Fish and Shrink. A next step towards efficient case retrieval in large scaled case bases

Jörg Walter Schaaf

GMD – Artificial Intelligence Research Division
e-mail:Joerg.Schaaf@gmd.de
Sankt Augustin, Germany

Abstract. Keywords: Case-Based Reasoning, case retrieval, case representation
This paper deals with the retrieval of useful cases in case-based reasoning. It focuses on the questions of what "useful" could mean and how the search for useful cases can be organized. We present the new search algorithm *Fish and Shrink* that is able to search quickly through the case base, even if the aspects that define usefulness are spontaneously combined at query time. We compare *Fish and Shrink* to other algorithms and show that most of them make an implicit *closed world assumption*. We finally refer to a realization of the presented idea in the context of the prototype of the FABEL-Project[1].
The scenery is as follows. Previously collected cases are stored in a large scaled case base. An expert describes his problem and gives the aspects in which the requested case should be similar. The similarity measure thus given spontaneously shall now be used to explore the case base within a short time, shall present a required number of cases and make sure that none of the other cases is more similar.
The question is now how to prepare the previously collected cases and how to define a retrieval algorithm which is able to deal with spontaneously user-defined similarity measures.

1 Motivation

Suppose two cases in your domain have just been considered similar with respect to a certain aspect, and just a few minutes later they have to be regarded as to be considerably dissimilar because another aspect is now being regarded.
If that cannot happen in your domain, then a lot of approaches making use of static similarities are available to organize your case base and it would be better to take one of them to realize your retrieval task.
None of them will fit the requirements if similarity between two cases depends on the context in which retrieval is evoked. The reason is that once built up,

[1] This research was supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 01IW104. Project partners in FABEL GMD – German National Research Center for Information Technologie, Sankt Augustin, BSR Consulting GmbH, München, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

the relations between cases in the case bases of concurrent approaches remain static over time. These relations always reflect similarity between cases which is calculated once and used to support further retrieval tasks.

The approach presented in this article will help to solve some of your problems if the following statements hold and some basic operations are computable.

- Cases in your domain are complex enough to be worth being regarded from different points of view.
- Frequently changing the point of view is essential to benefit from the knowledge stored in cases.
- Changing the point of view leads to different representations (aspects) of cases, each pointing out what emerges while looking at the cases from that point of view.
- It is possible to evaluate similarity for each pair of cases comparing the corresponding representations.
- Dissimilarity between a query and a case shrinks the possible range of similarity between that case and cases in the neighborhood.

If these preconditions are true, then the presented approach will help to organize, structure and handle your cases.

Each of the cases will be stored in all its possible and meaningful representations, ready to be compared with other cases in the case base. Similarity of all stored cases will be precalculated for each aspect and will be stored. In a retrieval situation you will have the option to select aspects that have to be regarded and to combine them in a weighted manner if more than one aspect should be regarded at a time. The spontaneously defined set of weighted aspects (your actual view on the cases) will be used to calculate similarity between your actual problem (query) and the stored cases. As a result you will get all cases of your case base that deliver useful information to solve your problem, if suitable aspects are represented and regarded in your actual view.

To prevent the user from waiting too long for a retrieval result (until all cases have been checked) a completely new retrieval algorithm is presented in this article. The main idea of this algorithm is to successively shrink the range of possible similarities between the query and an amount of cases by a few explicit tests. Each case can be represented by an interval describing this range. Several different user demands concerning the cases the user wants to get can be answered by interpreting upper and lower bounds of these intervals. The main advantage of this strategy is that the effective use of time depends on the precision the user asked for.

The following article introduces a data structure to hold case representations (aspects) and links to store aspect specific similarities between cases. We explain how changes in point of view on cases can be seen as a spontaneous and weighted combination of aspects. We show that this leads to the possibility of a context dependent redefinition of case similarity by using only low cost calculations.

2 Situation-conform assessment of similarity

What does it mean to assess similarity with respect to a certain situation? First and foremost it means that assessment should change when circumstances change. In this section, I will introduce the elements to construct a situation-specific similarity assessment. They are called "aspects" here. To work with them, cases must be accessible in a format that can be manipulated by computers. In this paper this is called the *source representation* of a case. Many approaches try to interpret or to abstract from the source representation in order to extract the main idea of a case and to make that idea comparable with those of other cases. As a result a case gets a position in a searching structure or an index.
If a position in a searching structure is precalculated, then the query guides through the structure and leads to the most similar case which is part of the structure. This kind of access only works if the meaning of all attributes remains constant throughout all case retrievals. Only then the costs for building the search structure are worthwhile. If, in contrast, spontaneous change of important attributes shall be allowed then a static search structure is not applicable.
If cases are indexed a new possibility of comparison arises. Now it is possible to state at query time which attributes are important and how important they are. There are some approaches in literature, for example [1, 6, 9], that offer spontaneous attribute weighting and try to avoid complete search. These approaches differ from the following one in some important points concerning complexity of attributes and combination.

Complexity of attributes: Attributes of collected cases are often of low complexity. This makes comparison easy. If the cases of a domain can be described sufficiently by using some attributes, then the usage of the concept described here is not indicated. Cases in the domain of building construction need complex attributes to be compared properly. The topology graph of design objects like columns, pipes or furniture is such a complex attribute. Topology of design objects is an essential criterion for reusability of a design. Representation of topology is expensive, and comparison of two topology graphs is NP-complete [14]. Because the term "attribute" hardly fits a topology graph for example, a useful but complex representation like this will be called an "aspect representation" of a case. It is not easy to describe what an aspect itself is. We found that it can be seen as an idea of how to represent a case. To find a good idea to represent cases is a hard problem. Discussion about the quality of suggestions is not the topic of this paper. To define a retrieval algorithm mentioned above, we only need a name for an aspect, a function that represents the case with respect to the aspect and a function that compares the representations of two cases.
Let Ω_0 be the space of potentially all cases in the data exchange format (we sometimes call this format the original representation). That means that each real case is a point in Ω_0.

Definition 1 aspect representation function. Let i be the name of an aspect. An aspect representation function of a case is a function α_i: $\Omega_0 \rightarrow \Omega_i$. Ω_0

is the set of all cases in their source representation.

Ω_i is the space of all possible representations with respect to aspect i. The aspect representation function α_i assigns a point in Ω_i to each point in Ω_0. In principle the design of the aspect representation function is free but it should be designed to simplify the space and to emphasize certain attributes (the aspects) of cases. The function α_i often has the nature of a transformation or a projection, but sometimes of a knowledge intensive interpretation.
In the project FABEL a case-based reasoner was developed which offers several different tools to retrieve cases [13]. Each tool delivers useful cases under certain conditions. To make them all work together, the main ideas of these methods have been regarded as "aspects" in the sense described above.
Four main parts have been found in each of the developed retrieval methods:

- An idea of what to represent and to compare could be worthwhile.
- A function to represent cases in a specific way.
- A function to compare case representations and to state their distances in terms of a real number.
- A well suited organization of the case base that stores case representations and supports the retrieval of cases that are similar to the query with respect to the representation.

With the help of this analysis, the idea of what a case could b has been elaborated. It should be a set of representations, each regarding one interesting aspect. Figure 1 represents a case as a polyhedron with one face for each aspect.

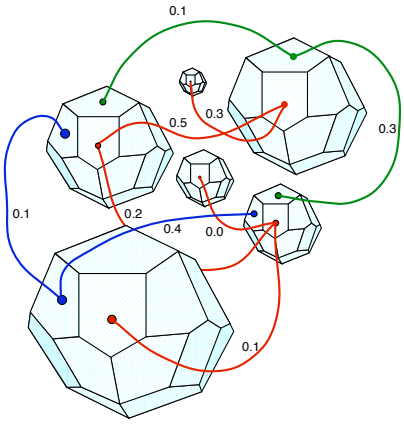

Fig. 1. The case base as a network of cases. A polyhedra corresponds to a case, a face of it to an aspect representation, the label of an edge to a calculated distance between two cases with respect to the connected aspect representations.

A case base can be seen as a network of cases. Weighted edges from face to face connect cases. The weight depends on the distance of the connected cases with respect to a certain aspect. Two cases are called (representation) neighbors with respect to an aspect, if they are connected by an edge concerning this aspect. Edges can be directed if the distance is not symmetric.
Consider the following data structure that defines cases more formal and implies a structure we call case base. It also introduces some terms used in further description.

```
case = record:
  identifier :string;
  representations-list :list of representation-node;
  mindist,maxdist :[0,1].

representation-node = record:
  aspect-name :string;
  aspect-representation :free type;
  neighbor-list :list of representation-neighbor.

representation-neighbor = record:
  reference :pointer to representation-node;
  distance :[0,1].
```

Combination: Comparison between a query and one of the collected cases is the basic operation of the retrieval task. This comparison is done stepwise by comparing aspect by aspect. We call the result of one step the *aspect distance* between two cases.

Definition 2 aspect distance. An aspect distance function δ_i is a distance function in Ω_i. For F_x and $F_y \in \Omega_0$ we abbreviate $\delta_i\big(\alpha_i(F_x), \alpha_i(F_y)\big)$ by $\delta_i(F_x, F_y)$.

The overall similarity between two cases is computed by a combination of the results coming from aspect specific comparisons. To distinguish this combination from an *aspect distance* it is called the *view distance*. The view distance is influenced by the actual importance of each contributing aspect and by the function that calculates the view distance by using the aspect distances as input.

Definition 3 view. A weight vector $\mathbf{W} = (w_1 ... w_n)$ with $\sum w_i = 1$ defines the "importance" for each aspect i. Each combination of weighted aspects defined by the vector $\mathbf{W}$ is called a view on a case. Aspect i belongs to the actual view if $w_i > 0$.

The importance of each aspect is given by the weight vector *view* ($\mathbf{W}$). The function to calculate the view distance is user defined. We call this function the *view distance function* or SD_{name}.

Definition 4 view distance. The view distance of two cases is defined[2] as a function $SD_{name} : \Omega_0 \times \Omega_0 \times \mathbf{W} \rightarrow [0, 1]$.

The view distance is a combination of aspect distances weighted by the view $\mathbf{W}$. Example:

$$SD_{med}(F_x, F_y, \mathbf{W}) := \sum_{i=1}^{n} w_i \delta_i(F_x, F_y)$$

If, for example, the user wants to consider a set of aspects in a weighted manner, the view distance function SD_{med} in the example above should be used. Further possible and useful view distance functions are discussed in [11]. They lead to more conservative or more creative case proposals.

View distances are calculated between cases. Some of them are of special interest for further explanation so we define them:

Definition 5 Test- and base distance and view neighbor. The view distance between query A and a test case T is called the "test distance". The view distance between test case T and one of its view neighbors V_i is called "base distance [3]". Two cases with a small view distance are called "view neighbors". As a first approach we define two cases as to be view neighbors if their view distance is below 1. As we will see, results of *Fish and Shrink* do not become incorrect by using any other threshold in [0,1].

The input to calculate the *view distance* (actual view and actual view distance function) normally changes when context or user intentions change. *Aspect distances* between two cases normally never change.
Spontaneous changes in case neighborhood (because of changing view distance) will raise difficulties in standard retrieval algorithms. It is principally impossible to precalculate static retrieval structures, like those used in search trees (for example k-d-trees [2, 16, 17]) or networks (for example, shared feature networks or discrimination networks [5]) that are based on a single evaluation of case distance.

3 Retrieval in large scaled case-bases

"Case-based reasoning will be ready for large-scale problems only when its retrieval algorithms are sufficiently robust and efficient to handle thousands and ten thousands of cases." (Kolodner in [8], page 289).

[2] Definition of view distances and their influence to case selection is discussed in [11]. $SD(x, x) = 0$ and SD is symmetric if all δ_i are symmetric.

[3] The notion "base distance" shall stress that this particular distance is stored in the case base.

3.1 Approaches with undeclared closed world assumption

We think that retrieval in large scaled case-bases can only be solved if memory is traded against time. The question arises what to store to speed up retrieval without limiting it too much.
In his hillclimbing approach Goos [6] stores distances between entire cases and focuses the search to the neighborhood of a case which are most promising. Other approaches based on case neighborhood are [9] and [18]. They use spreading activation to increase a score for cases with attribute values similar to the query.
In the following paragraph I will show that all, [6], [9] and [18], make use of an undeclared, implicit *closed world assumption* and that this assumption is the reason for a lot of problems. The implicit *closed world assumption* is:

If similarity between two cases is not explicitly stated, then these cases are definitively different.

This assumption is not necessary and is not used by the algorithm described in section 3.2.
Goos and Lenz try to benefit from similarities stated and stored between a directly tested case and its neighbors. They use information about the closeness of cases to decide where to go on searching [6] or which cases shall be presented to the user [9]. The authors assume that similarity not stated explicitly is equivalent with stated dissimilarity. They disregard that missing similarity statements can come from at least two reasons. On the one hand missing statements can express dissimilarity, on the other hand they can express that similarity has not yet been checked. That could happen if case representation or comparison is incorrect or if cases have been left unconsidered. The problem has been discussed in both papers without having been solved.
The main idea of the algorithm that I will present here is that only a well known and proven statement of similarity to a definitely dissimilar case leads to the deferment of cases.
Missing representation of important attributes leads to *no delay* and thus to early direct test. If the definitions of dissimilarity in [6], [9] and [18] fail, it is possible that similar cases are not found. According to the definition presented here, similar cases are tested later in the process at the worst.

3.2 Fish and Shrink. Case retrieval without closed world assumption

If one wants to benefit from one direct comparison between a query and a case, a direct test should reduce search space. The *Fish and Shrink* algorithm presented here realizes that without using any *closed world assumption* like this mentioned above. It is based (like the *Fish and Sink* algorithm [12]) on a metaphor of sinking cases and cases being dragged down (Figure 2). *Fish and Shrink* relies on the following assumption:

Presumption 1 *If a case does not fit the actual query, then this may reduce the possible usefulness of its neighbors.*

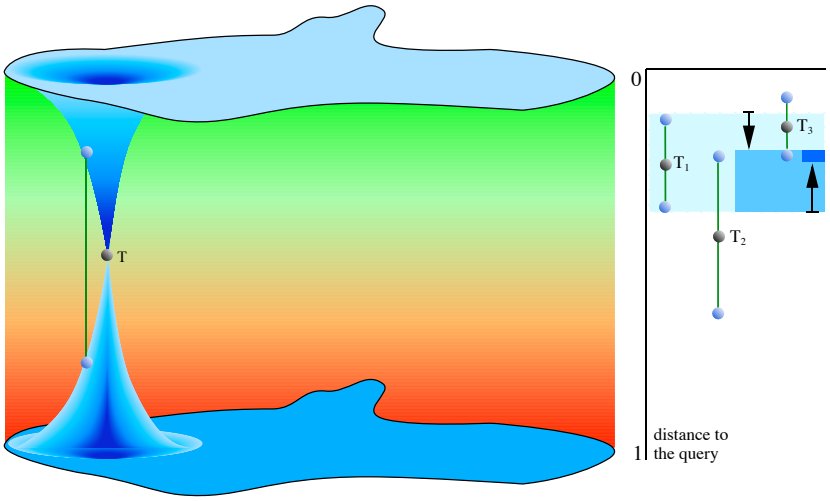

Fig. 2. Metaphor of sinking cases being dragged down by others (left). Intervals, shrinking from both ends, represent the still possible distance to the query (right).

This assumption is supported if the underlying distance function (aspect distance function) fulfills the following condition derived from the triangle inequality.

Presumption 2

$$\delta_i(A, V) \geq \delta_i(A, T) - \delta_i(T, V)$$

If the condition above is true for every triple (A, T, V) the *minimum distance* between the query and some not directly tested neighbors of the directly tested case can be stated. This means that `V.mindist` can be stated and eventually increased for some cases V, view neighbors of T. If a second condition is valid, an analogue reduction from the other side (`V.maxdist`) can be achieved. We assume:

Presumption 3

$$\delta_i(A, V) \leq \delta_i(A, T) + \delta_i(T, V)$$

If this condition is true, the interval of possible distances between query and not yet directly tested cases can be shrunk from both sides. That is what we describe with the term "shrink" in the nickname *Fish and Shrink* of the discussed algorithm. "Fishing" refers to the random access to the test case.
If one of the above conditions is violated then an error occurs. A possibility to avoid this is discussed in [11]. Figure 3 shows the algorithm *Fish and Shrink*.
The idea of *Fish and Shrink* is that the view distance between the query and a test case restricts both, the minimum distance as well as a maximum distance

1. Let CB be a list of all cases in the case base.
2. For each case $F \in CB$ let $F.mindist := 0$ and $F.maxdist := 1$;
3. While not OK and not interrupted
   (a) move precision line PL;
   (b) choose (fish) a case T with $T.mindist$ on PL;
   (c) $T.mindist := SD(A, T, \mathbf{W})$;
   (d) $T.maxdist := T.mindist$;
   (e) $\forall$ cases V that are view neighbors of case T
   with $V.mindist \neq V.maxdist$ do
      i. $V.mindist :=$
      $MAX\Big(testdistance - basedistance,\ V.mindist\Big)$;
      ii. $V.maxdist :=$
      $MIN\Big(testdistance + basedistance,\ V.maxdist\Big)$

Fig. 3. *Fish and shrink* algorithm. OK is a predicate which is TRUE if the user demands are fulfilled (see section 3.3). PL marks an increasing distance to the query. All cases with `mindist`=PL are candidates for the next direct test. All cases whose complete interval is passed by PL are offered to the user.

between query and some of the neighbors of the test case. The algorithm keeps the maximal *minimum distance* and the minimal *maximum distance* for each case. The interval defined by the two values `mindist` and `maxdist` describes the range of possible distances to the query for each case. Its upper bound only decreases and the lower bound can only rise. For historical reasons, lowering the upper bounds is also called *sinking*. This term shall stress that cases with bigger `mindist` are regarded to be not as promising as cases with smaller ones. Both movements of boundaries are results of one direct comparison between query and a test case.

Figure 2 tries to show this on the left. It describes the behavior of a case which is at first the *view neighbor* of test case T_1 and then also *view neighbor* of test cases T_2 and T_3. By knowing the distance between the query and T_1 the interval of possible distances between V and the query can be restricted without having it tested in any way. Direct test of T_2 and T_3 further restricts this interval. The test of T_2 lowers the upper bound (increases `mindist`), while test of T_3 raises the lower bound (decreases `maxdist`). To resume, every direct test leads to a further restriction of an interval that describes the possible distances between the not yet directly tested case V and the query.

By this means, derivable guesses become more and more precise until some definitive statements become possible. The ongoing process leads to the exact distance between each case and the query if it runs until its termination. Normally the process is manually or automatically interrupted before termination obtaining the demanded results. A discussion about the best time to interrupt, depending on the demanded results, can be found in the next section. Up to interruption, each direct test will at least shorten one interval (the one of the

tested case) and bring it to a single point. Additionally, the intervals of cases in the neighborhood of the tested case will be shortened.

3.3 Results from *Fish and Shrink*

Upon demand, *Fish and Shrink* supplies either all cases that are more similar than a given threshold S or delivers the k best cases optionally including a ranking or giving the exact distance for each of the k best cases.
Suppose the given threshold is S, then it is not necessary to test those cases directly whose maximum distance (`maxdist`) is lower than S (Figure 4*a* interval a). Such cases are better than S even if they are finally positioned at `maxdist`. With an analogue argument those cases whose minimum distance (`mindist`) is bigger than S (Figure 4*a* interval b) need not be tested. It cannot overcome the threshold S.

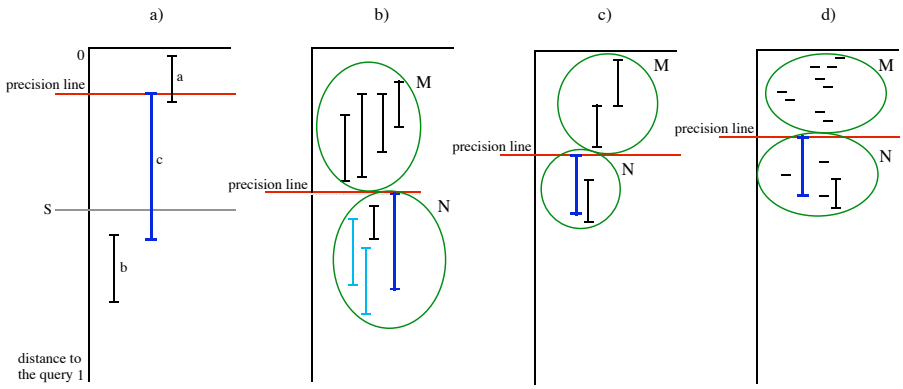

Fig. 4. User demand: all cases better than a threshold (a) and k best cases without giving the partial order of result cases (b). User demand: k best cases of the case base without (c) and with (d) exact distances between query and result cases.

Only those cases have to be tested whose intervals overlap S. The number of overlapping intervals decreases permanently because intervals become shorter. How much the number decreases after one direct test depends on the test distance and on the base distances. The bigger the test distance and the smaller the base distances, the more effective the direct test has been.
Before we give some suggestions for the predicate OK and the movement instructions for the precision line PL to realize various user requirements with *Fish and Shrink* (see Figure 3) we define some help predicates to make further terms more readable.

$$
\begin{aligned}
overlap\_eachother(F_i, F_j) := & F_i.mindist \leq F_j.mindist \leq F_i.maxdist \\
& \vee F_i.mindist \leq F_j.maxdist \leq F_i.maxdist \\
& \vee F_j.mindist \leq F_i.mindist \leq F_j.maxdist \\
overlap(F_i, S) := & F.mindist \leq S \leq F.maxdist \\
zero\_interval(F) := & F.mindist = F.maxdist \\
min\_mindist(F_i, M) := & F_i \in M \wedge \nexists F_q \in M : F_q.mindist < F_i.mindist
\end{aligned}
$$

$PL := F.mindist : overlap(F, S) \ \wedge \ min\_mindist(F, CB) \ \wedge \ \neg zero\_interval(F).$
$OK := \forall F \in CB : \neg overlap(F, S) \ \vee \ zero\_interval(F).$

Fig. 5. Movement of precision line and predicate OK for user demand: cases better than threshold S.

Three more types of user demands shall be discussed here to show how shrinking intervals can be interpreted. Figure 4*b* shows how to get the k best cases without a ranking (see Figure 4*b*). The rule is to continue until `maxdist` of k cases is passed by a line called *precision line*. The precision line is defined by the minimal value of `mindist` in the case base and marks this value by a line parallel to the X axis (e.g. in Figure 4).
OK and movement of the precision line for the user demand "best k cases without any ranking among them" is described in Figure 6.

$PL := F.mindist : \neg zero\_interval(F) \ \wedge \exists M, N \subset CB; M \cap N = \emptyset$
$\quad \wedge \forall m \in M, n \in N : m.maxdist \leq n.mindist$
$\quad \wedge \, min\_mindist(F, N).$
$OK := |M| \geq k.$

Fig. 6. Movement of precision line and predicate OK for user demand: k best cases without ranking.

If the user asks for k cases with ranking but without exact distances, *Fish and Shrink* stops if k intervals that do not overlap one another have been passed by the precision line (Figure 4*c*). OK and movement of the precision line is shown in Figure 7
If the user wants exact distances (and rankings) of the k best cases, *Fish and Shrink* stops if k intervals of length 0 have been passed by the precision line (Figure 4*d*).

$$
\begin{aligned}
PL \;&:= F.mindist : \neg zero\_interval(F) \;\wedge \exists M, N \subset CB; M \cap N = \emptyset \\
&\quad \wedge \forall m \in M, n \in N : m.maxdist \leq n.mindist \\
&\quad \wedge min\_mindist(F, N) \\
&\quad \wedge \forall m_1, m_2 \in M : \neg overlap\_eachother(m_1, m_2). \\
OK \;&:= |M| \geq k.
\end{aligned}
$$

Fig. 7. Movement of precision line and predicate OK for user demand: k best cases with rankings.

$$
\begin{aligned}
PL \;&:= F.mindist : \neg zero\_interval(F) \;\wedge \exists M, N \subset CB; M \cap N = \emptyset \\
&\quad \wedge \forall m \in M, n \in N : m.maxdist \leq n.mindist \\
&\quad \wedge min\_mindist(F, N) \\
&\quad \wedge \forall m \in M : zero\_interval(m). \\
OK \;&:= |M| \geq k.
\end{aligned}
$$

Fig. 8. Movement of precision line and predicate OK for user demand: k best cases with exact view distances to the query.

Besides the discussed results one can also activate *Fish and Shrink* to obtain the following:

- a sorted list of cases, recommending the most similar cases first;
- the best cases after a predefined run-time;
- the most promising cases at the time of a spontaneous user interruption;
- a sorted list of best cases up to the point where the quality of the next one is significantly worse than the previously recommended one;
- only one case of each group of very similar cases (a swarm), using a slightly extended algorithm as described below.

Figure 9 shows a diagram to control the ongoing process. It shows all intervals for each direct test, gives an overview to the user and supports the decision whether to interrupt *Fish and Shrink* or not.

4 How Fish and Shrink saves tests

It is an interesting question how *Fish and Shrink* manages to spare expensive direct tests and thereby keeps waiting time short. To explain that we have to look closer into two mechanisms.
First, not promising cases are sunk down so that they are tested – if at all – later in the process. Our last results from tests in the domain of Computer Aided Architectural Design (CAAD) show that *Fish and Shrink* delivers those cases early in the process which fit user specifications without testing too much. Speaking in terms of clusters, one may state that the view distance function

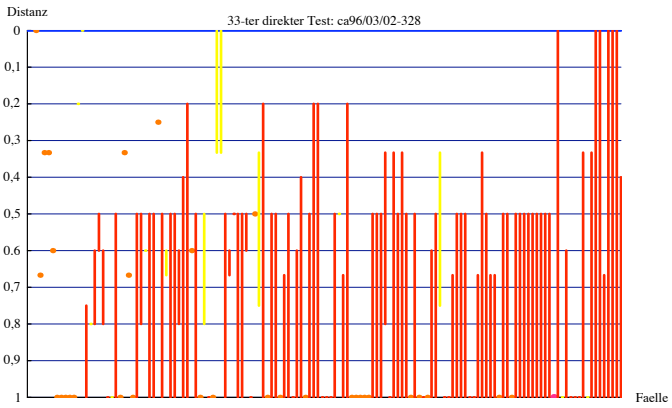

Fig. 9. A visualisation of the intervals in a test with a real database containing 150 cases, each in five different representations.

forms spontaneously a cluster and drags along all neighbors according to their individual (view) distance (see also Figure 2). This leads to a delay of a whole group of cases which can be seen as less promising.

The second mechanism is not yet discussed here and needs a closer look. Keeping with the metaphor of fishing, we call the mechanism the *swarm fishing*. The reason why it has been implemented is easy to explain by telling a joke:

Peter was asked by his teacher to give him ten animals living in Africa. After a short time of thinking he answered: monkey, lion, and ... and eight elephants.

This joke shows what humans expect when they are asked to give examples or even cases. They are expected to give *different* ones. I think it would become boring if the user of an case-based reasoning system would obtain hundreds of nearly equal cases, because he asked for all cases better than some threshold. Even if the user is clever and bounds the number it would be very unsatisfactory if all delivered cases were looking nearly equal. I think users want to get a big variety of cases, that means only one single representative for each cluster.

It is straight forward to satisfy such user demands with *Fish and Shrink*. Knowing the view distance function it is part of the algorithm to calculate the view distance between the test case and its view neighbors in order to sink (and lift) them if necessary. If this distance is recognized as to be below a certain bound, neighbors are seen to belong to the cluster (the swarm) of the test case. Close neighbors are not worthwhile to be tested in the ongoing process. In Figure 9 a case belonging to a swarm is shown in light grey color while intervals of other cases are black.

Further examination of a swarm can be evoked if the user explicitly wants to do that. Experience shows that cases in the swarm of a test case do not differ enough to provide additional information, so tests within a swarm are not recommended.

5 State of the work and outlook

The presented approach is completely worked out, formally described in [11] and implemented as a prototype. The retrieval algorithm *Fish and Shrink* delivers cases from the domain of building design that are similar in five freely combinable aspects. It is now being tested with larger case bases (about 400 cases in five different representations. Empirical results will be available by the end of 1996 ([15]). An evaluation by domain experts with respect to the quality of delivered cases is still to be done.
Up to now, we have integrated the aspects "case silhouette" [4] concerning two subsystems, "Gestalten" [10], "topological structure" [3] and a fuzzy representation of components [7] for cases in the domain of architectural design.
As an interesting additional feature, our concept allows to derive a quality assessment for aspects from working with it. The quality of an aspect could be determined by the support it gives to suggest the case that is finally chosen by the user. Aspects which do not support the delivering of "right" cases can disappear successively. This could happen step by step by disregarding the useless aspects for new cases, deleting aspect representations from older ones and thereby erasing the edges between case representations that are similar. This automatic assessment would make it possible for experts to define their own aspects in their domain, without having to fear that the case-based reasoner will permanently obtain useless cases due to a unfortunate aspect definition. We are still working on that.

References

1. K. D. Ashley and E. L. Rissland, 'Waiting on weighting: A symbolic least commitment approach', in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 239–244. Morgan Kaufmann, St. Paul, MN, (1988).
2. J. L. Bentley, 'Multidimensional binary search trees used for associative searching', *Communications of the ACM*, **18**(9), 509–517, (September 1975).
3. Carl-Helmut Coulon, 'Automatic indexing, retrieval and reuse of topologies in complex designs', in *Proceeding of the sixth International Conference on Computing in Civil and Building Engineering*, eds., Peter Jan Pahl and Heinrich Werner, pp. 749–754, Berlin, (1995). Balkema, Rotterdam.
4. Carl-Helmut Coulon and Ralf Steffens, 'Comparing fragments by their images', in *Similarity concepts and retrieval methods*, ed., Angi Voß, 36–44, GMD, Sankt Augustin, (1994).
5. E. A. Feigenbaum, 'The simulation of natural learning behaviour', in *Computers and Thought*, eds., E. A. Feigenbaum and J. Feldman, 297–309, McGraw-Hill, New-York, (1963).
6. Klaus Goos, *Fallbasiertes Klassifizieren. Methoden, Integration und Evaluation*, Ph.D. dissertation, Bayerische Julius-Maximilians-Universität, Würzburg, 1995.
7. Wolfgang Gräther, 'Computing distances between attribute-value representations in an associative memory', in *Similarity concepts and retrieval methods*, ed., Angi Voß, 12–25, GMD, Sankt Augustin, (1994).
8. Janet L. Kolodner, *Case-Based Reasoning*, Morgan Kaufmann, San Mateo, 1993.

9. M. Lenz and H. D. Burkhard, 'Retrieval ohne Suche', in *Fallbasiertes Schließen – Grundlagen und Anwendungen*, eds., Brigitte Bartsch-Spörl, Dietmar Janetzko, and Stefan Wess, pp. 1–10, Universität Kaiserslautern, Germany, (1995). Zentrum für Lernende Systeme und Anwendungen, Fachbereich Informatik.
10. Jörg W. Schaaf, 'Gestalts in CAD-plans, Analysis of a Similarity Concept', in *AI in Design'94*, eds., J. Gero and F. Sudweeks, pp. 437–446, Lausanne, (1994). Kluwer Academic Publishers, Dordrecht.
11. Jörg W. Schaaf, 'ASPECT: Über die Suche nach situationsgerechten Fällen im Case Based Reasoning', Fabel-Report 31, GMD, Sankt Augustin, (1995).
12. Jörg Walter Schaaf, '"Fish and Sink"; An Anytime-Algorithm to Retrieve Adequate Cases', in *Case-based reasoning research and development: first International Conference, ICCBR-95, proceedings*, eds., Manuela Veloso and Agnar Aamodt, 538–547, Springer, Berlin, (October 1995).
13. Barbara Schmidt-Belz, 'Scenario of FABEL Prototype 3 Supporting Architectural Design', Fabel-Report 40, GMD, Sankt Augustin, (1995).
14. S. Skiena, *Implementing Discrete Mathematics*, Addision-Wesley Publishing Co., 1990.
15. Ralf Steffens, *Kantenreduktion in ASPECT-Fallbasen*, Master's thesis, Universität Bonn, 1995. Diplomarbeit in der Entstehung.
16. Dagmar Steinert, *Effiziente Zugriffsstrukturen für die ähnlichkeitsbasierte Vorauswahl von Fällen*, Master's thesis, Lehrstuhl für Informatik VI (KI) der Universität Würzburg, 1993.
17. S. Wess, K.D. Althoff, and G. Derwand, 'Improving the retrieval step in case-based reasoning', in *Pre-Prints First European Workshop on Case-Based reasoning*, Universität Kaiserslautern, Germany, (1993). Zentrum für Lernende Systeme und Anwendungen, Fachbereich Informatik.
18. M. Wolverton and B. Hayes-Roth, 'Retrieving semantically distant analogies with knowledge-directed spreading activation.', Technical Report KSL 94-19, Knowledge Systems Laboratory, Stanford University, (March 1994).