# Approximative Retrieval of Attribute Dependent Generalized Cases

**Rainer Maximini, Alexander Tartakovski**
University of Hildesheim
Institute for Mathematics and Applied Computer Science
Data and Knowledge Management Group
D-31141, Hildesheim, Germany
{r_maximi|tartakov}@dwm.uni-hildesheim.de

## Abstract

In our current research project about electronic designs the problem to retrieve attribute dependent generalized cases occurs. This kind of cases cover an arbitrary subspace rather than a point in the space, spanned by the cases attributes and by a set of constraints. For such representations, the similarity assessment between a point query and generalized cases is a difficult problem for which this paper presents a solution. We developed and evaluated a converter, which samples the spawn subspace for each case, selects reasonable point cases out of it, and creates a new case base with only point cases belonging to the current generalized case. For this case base well-known retrieval engines can be used where only less modifications have to be made.

## 1 Introduction

In CBR applications, the traditional concept of a case is that of a point in the *attribute space*, spawned by the cases' attributes. Commonly, this space is called *problem–solution space* when the attributes can unambiguously be related to the problem description or the solution description, respectively. Driven by examinations of several new applications (see 1.1), we proposed the concept of *generalized cases* [Bergmann *et al.*, 1999; Bergmann and Vollrath, 1999; Bergmann, 2002]. A generalized case covers not only one point of the attribute space, but a whole subspace of it. A single generalized case immediately provides solutions to a set of closely related problems rather than to one single problem only. The solutions a generalized case represents are very close to each other; basically they should be considered as (slight) variations of the same principle solution. In general, a single generalized case can be seen as an implicit representation of a (possibly infinite) set of traditional "point cases". We assume that the similarity to a generalized case is the similarity to the most similar point of the case.

We also want to make clear that the idea of generalizing cases is not a radically new concept. It was already implicitly present since the very beginning of CBR and instance-based learning research [Kolodner, 1980; Bareiss, 1989; Salzberg, 1991]. However, in this paper we explore a more formal and systematic view on generalized cases by using constraints to express the dependencies between several attributes. This partially covers also the above mentioned related work.

This paper is structured in the following way: The end of the current section presents our application scenario followed by section two where the foundations of generalized cases are defined and where the research problem is clarified. The third section deals with existing solutions for a partial problem, whereas section four presents our approach to solve the whole retrieval problem. The succeding section five introduces the test domain and case base and evaluates our approach. We close our paper with a summary and give ideas for future developments in this area.

### 1.1 Our Application Domain: Electronic Design IPs

Increasingly, electronics companies integrate *Intellectual Properties* (IPs) from third parties within their complex electronic systems. An IP is a design object whose major value comes from the skill of its producer [Lewis, 1997], and a redesign of it would consume significant time. However, a designer who wants to reuse designs from the past must have a lot of experience and knowledge about existing designs, in order to be able to find candidates that are suitable for reuse in his/her specific new situation. Currently, searching electronic IP databases can be an extremely time consuming task because of two main reasons: On the one hand, the public-domain documentation of IPs is very restricted and on the other hand there are currently no intelligent tools to support the designer in deciding whether a given IP from a database meets (or at least comes close to) the specification of his/her new application. This is one objective of the current project *IPQ: IP Qualification for Efficient Design Reuse*[1] funded by the German Ministry of Education and Research (BMBF) and the related European Medea project *ToolIP: Tools and Methods for IP*[2].

IPs usually span a design space because they are descriptions of flexible designs that have to be synthesized to hardware before they actually can be used. The behavior of the final hardware depends on a number of *parameters* of the original design description. The valid value combinations for these parameters are constrained by different criteria for each IP.

[1] IPQ Project (12/2000 - 11/2003). Partners: AMD, Fraunhofer Institute for Integrated Circuits, FZI Karlsruhe, Infineon Technologies, Siemens, Sciworx, Empolis, Thomson Multi Media, TU Chemnitz, University of Hildesheim, University of Kaiserslautern, and University of Paderborn. See www.ip-qualifikation.de

[2] See toolip.fzi.de for partners and further information.

## 2 Foundations

### 2.1 Case Classification

Maximini, Maximini, and Bergman [Maximini *et al.*, 2003] introduced a formal classification schema for cases in attribute-value representation. We consider the *attribute space* $\mathbb{A} = T_1 \times \cdots \times T_n$. $T_i$ are atomic types which are data types with single values like integer, float, boolean, symbol, string, date, or time. Each case has $n$ attributes $A_1, \ldots, A_n$ with an associated type $T_i$.

We start our definitions with the easiest kind of cases, namely the *point cases*.

**Definition 2.1** *A* point case *PC is exact one element of the attribute space, $PC \in \mathbb{A}$, i.e., each attribute of PC has a concrete atomic value, unequal to $unknown$.*

A point case, e.g. c1 in figure 1, can be regarded as a special instance of another concept, namely *generalized cases*.

**Definition 2.2** *A* generalized case *GC is an arbitrary subspace of the attribute space, $GC \subseteq \mathbb{A}$.*

Generalized cases can be interpreted as an (infinite) set of point cases. The main difference in the following two definitions is the kind how this set is defined. The most simple form of a GC is the *attribute independent generalized case* (AIGC).

**Definition 2.3** *Assume there exists sets $\mathbb{A}_1, \ldots, \mathbb{A}_n$ with $\mathbb{A}_i \subseteq T_i$. An* attribute independent generalized case *AIGC is defined as $AIGC = \mathbb{A}_1 \times \cdots \times \mathbb{A}_n$. Consequently, an AIGC is a special subspace of the attribute space, $AIGC \subseteq \mathbb{A}$.*

Attribute independent generalized cases often occur in applications in which not all attributes of the case are known or where attribute values have been removed, e.g. by maintenance operations. Unknown attribute values are usually interpreted as arbitrary, i.e., they can be considered to spawn the whole attribute dimension in $\mathbb{A}$. Hence, this can be represented by $\mathbb{A}_i = T_i$, e.g. c2 in figure 1. Another common reason for a case to be an AIGC is because some attributes hold intervals (range) or sets as values, see c3. Further, a query is usually an AIGC, because the solution attributes are not defined.

The second form of GCs are the *attribute dependent generalized cases* (ADGC).

**Definition 2.4** *A generalized case $ADGC \subseteq \mathbb{A}$ that cannot be represented as an AIGC is called* attribute dependent generalized case.

Hence, a GC is attribute dependent if the subspace it represents cannot be decomposed into independent subsets for each attribute. Dependencies can be expressed by constraints like demonstrated by c4 and c5 in figure 1. Therefore, these cases are more complex and difficult to handle during retrieval, depending on the form of the spawned subspaces. Roughly speaking, AIGCs spawn a subspace which is orthogonal (except AIGC with sets) and are therefore manageable in case that similarity measures are also decomposed according to the attributes. For ADGCs the subspace is arbitrary and hence it may become computationally very hard to determine the similarity to a PC or another GC [Bergmann *et al.*, 1999].

The main focus in this paper is the retrieval step between a PC or AIGC query and a case base of ADGC. For this step it does not matter if the query is a PC or an AIGC.

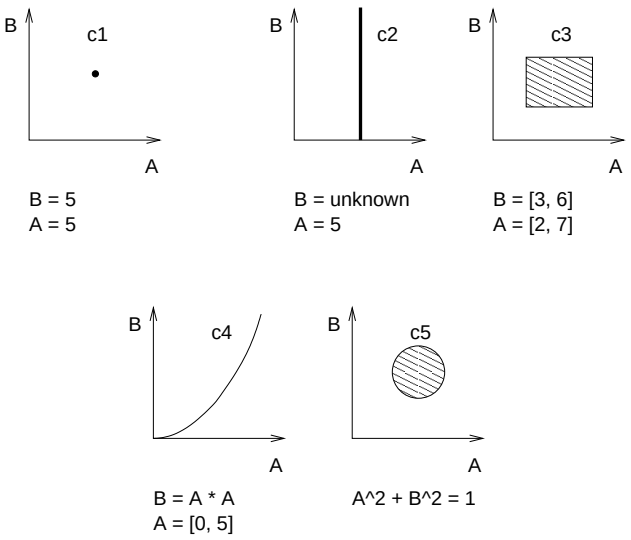



Figure 1: Examples for the three kind of cases

### 2.2 The Research Problem: Similarity Assessment and Retrieval

The important basic research issues involved when using ADGCs are related to representation formalisms, similarity assessment, and retrieval. One serious complication when studying these issues is that they are strongly connected with each other. Depending on the expressiveness of the representation formalism used for the ADGCs, similarity assessment is getting computationally more difficult, which also impacts the overall computational effort for retrieval from a large case base.

The similarity definition between a PC ($x$) and an ADGC ($y$) is presented in equation 1 and between an AIGC ($x$) and an ADGC ($y$) in equation 2.

$$\sup_{\forall y_t \in y} \{sim(x, y_t)\} \tag{1}$$

$$\sup_{\forall x_s \in x, y_t \in y} \{sim(x_s, y_t)\} \tag{2}$$

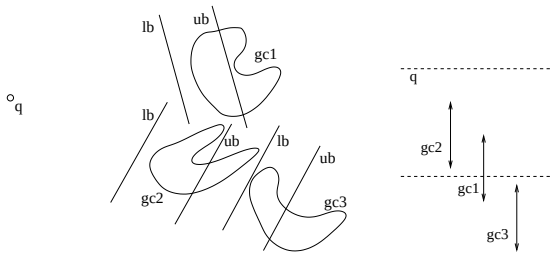
In these equations the association is used that a generalized case is a possibly infinite set of point cases and we are searching the two nearest points. Equation 1 is a special case of 2, because the query is a PC instead of an AIGC.

Nevertheless, the main problem is the computational complexity of such similarity functions why it is important to develop procedures that distinguish between an offline and an online phase. During the offline phase all computations should be done which are independent from the query, like creating an index. Unlike, the online phase is query dependent and should be very fast to reduce the response time of the retrieval. Consequently, the more calculations can be moved to the offline phase, the faster will be the online phase.

## 3 Existing solutions

Mougouie and Bergmann [Mougouie and Bergmann, 2002; 2003] present a mathematical optimization solution that instead of calculating the exact similarity between a query and each ADGCjust returns a ranking of the cases to find the most similar ones. Therefore, the upper and lower bounds for each case have to be calculated in relation to

the query so that they can be compared afterwards (see figure 2). This idea is similar to the fish and shrink algorithm [Schaaf, 1996].




(a) define (b) compare

Figure 2: Lower and Upper Bounds

A case can be ignored, if the lower bound of the case is higher than the upper bound of another case, e.g. in figure 2 the upper bound of $gc_2$ is lower than the lower bound of $gc_3$ why $gc_3$ can be removed. Until now, no statement can be made about $gc_1$ and $gc_2$, so their bounds have to be refined. The following algorithm presents the idea:

```
findMostSimilarCases(caseBase, query)
  remainingCases = caseBase
  while(isRefinementOfBoundsPossible())
    refineBoundsForEachCase()
    removePossibleCases(remainingCases)
  return remainingCases
```

This similarity assessment problem can now be formulated as an optimization problem:

$$\textbf{max}\ sim(q, x) \text{ s.t. } h(x) \leq 0,$$

where $h$ is a set of $m$ functions $h_1, h_2, \ldots$ that represents the ADGC by $h(x) \leq 0$, i.e., $x \in ADGC \iff \forall i\ h_i(x) \leq 0$. In this optimization problem, $max$ is the *objective function* to be maximized.

Mougouie and Bergmann use ADGCs that are represented through constraints over an n-dimensional real-valued vector space. It is shown that the difficulty depends on whether the ADGC spawns a convex or nonconvex subspace which is defined by the constraints. For convex constraints and by usage of convex similarity measures, the Topkis-Veinott method can be easily applied to determine exactly the similarity between a query (point case) and an ADGC. If the similarity measure is nonconvex or the generalized case contains also nonconvex constraints, the problem is more difficult. For this situation an algorithm is proposed that allows to incrementally compute sequences of upper and lower bounds for the similarity and assures the convergence of the algorithm. It allows to rank generalized cases without the exact computation of all similarity measures.

The current state of the algorithm has two main disadvantages. Firstly, it is only analyzed for real valued attributes. But in real live applications also symbolic values, integers or strings are needed. Secondly, the calculation of the bounds is complex and query dependent why it has to be done during the online phase. Presumably, this results in a high retrieval response time which is, for example in e-commerce application like our IPQ project, not acceptable.

# 4 Sampling Converter

As described above we made two main demands on our system:

- The retrieval response time should be short enough to use the system in e-commerce applications.
- All common attribute types should be handled.

Especially the last point forces us to reuse existing modules as much as possible, like retrieval engines for point cases. This results into a very simple concept:

- Transform the ADGC into point cases to use well researched and fast retrieval engines.
- To prevent a large increase of the case base size, which could destroy the fast response time, give the user influence onto the conversion process and use additional intelligent modules.

Of course, the quality of the retrieval result mainly depends on the sampling quality and the number of result cases. But the conversion is done in the offline phase where enough intelligence can be put into the converter to receive a sufficient case base quality which more likely results in a good retrieval quality. The online phase is nearly the same as for retrieval with traditional point cases. It has only to be granted that all retrieved cases originally belonged to different ADGCs or PCs, which will be described in section 4.2.

```
foreach adgc
  calculateTestCases(adgc)
  foreach testcase
    validateConstraints(testcase)
    if testcase valid
      pcCaseBase.add(testcase)
```

The retrieval result includes the concrete PC, its corresponding ADGC, and the similarity measure. With this information an additional retrieval explanation component can help the user to understand for which values the ADGC fits to the query. Additionally, the possible variations of the values can be explained and support can be given to refine the query.

## 4.1 Offline Phase

**Design of the Converter Core System**

To continuously evaluating and testing new ideas we have developed a modular converter with fixed interfaces to replace modules independently from others. This allowed us to implement first very simple modules which are exchanged by intelligent ones continuously. The four core modules are bounding box generator, test case generator, test case validator, and case base manager. Of course, there are more modules and components to realize such a converter, like case base reader and writer, but they are not of interest for the scope of this paper. The design of the converter is presented in figure 3 and explained in the following subsections.

The core system executes a loop for each ADGC whereby three case bases are used: origCaseBase is the original case base with ADGCs, pcCaseBase is the resulting case base with point cases only, and gcCaseBase is a temporary case base including the PCs of a current ADGC.

**Bounding Box Generator**

A big improvement was reached by restricting the possible values of each attribute. Therefore, for each attribute either the maximum and minimum values are calculated or a set of possible values. These values define a bounding box around the case for which several knowledge containers can be used. We have implemented two kinds of bounding box generators with different complexities and qualities:

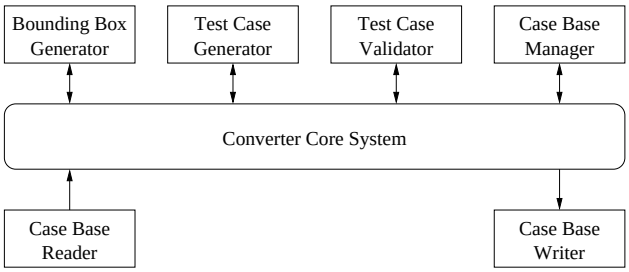



Figure 3: Modular Design of the Converter

1. The simplest one uses only the information from the model. For each attribute the data type definition is converted into the description of a bounding box. Any further information from other knowledge containers is ignored.
2. A more advanced one initializes the bounds like the previous one, but takes also the constraints of the ADGC into account. Each constraint is analyzed independently from other constraints or attributes, e.g., by a constraint with a case differentiation the possible result values are taken without checking the conditions.

Our tests have shown that the second one is a good balance between performance and quality. The additional computations to the first one are minimal why we used the second bouding box generator in our further tests.

**Test Case Generator**

The test case generator uses the information from the bounding box generator to generate test cases. Here we also have implemented three kinds of generator:

1. We started with a pure random generator which randomly selects values from the bounding box. In average this produces an equal distribution over the subspace of the case if enough test cases are generated.
2. Additionally to the previous one, information about the similarity measures are taken into account. Attributes with a high global similarity weight receive a finer distribution than attributes with a low weight. Although this does not lead to an equal distribution any more, the retrieval quality increases. Therefore, this solution should be preferred.
3. A further improvement was to increase the density at the border of the ADGC. The theory behind this extension is that the shortest distance between a query and a generalized case is on the border of the generalized case, except the query is an element of its subspace. To take this fact into account the probability for the attribute values at the edge of the bounding box are increased.

Each implementation returns a list of test cases which is furthermore sorted by the third algorithm, starting with the most important ones. This necessary later for the case base manager (see 4.1). For the evaluation we have used the second implementation, because the third one was not implemented so far.

**Test Case Validator**

It has to be checked if the test case is an element of the subspace of the ADGC. This is realized with a constraint checker by validating each constraint with the values of the test case. This is, of course, much more easy than propagating the constraints or using the mathematical optimization method as described in 3.

**Case Base Manager**

The case base manager creates for each generalized case an own case base where all corresponding point cases are collected. At the end of the ADGC conversion loop, this case base is merged with the global case base pcCaseBase. But some additionally tasks are further executed:

- The manager takes care about the size of the gcCaseBase. If the size grows above a user specified threshold all further cases are ignored. This implies that the most important cases should be added first which is granted by the test case generator.
- To recognize later which PC belongs to which ADGC, each PC is marked with the id of the ADGC. This mark is very important for the retrieval engine (see section 4.2) and for the result explanation component. This explanation component is necessary to explain the user why and with which concrete values his or her query fits to the retrieved cases.
- To improve the case base quality the case base manager takes care that the generated point cases are not too near to each other. This is done by a retrieval on the respective gcCaseBase with the possible new point case as query. Only if the similarity of the nearest case is lower than a user specified maximum similarity threshold ($maxSim$) the case is added to the gcCaseBase. Figure 4 presents the original point cases and the resulting point cases.

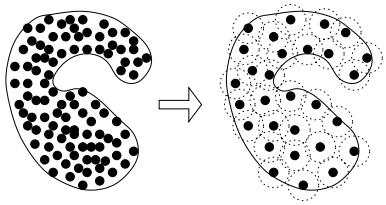


Figure 4: Retrieval Problem

The density can be adjusted by the similarity which is presented as the dotted circles around the point cases. This produces a good distribution over the ADGC and decreases the gcCaseBase's size without a lack of information or with only a small one for the retrieval. To guarantee a high coverage at the borders it is important to perform the adding in order of the importance of the cases.

## 4.2 Online Phase

After the conversion, the resulting point case base (pcCaseBase) includes a lot of PCs and AIGCs which belong to a few number of ADGCs. If a common retrieval engine would be used the retrieval result would be not the expected one. Imagine the case base in figure 5 with the four generalized cases, the 24 corresponding point cases, and the query.

If the three most similar cases would be requested the retrieval engine would only retrieve point cases which belong to the same ADGC. This is of course not the result expected by the user. To prevent this only one small modification to the retrieval engine had to be done. Each point case was marked by the case base manager (see 4.1) with a

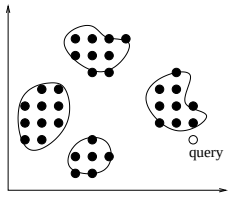



Figure 5: Retrieval Problem

label to identify the original ADGC. This label can now be used to decide if a new received point case $pc_n$ is added to the retrieval result list or not. We can distinguish between three cases:

1. The retrieval result list includes no case with the same referenced ADGC like $pc_n$, consequently $pc_n$ is added to the retrieval result list.
2. The retrieval result list includes a case $pc_l$ with the same reference and a lower similarity than $pc_n$, consequently $pc_n$ is added to the retrieval result list and $pc_l$ is removed from it.
3. The retrieval result list includes a case $pc_h$ with the same reference and a higher similarity than $pc_n$, consequently $pc_n$ can be ignored because $pc_h$ exists in the result list with a better similarity to the query.

# 5 Evaluation and Results

To evaluate the converter simplified IPs from our IPQ Project (see 1.1) are used. The original case model of an IP consists of more than 500 attributes, but by using them the results would have been too difficult to interpret. Therefore, we have selected seven typical attributes for our simplified model:

**FunctionalClass** : is an atomic value of 222 constant symbolic values which are ordered in a taxonomy.

**PowerConsumption** : is an atomic floating point value whose domain is defined in the range 0 to 500.0.

**MarketSegment** : is an atomic value of 51 constant symbolic values which are ordered in a taxonomy.

**CoreVoltage** : is a atomic floating point value whose domain is defined in the range 0 to 24.0.

**Hardness** : is an atomic value with the three symbolic constants Hard, Firm, and Soft.

**CacheSize** : is an atomic integer value whose domain is defined in the range from 0 to $2^{16}$ in exponential steps.

**ConfigurationRegister** : is an atomic integer whose domain is defined in the range from 0 to $2^8$.

Between these attributes several constraints can be defined for whose we have developed an own representation formalism [Maximini, 2002]. The following constraints are five typical examples:

$\texttt{<MarketSegment>} = \{Home, Military, Aerospace\}$

$\texttt{<Hardness>} = \{Hard, Firm\}$

$$\texttt{<CacheSize>} = \begin{cases} [0,32] & \text{if } \texttt{<Hardness>} == Hard \\ [0,512] & \text{if } \texttt{<Hardness>} == Firm \end{cases}$$

$$\texttt{<CoreVoltage>} = \begin{cases} 1.5 & \text{if } \texttt{<MarketSegment>} == Military \\ 1.1 & \text{if } \texttt{<MarketSegment>} == Aerospace \\ 2.5 & \text{else} \end{cases}$$

$$\texttt{<PowerConsumption>} = \texttt{<CoreVoltage>} * 0.5 + \texttt{<CacheSize>} * 0.2$$

The origCaseBase consists of 100 manually created ADGCs with different kinds of constraint. For the evaluation we focused our test on the following four questions:

- How much test cases should be generated?
- How stable is the conversion process?
- How stable is the retrieval?
- How good is the diversity optimization?

## 5.1 How much test cases should be generated?

To answer this question we have first calculated the maximum number of correct point cases manually. Thereby, the scala of the two attributes with floating point values had to be discretized (1000 values for each) to avoid an infinite set of point cases.

Figure 6 presents the relation between the number of generated test IPs (x-axis) and the percentage of coverage for 11 manually selected distinguished IPs (y-axis), that are identified by their ids. The percentage of coverage is the number of IPs (correct validated test IPs) divided by the manual calculated maximum number of PCs

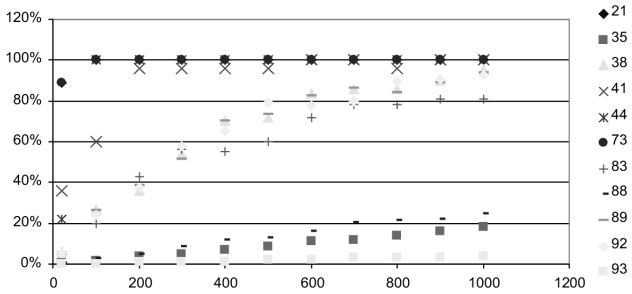



Figure 6: Case coverage for n test cases.

Three groups of ADGCs can be distinguished:

- ADGCs which reach very fast a coverage of 100% like case 21, 41, 44, and 73. The corresponding constraints consist in general of discrete values and of case differentiations whereby the results of the case differentiations are fixed values like for the CoreVoltage attribute. Consequently, the bounding box includes only these result values why the number of possible test cases is very small.
- ADGCs where the coverage increases very slow like case 35, 88, and 93. These cases contain constraints which are defined for attributes with floating point values and are mathematical functions for which the result values are not calculated. To get a coverage of 100% it would be necessary to generate an infinite number of test cases which is not possible.
- ADGCs where the coverage increases slowly like case 38, 83, 89, and 92. For these cases the constraints are defined for discrete values, but include also mathematical operations where the complete domain of the corresponding attribute is used by the bounding box generator. Other constraint types are case differentiations with intervals as result values. Consequently, there exist much more values than in the second case which increases the number of possible test cases.

Summarizing, the number of test cases depends on the kind of constraints and data types of the attributes. For

discrete values 1000 test cases are sufficient to get a coverage of at least 80%. But if the constraints are defined on attributes with floating point values the number has to be increased to 100.000 or more. Nevertheless, the correct number is very application dependent and has to be evaluated for each application by tests.

### 5.2 How stable is the conversion process?

Because of the random generation of test cases it is interesting to test the stability of the conversion process. Therefore, the number of test cases which are necessary to get a minimum coverage of 80% are calculated manually. For each case the conversion is done ten times with this minimum number of test cases and the corresponding coverage is calculated. Figure 7 presents the results.

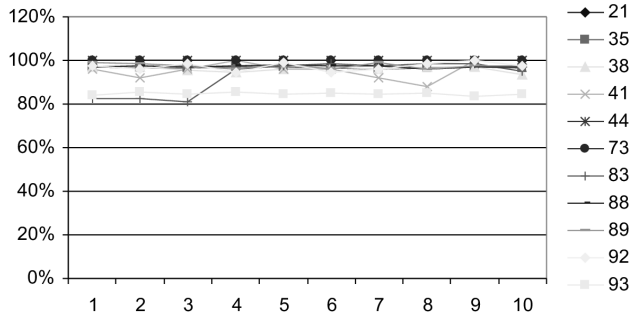



Figure 7: Stability of the Conversion Process

All cases reached the minimum coverage of 80% in each test run. For the case groups one and three of section 5.1 the coverage is constant, only in the second group are some variations of the coverage which is based on the poor handling of floating point value attributes.

### 5.3 How stable is the retrieval?

The previous subsection presented that an ADGC can be approximated by PCs which are randomly created. Now the question is analysed whether the retrieval on the resulting case base always returns the same cases in the same order. Therefore, the case base of 100 ADGCs was converted ten times in a case base of point cases. On each case base the same query was performed and the ranking of the 10 best cases is analysed. Figure 8 presents the results where the x-axis presents the ranking position and the y-axis the cases and the frequency how often it is returned at the corresponding ranking position.

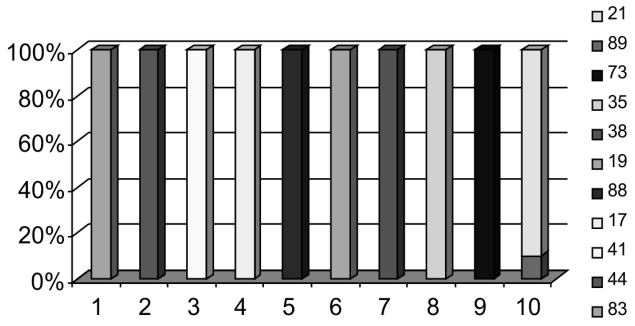



Figure 8: Stability of the Retrieval

The results have proofed that the retrieval is stable for the first ten best matching cases out of 100, except for one case base where at the tenth position another case was returned. Further analyses have shown that in this case case 21, which should have been returned at the tenth position was returned on the eleventh position. Several test runs with different queries have been performed and the stability of the retrieval could be reinforced.

### 5.4 How good is the diversity optimization?

Section 4.1 describes a similarity based optimization of the diversity of the PCs for each ADGC. For the previous evaluations a maximum similarity value ($maxSim$) of 1.0 was used, consequently no optimization was done. This section analyses the influence of different $maxSim$ values on the retrieval result as well as the performance improvement. Therefore, one reference case base with $maxSim = 1.0$ was created, four with $maxSim = 0.98$, and four with $maxSim = 0.95$. Furthermore, three different query cases have been build that have been used for all retrievals.

Table 1 presents the sizes of the case bases with different $maxSim$ values.

| $maxSim = 1.0$ | $maxSim = 0.98$ | $maxSim = 0.95$ |
|---|---|---|
| 85 MB (100%) | 195 KB (0.22%) | 110 KB (0.12%) |

Table 1: Case base size with different $maxSim$ values.

The following Table 2 presents the retrieval evaluation of three different queries. Each table contains the ranking of the case ids of the retrieval result for the nine case bases.

| **1.0** | **0.98** | | | | **0.95** | | | |
|---|---|---|---|---|---|---|---|---|
| 39 | 39 | 39 | 39 | 39 | 40 | 40 | 40 | 40 |
| 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 28 | 28 | 28 | 28 | 28 | 28 | 39 | 28 | 28 |
| 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 |
| 64 | 64 | 64 | 64 | 64 | 63 | 64 | 63 | 64 |
| 40 | 40 | 40 | 40 | 40 | 55 | 63 | 55 | 63 |
| 76 | 76 | 76 | 76 | 76 | 64 | 76 | 64 | 76 |
| 96 | 96 | 96 | 96 | 96 | 88 | 96 | 88 | 96 |
| 88 | 88 | 88 | 88 | 88 | 76 | 88 | 76 | 88 |

| **1.0** | **0.98** | | | | **0.95** | | | |
|---|---|---|---|---|---|---|---|---|
| 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 | 39 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 44 | 44 | 44 | 44 | 44 | 54 | 54 | 44 | 44 |
| 54 | 54 | 54 | 54 | 54 | 60 | 44 | 54 | 54 |
| 60 | 60 | 60 | 60 | 60 | 75 | 60 | 60 | 60 |
| 85 | 76 | 85 | 85 | 85 | 88 | 85 | 85 | 85 |
| 76 | 85 | 76 | 76 | 76 | 85 | 76 | 76 | 76 |
| 75 | 75 | 75 | 75 | 75 | 76 | 75 | 75 | 75 |

| **1.0** | **0.98** | | | | **0.95** | | | |
|---|---|---|---|---|---|---|---|---|
| 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| 54 | 55 | 54 | 54 | 55 | 54 | 54 | 54 | 54 |
| 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 |
| 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| 39 | 39 | 39 | 39 | 54 | 39 | 39 | 39 | 39 |
| 70 | 75 | 55 | 76 | 70 | 55 | 70 | 76 | 70 |
| 55 | 76 | 70 | 55 | 39 | 70 | 55 | 55 | 55 |
| 76 | 88 | 88 | 70 | 76 | 76 | 76 | 70 | 76 |

Table 2: Retrieval results of three different queries.

The ranking between $maxSim = 1.0$ and $maxSim = 0.98$ is nearly identically, only few cases differ in one position. Therefore, the retrieval result can be defined as

identically in respect of the random generation where also the ranking of $maxSim = 1.0$ can slightly differ as described in section 5.3. But with $maxSim = 0.95$ the situation has changed dramatically because the ranking differs very strong by several positions and some cases are missing completely in the top ten.

With the resulting case base's size in mind it is recommend to use this optimization, because it decreases the case base's size extremely without a significant lack of information for the retrieval, but to choose the $maxSim$ value carefully.

## 6 Summary and Outlook

We have presented a realization for a retrieval on attribute dependent generalized cases with a configurable converter. It transforms a case base with ADGCs into a case base with point cases or AIGCs for which common retrieval engines can be used with only few modifications.

Furthermore, we have done several tests to evaluate the quality of the converter, substantiating that it is a stable and sufficient technique for our IP domain in the IPQ project. Especially the similarity based optimization of the final case base decreases the case base's sizes dramatically without reducing the retrieval quality.

Nevertheless, the handling of attributes with floating point values has to be improved. One possibility is to adapt the constraint validater to make a similarity comparison of the values instead of an exact one. Also for the test case generator and for the bounding box generator exist enough space for improvements.

## References


[Ashley and Bridge, 2003] Kevin D Ashley and Derek G Bridge, editors. *Case-Based Reasoning Research and Development, Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCBR'03)*, Lecture Notes in Artificial Intelligence, 2689. Springer Verlag, 2003.

[Bareiss, 1989] Ray Bareiss. *Exemplar-Based Knowledge Acquisition: A unified Approach to Concept Representation, Classification and Learning*. Academic Press, 1989.

[Bergmann and Vollrath, 1999] R. Bergmann and I. Vollrath. Generalized cases: Representation and steps towards efficient similarity assessment. In W. Burgard, Th. Christaller, and A. B. Cremers, editors, *KI-99: Advances in Artificial Intelligence.*, LNAI 1701. Springer, 1999.

[Bergmann *et al.*, 1999] R. Bergmann, I. Vollrath, and T. Wahlmann. Generalized cases and their application to electronic designs. In E. Melis, editor, *7. German Workshop on Case-Based Reasoning (GWCBR'99).*, pages 6–19, 1999.

[Bergmann, 2002] Ralph Bergmann. *Experience Management - Foundations, Development Methodology, and Internet-Based Applications*. Lecture Notes in Artificial Intelligence 2432. Springer Berlin, Heidelberg, New York, Hong Kong, London, Milan, paris, Tokyo, 2002.

[Kolodner, 1980] Janet L Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory*. PhD thesis, Yale University, 1980.

[Lewis, 1997] Jeff Lewis. Intellectual property (IP) components. Artisan Components, Inc., [web page], http://www.artisan.com/ip.html, 1997. [Accessed 28 Oct 1998].

[Maximini *et al.*, 2003] Kerstin Maximini, Rainer Maximini, and Ralph Bergmann. An investigation of generalized cases. In Ashley and Bridge [2003], pages 261–275.

[Maximini, 2002] Rainer Maximini. Ipq: Concepts for the representation of parametrized ip by constraints in the sense of artificial intelligence. Technical report, University of Hildesheim, Data and Knowledge Management Group, P.O.Box 101363, D-31113 Hildesheim, Germany, August 2002.

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

[Mougouie and Bergmann, 2003] Babak Mougouie and Ralph Bergmann. Diversity-conscious retrieval from generalized cases: A branch and bound algorithm. In Ashley and Bridge [2003], pages 319–331.

[Salzberg, 1991] S Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:277–309, 1991.

[Schaaf, 1996] Jörg W. Schaaf. Fish and shrink: a next step towards efficient case retrieval in large scaled case bases. In Smith and Faltings [1996], pages 362–376.

[Smith and Faltings, 1996] Ian Smith and Boi Faltings, editors. *Advances in Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, 1186. Springer Verlag, 1996.