A Case Base Representation Technique to Support Case Based Reasoning

Content Areas: Case Based Reasoning
Tracking Number: A207

Abstract

A representation technique whereby Case Bases (CBs) used in Case Based Reasoning (CBR) applications can be encoded is described. The representation considers the CB in terms of a multi-dimensional space where each dimension represents a particular attribute, and the values for that dimension an enumeration for that attribute. The advantages offered are firstly that of compression — the CB is reduced to a set of integers. Secondly, unlike other CB compression techniques, the representation supports interaction without necessitating "de-compression"; further the numeric nature of the representation ensures that this interaction is computationally effective. Thirdly, and significantly in the context of CBR, the representation supports mechanisms whereby CBR queries can be relaxed/expanded to support ideas concerning "similarity" retrieval. Fourthly, within certain machine dependent limitations, the technique operates regardless of the number of "dimensions" (attributes) under consideration. Finally there is no corresponding increase in the level of complexity as the number of "dimensions" increases.

1 Introduction

Case Based Reasoning (CBR) has rapidly developed from the first reported laboratory systems such as CHEF that had a few dozen cases [Hammond, 1986] to commercial CBR systems with tens of thousands of cases [Kitano *et al.*, 1992; Waltz, 1996]. During this development the issue of case-representation has been a central feature.

Case-representation is important because the representation will determine what data can be stored, how many cases can be stored, how similarity can be assessed, and crucially for large commercial Case Bases (CBs), how quickly cases can be retrieved. CBs typically have to deal with multi-attribute (multi-dimensional) data, whereas traditional database management systems normally use variations of a Binary-tree for single-attribute indexing. Consequently, conventional relational databases are not ideally suited to CBR. However, using multi-dimensional access methods it is difficult to order multi-dimensional spatial objects whilst preserving their spatial proximity.

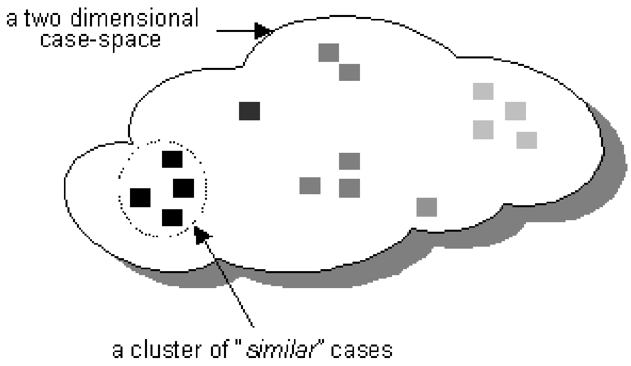

Figure 1: The "case space" concept central to CBR

In this paper an N-dimensional representation to compress and encode CBs is described. The key feature of the representation, unlike other CB compression techniques, is that spatial proximity is maintained. Further it does not require "de-compression" before anything useful can be achieved.

2 Previous Work

Central to CBR is the concept of similarity [Kolodner, 1993]. Cases may be considered as multi-dimensional objects existing within a multi-dimensional space — the case-space [Leake, 1996] as illustrated in Figure 1. Case objects that are similar to each other reside in close proximity to each other within the case-space. Case retrieval involves conceptually positioning a target case in the case-space and finding those cases close (similar) to it.

The case-space is a manifestation of the case-representation and it must be able to represent all the data types required by the cases and provide efficient retrieval. The problem has been that whilst it is easy to maintain the correct linear ordering of cases (i.e. an index) for single- dimensions it has not been possible to preserve the spatial proximity of multi- dimensional objects in a multi-dimensional index [Kamp *et al.*, 1998].

Research into spatial and multi-dimensional database systems has produced many techniques some of which have been used for CBR [Kamp *et al.*, 1998]. In general, multi-dimensional models vary according to whether they operate on *point data* or *spatial data*. The best known point data access method is the kd-tree [Bentley, 1975] first used for CBR by [Wess *et al.*, 1993]. The kd-tree is basically a generalisation of a binary decision tree to N-dimensions. Crucially it supports nearest neighbor search [Fukunaga and Narenda, 1975]. [Lin *et al.*, 1994] comment that there is a severe performance loss with most of these methods as the number of dimensions increase. Lin et al. show that the search of a kd- tree, as described by [Friedman *et al.*, 1997], is no more efficient than a linear search for more than nine dimensions.

An alternative to "point data" access is to model cases as spatial objects (in line with the more usual conceptual model of CBR). The representation described here adopts this second approach. Most current methods approximate spatial objects using minimum bounded rectangles in two dimensions (i.e. a multi-dimensional object is represented as points in a two dimensional space). Binary access methods can then be used to retrieve objects. However, this approach has several disadvantages:

- The calculation of point (i.e. retrieval) and range (i.e. similarity)is complicated
- Objects that are in close proximity in the N-dimensional space may be distant in the two dimensional space
- Complex distance based queries may not be possible at all [Kamp *et al.*, 1998].

There are several multi-dimensional access methods reported in the literature that can operate directly on the spatial data; either supporting ortho-rectangular objects, such as the R-tree [Guttman, 1984], or supporting objects with arbitrary shapes, such as the BV-tree [Freeston, 1995] and the Cell Tree [Gunter, 1989].

This paper describes a representational techniques that can support objects with arbitrary shapes (i.e. any number of dimensions), keeps similar objects in proximity to each other, supports complex distance-based queries (i.e. similarity-based retrieval), is computationally simple and does not suffer from performance loss as the number of dimensions increases.

3 Representation

The CB in a CBR system can presented as a table comprising N columns representing case attributes ($\{A_1, A_2, \ldots, A_N\}$), and M rows representing individual cases ($\{C_1, C_2, \ldots, C_M\}$). In common with relational database tables the CB has a schema associated with it. Attributes are typically either of some numeric type or are character arrays, and thus can be defined in terms of an enumeration of some kind. Consequently each attribute has a sequence of possible values associated with it:

$$A = \{v_0, v_1, \ldots, v_k\}$$

each identified by an "ordinal number" ranging from 0 to k. For many attributes there may be some significance to the ordering (e.g. age, monetary value etc.), however this is not always so. A case X can thus be described in terms of a *values-tuple*:

$$C_X = < v_{1_x}, v_{2_x}, \ldots, v_{N_x} >$$

where $1 \leq X \leq M$, and $0 \leq x \leq k$ (for the attribute in question). Of course knowledge of some attribute values may be unavailable and thus some fields may be left "blank".

Given the above each attribute is a dimension in an N-dimensional space (the *case space*), and consequently each case is a N-dimensional object (a *case object*) in this space whose location and shape is defined by the nature of the values for its associated attributes. Given a case for which all attribute values are known, these values act as coordinates which uniquely define a "cell" within the case space. Where a value for an attribute is unknown or not applicable all possible values for the attribute are included when defining the nature of the case object. For example Figure 2(a) shows a 3-D case space defined by three attributes:

$$\begin{aligned} A_1 &= \{0, 1, 2, 3, 4\} \\ A_2 &= \{0, 1, 2, 3\} \\ A_3 &= \{0, 1, 2\} \end{aligned}$$

where the sequence of integers are the ordinal numbers from 0 to k of the enumerations for each attribute. If we now consider a case:

$$C_1 = \{1, 1, 0\}$$

This will uniquely define a "single cell" case object of the from given in Figure 2(b). Consider a second case which has an unknown value for A_2:

$$C_2 = \{2, \_, 0\}$$

This will define a case object as given in Figure 2(c). Figure 2(d) then shows a case of the form:

$$C_3 = \{3, \_, \_\}$$

4 Compression

From the previous section each attribute (dimension) has a range of values associated with it $(0 \ldots k)$. Thus, for each dimension we allocate a number of bits in an integer type, sufficient for all the values plus an additional bit to act as a "sign" bit (see sub-section 4.3 for more detail concerning the latter). The number of bits (b_n)

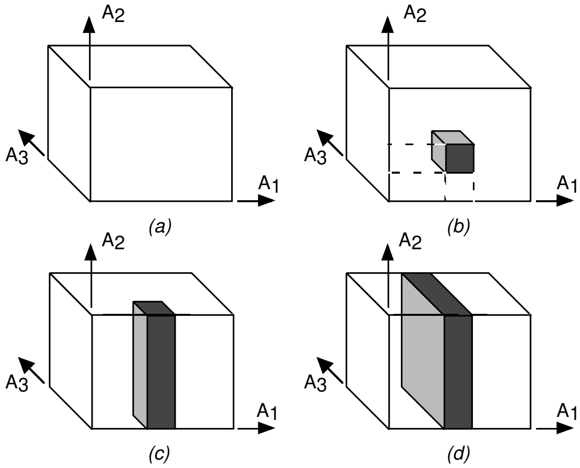

Figure 2: Example case objects

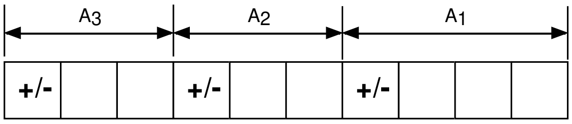

Figure 3: Bit allocation

required for each dimension N can be calculated using the following identity:

$$b_n = 1 + log_2 k$$

(rounded to the nearest whole number). Thus considering the example in Figure 2, a distribution of the form shown in Figure 3 would be appropriate. Given any coordinate tuple the address associated with that tuple can be calculated using the function:

$$\begin{aligned}
Dom(f_{cart2tess}) &= \{< v_1 \dots v_n >\mid v_1 \in A_1 \dots v_n \in A_n\} \\
Cod(f_{cart2tess}) &= \{t \mid t \in Int\} \\
Gr(f_{cart2tess}) &= \{< v_1 \dots v_n, t >\mid t = \sum_{i=1}^{n} v_i \times base_i\}
\end{aligned}$$

where Int is the set of all integers, and $base_n$ is then calculated as follows:

$$base_n = 2^{\sum_{i=0}^{n-1} b_i}$$

in which b_i equals the number of bits allocated to dimension i. Thus in the above example the dimensions will be:

$$\begin{aligned}
base_1 &= 2^0 = 1 \\
base_2 &= 2^{0+4} = 16 \\
base_3 &= 2^{0+4+3} = 128
\end{aligned}$$

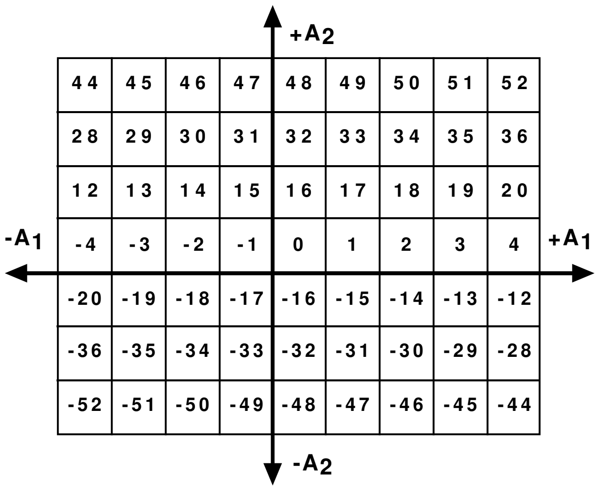

Figure 4: 2-D slice through example space ($D_3 = 0$)

in which case:

$$\begin{aligned}
f_{cart2tess}(v_1, v_2, v_3) &= base_1 v_1 + base_2 v_2 + base_3 v_3 \\
&= v_1 + 8v_2 + 128v_3
\end{aligned}$$

we can therefore uniquely identify each cell in the *case space*. A "slice" of this space in the $D_1 - D_2$ plane is given in Figure 4 (A_3 is constant at 0). The zero address is the *origin* for the representation, i.e. the reference point to which other points can be related. The figure includes negative addresses and also indicates the bottom-left to top-right linearisation which is a feature of the representation.

The advantages of the representation (with respect to Cartesian and other "tesseral" approaches) are:

1. The representation operates with any number of dimension without incurring any increase in complexity (i.e. there is no performance loss) as the number of dimensions under consideration increases.
2. There is no requirement for "de-compression", however, if desired addresses can be converted to and from Cartesian/value tuples in a computationally effective manner (not the case with standard tesseral approaches).
3. It results in a "left-to-right" linearisation of space (Figure 4) which in turn offers advantages of:
   - Efficient data storage using ideas founded on "run-line" encoding strategies (see Sub-section 4.2).
   - Computationally effective comparison and translation of sets of addresses.
4. Supports computationally efficient translation through the space using straightforward integer addition and subtraction (see Sub-section 4.3), unlike other tesseral systems which required recourse to "look-up" tables or specialised tesseral processors .

5. The addressing system is intuitive.

As a consequence of these advantages many of the disadvantages associated with alternative approaches to N-dimensional CB representations (see section 2) are avoided.

4.1 Note on tesseral representations

Tesseral representations are spatial representations which are derived by repeatedly dividing a space into isohedral sub-spaces until some predefined resolution is reached [Bell *et al.*, 1983; Diaz and Bell, 1986]. The resulting sub-spaces are then allocated (usually numeric) "addresses" in such a way as to reflect the hierarchical decomposition.

The representation described here is categorised as a tesseral representation because it is a mechanism for addressing spaces decomposed into isohedral subspaces, but at some predefined resolution. A more detailed explanation of the representation can be found in [Coenen *et al.*, 1998b] and detail concerning its wider application in [Coenen *et al.*, 1998a].

4.2 Note on data storage

Using the representation it is not necessary to store all addresses associated with a particular case object. Instead, case objects can be presented in terms of sequences of *N-cubes* where each cube is defined in terms of a prefix address and a postfix address separated by an operator "..". The prefix address is the address geometrically nearest to the origin, and the post fix address that furthest away. Thus the entire example case space would be defined by the N-cube:

$$\{0..308\}$$

and the cases presented in Figure 2 thus:

$$\{18\} \qquad \{2..50\} \qquad \{2..306\}$$

For convenience single address objects are not expressed in terms of a sequence. Finally a "plus" shape might be the defined as follows:

$$\{1..33, 16, 18\}$$

(this may be confirmed by reference to Figure 4). Note the ordering of cells according to the location of the prefix address within the linearisation.

4.3 Note on tesseral arithmetic

One of the advantages of the representation is that we can move through any space referenced in this manner using simple integer arithmetic. For example, to move one point to the right (given the orientation in Figure 4) from a given reference we simply add 1 ($Base_1$), to move in the opposite direction we subtract 1. If we wish to move the above "plus" shape two points to the right and one point upwards we simply add 18 to each address in the sequence:

$$\{1..33, 16, 18\} + \{18\} = \{19..51, 34, 36\}$$

(this can be verified by reference to Figure 4). If, alternatively, we wish to expand the "plus" in both directions along the D_1 axis by one point we add the N-cube $\{-1..1\}$. Moreover to complete this addition we only need to add the prefix operands in the first sequence to the prefix operands in the second sequence, and similarly with the postfix operands:

$$\{1..33, 16, 18\} + \{-1..1\} = \{0..34, 15..17, 17..18\}$$

(The overlap that this entails is not significant). From the above it is clear that to support translation in all directions through out the case space negative addressees are required. It is for this reason that the encoding includes negative equivalents of all values for each dimension

5 An Example Case Base

From the above we have identified an N-dimensional space D made up of the maximal set of addresses for a particular CBR application. The positive part of this space D_p is then the case space, i.e. the set of candidate addresses which may form part of any case. The CB, then comprises a set of sets of address such that each set within the overall set represents a case:

$$CB = \{C_1, C_2, \ldots, C_{M-1}, C_M\}$$

where cases C_1 to C_M are described in terms of subsets of D_p. For example the cases given in Figure 2 would be described thus:

$$CB = \{\{18\}, \{2..50\}, \{2..306\}\}$$

6 Case Retrieval

Given a CB of the above form we can now direct CBR queries at it. The most straight forward approach would of course be to use SQL style queries to find particular cases. Thus using the CB given in Figure 2 we might ask:

```
* SELECT case FROM
* casebase WHERE
* A_{1} = 0, A_{2) = 1, A_{3} = 2;
```

"Find all the cases with the given set of attribute values". Alternatively:

```
* SELECT case FROM
* casebase WHERE
* A_{2} = 1, A_{3) = 2;
```

where any value for A_1 is appropriate. Although this provides an ideal front end the resolution of such queries with respect to the representation requires translation into an intermediate form. A constraint formalism has thus been identified, the broad syntax for which is presented in Table 1. The second example SQL query will therefor translate into:

$$constraint([CB], \cap, [\{272..279\}]).$$

<constraints>	: <constraint>
	| <constraint> <constraints>
	;
<constraint>	: 'constraint(' <operand> ',' <operator> ',' <operand> ').' ;
<operand>	: '[' <identifier> ']'
	| '[' <identifier> ',' <quantifiers> ']'
<identifier>	: *A set of sets of addresses (e.g. the entire case base)*
	| *A single set of address (e.g. an individual case)*
	;
<quantifiers>	: <quantifier>
	| <quantifier> ',' <quantifiers>
	;
<quantifier>	: 'size(' *A positive integer* ')'
	| 'offset(' *A single set of address* ')'
	;
<operator>	: *The set of standard set operators* ;

Table 1: BNF Syntax for constraint expression

where the parameters for the "target value set", the sequence {272..279} (Figure 5(a)), are calculated as follows:

$$f_{cart2tess}(<0,1,2>) \rightarrow 272$$
$$f_{cart2tess}(<4,1,2>) \rightarrow 276$$

($A-1$ can take any value from 0 to 4). The constraint satisfaction mechanism used will then attempt to resolve this constraint by identifying those case objects in the CB that intersect with the target set {273..279}. In the example this will be the set {{2..306}}, i.e. there is only one case in the CB that satisfies the constraint (case 3). If no matching cases are found the above will return {}.

If exact matches are required we would need to refine our constraint as follows:

$$constraint([CB, size(1)], \subseteq, [\{272..279\}]).$$

"Find me all the cases described by a single cell (i.e. the set of addresses describing the case has a cardinality of 1) which are a subset of the N-cube 272..279." If we are only interested in cases for which all values have been instantiated then the constraint would need to be expressed as follows:

$$constraint([CB, size(1)], \cap, [\{272..279\}]).$$

or

$$constraint([CB, size(1)], \subseteq, [\{272..279\}]).$$

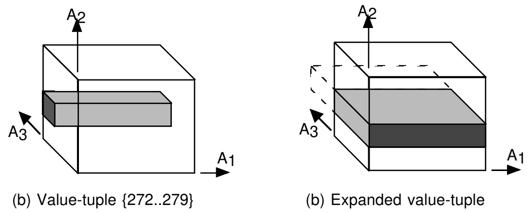

Figure 5: Example shape definitions that might form part of CBR queries

None of the cases (subsets of CB) in the example case base would satisfy the constraint.

In the context of the proposed CBR system all CB interaction can be carried out in this manner. Further, queries can be expressed as a sequence of constraints.

The advantage offered by the above in terms of computational effectiveness is that the number of comparisons required to establish the result is significantly less than that required to compare individual rows in (say) a table of cases — in many instances we only need to compare the single integers representing each case as opposed to groups of attribute values (integer or otherwise).

A further advantage offered by the approach is where no exact match is discovered. Because of the arithmetic that the representation supports (sub-section 4.3) the required value set can be expanded simply and efficiently. This can be achieved in a uniform manner, or along particular axes and/or particular directions. For example if we wish to extend our search by expanding the target value set by one cell in both directions parallel to the A_3 attribute we revise the above constraints as follows:

$$constraint([CB, size(1)], \cap, [\{272..279\},$$
$$offset(\{-256..256\})]).$$

The "offset" quantifier included here indicates that the set {−256..250} should be added to the target value set {272..279} before the intersect operation is performed. The effect is to produce a shape of the form given in Figure 5(b) (the portion of the shape outside the case space indicated by the broken line is of no interest in the context described here).

7 Conclusions and Discussion

In the foregoing we have used a N-dimensional spatial representation to reference a CB and interact with it using a constraint based formalism. The straight forward case retrieval is achieved in a similar manner as if an SQL style query had been used — although the result is arguably obtained in a more computational effective manner (especially given a large CB with many attributes).

The expansion technique (offsets) which may be invoked when ever a direct hit is not made is more interesting in that it is computationally cheap to expand a target set, particularly given a large number of attributes. In addition the expansion does not necessarily require detailed knowledge of the application — in the absence of any information dictating otherwise an attribute set can be expanded uniformly in all directions. Whereas, with the presence of knowledge, attributes can be expanded by different amounts along different dimension, equivalent to the weighting of attributes in a nearest neighbour algorithm.

The paper clearly establishes that the CB can be advantageously represented in a spatial context. The principal advantages are:

- A case and all its attributes (regardless of how many) can be represented as a single integer. This in turn offers computational advantages especially where a large number of cases with a high dimensionality are under consideration.
- The computational advantages are of particular relevance when we wish to expand a given set of attributes so as to identify similar cases
- The approach is also perfectly capable of operating with missing information or limited information unlike many other spatial access methods [Kamp *et al.*, 1998].

The described approach has been implemented in a "proof-of-concept" system which, although not a full CBR system, in that it does not include the revise and retain processes traditionally associated with CBR systems whereby the system can adapt and learn [Aamodt and Plaza, 1994], it does demonstrated the viability of the representation and retrieval approach. The authors are therefore encouraged by the results to date and are currently seeking to expand the work.

References

[Aamodt and Plaza, 1994] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(i):39–59, 1994.

[Bell *et al.*, 1983] S.B.M. Bell, B.M. Diaz, F.C. Holroyd, and M.J.J. Jackson. Spatially referenced methods of processing raster and vector data. *Image and Vision Computing*, 1(4):211–220, 1983.

[Bentley, 1975] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(ix):509–17, 1975.

[Coenen *et al.*, 1998a] F.P. Coenen, B. Beattie, T.J.M. Bench-Capon, B.M. Diaz, and M.J.R. Shave. Spatio-temporal reasoning using a multi-dimensional tesseral representation. In *Proceedings ECAI'98*, pages 140–144, 1998.

[Coenen *et al.*, 1998b] F.P. Coenen, B. Beattie, M.J.R. Shave, T.J.M. Bench-Capon, and B.M. Diaz. Spatial reasoning using the quad-tesseral representation. *Artificial Intelligence Review*, 12:321–343, 1998.

[Diaz and Bell, 1986] B.M. Diaz and S.B.M. Bell. *Spatial Data Processing Using Tesseral Methods.* Natural Environment Research Council publication, Swindon, England, 1986.

[Freeston, 1995] M. Freeston. A general solution of the n-dimensional b-tree problem. *ACM SIGMOD Record*, 24(ii):80–91, 1995.

[Friedman *et al.*, 1997] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM transactions Math Software*, 3:209–226, 1997.

[Fukunaga and Narenda, 1975] K. Fukunaga and P.M. Narenda. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Transactions on Computers*, 24:750–753, 1975.

[Gunter, 1989] O. Gunter. The design of the cell tree: an object-oriented index structure for geometric databases. In *Proceedings 5th. Int. Conf. On Data Engineering*, pages 598–605, 1989.

[Guttman, 1984] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormack, editor, *Proc. ACM SIGMOD 84*, pages 47–57, 1984.

[Hammond, 1986] K.J. Hammond. Chef: A model of case-based planning. In *Proceedings of, AAAI-86*, 1986.

[Kamp *et al.*, 1998] G. Kamp, S. Lange, and C. Globig. Related areas. In M. Lenz, B. Bartsch-Sporl, H.-D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology: From Foundations to Applications*, number 1400 in Lecture Notes in AI, pages 327–351, Berlin, 1998. Springer-Verla.

[Kitano *et al.*, 1992] H. Kitano, A. Shibata, H. Shimazu, and Kajiharaand A. Sato J. Building large-scale and corporate wide case-based systems. In *Proceedings of AAAI-92*, 1992.

[Kolodner, 1993] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufman, San Francisco, CA., 1993.

[Leake, 1996] D.B. Leake. Cbr in context: The present and future. In D.B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons and Future Directions*, pages 3–30, 1996.

[Lin *et al.*, 1994] K.I. Lin, H.V. Jagadish, and C. Faloutsos. The tv-tree: An index structure for high-dimensaional data. *VLDB Journal*, 3:517–542, 1994.

[Waltz, 1996] D. Waltz. Large-scale applications of cbr. In I. Smith and B. Faltings, editors, *Advances in Case-Based Reasoning*, number 1168 in Lecture Notes in AI, Berlin, 1996. Springer-Verla.

[Wess *et al.*, 1993] S. Wess, K.-D. Althoff, and G. Derwand. Using kd-trees to improve the retrieval step in case-based reasoning. In S. Wess, K.-D. Althoff, and M.M. Richter, editors, *Topics in Case-Based Reasoning*, number 837 in Lecture Notes in AI, pages 167–81, Berlin, 1993. Springer-Verla.