Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning*

Stefan Wess, Klaus-Dieter Althoff and Guido Derwand

University of Kaiserslautern
D-67653 Kaiserslautern, Germany
wess@informatik.uni-kl.de

Abstract. Retrieval of cases is one important step within the case-based reasoning paradigm. We propose an improvement of this stage in the process model for finding most similar cases with an average effort of $O[log_2 n]$, n number of cases. The basic idea of the algorithm is to use the heterogeneity of the search space for a density-based structuring and to employ this precomputed structure, a k-d tree, for efficient case retrieval according to a given similarity measure *sim*. In addition to illustrating the basic idea, we present the experimental results of a comparison of four different k-d tree generating strategies as well as introduce the notion of *virtual bounds* as a new one that significantly reduces the retrieval effort from a more pragmatic perspective. The presented approach is fully implemented within the (PATDEX) system, a case-based reasoning system for diagnostic applications in engineering domains.

1 Introduction

Retrieval of sufficiently similar cases is one of the main points in the process model [25] of case-based reasoning, i.e. before selecting the most useful case(s) for adaptation, the case base must be restricted to a small set of reasonable candidates. Retrieval and selection of cases are often distinguished by the kind of features they use for case comparison (*surface* versus *structural* similarity: [19]). To detect really useful cases for the problem at hand, the selection step has to consider all available knowledge of the underlying domain. Thus, computing this structural similarity match is very expensive. Unfortunately, the retrieval step which deals with all cases in the case base must be computed very fast. Therefore, this step can only rely on the comparison of syntactical features (*surface similarity*) [17]. Basically, there are two different approaches to similarity assessment in case-based reasoning [29, 6]: the *representational approach*, proposed by [22] using a structured memory of cases *(Dynamic Memory)*, and the *computational approach* e.g. [32, 1, 5], which is based on the computing of an explicit similarity function *sim* (e.g. [36]). In this work we will focus on this approach.

* Funding for this research has been partially provided by the Commission of the European Communities (ESPRIT contract P6322, the INRECA project). The partners of INRECA are AcknoSoft (prime contractor, France), tecInno (Germany), Irish Multimedia Systems (Ireland), and the University of Kaiserslautern (Germany).

A naive approach to case retrieval would be to compute the surface similarity by comparing syntactical features of every case in the case base to the current problem according to a given similarity measure *sim*. The set of cases which must be retrieved by the following selection procedure is then determined by the m-most similar cases (m fixed), or by all cases exceeding a given similarity threshold δ. Many known case-based reasoning systems use this simple kind of approach (at least hidden in the implementation).

```
PROCEDURE SearchLin(CaseBase[1..n],Query_Case,m)
VAR i, SimCaseQueue[1..m];
BEGIN
  FOR i:=1 TO SIZE(CaseBase) DO
    IF (sim(Query_Case,CaseBase[i]) > SimCaseQueue[m])
       THEN [Update SimCaseQueue with CaseBase[i]];
  RETURN(SimCaseQueue);
  END; (* FOR *)
END; (* SearchLin *)
```

Since the overall complexity of this retrieval procedure is $O[n]$, n number of cases, for small case bases this strategy is reasonable. But, for increasing case bases this procedure leads to a too time-consuming process that restricts this approach to toy domains.

Up to now, the improvement of the efficiency of the retrieval step has been the goal in different research projects, e.g. [33]. We can distinguish two main approaches: First, the brute-force methods using massively parallel architectures like [32, 24] which take up to one processing element for each case in the case base. Second, precomputation of good indices (*cf.* [35]) for rapid access to the case base, e.g. [8, 20]. The first approach needs a lot of hardware support for the speed up of the retrieval process. By using the second approach, it is difficult to guarantee the completeness of the retrieval according to the used similarity measure *sim*.

The problem of determining the most similar cases (best matches) based on a given case description and a measure *sim* is well known as *nearest neighbor search* [11]. Cases can then be interpreted as points within a multidimensional search space where each attribute implements one dimension that can be searched with an associative procedure. The main idea of the proposed approach is to structure the search space based on its observed density and using this precomputed structure for efficient case retrieval according to the given similarity measure [33]. We developed a retrieval mechanism [28] based on a k-d tree, a multi-dimensional binary search tree [9, 15, 10]. Within the k-d tree an incremental best-match search is used to find the m most similar cases (nearest neighbors) within a set of n cases with k specified indexing attributes (dimensions). The search is guided by application dependent similarity measures. The overall similarity measure is split into local measures for each value range and a global measure which is composed from the local ones [31]. A k-d tree as such is comparable to a discrimination net [13, 12] that has been optimized for similarity-based retrieval of cases.

We will introduce and compare four different strategies for generating k-d trees, which base on the notions of *interquartile distance, category utility, entropy, average similarity*, respectively. Then the basic associative search mechanism is presented [15, 10], and completed with an effective improvement of the underlying search procedures (*virtual bounds*) that is confirmed with the respective experimental results.

2 Building a *k*-d Tree

The basic idea of the approach is to build a tree [33] which splits the search space into parts which contain a number of similar cases according to the given similarity measure *sim* (Figure 1). Therefore, every node within the k-d tree

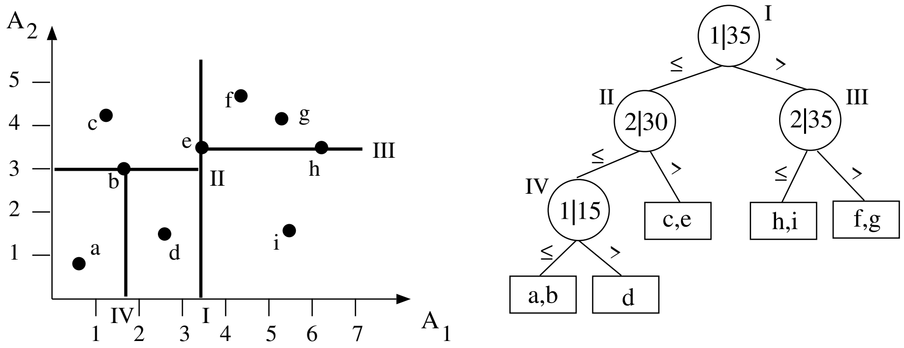


Fig. 1. An exemplary two dimensional search space and the corresponding k-d tree

represents a subset of the cases of the case base and the root node represents the whole case base. Every inner node partitions the case set into two disjoint subsets, storing the bounding values for each dimension (attribute). The leaves of the tree which contain a specific number of cases are called *buckets*. For the construction of the tree, we have to choose the best *partitioning attribute* [30] which divides the case base into two equally sized parts [15]. The process continues recursively for each of the constructed subsets of the case base until only a few cases *(bucket size)* remain which are stored together in one *bucket*. The determination of the partitioning attribute (dimension) is the most crucial part of the approach. For best speedup of the retrieval process the partition of the search space has to reflect the structure and the density of the underlying case base.

To estimate the dispersion, we first use a statistical measure, namely the *interquartile distance* [15] that can be used for both numeric and ordered nominal attribute value ranges. While the *median* splits a given distribution of values into two equally sized areas, quartiles split them into four. The first quartile q_1 (25 % quartile) divides the *lower half* of the distribution into two equally sized areas as the third quartile q_3 (75 % quartile) does with the *upper half* of the distribution.

The median is denoted as the second quartile. The greater the distance between these quartiles, the greater is the dispersion of the attribute values. During tree construction the attribute having the maximal dispersion with respect to the used similarity measure *sim* is selected as the discriminating one. Since we use similarities and not distances, we introduce the interquartile similarity as a new term. It denotes that we select that attribute for discriminating purposes where the respective quartiles have the lowest similarity (which corresponds to the greatest distance).

```
PROCEDURE Create_Tree(CB)
VAR Discriminator,PartitionValue,minSimilarity,i;
BEGIN
   IF MakeBucket(CB) THEN RETURN(MakeTerminalNode(CB));
   minSimilarity := infinity;
   FOR ALL Attribute[i] DO
      IF Spread(Attribute[i],S) < minSimilarity THEN
         minSimilarity := Spread(Attribute[i],CB);
         Discriminator := Attribute[i];
      END (* IF *)
   END (* FOR *)
   PartitionValue := Median(Diskriminator,CB);
   RETURN(MakeInternalNode(Diskriminator,PartitionValue,
     Create_Tree(LowerPartition(Diskriminator,PartitionValue,CB)),
     Create_Tree(UpperPartition(Discriminator,PartitionValue,CB)));
END (* Create_Tree *)
```

The procedures `Spread (A[i],CB)` and `Median(Discriminator,CB)` compute the dispersion of the values of attribute A_i for the cases in `CB` using the *interquartile similarity* with respect to *sim*, and the median of the discriminating attribute `Discriminator` based on the values of `Discriminator` given by `CB`. The procedure `MakeBucket(CB)` is `TRUE` if the number of values included in `CB` is less or equal than the bucket size b, which has been defined before, i.e. $|\mathcal{CB}| \leq b$ ($|\mathcal{CB}|$ denotes the cardinality of sets). The procedures `LowerPartition(A[i],Value,CB)` and `UpperPartition(A[i],Value,CB)` divide the set `CB` into two disjoint subsets where `LowerPartition` includes all cases of which the value of attribute A_i is less than `Value`. `MakeTerminalNode(S)` generates a leave node using the cases from `CB`. The procedure `MakeInternalNode(A[i], Value,lowerSon,upperSon)` generates an inner node that includes the discriminating attribute A_i, the partitioning value `Value`, and two pointers to the leave or inner nodes of the subtrees `lowerSon` and `upperSon`.

The average case effort [27] for generating a k-d tree is $O[k * n * log_2 n]$, for the worst case $O[k * n2]$. The average costs for retrieving the most similar case are $O[log_2 n]$, if the tree is optimally organized. For the worst case, the retrieval costs are $O[n]$. The retrieval mechanism is correct and complete in the sense that it always returns the m most similar cases according to the specified global similarity measure *sim*. In spite of these (theoretical) characteristics, our experiments showed that there are some disadvantages resulting from the fact that the interquartile similarity uses an a priori estimation and considers attributes in an isolated way that often comes up with unsatisfying results. Thus, we compared the above described way of attribute selection with three other approaches based on an a posteriori estimation that, after a trial wise splitting, evaluates the quality of a partitioning process by considering all dimensions (attributes).

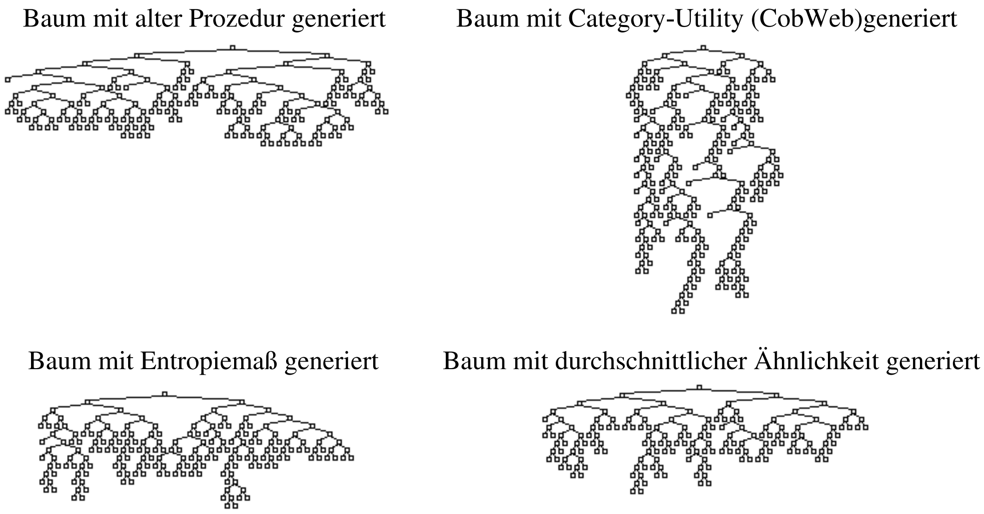


Fig. 2. Contrasting the Four Different Indexing Trees

Using the Category-Utility of CobWeb: The basic idea behind the category utility is to prefer attributes that have great effects on other attributes [14]. This is an advantage to the above described attribute selection based on the interquartile similarity. But, a disadvantage is the restriction to symbolic attributes, even the Classit approach [16] that uses the variance offers no satisfying solution here because the mean value is necessary for the computation of the variance but not available for all attributes. Another drawback of this approach is that the category utility measure [14] does not use the available similarity measures for the tree generation process.

Using the Entropy Measure: The entropy measure prefers attributes such that the information gain is maximized [30]. The measure is very general and has been used for a wide range of problems. Since it is based on probabilities, again the information included in the similarity measures is not used. A second problem arises because it very much focuses on the classes within the case descriptions. Such information is not always available. In addition, it is a drawback that the whole tree generation process heavily depends on the correctness of such classes.

Using the Measure of the Average Similarity: The idea behind this approach is to build up subtrees and buckets based on cases that are as similar as possible. Thus, all the information that is already included in the available cases and similarity measures is used. As a consequence, no class description has to be included in the case descriptions, and there is no restriction with respect the attribute types that can be used.

For our experiments we chose the *car database* [21] from the *UCI Repository of Machine Learning Databases and Domain Theories* (`ftp ics.uci.edu`). It includes 205 different cars that are described using 25 attributes like manufacturer, technical data, price, and a risk estimation for insurance companies. The resulting indexing trees for all four different strategies for generating k-d trees are shown in Figure 2.

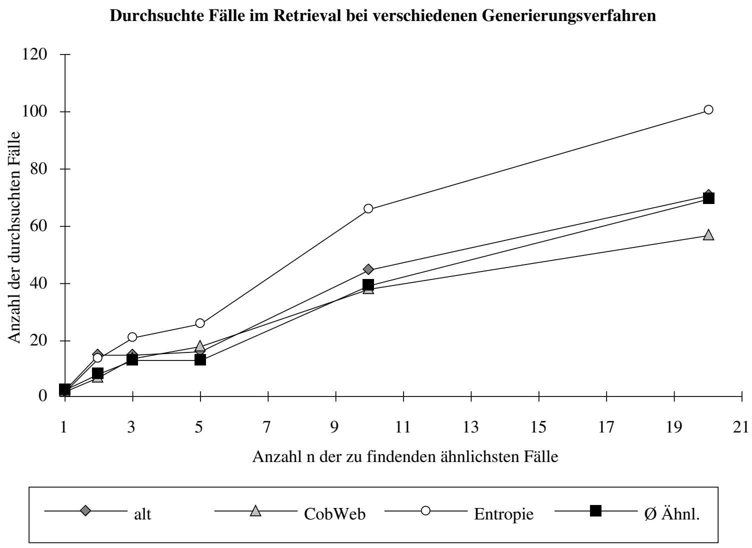


Fig. 3. Comparing the Number of Examined Cases to Find m most Similar Cases

We conducted many experiments to analyze the different approaches to the generation of indexing trees. Some of these results are summarized in Figure 3. Here the figure shows, for all four generation procedures, the number of examined cases for the task to find the m most similar cases (for $m = 1, 2, 3, 5, 10$, and 20, respectively). Up to $m = 10$, the *average-similarity* approach is the best. If the task is to find more than the ten most similar cases, the CobWeb approach is the best. But, if we also look at Figure 4, the *average-similarity* approach becomes clearly the best suited one. Here, the required time in seconds is shown for finding the m most similar cases.

3 Searching Similar Cases using a k-d Tree

The search for similar cases in the k-d tree is done via a recursive tree search procedure according to the global similarity measure *sim*. Normally, there are no fully identical cases in the case base and we have to look for the most similar ones. Using the tree as a kind of binary search tree leads to a *bucket* where a specific number b of cases are stored. At this stage, it is necessary to compute the similarity of each case stored in the bucket using the predefined similarity measure sim. If we are looking for the m most similar cases we can build up a queue containing these most similar cases. Using this queue, we draw a hyperball around the given problem that includes the m most similar cases found in the current bucket. Thus, every case which is at least as similar as the examined ones must be within this constructed k-dimensional hyperball. By using this hyperball we are able to decide which *buckets* we have to examine next. For

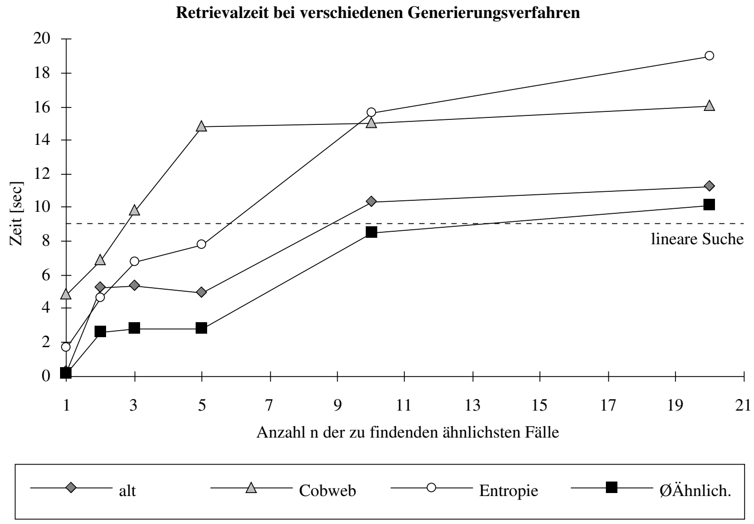
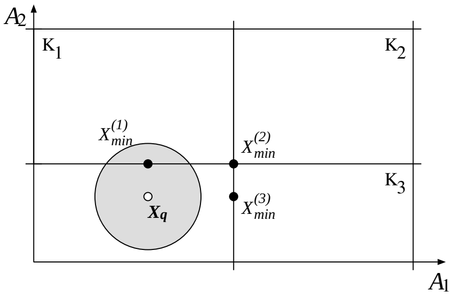


Fig. 4. Comparing the Respective Retrieval Time

an efficient implementation of this basic idea, we use two test procedures [15]: BALL-WITHIN-BOUNDS (BWB) and BOUNDS-OVERLAP-BALL (BOB) (Figure 5). These procedures check whether it would be reasonable to explore certain areas of the search space in more detail, or not. Such tests can be carried out without retrieving the respective cases. The geometric bounds of the considered subspaces are used to compute a *similarity interval* whose upper bound then *answers* the question to explore, or not. For finding the m most similar cases for a given working case (or query case), we apply recursive tree search. Thus, as input we need the query case X_q, the number m of most similar cases, the k-d tree represented by its root node, and the global similarity measure sim. During search a priority queue PQC is continuously updated which includes the m most similar cases (while `PQC[n]` denotes the nth most similar case, `PQS[n]` denotes the actual similarity value of the nth most similar case). If the recursive search procedure examines a leaf node, the similarity of all included cases is computed and, if necessary, the priority queue PQC is updated. If the examined node is an inner node, then the search procedure is recursively called for that son node which should include the query case. If this call terminates, it is tested whether it is also necessary to examine the other son node by using the **BOB** test.

The **BOB** test is **TRUE** if the cases of the actual tree node have to be explored. The inner nodes are correct generalizations of all the cases they represent in the sense that they include the geometric (upper and lower) bounds (for every indexing attribute) which correspond to the respective subspace.



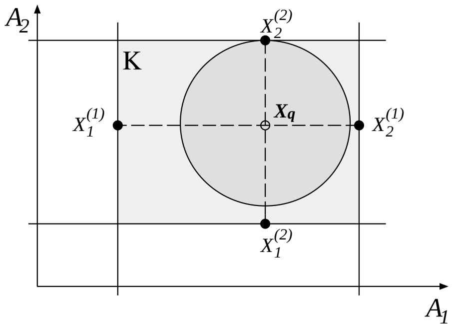


Fig. 5. Basic idea of the BOB and BWB test

$$\boxed{BOB \iff Sim(X_{min}, X_q) \geq PQS[m] \ (= Sim(PQC[m], X_q) \ )}$$

These geometric bounds are used to compute a similarity interval whose upper bound then answers the question to explore, or not. The closest point X_{min} within the actual node's subspace is computed as the projection onto the actual node's geometric bounds (Figure 5). X_{min} is on the actual node's bounding box on the edge facing the query case X_q. If there is no overlapping in any of the k dimensions between the node's bounding box and the k-dimensional ball round X_q, then X_{min} is a corner of the bounding box. If X_q is within the bounding box then $X_q = X_{min}$ (Figure 5). Before the recursive search procedure terminates, the **BWB** test is applied. This test is **TRUE** if the k-dimensional ball round X_q is completely within the bounding box of the actual tree node (Figure 5).

$$\boxed{BWB \iff Sim(X_1^{(j)}, X_q) < PQS[m] \quad \wedge \quad Sim(X_2^{(j)}, X_q) < PQS[m] \ \forall j = 1, \ldots, k}$$

In this case, no overlapping with other bounding boxes is possible. Thus, the search is finished, and the m most similar cases for the current problem according the given global similarity measure sim are found.

With the following procedure `RetrieveCases` a search for similar cases is performed as follows:

1. Initialization of the global array `Query`$[1 \ldots k]$ with the query case

2. Initialization of the global variables:
   `Queue[1...m].Case` :=nil
   `Queue[1...m].Sim` :=0
   `Bounds[1...k].Upper` :=∞
   `Bounds[1...k].Lower` :=$-\infty$
3. Call of `RetrieveCases(root)`, where `root` denotes a pointer to the root node of the respective k-d tree. After the procedure has finished `Queue[1..m]` will contain the m most similar cases of the `CaseBase` with respect to `Query` and the used similarity measure sim.

```
PROCEDURE RetrieveCases(treeNode)
VAR temp,Discriminator,PartitionValue;

BEGIN
   IF (TerminalNode(treeNode)) THEN
      [Update Queue[1..m] using Query, sim and the cases in treeNode];
      IF BallWithinBounds THEN [Finished] ELSE RETURN; (* BWB *)
   END;

   Discriminator := treeNode.discriminator;
   PartitionValue := treeNode.partitionValue;
   IF Query[Discriminator] <= PartitionValue THEN
      temp := Bounds.Upper[Discriminator];
      Bounds.Upper[Discriminator] := PartitionValue;
      RetrieveCasesh(treeNode.lowerSon);
      Bounds.Upper[Discriminator] := temp;
   END
   ELSE
      temp := Bounds.Lower[Discriminator];
      Bounds.Lower[Discriminator] := PartitionValue;
      RetrieveCases(treeNode.upperSon);
      Bounds.Lower[Discriminator] := temp;
   END;

   IF Query[Discriminator] <= PartitionValue THEN
      temp := Bounds.Lower[Discriminator];
      Bounds.Lower[Discriminator] := PartitionValue;
      IF BoundsOverlapBall THEN RetrieveCases(treeNode.upperSon); (* BOB *)
      Bounds.Lower[Discriminator] := temp;
   END
   ELSE
      temp := Bounds.Upper[Discriminator];
      Bounds.Upper[Discriminator] := PartitionValue;
      IF BoundsOverlapBall THEN RetrieveCases(treeNode.lowerSon). (* BOB *)
      Bounds.Upper[Discriminator] := temp;
   END;
   IF BallWithinBounds THEN [Finished] ELSE RETURN; (* BWB *)
END; (* RetrieveCases *)
```

4 Improved Retrieval using Virtual Bounds

We now improve the above introduced bounds tests based on the known cases. The basic idea is to describe the subspaces that really include cases more precisely. Therefore, all occurring maximal and minimal values for each attribute are stored. This led us to the definition of the notion of *Minimal Virtual Bounds* (cf. Figure 6).

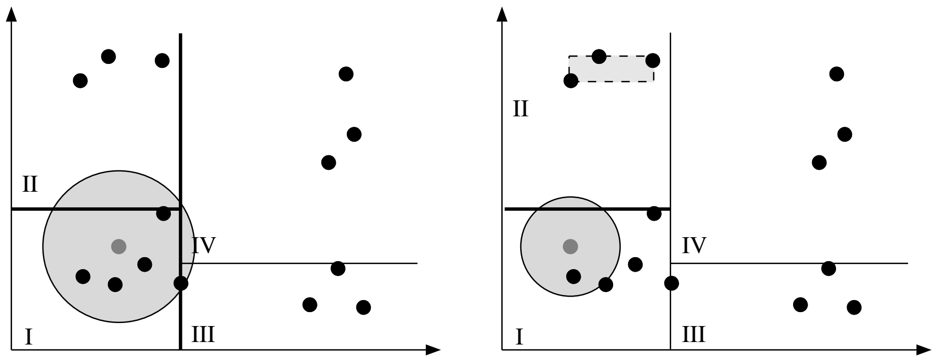


Fig. 6. `BOB` Tests and Minimal Virtual Bounds

In the right part of Figure 6, we can get an intuitive understanding how much from the description space need not to be considered during search by looking at the "white areas". A `BOB` test that uses minimal virtual bounds recognizes that Bucket II does not include any *better cases* (cf. Figure 6).
The *Minimal Virtual Bounds* of a tree node for the dimension k are defined as follows:

$$minBounds[k].Upper := max(\{Case_i[k]\})$$
$$minBounds[k].Lower := min(\{Case_i[k]\})$$

$\{Case_i[k]\}$ denotes the set of all attribute values of attribute k of all cases $Case_i$ being represented by the respective tree node.

While minimal virtual bounds led to an improvement of the `BOB` tests, an analogous idea (of *Maximal Virtual Bounds*) can be used to improve the `BWB` tests. For the latter, it is reasonable to describe the searched subspace as precise as possible such that the k-dimensional hyperball around the query case has the maximal chance to be completely within that ball. Thus, we introduced *Maximal Virtual Bounds*, as described in Figure 7.

Within such maximal virtual bounds it is guaranteed that no more similar cases can be found within those borders. The computation of the maximal virtual bounds requires more effort because it is not based on the analysis of the cases

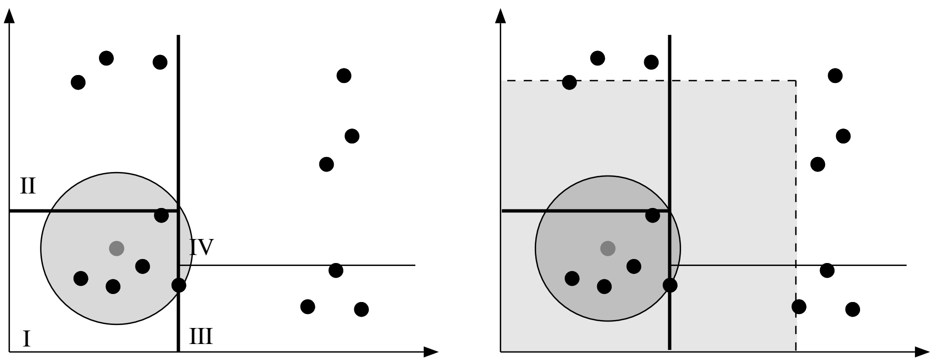


Fig. 7. `BWB` Tests and Maximal Virtual Bounds

but on the analysis of all neighboring subspaces. Again the virtual bounds can be computed during tree generation.

Within the maximal virtual bounds, it is guaranteed that only cases of the respective subspace itself belong to it. There are no more similar cases outside these boundaries. Thus, the search is finished.

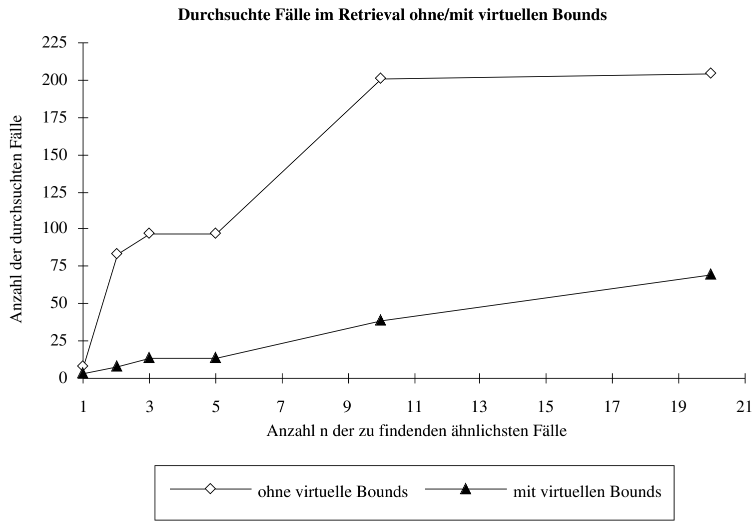


Fig. 8. Improved Performance (Used Cases) with Virtual Bounds

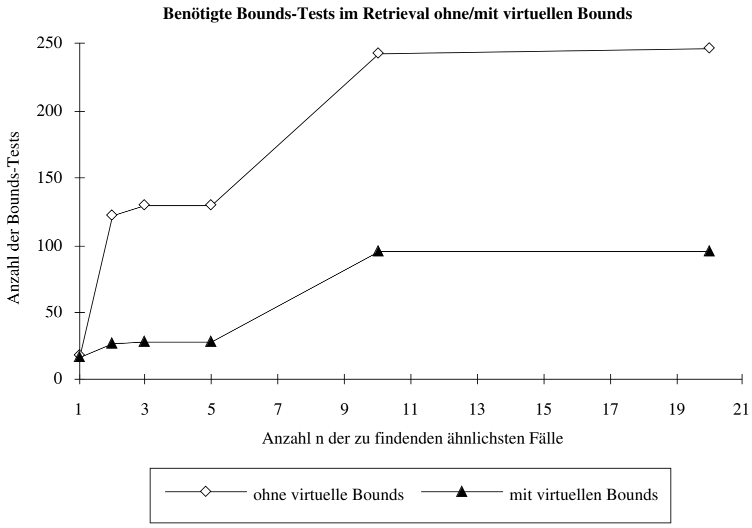


Fig. 9. Improved Performance (Retrieval Time) with Virtual Bounds

5 Evaluation

From a more general perspective, we showed that choosing those attributes for discrimination purposes that maximize the average similarity within the partitions in the k-d tree result in the best performance. The taken approach using an a posteriori evaluation goes a step away from the original idea of the k-d tree approach [15] and a step towards the conceptual clustering approach [16]. The resulting k-d tree already represents a certain classification of the cases. But, it also differs from known approaches by the following aspects:

- Our implementation of the original k-d tree approach [15] has been combined with an a posteriori evaluation like in conceptual clustering systems.
- The a posteriori evaluation is based on the new measure of *average similarity* within the respective k-d tree partitions.
- We use class-dependent information to dynamically decide on the selection of an appropriate global similarity measure *sim* [5, 31].
- The global similarity measures can be automatically improved by a competitive learning strategy [37].
- The described case retrieval approach is only one subcomponent of the PATDEX case-based reasoning system [38] being combined with other techniques.
- PATDEX itself is an integrated subpart of the MOLTKE knowledge acquisition workbench [3, 4, 2]. Therefore, the similarity assessment process can be improved by the use of additional knowledge like default attribute values as well as causal and heuristic determination rules.

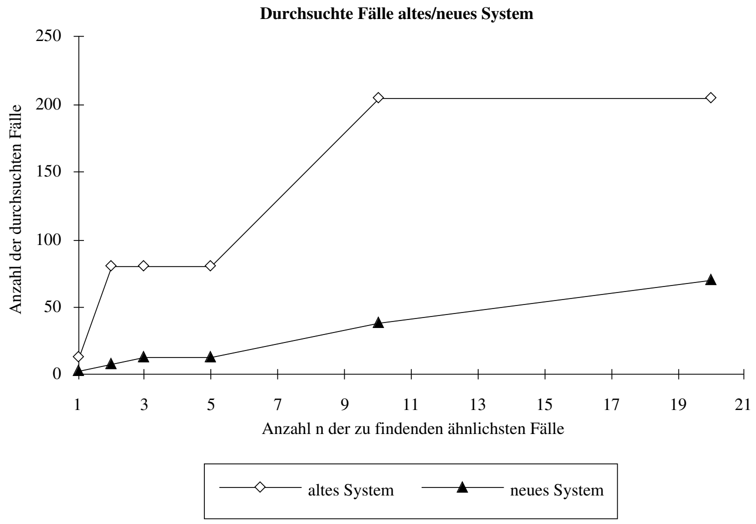


Fig. 10. Used cases with the new approach

From a k-d tree perspective [15], we showed that the retrieval can be significantly improved by the use of virtual bounds. They are a refinement of the retrieval procedures within the k-d tree. The virtual bounds are directly stored in tree nodes of the k-d tree. The overall speedup of the proposed approach (improved generating and retrieval procedures) compared to the original approach [15] is about 5 to 10 times. Figure 10 shows the number of used cases for different retrieval tasks. Compared to the old approach [15] much less cases have to be examined.
If we look at the comparison of the retrieval times of the new and the old approach (cf. Figure 11) we can state also a speed up. Compared to the linear approach (cf. section 1) the power of the new approach becomes clear. Within our experiments (cf. Figure 11) the search for 20 similar cases (10% of the whole case base) the retrieval time in the new k-d-tree is faster than the linear approach. With the old approach [15] this limit was reached with the task to retrieve just two cases (cf. Figure 11).

6 Conclusions

A k-d tree, together with appropriate generation and retrieval algorithms, enables an efficient support for the retrieval task in case-based reasoning. Since the number of cases that need to be searched can be significantly reduced, this approach is especially applicable to huge case bases.
In our laboratory, several experiments have been conducted dealing with other

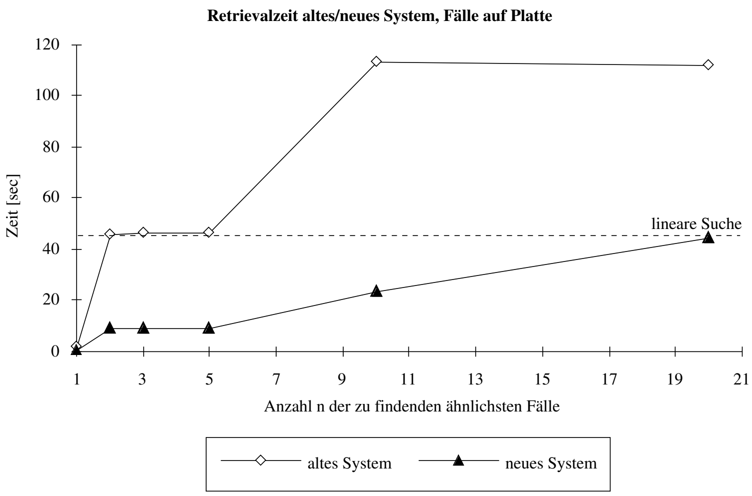


Fig. 11. Overall Performance of the proposed approach

parameters such as selection of the partitioning value of the respective discriminating attribute, dynamic determination of the bucket size as well as incremental insert and incremental retrieval of cases. For the results cf. [38]. Our approach can efficiently handle missing attribute values during consultation [26] and unordered value ranges by (partially) extending the k-d tree to an n-ary tree [38]. Since the used search procedures are strongly connected to the notion of "distance", the used global and local similarity measures have to meet certain compatibility requirements, namely to be both monotonic and symmetric.

The combination of information retrieval/nearest neighbor classification and case-based reasoning has been of increasing interest recently, e.g. [7, 34]. Nevertheless, we are not aware of any similar approach that is also correct, complete, and efficient.

7 Acknowledgement

The authors would like to thank Hannes Öchsner, Frank Goebel and Michael M. Richter for their contributions to this work and the reviewers for their helpful comments on an earlier version of this paper.

References

1. David W. Aha. Case-Based Learning Algorithms. In Ray Bareiss, editor, *Proceedings: Case-Based Reasoning Workshop*. Morgan Kaufmann Publishers, 1991.

2. Klaus-Dieter Althoff. *Eine fallbasierte Lernkomponente als integrierter Bestandteil der* MOLTKE *Werkbank zur Diagnose technischer Systeme.* PhD thesis, Fachbereich Informatik, Universit"at Kaiserslautern, 1992.
3. Klaus-Dieter Althoff, Benedikt Faupel, Sabine Koksk"amper, Ralf Traph"oner, and Wolfgang Wernicke. Knowledge Acquisition in the Domain of CNC Machining Centers: The MOLTKE Approach. In *Proceedings of 3.rd European Workshop on Knowledge Acquisition EKAW*, 1989.
4. Klaus-Dieter Althoff, Frank Maurer, and Robert Rehbold. Multiple Knowlegde Acquisition Strategies in MOLTKE. In *Proceedings of 4.th European Workshop on Knowledge Acquisition EKAW*, 1990.
5. Klaus-Dieter Althoff and Stefan Wess. Case-Based Knowledge Acquisition, Learning and Problem Solving for Diagnostic Real World Tasks. In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91*, 1991.
6. Klaus-Dieter Althoff and Stefan Wess. Case-Based Reasoning and Expert System Development. In Franz Schmalhofer, Gerhard Strube, and Thomas Wetter, editors, *Contemporary Knowledge Engineering and Cognition.* Springer-Verlag, Berlin, 1992. In preparation.
7. Peter Anick, Evangelos Simoudis, Bruce Croft, William Mark, and Chris Riesbeck, editors. *Case-Based Reasoning and Information Retrieval.* AAAI - Spring Symposium Series, 1993.
8. R. Barletta and W. Mark. Explanation-Based Indexing of Cases. In Kolodner [23], pages 50–61.
9. J.L. Bentley. Multidimensional binary search trees used for associative searching. *Com. of the ACM*, 18:509–517, 1975.
10. A.J. Broder. Strategies for efficient incremental nearest neighbor search. *Pattern Recognition*, 23:171–178, 1990.
11. Belur Dasarathy. *Nearest Neighbor Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, 1990.
12. D. McDermott E. Charniak. *Introduction to AI.* Addison-Wesley, 1985.
13. E. A. Feigenbaum. The simulation of verbal learning behavior. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought.* McGraw-Hill, 1963.
14. D. Fisher. Cobweb: Knowledge acquisition via conceptual clustering. *Machine Learning*, 2:139–172, 1987.
15. J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. math. Software*, 3:209–226, 1977.
16. J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40:11–61, 1989.
17. Dedre Gentner and Kenneth D. Forbus. MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pages 504–509, 1991.
18. Kristian J. Hammond, editor. *Proceedings: Case-Based Reasoning Workshop.* Morgan Kaufmann Publishers, 1989.
19. K. J. Holyoak and K. Koh. Analogical Problem Solving: Effects of Surface and Structural Similarity in Analogical Transfer. In Midwestern Psychological Association, editor, *Proceedings of the Conference of the Midwestern Psychological Association*, 1986. May 1986.
20. Dietmar Janetzko, Stefan Wess, and Erica Melis. Goal-Driven Similarity Assessment. In Hans-Jürgen Ohlbach, editor, *Proc. German Workshop on AI (GWAI'92).* Springer Verlag, 1992. (in Vorbereitung).

21. Dennis Kibler, David Aha, and Marc Albert. Comparing instance-averaging with instance filtering learning algorithms. In *Proceedings of the third European Working Session on Learning*, pages 63–80, 1988.
22. Janet L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory*. PhD thesis, Yale University, 1980.
23. Janet L. Kolodner, editor. *Proceedings Case-Based Reasoning Workshop*. Morgan Kaufmann Publishers, 1988.
24. Janet L. Kolodner. Retrieving Events from a Case Memory: A Parallel Implementation. In *Proc. Case-Based Reasoning Workshop* [23], pages 233–249.
25. J.L. Kolodner, R.L. Simpson, and K. Sycara. A Process Model of Case-Based Reasoning in Problem Solving. In IJCAI, editor, *Proc. IJCAI 1985*, pages 284–290. Morgan Kaufmann Publishers, 1985. Los Angeles, California, USA.
26. M. Manago, K.D. Althoff, E. Auriol, R. Traphöner, S. Wess, N. Conruyt, and F. Maurer. Induction and reasoning from cases. In M.M. Richter, S. Wess, K.D. Althoff, and F. Maurer, editors, *First European Workshop on Case-Based Reasoning (EWCBR-93)*, pages 3313–318, 1993.
27. Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. Springer, 1984.
28. Hannes Öchsner and Stefan Wess. Ähnlichkeitsbasiertes retrieval von fallbeispielen durch assoziative suche in einem mehrdimensionalen datenraum. In K-D. Althoff, S. Wess, B. Bartsch-Spörl, and D. Janetzko, editors, *Ähnlichkeit von Fällen in Systemen des fallbasierten Schliešens*, SEKI-Report, Universität Kaiserslautern, SFB 314, 25.-26. Juni, June 1992.
29. B. W. Porter. Similarity Assessment: Computation vs. Representation. In Hammond [18], pages 82–84.
30. J.R. Quinlan. Learning efficient classification procedures and their application to chess endgames. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Maschine Learning*. Tioga Press, 1983.
31. Michael M. Richter and Stefan Wess. Similarity, Uncertainty and Case-Based Reasoning in PATDEX. In Robert S. Boyer, editor, *Automated Reasoning*, Essays in Honor of Woody Bledsoe, pages 249–265. Kluwer Academic Publishing, 1991.
32. Craig Stanfill and David Waltz. Toward Memory-Based Reasoning. *Com. of the ACM*, 29(12):1213–1229, 1986.
33. R. H. Stottler, A. L. Henke, and J. A. King. Rapid Retrieval Algorithms for Case-Based Reasoning. In *Proceedings of the 11th International Conference on Artificial Intelligence IJCAI-89*, pages 233–237, 1989. Detroit, Michigan, USA.
34. Toshikazu Tanaka and Naomichi Sueda. Combining strict matching and similarity assessment for retrieving appropriate cases efficiently. In Anick et al. [7].
35. P. Thagard and K. I. Holyoak. Why Indexing is the Wrong Way to Think About Analog Retrieval. In Hammond [18], pages 36–40.
36. A. Tversky. Features of Similarity. 84:327–352, 1977.
37. Stefan Wess. PATDEX - Inkrementelle und wissensbasierte Verbesserung von Ähnlichkeitsurteilen in der fallbasierten Diagnostik. In *Tagungsband 2. deutsche Expertensystemtagung XPS-93*, Hamburg, 1993. Springer Verlag.
38. Stefan Wess. *Fallbasiertes Problemlösen in wissensbasierten Systemen*. PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, 1994. (in Vorbereitung).

This article was processed using the LaTeX macro package with LLNCS style