

Increasing Levels of Assistance in Refinement of Knowledge-Based Retrieval Systems

Catherine Baudin*
Artificial Intelligence Research
Branch
NASA Ames Research Center
baudin@ptolemy.arc.nasa.gov

Smadar Kedar
Institute for the Learning
Science
Northwestern University
kedar@ils.nwu.edu

Barney Pell**
Artificial Intelligence Research
Branch
NASA Ames Research Center
pell@ptolemy.arc.nasa.gov

Abstract

This paper is concerned with the task of incrementally acquiring and refining the knowledge and algorithms of a knowledge-based system in order to improve its performance over time. In particular, we present the design of DE-KART, a tool whose goal is to provide increasing levels of assistance in acquiring and refining indexing and retrieval knowledge for a knowledge-based retrieval system. DE-KART starts with knowledge that has been entered manually, and increases its level of assistance in acquiring and refining that knowledge, both in terms of the increased level of *automation* in interacting with users, and in terms of the increased *generality* of the knowledge. DE-KART is at the intersection of machine learning and knowledge acquisition: it is a first step towards a system which moves along a continuum from interactive knowledge acquisition to increasingly automated machine learning as it acquires more knowledge and experience.

Keywords: Conceptual indexing and retrieval, increasing levels of assistance, inductive learning.

1. Introduction and Motivation

Knowledge-based conceptual retrieval systems assist users in retrieving information stored in a conceptual rather than literal form. These systems use models of a document's content to index the information, and use a reasoning component to search and retrieve information related to a user's query. Such systems need to interact with users at every stage of their life cycle. First, system builders interact with a bare-bones system in order to encode and debug the algorithms and heuristics which perform the retrieval task. Then, knowledge engineers encode and refine the domain knowledge. Finally, end-users interact with the system as it performs the retrieval task.

As in most realistic knowledge-based systems, the algorithms, heuristics and domain knowledge initially built into the system may be incomplete or incorrect. If the system operates incorrectly during system testing and operation, (e.g. the wrong information is retrieved, or existing

(*) Catherine Baudin is an employee of RECOM Technologies.

(**) Barney Pell is an employee of the Research Institute for Advanced Computer Science (RIACS).

information is not found), the end-user may provide feedback on the failure. The knowledge engineer and system designer then need to diagnose the failure and refine the knowledge, heuristics or algorithms to improve the system's behavior. Such a system refinement cycle never ceases--it is done incrementally over the life of a knowledge-based retrieval system.

In order to perform system refinement more precisely and efficiently, there is a need for tools that provide assistance throughout the life of the system. These tools should be able to monitor the performance of the task during operation; detect which portions of the knowledge are responsible for faulty behavior; and assist the knowledge engineers and system builders in refining the knowledge and algorithms to improve performance over time.

In this paper we present the design of such a knowledge acquisition and refinement tool, DE-KART (for DEDAL Knowledge Acquisition and Refinement Tool). It is designed to assist DEDAL [Baudin *et al.* 1992a], a knowledge-based conceptual retrieval system currently applied to documents of mechanical engineering design. DE-KART is a tool at the intersection of machine learning and knowledge acquisition. Instead of performing acquisition and refinement at one endpoint of the spectrum or the other--that is, totally automated or totally interactive--DE-KART is a first step towards a system which moves along a continuum from interactive to increasingly automated, as the system acquires more knowledge and experience. We call this movement *increasing levels of assistance*. What the user does manually at one level, the system provides assistance and suggestions for at the next level of assistance, and performs some of the task more automatically at yet a higher level. A level of assistance is measured by the level of *generality* of the knowledge acquired, and by the level of *automation* provided by the system to acquire and refine the knowledge. The level of generality of the knowledge can range from project specific knowledge to generic task knowledge, and the level of automation can range from totally manual to interactive to totally automated.

2. Overview of DEDAL and DE-KART

The architecture of our system consists of two main components: the *performance system* DEDAL, and the *knowledge acquisition and refinement tool*, DE-KART. Section 2.1 gives background on DEDAL, and Section 2.2 provides a sample of the interaction with DEDAL and DE-KART.

2.1 Background: Document indexing and retrieval in DEDAL

DEDAL stores mechanical engineering design documents in the form of text, graphics and videotaped information such as meeting summaries, pages of a designer's notebook, technical reports, CAD drawings and videotaped conversations between designers. It uses a *conceptual* indexing and query language to describe the content and the form of design information. The query language is the same as the indexing language, and uses concepts from a *model* of the artifact being designed, and from a *task vocabulary* representing the classes of *topics* usually covered by design documents [Baya *et al.*, 1992], [Baudin *et al.*, 1992a]. A conceptual index can be seen as a structured entity consisting of two parts: the *body* of the index which represents the content of a piece of information, and the *reference* part that points to a region in a document. For instance:

"The inner hub holds the steel friction disks and causes them to rotate" is part of a paragraph in page 20 of the record: report-333. It can be described by two indexing patterns:

<topic **function** subject **inner-hub** level-of-detail **configuration** medium **text** in-record **report-333** segment **20**>.

<topic **relation** subject **inner-hub** and **steel-friction-disks** level-of-detail **configuration** medium **text** in-record **report-333** segment **20**>

The queries have the same structure as the body of an index and use the same vocabulary. A question such as: "How does the inner hub interact with the friction disks?" can be formulated in DEDAL's language as the query:

<get-information-about topic **relation** subject **inner-hub** and **steel-friction-disks** with preferred medium **equation**>.

For mechanical engineering design, the domain model includes aspects of the artifact structure, the requirements, the main design decisions and alternatives considered. The model (as illustrated in Figure 1) includes hierarchical relations such as *isa* and *part-of* that are used by DEDAL to match a query with a given index.

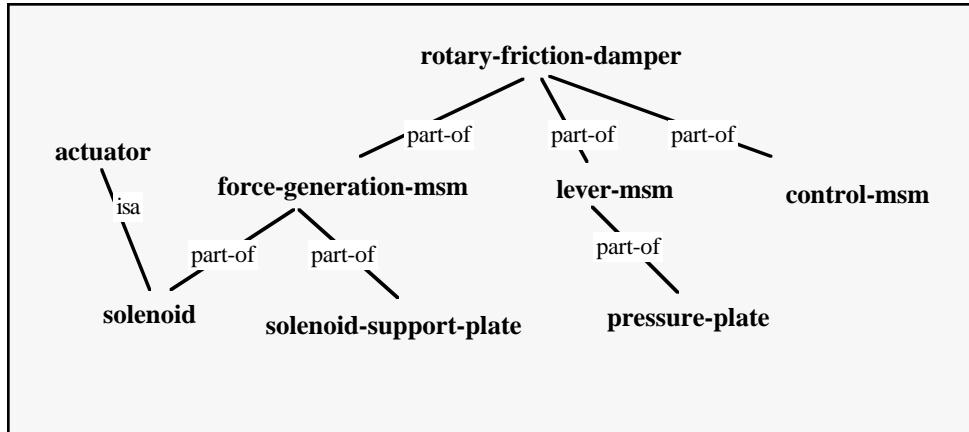


Figure 1: Part of the component hierarchy in the domain model.

The retrieval module takes a query from the user as input, matches it to the set of conceptual indices and returns an ordered list of references related to the question. The retrieval proceeds in two steps: (1) exact match: find the indices that exactly match the query and return the associated list of references. If the exact match fails: (2) heuristic match: activate the *proximity retrieval heuristics*. DEDAL currently uses 20 proximity retrieval heuristics to find related answers to a question. For instance, a heuristic may state that segments described by concepts like "*decision*" and "*alternative*" for a given part are likely to be located in nearby regions of the documentation. The retrieval procedure and the heuristics are described in [Baudin *et. al.*, 1992b].

Each retrieval step returns a set of references ordered according to a set of priority criteria. The user selects a reference and goes to the corresponding segment of information¹. If a user is not satisfied with the references retrieved he or she can request more information and force DEDAL to resume its search and retrieve another set of references.

2.2. A Sample Interaction with DEDAL and DE-KART

Initially both the domain knowledge and general retrieval heuristics are entered manually. Then, as the end-user queries the system to retrieve documents, he or she provides feedback to the system on the relevance of the documents retrieved. DE-KART monitors the different layers of knowledge involved in the retrieval, and assists the knowledge engineer and system builder in diagnosing and refining the knowledge, from the most specific domain-dependent layer, to the more general retrieval heuristics.

In this sample interaction, a mechanical engineering designer queries DEDAL to retrieve information about the "function of the solenoid", which is a subcomponent of the "force generation mechanism" in the "rotary friction damper" (a kind of shock absorber), as illustrated in Figure 1.

DEDAL attempts to find an index matching the query. It cannot find an index that exactly matches the query, so it uses a proximity heuristic: "to find information about the *function* of a subcomponent, look for information about the *operation* of a mechanism that includes this component". Using this heuristic, the retrieval component assumes that the "function of the solenoid" might be documented in sections describing the "operation of the force generation mechanism". DEDAL presents the user with the retrieved information about the "operation of the

force generation mechanism", and asks for feedback on whether or not the reference retrieved was relevant to the query.

If the user finds relevant information about the function of the solenoid, then the user's feedback is that the retrieved reference is *relevant*. Given a relevant reference, DE-KART automatically acquires a new index, associating the reference with the user's query (see section 3.1).

After some time, the user may query the system to retrieve "the function of the lever mechanism", another subcomponent of the damper. Again, no direct index exists. DEDAL uses the same proximity heuristic once again, and retrieves a part of the document associated with the "operation of the damper", although this time the user cannot find information about the function of the level mechanism in that part of the document and provides feedback that this retrieval is *irrelevant*. No index is acquired.

After the system has been in operation for a while, some indices will fail to match a query more often than they succeed. DE-KART can provide assistance to the knowledge engineer to re-prioritize, refine or even remove the offending indices. Similarly, after some time, some retrieval heuristics will prove to be more successful than others in retrieving relevant information in response to a query. Based on the monitoring information about the success and failure rates of the indices and the heuristics used to retrieve the information, DE-KART provides increasing levels of assistance to the knowledge engineer and the system builder to *detect* the parts of the knowledge that seem to fail, *diagnose* the causes of failure (incorrect index, unsuccessful heuristics, ill-formulated query), *refine* the offending knowledge, *validate* that the refinement in fact improves the retrieval performance over time, and continue to *incrementally refine* the knowledge until retrieval performance is satisfactory, according to some measure.

3. Acquisition and Refinement of Indexing Knowledge

As seen in Figure 2, knowledge in DEDAL exists at three levels of abstraction. At the top level are *task-dependent* retrieval heuristics. These heuristics are valid for the task of document retrieval in mechanical engineering. Below that are *domain-dependent* heuristics derived by applying task-dependent heuristics to particular user queries. These use terms defined within a specific domain model, as seen in Figure 1. These domain-dependent heuristics match a user query with *document-dependent* conceptual indices. This section shows how these different levels of knowledge are acquired, used and refined.

3.1. Index acquisition

The conceptual indices are initially entered manually. The user poses queries, and the system uses its heuristic retrieval strategies to find a match between a query and a "related" index. After a successful retrieval, DE-KART updates the document descriptions by turning this query into a new index. This *query-based* index acquisition and refinement phase is described in more detail by Baudin *et. al.* [1993]. It is performed in five steps.

1. Query formulation: The end-user formulates a query using the conceptual language. For instance, "what is the function of the solenoid?" is translated as: <topic **function** subject **solenoid**>.

2. Information retrieval: DEDAL searches for an index that exactly matches the query. In this case, it does not find an exact match and applies a proximity heuristic to "guess" where the required information may be located. For instance, H1 (see Figure 2) states that any information describing the *operation* of a mechanism might also describe the *function* of its parts. Given that the solenoid is a subpart of the force-generation mechanism (Figure 1), it finds a more general index: "operation of the force-generation-msm".² In this case DEDAL found two pages describing the operation of the force generation mechanism, pointed to by indices I1 and I2 (only I1 is shown in the figure).

3. **Relevance Feedback:** The user looks at the two references retrieved, finds that the reference pointed to by the index I1, page 23 in progress report damper-spring-90, describes the function of the solenoid, while the document associated with index I2, CAD document-4566, does not. The user rates the reference of I1 as *relevant* and the reference of I2 as *irrelevant*.

4. **Index generation:** Each time a reference is retrieved by the heuristic match and is rated relevant, DE-KART attaches the reference of the selected index to the query, turning the query into a new conceptual index . In this case DE-KART creates a new index I1-1 (see Figure 2). The system now knows that page 23 of progress report damper-spring-90 explicitly describes the function of the solenoid. Note that no new index was created for I2, as it was judged to be irrelevant. This process of index generation extends *relevance feedback* techniques [Salton & Buckley, 1990] to the acquisition of *conceptual* indices, and is described in more detail by Baudin *et. al.* [1993].

In addition, each time a heuristic is used to retrieve a reference, the system records the instance of the heuristic that was used. These *domain dependent rules* are created by instantiating a generic heuristic with the arguments in the user query. In our example R1 is the domain dependent rule used to retrieve the relevant reference pointed to by I1. It states that: "if a question is about the function of the solenoid, then look for indices about the operation of the force generation mechanism". In the figure, R2 and R3 are other instances of H1. This illustrates that domain dependent rules are stored each time a heuristic is instantiated, whether or not their use led to relevant retrievals for the given query.

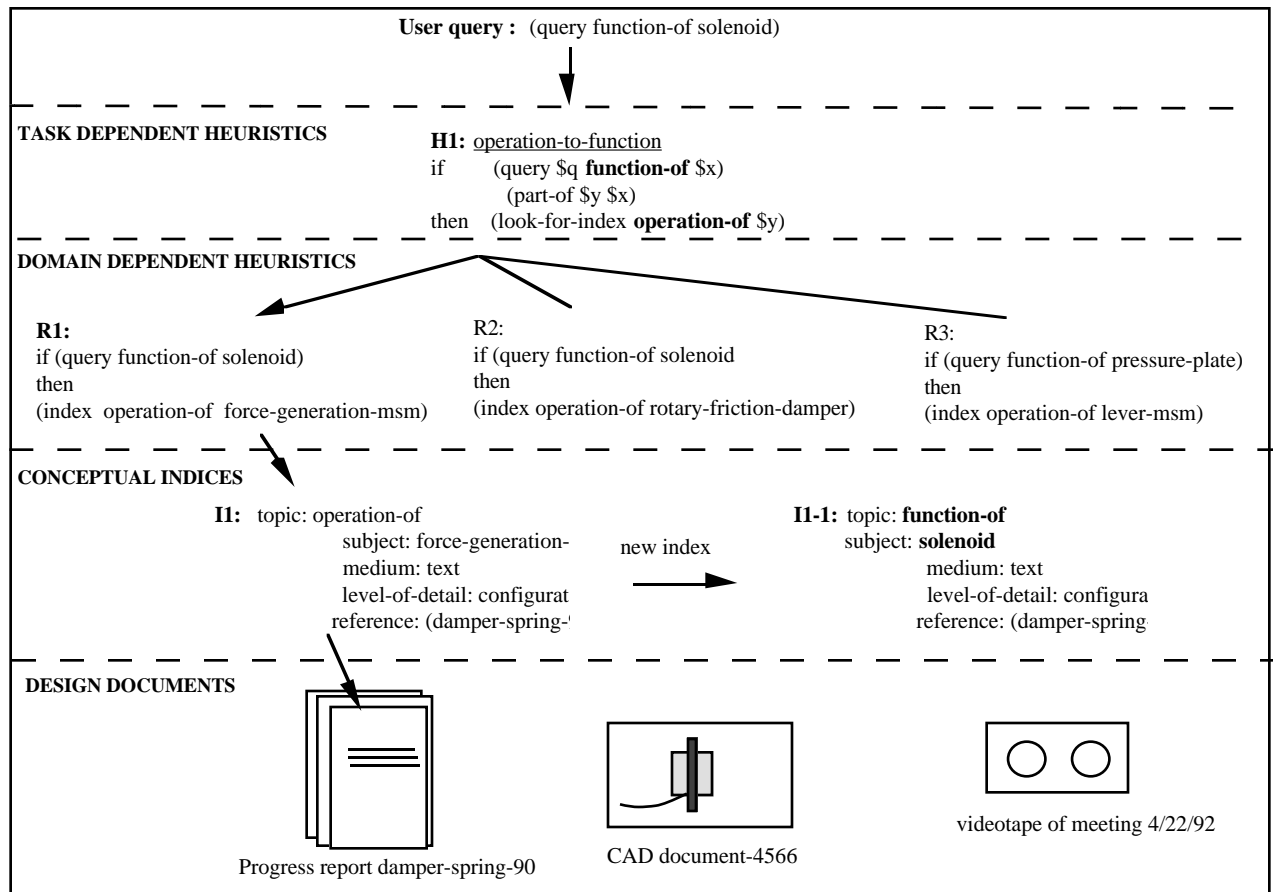


Figure 2: Three layers of indexing and retrieval knowledge in DEDAL.

5. Index Monitoring: After an index is created, the index and the heuristic used to create it are monitored for relevant and/or irrelevant retrievals. DE-KART increases a *success* or *failure* count each time a created index is matched exactly and judged to be relevant or irrelevant, respectively. The *success rate* of an index is computed as follows: $\text{success-rate} = \text{success-count} / (\text{success-count} + \text{failure-count})$. The procedure also keeps track of the relevance of each subject in an index [Baudin *et. al.*, 1993].

3.2. Index refinement

When the success rate of an index suggests that it should be repaired, the system provides two types of assistance to repair the faulty index depending on the type of user who is interacting with the system. The first level of repair is attempted during the system's operation with the end-user asking queries of the system. Based on the low rate of the index, the retrieval component assigns it a low priority so as to prevent the system from selecting it during subsequent retrievals. This level of assistance to the end-user is performed entirely automatically.

If the user is the knowledge engineer, the system engages him or her in a repair dialogue by first displaying the suspected indices along with the associated text, graphic or videotaped information. The system then enables the knowledge engineer to modify the parameters in the index, or to remove the index altogether. In this phase the system displays the rate of the index along with the rate of the subjects associated with the index.

4. Domain dependent retrieval rule refinement

If the index was retrieved by a proximity retrieval heuristic, the system increases the success or failure count of both the domain dependent retrieval rule (R1 in our example) and the generic heuristic used to generate the rule from the user query (H1 in our example).

When a particular instance of a retrieval heuristic seems to fail often, the system attempts two levels of repair. During the system's operation with the end-user the system lowers the priority of the domain-dependent rule, preventing it from being selected in subsequent retrievals. If the user is the knowledge engineer, the system displays the rule and enables the knowledge engineer to manually modify the rule by adding additional constraints to be matched with a given query. It then verifies that the previous indices that were created using this rule would still be generated.

For instance, in Figure 2, the rule R2 states that: "if the query is about the function of the solenoid, then retrieve information on the operation of the rotary friction damper". Given that R2 retrieved only irrelevant references, the knowledge engineer could attempt to specialize the rule by specifying, for example, that the query "function of solenoid" can only be matched with an index that points to information expressed in a *textual* medium. The rule then becomes:

R2':

```
if      (query function-of solenoid)
then    (index operation-of rotary-friction-damper medium text)
```

A refinement at the level of domain-dependent rules is more general than a refinement of a specific index. For instance, the domain-dependent rule R2 can be reused for other documents in the same domain, whereas fixing an index is only a document-dependent repair.

5. Refinement of generic retrieval heuristics

As a start, our goal is to decrease the number of irrelevant documents presented to a user (increasing precision) while maintaining the same number of relevant documents (preserving recall). With respect to this goal, we will consider that a heuristic *fails* if its failure count is high--that is, if it has retrieved many irrelevant references. At present, we do not consider recall failures, in which a heuristic fails to retrieve a relevant reference.

H1: function-to-operation if (query function-of \$X) and (part-of \$Y \$X) then (index operation-of \$Y)	
Positive instances of H1	Negative instances of H1
R1: if (query function-of solenoid) then (index operation-of force-generation-msm)	R2: if (query function-of solenoid) then (index operation-of rotary-friction-damper)
R3: if (query function-of pressure-plate then (index operation-of lever-msm)	R4: if (query function-of pressure-plate then (index operation-of rotary-friction-damper)

Figure 3: Positive and negative instances of H1.

A set of proximity retrieval heuristics is first entered manually by the system builder. During system operation when DE-KART detects that a heuristic fails for specific queries, the conditions in the *if* part of the heuristic, the *applicability conditions*, need to be refined so as to restrict the irrelevant retrievals but preserve the successful ones and thus improve the *precision* of the retrieval (the percentage of relevant references retrieved over the total number of references retrieved for the user)

Refinement of generic heuristics is performed by adapting an *inductive learning* technique. To review, inductive learning searches for a concept in some given generalization language which covers the *positive* instances of the concept, and none of its *negative* instances. In our case, the task of refining heuristics in DE-KART is to help the knowledge engineer find the concept which describes the applicable retrievals for a proximity heuristic. The positive and negative instances of the concept to be acquired are the instances of successful and failed retrievals of the heuristic. The induction task amounts to finding a concept to specialize the current applicability conditions of a heuristic, which is over-general, such that none of the failed retrievals would be generated, while all the successful ones would be.

Our approach differs from the classical automated approach to induction in that the system does not automatically perform the entire induction task, but provides the user with increasing levels of assistance in performing this induction interactively. We present here two levels of assistance. One, *interactive inductive refinement* assists the knowledge engineer in discovering a concept which covers the positive and none of the negative examples by focusing his attention on the positive and negative instances of a faulty heuristic (Figure 3) and by validating the refinement proposed by the user on previous and future retrievals. The second level of assistance, *autonomous inductive refinement*, performs induction using an off-the-shelf ID3 machine learning algorithm to automatically refine the applicability conditions for the faulty heuristics.

5.1. Interactive inductive refinement of heuristics

We first illustrate the interactive inductive refinement task for proximity heuristics.

1. Failure Detection and Diagnosis: Both domain dependent and generic task retrieval heuristics are monitored during the system's operation (see section 3.3). When the failure rate of a heuristic exceeds some threshold, the heuristic refinement dialog can be initiated. The typical user who interacts with the system during this dialog is the knowledge engineer.

The system presents the knowledge engineer with the suspect heuristic H1 (see Figure 3). It displays negative and positive instances of the heuristic H1. In our example, the instances of heuristic H1, R2 and R4, retrieved irrelevant references in response to the queries: "function of solenoid" and "function of pressure plate".

2. Refinement: The knowledge engineer interacts with the system in an attempt to find a concept which would cover all of the positive instances and none of the negative ones.

a. concept selection: If the system has a base of pre-existing concepts, the knowledge engineer can select an appropriate concept, and apply it to the variables of the heuristic.

b. language extension: If no such concept exists, the knowledge engineer can extend the language and enter the definition of a new concept. In our example (see Figure 3), the knowledge engineer defines a concept that would prevent the generation of R2 and R4 while still enabling the generation of the successful retrievals R1 and R3. Looking at the instances of H1 he may notice that in both negative instances R2 and R4 the variable \$Y was instantiated with the value "rotary-friction-damper" which is the top-level component of the whole device. The intuition for why retrievals failed in these instances is that documents that describe the operation of the top level component are so general that the end-user could not find the description of a function of a subcomponent he or she was looking for, and deemed this retrieval "irrelevant". The knowledge engineer defines the new concept: (top-level-component \$X) which is true if X is the top component of the part-of hierarchy (true in our example of the rotary friction damper).

c. heuristic repair: The system now assists the user in editing the applicability conditions of H1. In this example, the user restricts the variable \$Y in H1, specifying that a query: <function of \$X> can match an index <operation of \$Y> only when \$Y is *not* the top level component in the part-of hierarchy.

The modified heuristic H1' is then:

H1': function-to-operation

```
if      (query function-of $X)
        (subpart $Y $X)
        (not (top-level-component $Y))
then    (index operation-of $Y)
```

The new concept "top-level-component" is added to the generalization vocabulary of the system and becomes usable to refine other heuristics.

3. Validation: The system checks that this new applicability condition prevents the generation of the negative instances of H1 while still enabling the generation of the positive instances. In our example (see Figure 2), any domain-dependent rule which instantiates \$Y with the top level component "rotary-friction-damper" would be prevented from being generated. The system then validates that the positive instances R1 and R3 would still be generated given this new restrictive applicability condition. The new heuristic H1' is added to the retrieval knowledge. H1 however remains in the system to enable the retrieval component to fall back on it if the existing set of refined heuristics fail to retrieve relevant answers. DE-KART classifies the new heuristic as "tentative" until the system has enough experience to establish that no positive instances of H1 are prevented from being generated.

4. Incremental Refinement: After some time, the user submits a new query about "function of the control mechanism". No index exactly matches the query and no heuristic (including H1') succeeded in retrieving a reference. The system then falls back on H1 and generates the instantiated rule R5: "if the question is about the function of the control mechanism, then look for indices about the operation of the rotary friction damper". This rule retrieves a document based on the index: <operation of rotary-friction-damper>. In this case the retrieved document is rated as relevant by the user. The failure of H1' to generate the positive instance R5 suggests that the concept (not (top-level-component \$X)) is too restrictive, and H1 must be refined differently. DE-KART provides the knowledge engineer with a heuristic refinement dialog and now focuses the attention of the user on R2, R4 and R5 (see Figure 4).

Positive instances of H1	Negative instances of H1
<p>R1: if (query function-of solenoid) then (index operation-of force-generation-msm)</p> <p>R3: if (query function-of pressure-plate then (index operation-of lever-msm)</p> <p>R5: if (query function-of control-msm) then (index operation-of rotary-friction-damper)</p>	<p>R2: if (query function-of solenoid) then (index operation-of rotary-friction-damper)</p> <p>R4: if (query function-of pressure-plate then (index operation-of rotary-friction-damper)</p>

Figure 4: Positive and negative instances of domain rules.

At this point the knowledge engineer provides the concept, (parent \$X \$Y), which is true if Y is the immediate ancestor of X in the part-of hierarchy. This concept will cover all of the positive instances of H1 since in all of them, the two components are close enough in the hierarchy (see Figure 1). The intuition is that the end-user was able to find information about the control mechanism where the operation of the "rotary-friction-damper" is described, whereas the "solenoid" component is too detailed a component to be documented at such a high level. The system assists the user in editing the applicability conditions of H1' and generates a new heuristic H1'':

H1'': function-to-operation

```

if      (query function-of $X)
        (subpart $Y $X)
        (parent $X $Y)
then    (index operation-of $Y)

```

This time the new applicability condition of the heuristic prevents the generation of the negative instances of H1' (R2, and R4) while enabling the positive instances (R1, R3 and R5) to be generated.

At this point DE-KART has refined the applicability conditions of heuristic H1, increasing the precision of future retrievals. The system also acquired two new concepts for the generalization hierarchy: (top-level-component \$X) and (parent \$X \$Y).

Once enough concepts are entered in the generalization hierarchy and enough examples of successful and unsuccessful matches between queries and indices are recorded, DE-KART can provide a greater level of assistance to the system builder by autonomously selecting and adding restrictive applicability conditions for a faulty heuristic.

5.2. Autonomous inductive refinement of heuristics.

At this level of assistance the system automatically suggests possible heuristic refinements to the system builder. This is accomplished using an inductive learning algorithm. This section provides a brief review of induction in general, and then describes our application of induction to the refinement of retrieval heuristics.

Induction works by extracting patterns, or regularities, in the training data which can then be used to predict properties of new examples. In the most common use of induction, called supervised learning, we are interested in predicting one property of an example, its membership in a class, based on features of the example. For example, it is possible to predict whether an animal is a mammal or reptile (the class) when told whether the animal has fur and is warm-blooded. The task of a supervised learning tool in this example would be to extract such decision rules from a database of animal descriptions in which each animal is pre-classified as mammal or reptile. Out of the set of all features describing different animals (such as furriness, lifespan, blood temperature, and so on), the algorithm uses the examples and decides for itself how to use those features to distinguish mammals from reptiles.

Inductive learning has been an active area of research, and many off-the-shelf algorithms and tools are commercially available. These algorithms and tools vary along a number of important dimensions: how easy they are to implement, how easy they are to use, in what form the examples have to be described, how many examples are required to form good decision rules, how general those resulting rules are, how easily they can be used to classify new examples, and how well the algorithms perform when some examples are misclassified (partially incorrect data is called noisy data).

For our purposes, we chose an implementation of the ID3 Inductive Learning Algorithm [Quinlan, 1986]. We chose this algorithm for three reasons. For one, it classifies the kind of examples we have in our system (examples described by a set of discrete features and their values). Secondly, the algorithm is easy to use and readily available. Thirdly, the output of the algorithm is a set of compact rules which can be understood by a human (as opposed to, for instance, the opaque set of rules produced by neural networks that perform the same induction task).

Briefly, ID3 extracts regularities in features of examples, and creates decision rules in the form a tree. Each node in the decision tree represents a test on one feature of an input example (for example, whether the animal has fur or not), and each possible value for the tested feature (for example, "has fur" or "no fur") leads to a distinct subtree. When the tree is used to classify an example, the tests are applied in turn to the example, starting with the test at the root of the tree, and following the branch which corresponds to the value the example has on the tested feature (for example, if the animal to be classified does have fur, it would proceed down the "has fur" branch of the tree). When a leaf of the tree is reached (there are no more tests), the example is classified according to the class stored on that leaf (for example, if an animal example proceeded down the "has fur" and "gives live birth" branches, it would then be classified as a "mammal").

The details of the ID3 decision-tree construction algorithm are as follows: ID3 takes as input a set of pre-classified examples. Each example consists of a fixed set of feature-value pairs (such as "blood temperature=warm, weight=120"...), and a final classification (e.g. mammal). If the examples are all of the same class (e.g. all mammals), the algorithm produces a leaf node labeled with that common class. Otherwise, the algorithm chooses the most discriminating feature which best separates the examples into different classes. For example, the feature "blood temperature" might take on one value ("warm") for most of the mammals in the example set, and another value ("cold") for most of the reptiles, so this would be a good discriminating feature. By contrast, the feature "body weight" might not be useful for distinguishing mammals from reptiles, as both classes of animal come in a variety of sizes. The algorithm then groups together all examples which have the same value for that feature. For instance, all examples with "blood temperature=warm" would form one group, and those with "blood temperature=cold" would form another. The ID3 algorithm is then called recursively to produce a subtree for each of these subgroups. A node is then created which corresponds to the tested feature ("blood temperature"), and for each possible value of this feature a link is created to the corresponding subtree. For instance, "blood temperature = cold" would be linked to the subtree constructed by calling the algorithm on the cold-blooded subgroup, as discussed above. This tree, consisting of the created node and the links to all the subtrees, is the output of the algorithm. It can then be used to efficiently classify new input examples.

For our purposes, we apply ID3 to automatically suggest possible heuristic refinements to the system builder. Given a set of relevant and irrelevant instances of the retrieval heuristics, the algorithm produces a set of possible restrictions to the applicability conditions of those heuristics. In the language used to describe ID3 above, our features include "medium" "subject", and "retrieval-heuristic", and values for those features include "text", "solenoid", and "function-to-operation", respectively. The classes are "relevant retrieval" and "irrelevant retrieval". An example consists of a set of "feature=value" pairs (for example, "subject=solenoid, medium=video, topic=function, page=22, retrieval-heuristic=function-to-operation") and a pre-classified example

is one in which an example is labeled as either an instance of a relevant or an irrelevant retrieval. A rule extracted from a decision tree in our case might read: "if a new example is one in which the query's subject is solenoid, the medium of the index is video, and the heuristic that generated the retrieval is function-to-operation, then predict that example to be an irrelevant retrieval."

The autonomous inductive refinement consists of the following steps:

1. Failure Detection and Diagnosis: This technique can take place after a preferably large and varied set of positive and negative examples of proximity heuristic retrievals have been collected. The system monitors the failure rates of the heuristics, and identifies those heuristics whose failure rate has passed some threshold.

2. Example Pre-processing: The input to the induction algorithm is a set of positive and negative examples of a heuristic, each consisting of a set of features describing the query, the matching index, and relations among the query and index. The examples are generated from successful and failed retrievals during system operation. To make possible the generation of domain independent refinements, DE-KART adds some features to each example. These features represent different types of relations between the subjects in the query and the subjects in the matching index. For instance the feature "subjects-in-query-equal-subjects-in-index =yes" means that the subjects in the query are the same as the subjects in the matching index. In the same way, "subjects-query-is-part-of-subjects-index = yes" means that the subjects in the query are part-of some subjects in the matching index.

Figure 5 shows a positive example for the heuristic H2 "operation-to-construction" which retrieves information about construction of a mechanism, in response to a query about the operation of that mechanism.

H2: Operation-to-construction

if (query operation-of \$X)
then (index construction-of \$X)

For this example, the query was about the "operation of the solenoid" <query-topic = operation-of, query-subject = solenoid> and the matching index was construction of solenoid <index-topic = construction-of, index-subject = solenoid>. This is a positive example, meaning that a piece of information describing the "construction of the solenoid", in a textual medium, and a conceptual level of detail, was relevant to the query: "operation of solenoid". The second part of this example (in bold) shows a set of relations between the subject(s) of the query and the subject(s) of the index, that are automatically added. For instance, in this example the subject of the query was the same as the subject of the index (subjects-query-equal-subjects-index = yes). In our initial tests, the parts of the example that are in italic are domain dependent terms and were not fed to the induction algorithm.

```

example 165 positive
retrieval-heuristic: operation-to-construction
query-topic = function-of
query- subjects= solenoid
index-topic = operation-of
index-subjects = force-generation-mechanism
index-medium = text
index-level-of-detail = conceptual

subjects-query-equal-subjects-index = yes
subjects-query-is-attributes = no
subjects-query-is-requirements = no
subjects-query-is-part-of-subjects-index = no
subjects-index-is-part-of-subjects-query = no
subjects-query-is-kind-of-subjects-index = no
subjects-index-is-kind-of-subjects-query = no
subjects-query-depends-on-subjects-index = no
subjects-query-influence-subjects-index = no
...

```

Figure 5: A positive example of query/index match.

3. Refinement: The system runs the ID3 induction algorithm on the faulty heuristics to characterize when they fail and when they succeed. The result is an *applicability tree*, a decision tree which describes the applicability conditions of the heuristic covering the positive examples seen so far, and none of the negative ones. Figure 6 shows an applicability tree for the proximity heuristic “operation-to-construction”.

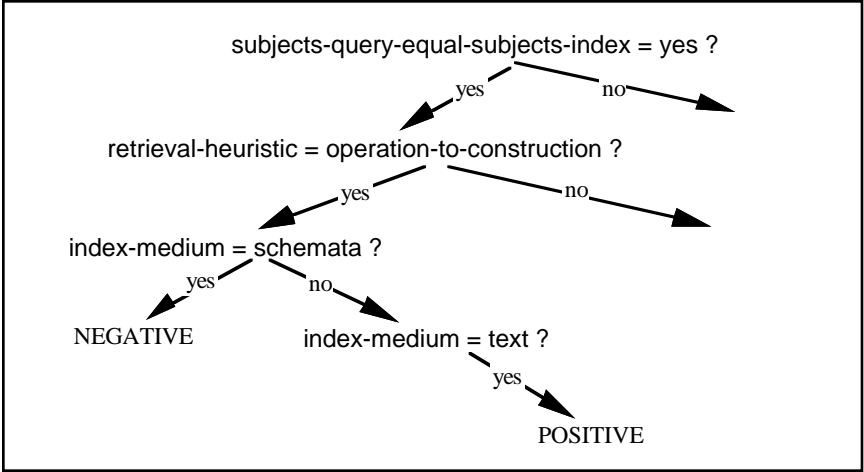


Figure 6: A portion of an applicability tree generated by the ID3 induction algorithm.

The tree refines the applicability conditions for the heuristic in the following way: the initial applicability conditions state that any query about operation of a mechanism X can be matched with an index about construction of X (intuitively, the structure of an assembly reveals insights about its operation). The applicability tree refines this condition by specifying all of the conditions already in the heuristic, plus that any information about construction of X may be relevant to a query about operation of X *only if the medium of the document is text*, not if it is, say, a schemata.

H2': Operation-to-construction

if (query operation-of \$X)
(index-medium text)
then (index construction-of \$X)

4. Validation and transfer: The system validates that the applicability conditions of the heuristic, plus the applicability tree for it, do indeed prevent the generation of the negative instances of the heuristic while enabling the generation of the positive ones. It does so by classifying positive and negative examples from sessions with the user other than the session on which the training was done. In doing this, the system can compare its prediction of which examples should be negative and positive, with the user's classification of positive and negative examples.

An alternative to this automatic validation procedure is to rely on the knowledge of the knowledge engineer familiar with the domain to validate a refinement. The advantage is that this interactive validation can filter those refinements that are clearly incorrect, while enabling the system to experiment and validate interesting refinements with end-users.

For instance in the case of the heuristic "operation-to-construction", the refinement proposed by the system corresponds to the intuition that a piece of text describing how a mechanism is assembled is more likely to provide information about how the mechanism works (operates) than a photo or a schemata representing the mechanism. At this stage a system builder is likely to select this refinement and try it even though it could not yet be validated on a large sample of queries.

5. Incremental Refinement: If the system discovers false negatives and false positives in its classification, it can now rerun the induction algorithm on the combined previous and current sessions. When the false positives and false negatives fall below some threshold, then the system can insert additional applicability conditions to provide a refined heuristic whose precision has been improved.

6. Discussion

We have presented the design of DE-KART, a knowledge acquisition and refinement tool that provides increasing levels of assistance in acquiring indexing and retrieval knowledge for a knowledge-based retrieval system. In the scenario presented, the knowledge of DEDAL is refined by: (1) increasing the *level of generality* of the knowledge repaired (from document-dependent indices, to domain-dependent rules, to task-dependent retrieval heuristics) as the system gains experience with different end-users and different queries, and (2) increasing the *level of automation* of the assistance provided to the knowledge engineer and to the system builder: from manual to semi-automated query-based indexing when repairing the conceptual indices; and from manual to automated with the knowledge engineer and the system builder when repairing the retrieval heuristics.

The system starts with a core of conceptual indices entered manually and refines this core of knowledge using the queries of the end-user and feedback about the relevance of the document retrieved in response to the query. This increases the level of automation of the index acquisition from manual to semi-automated when interacting with the end-user of the retrieval system. The level of generality remains at the document or project-specific level.

In the same way, the generic retrieval heuristics are first entered manually. This knowledge increases the recall of the system [Baudin *et al.*, 1993], but is over-general and occasionally leads to the retrieval of irrelevant references in response to a query. These failures provide the opportunities for refinement and repair. While the system is in operation with end-users, DE-KART tries to recover from failing retrieval heuristics by lowering their priority for retrieval. When the system starts interacting with the knowledge engineer it engages in a dialogue to refine the retrieval heuristics. During this refinement phase, the system increases the level of assistance provided to the knowledge engineer. If the system has no knowledge of concepts with which to repair a given

heuristic, and few examples of its success and failure, it interacts with the knowledge engineer to refine the representation of the examples and to validate the proposed refinement on previous and future retrievals. When more knowledge of the relevant concepts used to fix heuristics is entered and enough examples of relevant and irrelevant retrievals are generated, the system can suggest a set of repairs by running an induction algorithm on these examples. Here again, the refinement proposed by the system can be first validated by the knowledge engineer before enabling end-users to experience the effects of the refinement.

In our current scenario, the role of the end-user is to provide feedback on the performance of the system during its operation, thus enabling examples of relevant and irrelevant retrievals to be generated. Because this relevance feedback is performed by the end-users, the resulting data are noisy and must be further monitored by the system. For instance, a user might have misformulated a query and thus he or she will find a retrieval irrelevant although the proximity heuristic used was correct. Some of these false negatives can be filtered automatically by disregarding examples where the same question/index pair was judged relevant one time and irrelevant another time. However, it is still advisable to have the human expert review domain rules that seem to fail often. Moreover, the automated refinement based on this feedback should be designed to take such noise into account [Julian & Fenves, 1994]

Of course an advantage of being able to gather examples during the system's operation is that: (a) it enables the system to experience a wide variety of situations that might be difficult to anticipate beforehand, (b) it provides some simple information retrieval enhancement in a non-obtrusive way by reprioritizing the rules and the indices during operation and (c) it can monitor its knowledge in the background and decide, for instance, when to increase its level of automation [Maes & Kozierok, 1993]. This end-user-centered knowledge acquisition aspect is enabled by two factors. The first factor is that in an information retrieval task the end-user is able to *judge* the relevance of an answer provided by the system even though he might not be expert in the domain. This is not the case in most diagnostic tasks, in which only the domain expert is able to judge whether a diagnosis is relevant. The second factor is that, in an information retrieval task, there are generally no dire consequences in retrieving false positives. This means that a system is still usable in early stages of development, before extensive refinement with an expert has taken place.

One phase of the heuristic repair in which the human plays a vital role is the *concept definition* phase where the knowledge engineer decides on new concepts to be added to the representation in order to discriminate positive and negative examples. From our experience with DEDAL, it seems that this concept definition phase is the activity that has the greatest impact on the performance of the system in the early stages of its development. At present, concept definition happens during the language extension part of the interactive refinement phase (see section 5.1). This functionality still needs to be integrated into the automated refinement phase (section 5.2), which could then be viewed as a form of *interactive induction* [Tecuci & Hieb, 1993; Buntine & Stirling, 1990; Shapiro, 1983; Muggleton, 1987].

7. Related Work

Knowledge acquisition can be broadly classified into three stages: *elicitation*, *refinement* and *restructuring* [Bareiss *et al.*, 1989]. DE-KART focuses on the refinement stage. Recently there has been work on integrating machine learning and knowledge acquisition techniques in the area of knowledge base refinement (see this issue, Tecuci *et al.*, 1993, and the survey by Wilkins, 1991). Some of this work tries to automate the refinement process by combining knowledge acquisition with various machine learning techniques using: induction [Feldman *et al.*, 1994], explanation-based learning [Reinartz & Schmalhofer, 1994; Pazzani *et al.*, 1991], a combination of inductive and analytical techniques [Tecuci & Duff, 1994], or analogical reasoning [Gil & Paris, 1994].

Our work differs from most other efforts to integrate knowledge acquisition and machine learning in the area of knowledge base refinement in that it performs refinement at various levels of automation

and generality. In this respect it is similar to recent work on procedure acquisition [Mathe & Kedar, 1992] which also provides increasing levels of assistance. DISCIPLINE [Tecuci & Kodratoff, 1990] and NeoDISCIPLINE [Tecuci, 1992] can be viewed as providing increasing levels of assistance, although the emphasis is on machine learning methods with rudimentary knowledge acquisition, whereas our emphasis is on knowledge acquisition, with some rudimentary machine learning. Given a taxonomy of interactive knowledge acquisition methods such as that proposed by Musen [1989], DE-KART cannot be placed at a fixed level of the taxonomy, but can be viewed instead as moving through various levels of the taxonomy as it increases its level of assistance, e.g. from a "task level" tool for domain experts to a "method level" tool geared toward knowledge engineers.

One interesting aspect of our work is that DE-KART acquires knowledge both from the novice end-user during the system's operation, and from the domain expert or knowledge engineer in a separate refinement phase. The presence of these distinct contexts for knowledge acquisition places DE-KART midway between systems which learn primarily from the end-user, and can provide immediate but shallow knowledge refinement during performance (such as Maes & Kozierok, 1993; Clause & Utgoff, 1992) and knowledge acquisition tools that can perform deep knowledge restructuring but do so primarily through interaction with the domain expert or the knowledge-engineer.

8. Future Work and Conclusion

We need to conduct more experiments to evaluate our knowledge acquisition and refinement approach for indexing and retrieval knowledge. So far, we have acquired and collected examples from the queries asked by two mechanical designers while they were accessing information in the context of the "rotary friction damper" problem in the mechanical engineering design domain [Baudin & Kedar, 1993]. During the first year of the system's operation we acquired new concepts to discriminate between positive and negative examples. We then applied the induction algorithm on 300 examples of successful and unsuccessful matches. The refinements proposed by the induction algorithm were reviewed by the knowledge engineer, who decided which ones should be incorporated into the system. Among the proposed refinements, the ones that involved additional conditions on the medium and level of detail of the retrieved information usually appealed to the human's intuition even though the system builder and the knowledge engineer could not always give an opinion on the quality of the proposed refinement. This manual selection phase is necessary in the first stages of the life of the system where its experience with different users and different domains is still limited. However, we need to conduct more experiments to confirm the human intuition and measure the impact of the acquisition on the *precision* and *recall* of future retrievals.

To evaluate the generality of the acquired knowledge we are experimenting not only with different classes of users but with different domains. In particular, we are now running the system to retrieve text, graphics and video records for the design of an innovative bioreactor, a device that will enable NASA life scientists to study microbial growth.

While our acquisition and refinement methods are not dependent on the specifics of the mechanical engineering domain, they currently address only some of the issues in supporting acquisition and refinement in knowledge-based retrieval systems. Currently, DE-KART can refine a heuristic by constraining the variable in the applicability conditions of the heuristic. However, some refinements involve the addition of new variables. We also need to provide assistance when heuristics are too specific, in order to enhance the system's recall. This will involve fixing the domain model representing the device being designed as well as the retrieval heuristics. Finally, this methodology assumes that acquisition and refinement are ongoing tasks performed during the life of the system, and that time is available to interact with the system to correct the knowledge base.

Currently, DE-KART is not integrated with the induction algorithm, and the refinements proposed by the system must be manually integrated in DEDAL to enable the system to monitor the repair.

We also need to provide more guidance in the dialogue with the user to select the different levels of assistance.

In conclusion, this paper has presented a tool which acquires knowledge through interaction both with possibly naive end-users, during performance of a task, and with domain experts and knowledge engineers, in an incremental but off-line refinement phase. These dual contexts enable the system to continually improve its performance throughout its lifetime, to acquire knowledge at several levels of generality, and to increase its level of automation as its base of knowledge and experience accumulate. So far, this functionality has been implemented within the context of a knowledge-based information retrieval task, in which the end-user, while not an expert, is still able to provide feedback on the performance of the system. An important area for future work is to identify the class of problems for which this dual-context approach to continuous knowledge acquisition is applicable.

Acknowledgments

Thanks to Vinod Baya, Ade Mabogunje and Jody Gevins Underwood who helped us evaluate the results of our experiments and for fruitful discussions about question-based indexing. Thanks to Larry Leifer and to the other members of the GCDK group for their feedback and support on this project, to Wray Buntine, Tom Hinrichs, and members of NASA Ames and the Institute for the Learning Sciences for useful interactions and comments on earlier drafts.

References

- Bareiss, R., Porter, B.W., Murray, K.S., Supporting Start-to-Finish Development of Knowledge Bases. *Machine Learning*, 4(3-4):259-283, 1989.
- Baudin, C., Gevins, J, Baya, V, Mabogunje, A., DEDAL: Using Domain Concepts to Index Engineering Design Information. *Proceedings of the Meeting of the Cognitive Science Society*, Bloomington, Indiana, 1992a.
- Baudin, C., Gevins, J., Baya, V., "Using Device Models to Facilitate the Retrieval of Multimedia Design Information", in proceedings of IJCAI 93, Chambéry, France, Vol. 2, 1992b.
- Baudin, C., Kedar, S., Gevins, J., Baya, V., Question-Based Acquisition of Conceptual Indices for Multimedia Design Documentation. *Proceedings of the Eleventh National Conference on Artificial Intelligence*; Washington, D.C., 1993.
- Baya, V, Gevins, J, Baudin, C, Mabogunje, A, Leifer, L., An Experimental Study of Design Information Reuse. *Proceedings of the 4th International Conference on Design Theory and Methodology*, ASME, Scottsdale, Arizona, pages 141-147, 1992d.
- Buntine, W, & Stirling, D. Interactive Induction. In Hayes, J., Michie, D., and Tyugo, E., editors, *Machine Intelligence 12, Analysis and Synthesis of Knowledge*, pages 121-137. Oxford University Press, New York, 1990.
- Clouse, J. & Utgoff, P, A, Teaching Method for Reinforcement Learning. In D. Sleeman and P. Edwards (eds), *Proceedings of the 9th International Workshop on Machine Learning*, pp 92-101, Morgan Kaufman, 1992.
- Feldman, R., Koppel, M, & Segre, A., Biased Revision of Approximate Domain Theories. This issue.
- Julian, B. & Fenves, S., Guiding Induction with Expert Knowledge. This issue.
- Gil, Y. & Paris, C. Toward Method-Independent Knowledge Acquisition. This issue.

- Maes, P. & Kozierok, R., "Learning Interface Agents" *Proceedings of the Eleventh National Conference on Artificial Intelligence*; Washington, D.C., 1993.
- Mathe, N. & Kedar, S., Increasingly Automated Procedure Acquisition in Dynamic Systems. *Proceedings of the 7th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Vol. 2, Banff, Canada, October, 1992.
- Muggleton, S., Structuring Knowledge by Asking Questions. In I. Bratko and N. Lavrac (eds), *Progress in Machine Learning*, Wilmslow, England: Sigma Press, 1987, pp. 218-229.
- Musen, M. A., Conceptual Models of Interactive Knowledge Acquisition Tools. *Knowledge Acquisition*, 1(1):73-88, 1989.
- Pazzani, M.J. & Brunk, C.A., Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning. *Knowledge Acquisition*, 3:157-173, 1991.
- Quinlan, J.R. 1986, Induction of decision trees. *Machine Learning* 1(1):81-106.
- Reinartz, T. & Schmalhofer, F. An Integration of Knowledge Acquisition Techniques and EBL for a Complex Application. This issue.
- Salton, G. & Buckley, C., Improving Retrieval Performance by Relevance Feedback. *J. of ASIS*, 41:288-297, 1990.
- Shapiro, A.D., *The Role of Structured Induction in Expert Systems*. PhD thesis, University of Edinburgh, 1983.
- Tecuci G. & Kodratoff Y., "Apprenticeship Learning in Imperfect Domain Theories," in Kodratoff Y. and Michalski R.S. (eds), *Machine Learning: An Artificial Intelligence Approach*, vol III, Morgan Kaufmann, 1990.
- Tecuci G., "Automating Knowledge Acquisition as Extending, Updating and Improving a Knowledge Base," *IEEE Transactions on Systems, Man, and Cybernetics*, 22:6, pp. 1444- 1460, November/December 1992.
- Tecuci G. & Hieb M., "Consistency-driven Knowledge Elicitation: Using a Learning-oriented Knowledge Representation that Supports Knowledge Elicitation in NeoDISCIPLINE," *Knowledge Acquisition* , December 1993. vol 5.
- Tecuci, G. & Duff, D. A Framework for Knowledge Base Refinement Through Multistrategy Learning and Knowledge Acquisition, this issue.
- Wilkins, D.C., A Framework for Integration of Machine Learning and Knowledge Acquisition Techniques. *Proceedings of the 6th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Vol. 2, Banff, Canada, October 6-11, 1991.

¹ Dedal communicates with a system with hypertext access to its text and graphic documents. Audio documents are accessible on a Sparc workstation. Video is not yet directly accessible, but indices to video clips are provided to the user who can then view the video on a separate device.

²force-generation-msm stands for force-generation-*mechanism*. In the rest of the paper we will use the abbreviation *msm* for the word *mechanism*.