Lecture Notes in Computer Science, IEA/AIE*98, Springer-Verlag

An Efficient Approach to Iterative Browsing and Retrieval for Case-Based Reasoning*

Igor Jurisica[1] and Janice Glasgow[2]
[1] Univ. of Toronto, Faculty of Information Studies, 140 St. George Str., Toronto, ON M5S 3G6, Canada, jurisica@fis.utoronto.ca
[2] Queen's Univ., Dept. of Computing and Inf. Sci., Kingston, ON K7L 3N6, Canada, janice@qucis.queensu.ca

Abstract. A case base is a repository of past experiences that can be used for problem solving. Given a new problem, expressed in the form of a query, the case base is browsed in search of "similar" or "relevant" cases. One way to perform this search involves the iterative evaluation of a series of queries against the case base, where each query in the series is obtained by restricting or relaxing the preceding query.
The paper considers alternative approaches for implementing iterative browsing in case-based reasoning systems, including a naive algorithm, which evaluates each query independent of earlier evaluations, and an incremental algorithm, which reuses the results of past query evaluations to minimize the computation required for subsequent queries. In particular, the paper proposes an efficient algorithm for case base browsing and retrieval using database techniques for view maintenance. In addition, the paper evaluates the performance of the proposed algorithm with respect to alternative approaches considering two perspectives: *(i)* experimental efficiency evaluation using diverse application domains, and *(ii)* scalability evaluation using the performance model of the proposed system.

1 Introduction

There are many applications where iterative browsing is an effective way of obtaining information and supporting collaboration [26]. It is particularly useful for exploratory search in complex domains and for problem solving tasks that involve considering alternative situations (e.g., examining "what-if" scenarios). Such applications include knowledge mining, on-line analytical processing, case-based reasoning (CBR), etc. In order to support iterative browsing, a system must be able to respond to a series of queries, where the system or the user may alter a current query and re-submit it for further evaluation. This process may be repeated until the desired quality and quantity of information is obtained. The iterative process is aimed at maintaining high recall while improving precision.

* Authors are indebted to J. Mylopoulos for comments and suggestions on an earlier draft. The research has been supported by IBM Centre for Advanced Studies, Canadian Software Engineering Research, and Information Technology Research Centre of Ontario.

A naive approach to iterative browsing involves evaluating each query independently. A more sophisticated approach involves incremental computation [5, 9, 10], where the result of a query is reused to evaluate a subsequent query more efficiently. The amount of necessary modification to a query in an incremental approach can be, to some degree, controlled by collecting and using extra information produced during query evaluation. Although this may require additional storage, overall performance is usually improved.

This paper proposes an incremental retrieval algorithm for CBR systems. Retrieval is based on a nearest-neighbor matching algorithm [28], which was modified as follows: *(i)* attributes are grouped into categories of different importance to help control the matching process and diminish the negative effect of irrelevant attributes on performance; *(ii)* an explicit context is used during similarity assessment to ensure that only relevant cases are retrieved; and *(iii)* incremental context transformation are applied during query relaxation to speed-up query-processing.

Informally, a case comprises an input, an output and a feedback (the problem, the solution and an evaluation of the solution). After a CBR system accepts a new problem description (a case without a solution and feedback) from a user or another part of the system, it retrieves relevant cases from a case base (past problems with similar input). These cases are then adapted (either by an expert or by the system) to produce a solution for the input problem. The final step of CBR involves evaluating the solution and producing feedback information.

CBR relies on the concept that similar problems have similar solutions. Thus, a case retrieval algorithms must find the most similar/relevant cases for a particular query. The more relevant a case is to a problem, the less adaptation is needed and the more precise the solution. Generally, it is better to retrieve fewer cases than to retrieve less relevant cases that result in a poor solution. A major task in CBR is to measure case relevance so as to retrieve only highly-relevant cases. We propose an iterative process, which incrementally modifies a set of retrieved cases, to address this goal.

The research presented in this paper has three primary objectives: *(i)* to define an iterative browsing system for case retrieval; *(ii)* to adapt an incremental view maintenance algorithm used in database systems for use in CBR; and *(iii)* to theoretically and experimentally evaluate the resulting system. An emphasis is put on the performance of the proposed retrieval component for a CBR system. Results of performance evaluation suggest that a CBR system with the iterative retrieval method would scale up for large and complex case bases. However, in some well-defined and relatively-small domains, traditional CBR algorithms currently provide an adequate solution for case retrieval.

2 System Description

A *case*, $\mathcal{C}$, is a representation of a real-world experience in a particular representation scheme. A case will be represented as a finite set of attribute-value pairs (descriptors): $\mathcal{C} = \{\langle a_0.V_0\rangle, \langle a_1.V_1\rangle, \cdots, \langle a_n.V_n\rangle\}$, where $A_i = \langle a_i.V_i\rangle$ is an attribute-value pair. Individual attributes and values are grouped into one or more categories. *Category* membership is defined either using domain knowledge (if available) or using a knowledge-discovery algorithm [7]. Different constraints may be ascribed to different categories, i.e. individual groups of attributes, which helps diminish the negative effect of irrelevant attributes.

Context plays an important role during case retrieval. Our primary goal is to retrieve only useful cases, i.e., cases highly relevant to a problem represented as a partially developed target case (a query). Since the usefulness of individual attributes during matching varies, we propose to incorporate the notion of a context to define what attributes are relevant in a given situation.

A *context*, Ω, is defined as a finite set of attributes with associated constraints on the attribute values: $\Omega = \{\langle a_0.CV_0\rangle, \cdots, \langle a_k.CV_k\rangle\}$, where a_i is an attribute name and constraint CV_i specifies the set of "allowable" values for attribute a_i. A context specifies relevant attributes and how "close" an attribute value must be in order to satisfy the context.

A context is formed using the most relevant attributes for a given target case; similarity is determined as a closeness of values for the prescribed attributes, taking attribute value distribution and the task being solved into account. The attributes and constraints for a context can be specified by a user or derived automatically. In general, the context can be specified using the following scenarios: *(i)* Task-based retrieval: The user specifies the task to be solved and the system selects an appropriate context. *(ii)* Query-by-example: The user selects a source case from a case base to be used as a problem example. Case attributes and values are then used as attributes and constraints in a context. *(iii)* The user has enough domain knowledge[2] to specify the context directly. *(iv)* Retrieval-by-reformulation: The user submits an initial unrefined query, reviews the resulting solution, and then iteratively modifies and resubmits the query. This approach is best suited for repository browsing.

If the interpretation of the case in a case base (a source case) satisfies the given constraints for the specified attributes, then it is considered *relevant* for a given context. A case $\mathcal{C}$ *satisfies* a context Ω, denoted $sat(\mathcal{C}, \Omega)$, if and only if for all pairs $\langle a_i.CV_i\rangle \in \Omega$, there exists a pair $\langle a_i.V_i\rangle \in \mathcal{C}$ such that V_i is in CV_i:

$$sat(\mathcal{C}, \Omega) \ iff \ \forall a_i \langle a_i.CV_i\rangle \in \Omega \rightarrow \exists V_i \langle a_i.V_i\rangle \in \mathcal{C} \wedge V_i \in CV_i. \qquad (1)$$

All cases relevant with respect to a given context are considered similar in a given context. The process of retrieving relevant cases can then be described as a constraint satisfaction process [25].

Two types of constraints can be imposed on attributes and their values. *Cardinality constraints* specify the number of attributes required to match for a particular category. This is an extension of an *x-of-n* matching algorithm [19], since different categories may have different matching requirements, i.e., different x. Individual categories are ordered according to the importance of their attributes. If no priority is assigned to a category, they are accessed sequentially. If the case has only one category, the matching is equivalent to *x-of-n* matching.

[2] Relevant attributes can be located using a knowledge-discovery algorithm [17].

However, if more categories are defined, then important attributes may require *n-of-n* matching, less important attributes *x-of-n* matching (for $x < n$) and irrelevant attributes may be eliminated from consideration altogether.

Value constraints specify constraints on attribute values. They include: *(i) instance matching* – the attribute value in the source case must match the attribute value in the target case; *(ii) set matching* – the attribute value in the source case must be included in the set of allowed values specified in the context; *(iii) interval matching* – the attribute value in the source case must be within an interval specified by the context; and *(iv) domain matching* – the attribute value in the source case must be an instance of the constraint in the context.

Variable-context similarity-based retrieval is monotonic [16]. For the purpose of case retrieval, the similarity relation maps a context Ω and a case base Δ onto the set of cases SC in the case base that satisfy the context. Given a case base Δ and a context Ω, the retrieval function returns a non-empty set of relevant cases $SC \subseteq \Delta$ such that all cases in SC satisfy the given context. The retrieval function is complete in the sense that it returns all relevant cases and only relevant cases. In the next section we define context transformations that relax or restrict category and value constraints during iterative case retrieval.

3 Context Relaxation and Restriction

An explicitly defined context controls the closeness of retrieved cases. If too many or too few relevant cases are retrieved using the initial context, then the system automatically transforms the context or the user manually modifies it. The context transformation process controls the quality and quantity of retrieved cases and thus when transforming the context, the system may return an approximate answer quickly or may spend more resources to calculate a more accurate answer [14]. An approximate answer can be iteratively improved, so that the change between an approximate and an accurate answer is continuous. This is an important feature for bounded resource computation [6, 12].

We propose two context transformations as a foundation for supporting iterative retrieval and browsing: *relaxation* – to retrieve more cases, and *restriction* – to retrieve fewer cases.

3.1 Context Relaxation

A context Ω_1 is a *relaxation* of a context Ω_2, denoted $\Omega_1 \succ \Omega_2$, if and only if the set of attributes for Ω_1 is a subset of the set of attributes for Ω_2 and for all attributes in Ω_1, the set of constraints in Ω_2 is a subset of the constraints in Ω_1. As well, contexts Ω_1 and Ω_2 are not equal.

$$\Omega_1 \succ \Omega_2 \quad \mathit{iff}\ \forall \langle a_i.CV_i \rangle \in \Omega_1, \exists \langle a_i.CV_j \rangle \in \Omega_2 : CV_i \supseteq CV_j \wedge \Omega_1 \neq \Omega_2. \qquad (2)$$

We propose to implement context relaxation as *reduction* and *generalization*.[3] *Reduction* removes an attribute-value pair from a context, either permanently

[3] Reduction and generalization implement cardinality and value constraints respectively.

or dynamically – given *x-of-n* matching, the required number of attributes is reduced from x to y, where $0 < y < x \leq n$. *Generalization* relaxes the context by enlarging the set of allowable values for an attribute.

3.2 Context Restriction

A context Ω_1 is a *restriction* of a context Ω_2 if and only if Ω_2 is a relaxation of Ω_1: $\Omega_1 \prec \Omega_2$ *iff* $\Omega_2 \succ \Omega_1$. We implement context restriction as *expansion* and *specialization*. *Expansion*, an inverse operation to reduction, strengthens constraints by enlarging the number of attributes required to match. *Specialization* strengthens constraints by removing values from a constraint set for an attribute. This may lead to a decreased number of cases that satisfy the resulting context.

3.3 Transforming Context Iteratively

The relaxation technique can advantageously be used for returning answers to a specific query as well as returning related answers [8]. Without an automatic query relaxation, users would need to submit alternative queries. The restriction technique works analogously, but is used mainly for controlling the amount of returned information, preventing information overload. Since the search for relaxed or restricted contexts could be infinite, there must be a mechanism for controlling it, either by user intervention (via user preferences) or by other means.

Iterative restrictions or relaxations of a context produces a chain of contexts $(\Omega_0, \Omega_1, \cdots, \Omega_n)$ and corresponding sets of retrieved cases $(SC_0, SC_1, \cdots, SC_n)$. This process is monotonic in the sense that if the successive contexts are partially ordered then the retrieval function produces a corresponding effect on the sets of cases retrieved. More formally, if SC_i is the set of cases retrieved for context Ω_i, then a partial order (by relaxation) of contexts results in a partial order (by subset) of case sets: If $\Omega_0 \prec \Omega_1 \prec \cdots \prec \Omega_n$, then $SC_0 \subseteq SC_1 \subseteq \cdots \subseteq SC_n$. Previously we have shown that if Ω_1 is a relaxation of Ω_2 then all cases that satisfy Ω_2 also satisfy Ω_1 [16]:

$$\forall \mathcal{C} \in \Delta : (sat(\mathcal{C}, \Omega_2) \wedge \Omega_1 \succ \Omega_2) \rightarrow sat(\mathcal{C}, \Omega_1). \tag{3}$$

A context can also be iteratively restricted by making it progressively more specific, i.e., allowing fewer cases to satisfy it. As for relaxation, expansion and specialization produce a partial order of contexts.

The process of restricting and relaxing contexts can be repeated and interwoven until the agent is satisfied with the quantity and the relevance of the retrieved cases. It is apparent that after the context is modified by relaxation/restriction, the system must re-evaluate the query. A naive approach takes the new query and submits it to the system (see Figure 1). A more sophisticated approach could take advantage of an already processed query by incrementally modifying its result [2]. In the next section, we introduce such an algorithm.

```
iterativeRetrieve (Ω, Δ, LowerLimit, UpperLimit)
  initialize SC to ∅
  for all cases in the case base
      retrieve (C_i)
      if  C_i satisfies the context (see Equation 1) then
          add (C_i, SC)
  if  | SC | ≤ LowerLimit then
      iterativeRetrieve (relax (Ω, Category), Δ, LowerLimit, UpperLimit)
  else if | SC |> UpperLimit then
      if the Ω was not previously relaxed then
         iterativeRetrieve (restrict (Ω, Category), Δ, LowerLimit, UpperLimit)
  else return (SC)
end
```

Fig. 1. Naive iterative case retrieval algorithm. Context is initialized with the attributes and constraints from the input case. Special counters are used to prevent repeating context restrictions and relaxations forever. Context transformations modify attributes of the least important category first. Only one category is transformed at a time.

4 Incremental Context Modifications

Incremental algorithms have previously been successfully applied to database systems for view maintenance [1, 5, 9, 10]. In general, an incremental view maintenance algorithm handles deletions, negations, updates, aggregation and recursion. Various approaches have been proposed to tackle these problems. One well known incremental view maintenance algorithm is a counting algorithm, which supports delete and re-derive operations [11]. It assumes universal materialization of predicates and stores counts of the number of derivations to be associated with each tuple. Another algorithm derives production rules to maintain selected SQL views, namely views without duplicates, aggregation and negation [5].

Because an incremental algorithm changes only relevant parts of the answer or view, instead of completely re-computing the answer, it is usually substantially more efficient than a naive one. Efficiency improvement is more visible when several consecutive changes to the query are required, i.e., during iterative browsing, or when a large information base is used. Small updates to the query generally produce only small changes to the query result. Thus, an incremental approach requires only local changes to the query [10].

The basic idea of incremental query processing is to store query results and reuse them when related queries are computed [1]. Assuming that the number of attributes per case is significantly smaller than the total number of cases in the case base suggests that incremental context modification outperforms the process of re-computation of the answer from scratch. Our system supports incremental context modification when partial results are kept at the attribute level or at the category level. The first approach requires extra storage space, but is more versatile and is thus suitable for case bases with frequent context modifications and with fewer attributes per case. The second approach is a compromise between performance gain from an incremental approach and modest storage requirement. This approach is useful for less frequent context changes and for case bases containing cases with a large number of attributes. Next, we explain the rationale behind the first approach.[4]

During iterative retrieval of cases, the system modifies constraints on attributes. A *naive retrieval algorithm* (see Figure 1) produces SC' by determining which cases in a case base satisfy constraints on all attributes defined in the context, i.e., for all $SC.a_i$. *Incremental retrieval algorithm* reuses $SC.a_i$ to produce the set of relevant cases SC' (see Figure 2). Relaxing attribute a_i changes a set of matching cases included in $SC.a_i$. However, all remaining partial answers remain unchanged. Thus, SC' can be constructed by adding cases that satisfy both the initial context and the relaxed constraint on attribute a_i to the SC. Restricting attribute a_i results creating SC' by removing cases that do not satisfy additional constraint from SC. Determining satisfiability only requires testing if the case in the set of retrieved cases must be removed either because it needs excluded value to match or because it cannot match an added attribute.

The idea of incremental context relaxation and restriction has evolved from the notion of differential queries [4]. First, parts of the context affected by the transformation are determined. Second, only those parts are recomputed. We express this process using context addition and difference.[5] Thus, the incremental context transformation algorithm can be formalized as follows: $\Omega' = \Omega + \delta^+ - \delta^-$, where δ^+ and δ^- denote a context that needs to be added or removed.

Without loss of generality, it is assumed that δ^+ and δ^- are contexts with single attribute-constraint pair. Because the context transformation process is iterative, more complex constraints can be created through multiple iterations. Next we formalize context transformations for our incremental retrieval algorithm (see Figure 2) and show how the final set of retrieved cases (SC') can be constructed using partial results from previous query evaluation (SC).

Reduction involves removing an attribute-value pair from a context. This can be done either *permanently* – an attribute a_k is removed from the context: $\Omega' = \Omega - a_k$, or *dynamically* – an *m-of-n* matching is used. Thus, for the reduced context, the resulting set of cases is generated as a union of the partial results of the set of cases that satisfy individual attribute constraints of the original context ($SC.a_i$), without considering the constraints on the removed attribute: $SC' = \cap SC.a_i / SC.a_k$.

Expansion involves adding an attribute-value pair to a context: $\Omega' = \Omega + \delta^+$, where $\delta^+ = \{a_i, \{V_i\}\}$. Thus, removing cases that do not satisfy the constrained context from the set of cases generates the set of retrieved cases: $SC' = SC \cap SC.a_i'$, where $\mathcal{C}_i \in SC.a_i'$ *iff* $sat(\mathcal{C}_i, \delta^+)$.

Generalization involves enlarging the set of allowable values for a given context: $\Omega' = \Omega + \delta^+$, where $\delta^+ = \{a_i, \{V_i\}\}$. Thus, the set of retrieved cases is

[4] The second approach works analogously.

[5] We define context addition and difference in terms of standard set-theoretic operations.

generated as an intersection of the set of cases that satisfy the original context and the set of cases that satisfy the context change δ^+: $SC' = SC \cap SC.a'_i$, where $\mathcal{C}_i \in SC.a'_i$ *iff* $sat(\mathcal{C}_i, \delta^+)$.

Specialization involves removing values from a constraint set for an attribute defined in a context: $\Omega' = \Omega - \delta^-$, where $\delta^- = \{a_i, \{V_i\}\}$. Thus, the set of retrieved cases (SC') is generated by removing cases that do not satisfy the restricted context from SC: $SC' = SC \cap SC.a'_i$, where $\mathcal{C}_i \in SC.a'_i$ *iff* $sat(\mathcal{C}_i, \delta^-)$.

```
retrieveIncremental (SC, Ω, Δ, LowerLimit, UpperLimit)
  for all cases in the Δ
      if a case C_j satisfies constraints on attribute a_i (see Equation 1) then
         add (C_j, SC_i)
  for all attributes in the Ω
      SC = ⋂ SC_i
  if consecutive relaxation-restriction or restriction-relaxation then
      return (SC)
  else if | SC | ≤ LowerLimit then
      relax (Ω, Category)
      set SC.a'_k to cases that satisfy the relaxed category
         if reduction then
            SC' = ∩ SC.a_i/SC.a_k
         else if generalization then
            SC' = SC ∩ SC.a'_k
  else if | SC |> UpperLimit then
      if the Ω was not previously relaxed then
         restrict (Ω, Category)
      set SC.a'_k to cases that satisfy the relaxed category
      SC' = SC ∩ SC.a'_k
  retrieveIncremental (SC', Ω, Δ, LowerLimit, UpperLimit)
end
```

Fig. 2. Incremental case retrieval algorithm. Initial SC is set to $\emptyset$. A user specifies LowerLimit and UpperLimit to set a desired amount of cases to be retrieved.

5 Performance Evaluation

Various approaches can be used to evaluate system performance. Available methods evaluate either *competence* of the system or its *scalability*. The former measures the capabilities of the system and can be assessed by precision/recall or accuracy/coverage measures. Scalability assesses system dependence on important factors, such as case base size, case and context complexity, and number of context transformation applied during iterative case retrieval.

We have tested both the competence and scalability of the proposed system: *(i)* learning control – solving the inverse kinematic task for a three-link spherical, angular robot [15], *(ii)* classification into a continuous class – predicting the rise time of a servomechanism in terms of two continuous gain settings and two discrete choices of mechanical linkages [15], *(iii)* prediction in medicine – suggesting a cost-effective treatment for in-vitro fertilization patients without compromising the probability for successful pregnancy [17], *(iv)* letter recognition [16] and *(v)* software reuse – similarity-based retrieval of software artifacts [13]. In [18] we show how a generic CBR system prototype $\mathcal{TA}\mathbf{3}$ can be custom-tailored to satisfy specific requirements in individual domains. Here we report on efficiency improvement due to using an incremental query modification.

There are several factors affecting performance of a CBR system: the size of a case base, size of a case and context, query complexity, number of iterations during iterative browsing, and the query relaxation/restriction strategy used. Here we present two efficiency evaluations of $\mathcal{TA}\mathbf{3}$: *(i)* Experimental efficiency evaluation using application domains described above and *(ii)* Scalability evaluation using the performance model of $\mathcal{TA}\mathbf{3}$.

To create the model we considered results of $\mathcal{TA}\mathbf{3}$'s experimental efficiency evaluation on various real-world domains and complexity properties of retrieval algorithms (both naive and incremental). The model simulates $\mathcal{TA}\mathbf{3}$'s scalability as a function of case base size, case representation complexity, query complexity, and context-modification strategy used. As a validation of this model, we compared experimental efficiency evaluation and scalability evaluation of $\mathcal{TA}\mathbf{3}$ on respective case base sizes, case representation complexity, and context complexity, when using query relaxation and restriction during iterative browsing. Figure 3 presents the cost of retrieval for naive and incremental reduction, generalization, expansion, and specialization algorithms as a function of case base size and size of the context using the performance model.

6 Discussion

An incremental algorithm for adapting a view in response to changes in the view definition is presented in [10]. Authors consider an SQL *Select-From-Where-GroupBy*, *Union views*, *Except views*, and present local adaptation strategies that use old view materialization to compute changes in the view. Their methods for adapting the *Where* part of an SQL are similar to our context transformations; the difference lies in handling reduction and expansion, which is not supported or necessary in database view maintenance. Blakeley et al. [4] propose an approach where an incremental view maintenance is supported for base relations updates but not for updates to views, as in our system.

FRANK is a case retrieval system, applied to the problem of back-injury diagnosis [21]. The user provides a description of a patient's symptoms and selects from a hierarchy of report types. A flexible control mechanism filters user's top-level considerations through the system's processing. A plan is then selected based on the report type. The task mechanism controls queries to the case base – if a query is not successful, then the system resubmits the query with altered initial values. This is similar to our notion of iterative query modification.

CBR has previously been successfully applied in various domains. However, in many of these studies, only small case bases were considered, often because the implemented system could not efficiently support retrieval for larger case bases. Various approaches, such as indexing [3, 20, 22] and selective forgetting [23, 27], have been proposed to improve the performance of CBR systems without decreasing competence. Parallel architectures have also been suggested as a way of increasing efficiency [24]. Our approach to performance improvement is based on incremental context modification, an approach which has previously proven successful in database applications.

The presented results support the claim that incremental context manipulation is generally more efficient then a naive approach. Performance improvement is increased when a large case base is used, when cases have many attributes or when several subsequent context modifications are required. In other words, this approach is most suitable for iterative browsing in complex domains. For simple retrievals involving a small case base and/or no consecutive context modifications, the naive approach may be preferable since time performance is not an issue and no extra storage space is required.

The proposed incremental algorithm is general in the sense that it is applicable for retrieval in any automated reasoning or decision support system that represents knowledge as attribute-value pairs and problems are solved by iteratively accessing and using previously derived information.

References

1. L. Bækgaard and L. Mark. Incremental computation of time-varying query expressions. *IEEE Trans. on Knowledge and Data Engineering*, 7(4):583–589, 1995.
2. F. Bancilhon. Naive evaluation of recursively defined relations. In M.Brodie and J.Mylopoulos, editors, *Knowledge Base Management Systems*, pages 165–178. 1986.
3. R. Barletta and W. Mark. Explanation-based indexing of cases. In *Proc. AAAI*, pages 541–546, 1988.
4. J. A. Blakeley, P.-A. Larson, and F. Wm. Tompa. Efficiently updating materialized views. In *ACM-SIGMOD*, pages 61–71, 1986.
5. S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *VLDB-91*, pages 577–589, Barcelona, Spain, 1991.
6. B. D'Ambrosio. Process, structure, and modularity in reasoning with uncertainty. In R. Shachter, T. Levitt, L. Kanal, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence 4*, pages 15–25. North-Holland, 1990.
7. J. Frawley and G. Piatetsky-Shapiro. *Knowledge Discovery in Databases.* AAAI Press, 1991.
8. T. Gaasterland. Cooperative answering through controlled query relaxation. *IEEE Expert, Intelligent Systems and their Applications*, 12(5):48–59, 1997.
9. T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *ACM SIGMOD*, pages 328–339, San Jose, CA, 1995.
10. A. Gupta, I. Mumick, and K. Ross. Adapting materialized views after redefinitions. In *ACM SIGMOD*, pages 211–222, San Jose, CA, 1995.

11. A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proc. of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 157–166, 1993.
12. E. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proc. AAAI*, pages 111–116, 1988.
13. I. Jurisica. A similarity-based retrieval tool for software repositories. In *The 3rd Workshop on AI and Software Eng.: Breaking the Mold. IJCAI-95*, Montreal, 1995.
14. I. Jurisica. Supporting flexibility. A case-based reasoning approach. In *The AAAI Fall Symposium. Flexible Computation in Intelligent Systems: Results, Issues, and Opportunities*, Cambridge, MA, 1996.
15. I. Jurisica and J. Glasgow. A case-based reasoning approach to learning control. In *5th Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, DKSME-96*, Phoenix, AZ, 1996.
16. I. Jurisica and J. Glasgow. Case-based classification using similarity-based retrieval. *International Journal of Artificial Intelligence Tools. Speciall Issue of IEEE ICTAI-96 Best Papers*, 6(4):511–536, 1997.
17. I. Jurisica, J. Mylopoulos, J. Glasgow, H. Shapiro, and R. Casper. Case-based reasoning in IVF: Prediction and knowledge mining. *AI in Medicine*, 12(1):1–24, 1998.
18. I. Jurisica and B. Nixon. Building quality into case-based reasoning systems. In *CAiSE*98, Lecutre Notes in Computer Science*. Springer-Verlag, 1998.
19. J. Ortega. On the informativeness of the DNA promoter sequences domain theory. *Journal of Artificial Intelligence Research*, 2:361–367, 1995. Research Note.
20. A. Ram. Indexing, elaboration and refinement: Incremental learning of explanatory cases. *Machine Learning*, 10(3):201–248, 1993.
21. E. L. Rissland, J. J. Daniels, Z. B. Rubinstein, and D. B. Skalak. Case-based diagnostic analysis in a blackboard architecture. In *Proc. of AAAI-93*, 1993.
22. C. M. Seifert. *et al.* Case-based learning - predictive features in indexing. *Machine Learning*, 16(1-2):37–56, 1994.
23. B. Smyth and M. T. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proc. of the 14th IJCAI*, pages 377–382, Montreal, Quebec, 1995.
24. E. Sumita, N. Nisiyama, and H. Iida. The relationship between architectures and example-retrieval times. In Proc. of AAAI, pages 478–483, Seattle, 1994.
25. P. R. Thagard, K. J. Holyoak, G. Nelson, and D. Gotchfeld. Analog retrieval by constraint satisfaction. *Artificial Intelligence*, 46:259–310, 1990.
26. M. B. Twidale, D. M. Nichols, and C. D. Paice. Browsing is collaborative process. *Information Processing & Management*, 33(6):761–783, 1997.
27. H. Watanabe, K.Okuda, and S.Fujiwara. A strategy for forgetting cases by restricting memory. *IEICE Trans. on Information and Systems*, E78D(10):1324–1326, 1995.
28. D. Wettschereck and T.Dietterich. An experimental comparison of the nearest neighbor and nearest hyperrectangle algorithms. *Machine Learning*, 19(1):5–27, 1995.

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

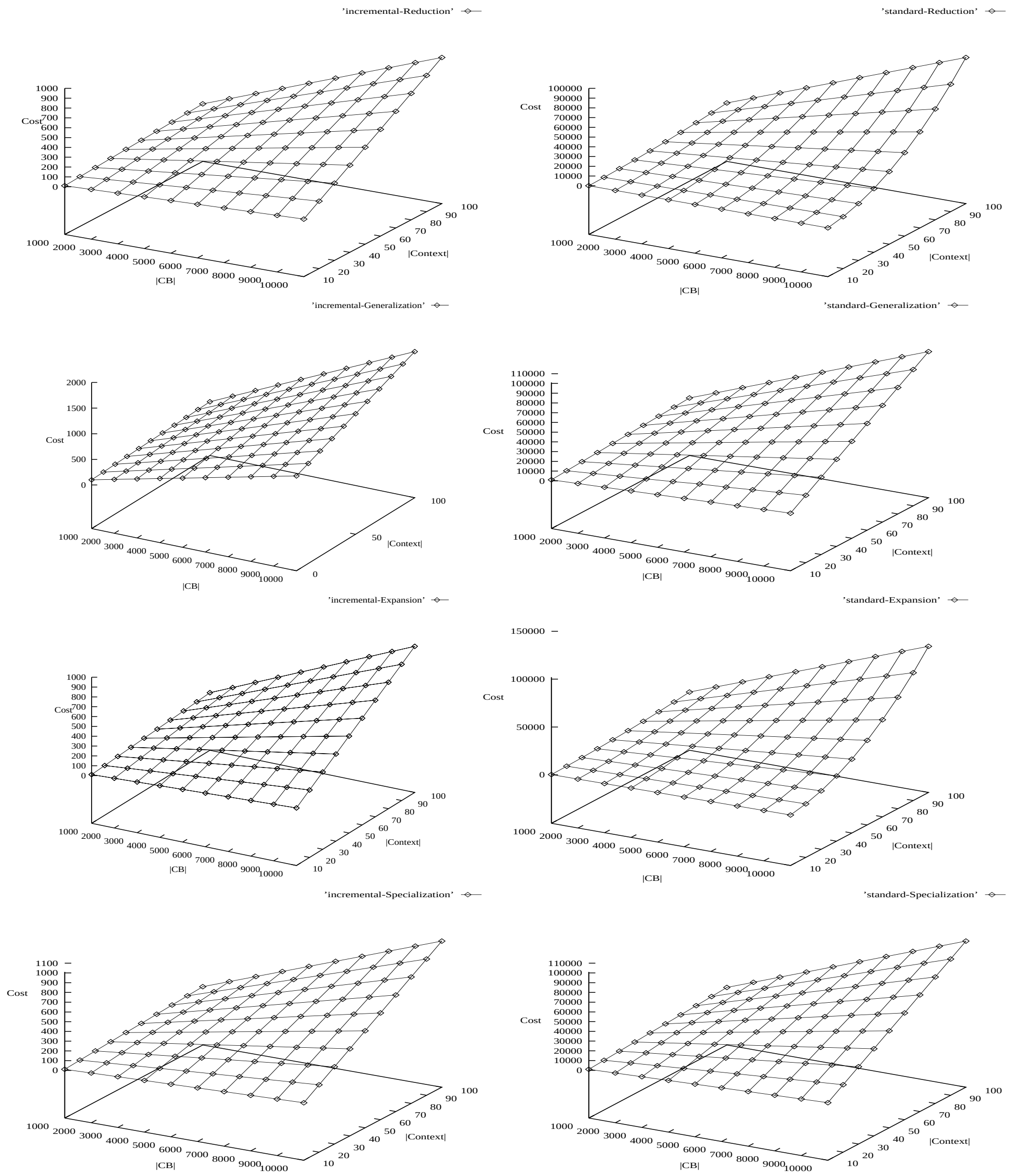

Fig. 3. Cost of retrieval for naive and incremental reduction, generalization, expansion, and specialization as a function of case base size ($|\ CB\ |$) and size of the context ($|\ Context\ |$).