them from the search focus (see figure 1).

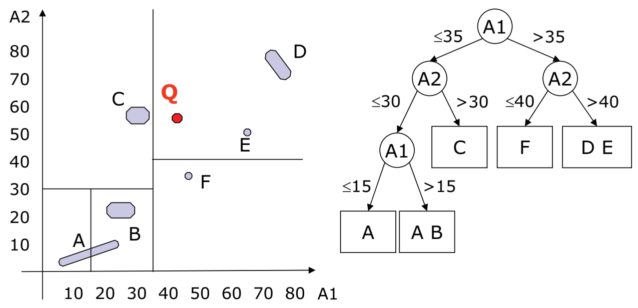


Figure 1: kd Tree for Generalized Cases

A kd-tree is a binary tree with nearly the same structure as a decision tree. Its inner nodes are labeled with attribute names, the edges with separation constraints. The leave nodes (also called *buckets*), are labeled with a disjoint subset of the cases. Every node of a kd-tree represents a subset of the case base. The root node represents the whole case base. Each inner node, labeled e.g. with an attribute A_i, partitions the represented set of cases into two disjoint subsets.

The construction of the kd-tree is organized recursively, similar to the construction of a decision tree. Beginning with a root node the represented set of cases is partitioned according to a *chosen discriminator attribute* and a chosen *separation value*. This recursive procedure carries on until a certain *termination criterion* is fulfilled.

When applying this algorithm to generalized cases, the methods for the selection of the discriminator attribute and separation value as well as the procedure for partitioning the cases that belong to a node into the respective subsets are becoming different compared to standard kd-trees (Bentley 1975; Friedman, Bentley, & Finkel 1977). The first extension is made in the partitioning function. This function contains a test checking if a given case belongs to a given subspace of the description space. This check is quite simple for point cases, but not for generalized cases. A generalized case gc belongs to some subspace S if and only if their intersection $gc \cap S$ is not empty. In general a set of generalized cases associated with a current node cannot be divided into two disjoint subsets described above. The reason for this is that a single generalized case may have non-empty intersection sets with the both hypercubes given by the child nodes (e.g. case A in figure 1). Hence, for case bases including generalized cases the usual requirement of disjunctive successor nodes must be dropped. The test providing information whether there is an intersection between a given generalized case and a hypercube is a special case of a *feasibility problem* (Horst & Tuy 1993) and can be performed by various optimization software.

The heuristic for determining the discriminator attribute aims at producing a balanced kd-tree and to reduce a backtracking during the search. It determines the projections of generalized cases onto the attribute's range (see figure 2). It must be pointed out that the standard heuristic for kd-trees cannot be applied since the projections are not points, but intervals. The projection of a single generalized case can be found by solving two special optimization problems. Analogous to the standard kd-tree selection heuristic, our heuristic prefers among other things attributes with a great dispersion of projections of cases. Figure 2 shows two attributes having a smooth dispersion of projections of cases. However, there is a significant difference between them: the left attribute allows to find a splitting value (here 45) such that the two resulting partitions contain half of the original cases in the node. The right attribute does not enable such a splitting. Our heuristic for attribute selection therefore, does not only regard the dispersion, but also lengths and intersections of the intervals. The details are described by Tartakovski (2008).

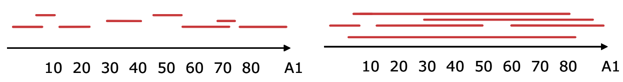


Figure 2: Projections for different attributes

Retrieval Using Standard kd-Tree Search

During retrieval the kd-tree is traversed to find the most similar generalized cases for a given query q. This approach only differs in minor details from the standard kd-tree retrieval for point cases as described by Wess et al. (1993).

In a nutshell, the algorithm is a recursive tree search procedure starting at the root node of the kd-tree. It descends the tree following the branch whose separation constraint matches the current value for this attribute in the query. If a bucket is reached, the similarity sim* between the query and all generalized cases in the bucket is computed. A sorted similarity list stores the k most similar generalized cases determined thereby. Then, two particular tests, originally called Ball-Within-Bounds and Ball-Overlap-Bounds test (Bentley 1975) are used to determine whether cases that are more similar than the k most similar case identified so far could exist in a neighbor bucket or subtree. In this case, the sibling bucket or subtree is examined as well and traversed to the buckets. This retrieval approach guarantees that the k most similar cases are found. However, in the worst case it can become necessary to search the whole case base depending on the outcome of the two tests during retrieval.

Retrieval Using Priority Queues

Arya et al. (1993) present a completely different search algorithm for original kd-trees called *priority kd-tree search*. The main idea of this method is to estimate the shortest distances between a query and all regarded nodes/hypercubes (see figure 1). The node having the shortest distance (highest similarity) to the query is considered as a node with a high chance of containing the most similar elements in the sub-tree below. Hence, it is examined before nodes with a higher distance.

The original algorithm developed by Arya et al. implements a search for the first nearest neighbor only and is restricted to the use of the Euclidian distance as a comparative measure. We have adapted and transferred the idea behind this algorithm for the purpose of retrieving the k most similar generalized cases applying an arbitrary similarity measures. The algorithm (see figure 3), maintains a list $nodes$ of open nodes during search in the kd-tree data structure. This is what Arya et al. calls a priority queue. This list is sorted according to the similarity of the nodes (hypercubes) to the query, such that the nodes with the hightest similarity are listed first. Further, the algorithm maintains a list scq of the already found k most similar cases sorted according to their similarity to the query. Finally, a $cache$ of cases is introduced to avoid that the similarity of a case is computed more than once. This is necessary since a generalized case may be included in more than one kd-tree bucket.

```
Input: kd tree root node $T$, query $q$
Output: Sorted list of cases $scq$

procedure Retrieve($T, q$)
  $nodes$: List of nodes sorted by similarity
  $scq$: Sorted list of best k cases
  $cache$: Set of cases
  $s : [0,1]$
  $T', T''$: kd-tree nodes
begin
$cache, scq, nodes := empty$
add($nodes, < T, 1 >$)
while $k$-th($scq$)$.sim$ <first($nodes$)$.sim$ do
  $T'$= first($nodes$)
  delete_first($nodes$)
  if $T'$ is inner node
    then
    for each node $T''$ of successors($T'$) do
      $s := sim^*(q, T'')$ (* similarity assessment *)
      add($nodes, < T'', s >$)
    else (* $T'$ is bucket *)
    for each case $c$ of $T'$ do
      if $c \notin cache$
        then
        $s := sim^*(q, c)$ (* similarity assessment *)
        add($cache, c$)
        if $s >$k$-th$(scq).sim then insert($scq, < c, s >$)
```

Figure 3: Improved kd-tree Retrieval

The search algorithm starts with an initialization of $nodes$ with the root node of the tree. Then the following procedure is carried out iteratively: the node T' with the highest similarity is extracted from the open list $nodes$. If the node T' is an inner node then its successors are inserted in the priority queue according to their similarity to the query. For determining the similarity between a query and a node, the hypercube that the node represents is considered an attribute *independent* generalized case itself. Hence, we can apply sim* to compute its similarity, i.e. similarity between the query and the closed point within the hypercube. Please note that this similarity computation is highly efficient (Tartakovski 2008) and does not require to solve an optimization problem. If node T' is a bucket, then the similarity values between the query q and all cases the bucket includes are calculated. These similarity computations involve solving the respective expensive optimization problems. Here, the cache is considered to avoid multiple similarity computations for a generalized case. If new cases are found that are more similar than the k-best known case in scq, they are inserted. The search procedure terminates if the similarity between the query q and the k-th best known case from the list scq is greater than the similarity between the query and the first node in the priority queue. Then, the not yet expanded nodes cannot contain any cases that are more similar.

Improvement: Minimal Dynamic Bounds

As an improvement to standard kd-trees for the retrieval of point cases, Wess et al. (1993) propose the use of *minimal dynamic bounds*. The idea of this approach is to increase the precision of the kd-tree structure and thereby to improve the retrieval efficiency. We transferred this idea to the priority k-d tree retrieval of generalized cases.

The main idea of this improvement is to define the bounds of every node more accurately. While a node within a standard kd-tree is associated with a hypercube including all cases belonging to the node, the minimal dynamic bounds define the smallest hypercube including the same cases. An example for such an improvement is demonstrated by figure 4. The left side of this figure demonstrates the standard node/hypercube S, while the right side of this figure demonstrates the minimal dynamic bounds that define the smallest hypercube S' including all cases of the node.

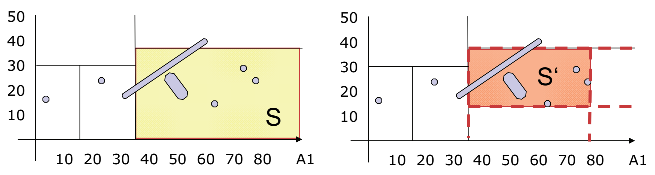


Figure 4: Dynamic Minimal Bounds Approach

The remaining area S - S' contains no cases and can be excluded from search. The minimal dynamic bounds can be computed during the construction of the kd-tree. For case bases consisting of traditional cases, the minimal bounds of any node can be calculated as follows:

$$minBounds[i].Upper := max\{c_1[i], \ldots, c_m[i]\}$$
$$minBounds[i].Lower := min\{c_1[i], \ldots, c_m[i]\}$$

Here, the term $\{c_1[i], \ldots, c_m[i]\}$ denotes the set of all values of the attribute i occurring in cases $c_1, \ldots, c_m$ associated with the regarded node (which is the total of all cases in the buckets of the sub-tree below).

For case bases including generalized cases, the bounds' calculation has to be adapted. Since an orthogonal projection of a single generalized case gc_j on any numeric attribute's range is an interval or a set of intervals, the interval/s beginning $gc\_min_j[i]$ and the interval/s end $gc\_max_j[i]$ have to be determined. In order to obtain these values two special optimization problems have to be solved:

$$gc\_min_j[i] := min\ x_i\ s.t.(x_1, \ldots, x_i, \ldots, x_n) \in gc_j \cap S$$
$$gc\_max_j[i] := max\ x_i\ s.t.(x_1, \ldots, x_i, \ldots, x_n) \in gc_j \cap S$$

Here S is the hypercube associated with the current node, and hence $gc_j \cap S$ is the fraction of the generalized case gc_j that is inside S.

Now, the minimal bounds can be calculated as follows:

$$minBounds[i].Upper := max_{j=1 \ldots m}\ gc\_max_j[i]$$
$$minBounds[i].Lower := min_{j=1 \ldots m}\ gc\_min_j[i]$$

These dynamic minimal bounds are used by the priority kd-tree retrieval algorithm in Fig. 3 when the similarity between the query and a node is determined in line $s := sim^*(q, T'')$. Instead of the whole hypercube of T'', the dynamic minimal bounds are used. The *priority k-d tree retrieval* algorithm thereby excludes the area that does not contain any cases and is therefore able to determine more precisely the highest possible similarity between the query and the generalized cases in the subtree below the node T''.

Experimental Evaluation

In an empirical evaluation we investigated the performance of the proposed retrieval methods. The objective is to investigate how far the kd-tree-based retrieval with the proposed improvements reduces the number of expensive similarity computations compared to the sequential retrieval. For experimental purposes we used a case generator that is able to produce artificial generalized cases with characteristics similar to those that occur in the domain of brokerage of electronic chip designs (Mougouie & Bergmann 2002). This case generator is able to produce generalized cases with a predefined number of variables and constraints. We created three case bases with increasing complexity in size and number of variables:

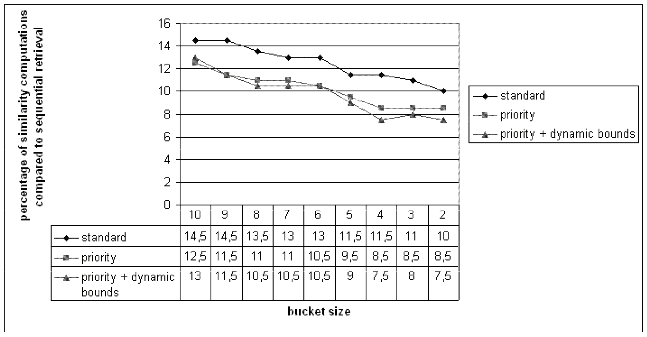
case base	# cases	# attributes
cb_1	200	2
cb_2	400	4
cb_3	400	8

For each case base, the *standard kd-tree-based retrieval*, the *priority kd-tree search*, and the priority k-d tree search combined with the *method of minimal dynamic bounds* are compared concerning the number of required similarity computations. All three methods implement a cache, which avoids redundant computationally expensive similarity assessment computations.

For each case base several kd-trees of different sizes are constructed. For the construction of the trees the proposed methods for selection of the discriminator attribute and the separation value are used. The size of the trees is controlled by a termination criterion during tree construction that restricts the number of generalized cases in a bucket of the tree. However, due to the potential overlap of generalized cases, it is possible that the number of cases in a node cannot be reduced to the required bucket size, even after a sequence of nodes is introduced. In this case the tree construction in that branch is terminated if for a number of times (defined by a parameter called *number of separation attemts*) the number of cases in a node is not reduced.

In the experiment each of the three retrieval methods is performed 100 times with random queries for each kd-tree. The size of the result list is set to k=10 cases.

Case Base 1. Figure 5 shows the performance results for case base cb_1. For this case base the number of separation attempts is set to 2 attempts, the bucket size varies from 2 to 10. The y-axis shows the percentage of similarity computations compared to the sequential retrieval for each retrieval method dependent on the bucket size (x-axis).

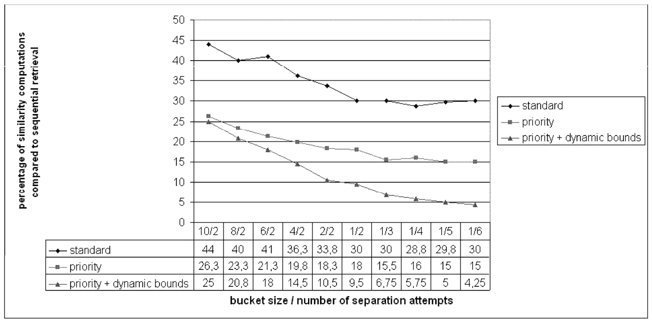


	10	9	8	7	6	5	4	3	2
standard	14,5	14,5	13,5	13	13	11,5	11,5	11	10
priority	12,5	11,5	11	11	10,5	9,5	8,5	8,5	8,5
priority + dynamic bounds	13	11,5	10,5	10,5	10,5	9	7,5	8	7,5

Figure 5: Percentage of similarity computations for cb_1

This experiment shows that all three retrieval methods are quite efficient, when being applied to case bases with cases having only two variables. However, there is a small advantage for the priority k-d tree search combined with minimal dynamic bounds.

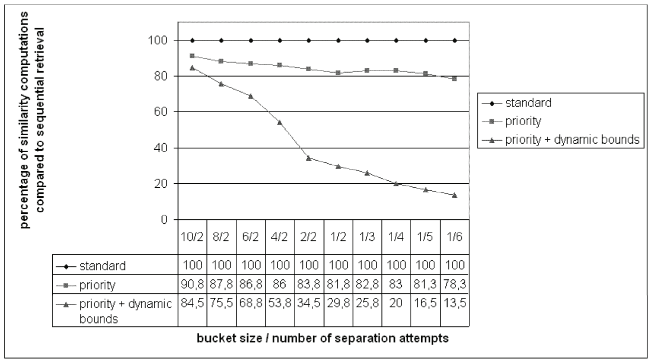
Case Base 2. Figure 6 shows the performance results for case base cb_2. The bucket size varies from 10 to 1 and subsequently the number of separation attempts is increased from 2 to 6. This experiment shows that the standard kd-tree retrieval becomes inefficient for case bases containing cases with four generalized attributes. Also the priority k-d tree search has to perform many similarity computations, in average 15% of the similarity computation calls. However, the priority k-d tree search combined with the method of minimal dynamic bounds requires in average only 4,25% (17 cases) of the similarity computation calls the sequential retrieval would perform. In the average, it regards only 7 cases which are not included in the result list.



	10/2	8/2	6/2	4/2	2/2	1/2	1/3	1/4	1/5	1/6
standard	44	40	41	36,3	33,8	30	30	28,8	29,8	30
priority	26,3	23,3	21,3	19,8	18,3	18	15,5	16	15	15
priority + dynamic bounds	25	20,8	18	14,5	10,5	9,5	6,75	5,75	5	4,25

Figure 6: Percentage of similarity computations for cb_2

Case Base 3. The results obtained with case base cb_3, shown in figure 7 demonstrate that the priority k-d tree search combined with minimal dynamic bounds is the best of all approaches and requires in average 13,5% of the similarity computation calls the sequential retrieval would perform. However, the absolute retrieval time is still about 9.7 seconds on a Pentium 4 PC (3 GHz, 3 GB RAM). The standard kd-tree retrieval method regards all cases even for the largest index-structure and requires on average about 72 seconds retrieval time.



	10/2	8/2	6/2	4/2	2/2	1/2	1/3	1/4	1/5	1/6
standard	100	100	100	100	100	100	100	100	100	100
priority	90,8	87,8	86,8	86	83,8	81,8	82,8	83	81,3	78,3
priority + dynamic bounds	84,5	75,5	68,8	53,8	34,5	29,8	25,8	20	16,5	13,5

Figure 7: Percentage of similarity computations for cb_3

Conclusion

We proposed several new improvements for kd-tree based retrieval of attribute dependent generalized cases. Our experimental evaluation shows that the most efficient retrieval method is the priority k-d tree search combined with the method of minimal dynamic bounds. This method improves the retrieval step significantly compared to any other index-based retrieval method known. From our experience with two application domains we are quite confident that this method enables efficient retrieval of generalized cases for medium sized applications with cases bases of a few hundred cases on a single high-performance server PC.

We think that further significant improvements are hardly possible by further optimizing the index structure. A different and more promising route in this direction could be to investigate methods that enable to subdivide generalized cases into several sub-cases in such a way that the resulting similarity assessment problem leads to optimization problems of smaller complexity.

References

Arya, S., and Mount, D. M. 1993. Algorithms for fast vector quantization. In Storer, J. A., and Cohn, M., eds., *Proceedings DCC'93 (IEEE Data Compression Conference)*, 381–390.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9):509–517.

Bergmann, R., and Vollrath, I. 1999. Generalized cases: Representation and steps towards efficient similarity assessment. In *Proceedings of the 23rd Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence*, volume 1701, 195–206.

Friedman, J. H.; Bentley, J. L.; and Finkel, R. A. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions Math. Software* 3:209–226.

Horst, R., and Tuy, H. 1993. *Global Otimization: Deterministic Approaches*. Springer.

Maximini, K.; Maximini, R.; and Bergmann, R. 2003. An investigation of generalized cases. In Ashley, K. D., and Bridge, D., eds., *Proceedings of the 5th International Conference on Case Base Reasoning (ICCBR'03)*, volume 2689 of *LNAI*, 261–275. Trondheim, Norway: Springer.

Mougouie, B., and Bergmann, R. 2002. Similarity assessment for generalizied cases by optimization methods. In *Proceedings of the European Conference on Case-Based Reasoning ECCBR-02*. Springer.

Tartakovski, A.; Schaaf, M.; Maximini, R.; and Bergmann, R. 2004. Minlp based retrieval of generalized cases. In Funk, P., and Calero, P. A. G., eds., *Advances in Case-Based Reasoning*, LNAI3155, 404–418. Madrid, Spain: Springer Verlag, Berlin-Heidelberg.

Tartakovski, A.; Schaaf, M.; and Bergmann, R. 2005. Retrieval and configuration of life insurance policies. In Munoz-Avila, H., and Ricci, F., eds., *Sixth International Conference on Case-Based Reasoning (ICCBR 2005)*, volume 3620, 552–565. Chicago, Illinois (USA): Springer.

Tartakovski, A. 2008. *Reasoning with Generalized Cases*. Dr. Hut Verlag, Muenchen, PhD Thesis.

Wess, S.; Althoff, K.-D.; and Derwand, G. 1993. Using *k*d-trees to improve the retrieval step in case-based reasoning. In Wess, S.; Althoff, K.-D.; and Richter, M. M., eds., *Topics in Case-Based Reasoning. Proc. Of the First European Workshop on Case-Based Reasoning (EWCBR-93)*, Lecture Notes in Artificial Intelligence, 837, 167–181. Springer Verlag.