Condorcet Query Engine: A Query Engine for Coordinated Index Terms

Paul E. van der Vet and Nicolaas J.I. Mars
Knowledge-Based Systems Group, Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands. E-mail: vet@cs.utwente.nl

On-line information retrieval systems often offer their users some means to tune the query to match the level of granularity of the information request. Users can be offered a far greater range of possibilities, however, if documents are indexed with coordinated index concepts. Coordinated index concepts are compound index concepts that express a relation between concepts that function as simple subject descriptors, as in `cures(headache, aspirin)`. We show the point by giving a functional specification of a query engine that can handle both simple and coordinated index concepts, Boolean combinations, and superconcept expansion. The Condorcet Query Engine, a prototype query engine that can be run over the World Wide Web (WWW), demonstrates the feasibility of these ideas. The combination of known and novel techniques ensures a smooth migration path from existing query engines to more advanced engines that handle coordinated index concepts.

1. Querying at Different Levels of Granularity

On-line information retrieval (IR) systems often offer their users some means to tune the query to match the level of granularity of the information request. For instance, several on-line bibliographic databases, like Elsevier's EMBASE (the on-line version of *Excerpta Medica*), offer term expansion. In term expansion, a term in a query is expanded to all its narrower terms as defined in the accompanying thesaurus (EMTREE for EMBASE); the search is performed for each narrower term; and the union of the sets found is returned as a result. Term expansion thus widens the set of selected documents. In a Boolean search, still the most popular search method by far, the familiar operators AND and NOT serve to narrow the set of selected documents while OR serves to widen the set.

A Boolean search is an example of combining index terms to form larger, meaningful wholes. The traditional library designation for this practice is *coordination* (see Foskett, 1982, for an elaborate discussion of these matters). The term *postcoordination* refers to the user making the combinations while searching, as in a Boolean search. *Precoordination*, by contrast, refers to the indexer making the combinations while indexing the documents. For example, the bibliographic database *Engineered Materials Abstracts* (EMA) of Materials Information uses modifiers like "Temperature effects" to obtain index terms such as "Wear rate, Temperature effects."

In existing systems, however, precoordination is used sparingly if at all. This must be called a missed opportunity. The *Unified Medical Language System* (UMLS) of the U.S. National Library of Medicine (NLM, 1997) defines about fifty relations that may hold between index terms. Universal Decimal Classification (UDC), a classification system that predates on-line IR by several decades, specifies rules for making very expressive combinations. Why not use these techniques in IR systems?

The advantage of using relations can be illustrated by a simple example. In a traditional IR system that employs a Boolean search, a document about aspirin as a cure for headache and a document about aspirin as a cause of headache will both be indexed by means of the index terms `aspirin` and `headache`. A user who searches only one of the two has no means to discriminate and will therefore always have to select both in order to find the document sought. To improve the situation, we can express the relation between the two index terms by means of what we call *coordinators*. In the present example, we will employ the coordinators `cures` and `causes`. The document about aspirin as a cure for headache now receives the index term `cures(aspirin, headache)`; the one about aspirin as a cause of headache receives the index term `causes(aspirin, headache)`. Users can limit their query to find only the documents with the relation they seek. In other words, precision is enhanced without any apparent effect on recall (assuming perfect indexing).

The combination of coordinated index terms with a Boolean search and term expansion makes for a very pow-

Received June 2, 1997; revised June 23, 1998; accepted August 4, 1998.

erful search engine. In the course of an IR project called Condorcet, we have developed the Condorcet Query Engine (CQE), a search engine that meets these specifications. Since CQE combines familiar and novel search modes, it offers a smooth migration path from traditional searching to the present, more advanced method. The CQE has been realized with off-the-shelf technology, drawing on knowledge-based techniques. It can be run on the examples used in this article from the Condorcet WWW site at URL:

```
http://wwwis.cs.utwente.nl:8080/kbs/
condorcet/
```

We will be concerned mainly with the functional specification of CQE and the way CQE is currently implemented. Although CQE combines coordination with exact search, there is no principled objection against the combination of coordination and document ranking. We have not investigated this, however.

The way coordinated terms are assigned to documents is not at issue here. For the query engine it does not matter how the terms were assigned. The Condorcet project investigates semiautomatic assignment of coordinated index terms to documents but the realization of practical systems based on this technology is not foreseen for the near future. Currently, controlled terms are assigned by means of manual indexing (for instance at Chemical Abstracts Services). Indexers may assign coordinated terms in a way not very different from that in which they now assign uncoordinated terms.

The organization of the article is as follows. We first discuss the Condorcet project (Section 2), to provide the context within which the present work was done, followed by a discussion of related work (Section 3). In Section 4 we turn to the semantics and syntax of simple and coordinated index concepts as used for indexing and for querying. Section 5 discusses the functional specification of CQE and illustrates the ideas by means of a simple example. Section 6 outlines the current implementation. Section 7 closes with a few concluding remarks.

2. The Condorcet Project

The setting of the present research is the Condorcet project carried out at the Knowledge-Based Systems Group of the University of Twente. We will only briefly outline the project; more information can be found elsewhere in the literature (van Bakel, Boon, Mars, Nijhuis, Oltmans, & van der Vet, 1996; van der Vet & Mars, 1996) and at the project's WWW pages, of which the URL has been given above.

Condorcet takes a controlled-term approach to indexing. Although uncontrolled-term approaches currently dominate IR research, the shortcomings of uncontrolled terms have been identified more than ten years ago by Blair and Maron (1985). Experiences at recent Text Retrieval Conferences (*TREC*) conferences show their doubts are justified. At TREC-3 (Harman, 1995), the best-performing systems typically reached a recall of about 30% at a precision of about 55%. This is impressive given the task and sufficient for many applications, but not sufficient for demanding applications like the one described by Blair and Maron. It appears that sizable improvements are only possible by switching to other, more resource-intensive strategies. The uncontrolled-term approach is also very sensitive to the length of the queries: TREC-4 (Harman, 1996) showed that short queries influence performance negatively.

These results can be understood in terms of what we call the *prediction game*, played by indexers and users of IR systems. Indexers attempt to predict which terms a user will employ when searching for a particular document. In formulating a query, a user attempts to predict which terms occur in the representation of the document he/she seeks. The various IR approaches can all be formulated as strategies to enhance predictability in both directions. Uncontrolled-term approaches hamper predictions by the user because natural language is simply too rich. Controlled terms abstract from the language in which a document happens to be written and thus reduce ambiguity. For the same reason, they allow retrieval of documents using the same terms irrespective of media (text, images, video, sound, or any combination).

Condorcet's strategy in the prediction game employs controlled index *concepts* rather than terms, and proposes to partially automate the indexing process to combat the traditionally high costs of indexing. The distinction between terms and concepts is important, as anyone who has entered keyboard input for a computer program will realize. Suppose the program can handle text and perform arithmetical calculations. When entering the string "115.95" it makes quite a difference whether it is intended to be: (1) A number, or (2) a piece of text that happens to consist of digits and a period. This difference corresponds to the different underlying concepts. Numbers are arithmetical concepts, text constituents are not. Numbers can be added, subtracted, and multiplied, but text constituents cannot. Conversely, there is no unique correct way to write a number as a text constituent: The number given above might just as well have been written as "1.1595 $\times$ 100" and in countless other ways without changing the meaning of the arithmetical concept.

Condorcet exploits recent results in computer science, and in knowledge-based systems research in particular. Workers in these fields have developed logical languages that allow a rich set of relations with formal semantics and computer programs able to process expressions in such languages (see Russell & Norvig, 1994, for a modern textbook). Coordinated index concepts are easily handled by these languages. Index concepts are taken from a predefined concept system or *ontology*, the technical term now commonly used (Hayes, 1985; Gruber, 1993, 1995; Mars, 1995). From an IR perspective, an ontology is like a classification system in that its concepts are subject to stringent syntactic rules and like a thesaurus in that it is specialized for a particular domain. Unlike classification systems and

thesauri, however, ontologies are designed and built for computer manipulation from the start. By making available concepts rather than terms, ontologies help combat natural-language ambiguities that remain even in thesauri. Ontologies also obviate the use of adjacency operators familiar from Boolean searching: A phrase like "upper respiratory tract" leads to a *single* concept in the document representation if it leads to a concept at all.

In order to avoid biases caused by the domain chosen, Condorcet works with two corpora of bibliographic descriptions. The first is taken from EMBASE, the other from *Engineered Materials Abstracts*. The index concepts for the former corpus are taken from the UMLS (NLM, 1997); the coordinators are a subset of the UMLS relations. The ontology we propose for the latter corpus lays down a precise semantics and a formal syntax for index concepts. It is structured in two orthogonal ways. First, it is possible to build up concepts out of other concepts. Such concepts are called composite. We have discussed composite concepts elsewhere (van der Vet & Mars, 1993, 1994, 1998; van der Vet, Speel, & Mars, 1995). Second, as for the EMBASE corpus, index concepts can be coordinated by means of so-called coordinators. This time, however, the coordinators are defined by us.

This article concentrates on the functional specification of a query engine that can handle coordinated concepts, in addition to superconcepts and Boolean combinations. For purposes of exposition, we will employ simplistic examples involving concepts that we will write as `aspirin`, `paracetamol`, `headache`, and a few relations, thus hiding their actual (and possibly composite) shape from view.

3. Related Work

The idea of using precoordination to raise precision arose in the context of an earlier project called Sapiens (Speel, Mars, & van der Vet, 1991) but could not be realized there because of constraints imposed by the funding body. The present proposal combines this idea with our work on ontologies. Other workers have proposed similar ideas, in particular in the context of so-called KL-ONE-like knowledge representation languages (Brachman, 1977; Brachman & Schmolze, 1985). The idea to use a KL-ONE-like language for defining index concepts is straightforward (Brachman, McGuiness, Patel-Schneider, Alperin Resnick, & Borgida, 1991) and elaborations have been reported in the literature. The idea has been put to practice in the MIRTL system (Meghini, Sebastiani, Straccia, & Thanos, 1993; Sebastiani, 1994) where, however, the issue of semantics is not treated satisfactorily. Our approach offers a smooth migration path from traditional Boolean searching to more advanced methods, because only the concepts are richer. By contrast, a KL-ONE-like approach represents a break with tradition so that its introduction in organizations will be more difficult.

A KL-ONE-like approach is particularly suited for building and maintaining a concept hierarchy, relations included. We are not convinced of the superiority of KL-ONE-like systems when it comes to actual use. For one thing, recent empirical work on implemented KL-ONE-like systems has shown that their performance quickly deteriorates as the hierarchy gets larger (Speel, 1995; Speel, van Raalte, van der Vet, & Mars, 1995). This effect is less pronounced for systems whose reasoning is incomplete, but using such a system for IR purposes would mean that we trade increased precision for lower recall. The observations by Speel are corroborated by recent research within our Condorcet project. One of the project collaborators, Jeroen Nijhuis, has implemented a concept hierarchy in the KL-ONE-like knowledge representation language BACK (Hoppe, Kindermann, Quantz, Schmiedel, & Fischer, 1993; Peltason, 1991). BACK certainly makes for very convenient maintenance, but its performance on large concept hierarchies turned out to be insufficient for deployment in real-life IR systems.

4. Coordinated Index Concepts

4.1. Semantics

There are two perspectives on the semantics of index concepts. The easy perspective is that of the query engine. This perspective is adopted by many workers in IR, and we will take it, too, when discussing the query engine below. The query engine perspective interprets any index concept c as the set $S(c)$ of those documents which have c in their representation (the "S" stands for "semantics"). In particular, for Boolean operators we have the familiar relations:

$$S(c_i \text{ OR } c_j) = S(c_i) \cup S(c_j)$$

$$S(c_i \text{ AND } c_j) = S(c_i) \cap S(c_j)$$

$$S(c_i \text{ NOT } c_j) = S(c_i) - S(c_j)$$

where '–' stands for asymmetric set difference.

The query engine perspective can be turned into a formal model–theoretic semantics if wanted, but we will not do that here. The ease of the query engine perspective comes at a price. It assumes perfect knowledge about the representations of the documents in the collection. For obvious reasons, neither indexers nor searchers have this knowledge.

The other perspective is constituted by the prediction game. It is the proper perspective for indexers and searchers. Unfortunately, it involves many unclarities. Both indexers and searchers are said to be guided by the notion of relevance. Relevance, however, is a subjective notion that defies easy formalization. A process that automatically assigns index concepts to documents can be regarded as a formalization of "relevance." But in Condorcet, the process will be *semi*-automatic rather than fully automatic, and even in case of a fully automatic process, the searcher receives only modest assistance from a process specification.

TABLE 1. BNF-specification of index concepts.

index_concept	: :=	simple_concept | coordinated_concept
coordinated_concept	: :=	coordinator "(" simple_concepts ")"
simple_concepts	: :=	simple_concept | simple_concept "," simple_concepts

The symbols simple_concept and coordinator are preterminals. In a concrete application, they are expanded by means of the ontology. The ontology specifies which concepts are valid simple concepts and which are coordinators.

4.2. Syntax of the Document Representation

When a document *d* is indexed with the concept *c*, we will write this as the first-order assertion:

$$\texttt{about}(d, c)$$

meaning, roughly, "document *d* is about *c*." A document representation can contain several concepts $c_1, c_2, \ldots, c_n$. We write such a document representation as a conjunction of about-statements:

$$\texttt{about}(d, c_1) \wedge \texttt{about}(d, c_2) \wedge \cdots \wedge \texttt{about}(d, c_n)$$

Any index concept can be either *simple* or *coordinated.* Simple concepts are obvious: Aspirin and headache are two examples. A coordinated concept is constructed by relating two or more concepts by means of a so-called *coordinator.* A paradigm example is:

```
cures(aspirin, headache)
```

where cures is the coordinator. See Table 1 for a BNF-specification of index concepts.

The concept cures(aspirin, headache) might end up in a document representation if the document is about aspirin as a cure for headache. For this to happen, it is unimportant whether the cure is called effective or worthless. Therefore, a coordinated index concept is *not* an assertion (but the about-statement in which it occurs is). Formally, a coordinator is a function with nominal range and domain. The function cures takes two index concepts as arguments and then returns a new index concept.

4.3. Hierarchies

The ontology from which our index concepts are taken incorporates a number of subconcept–superconcept hierarchies. For instance, all concepts that refer to medicines are subconcepts of the concept medicine. The concept hierarchy governs superconcept expansion by the query engine, see Section 5.3. It is also used to enforce strong typing of the arguments of coordinators. Thus, the coordinator cures takes a first argument that is a subconcept of medicine and a second argument that is a subconcept of disease. The expression:

```
cures(headache, aspirin)
```

then is ill-formed.

Here, we will mimic the concept hierarchy by a set of assertions of the form:

```
subconcept(aspirin, medicine)
```

with an obvious interpretation. Since medicine is a concept like any other, it can be assigned to a document (say, when it discusses medicines in general). The concept hierarchy can be used to increase searching facilities in a way that will be described below.

There are limits to the strength with which typing can be enforced. A typing scheme may come to depend on the state of medical knowledge, so that it has to be revised each time something new is discovered. For a somewhat contrived example, suppose that for the coordinator causes we only allow concepts of type medicine as first argument and concepts of type disease as second argument. The notion of secondary infections, diseases caused by diseases, then necessitates a revision either of the typing scheme or of the ontology (to introduce a new coordinator).

5. Functional Specification of the Query Engine

5.1. Syntax of Queries

In contrast to index concepts, queries may include Boolean combinations and superconcepts that have to be expanded. The full range of queries is specified in Table 2. We will distinguish throughout between *index concepts* (as defined in Table 1) and *search concepts.* A search concept can only occur in a query. It can be syntactically identical to a simple or coordinated index concept, but it can also contain one or more superconcepts that have to be expanded. A search concept on its own can be a query. Search concepts can also be combined by means of Boolean operators, possibly nested, to form a query. Theoretically, there is no limit to the number of nesting levels (but practically, of course, there is).

In this section, we will discuss every kind of query and specify what the query engine has to do. We will employ a

TABLE 2. BNF-specification of queries.

query	: :=	search_concept | boolean_combination
boolean_combination	: :=	"(" query boolean_operator query ")"
boolean_operator	: :=	"AND" | "OR" | "NOT"
search_concept	: :=	c_search_concept | s_search_concept
c_search_concept	: :=	search_coordinator "(" s_search_concepts ")"
search_coordinator	: :=	coordinator | "any(" coordinator ")"
s_search_concepts	: :=	s_search_concept | s_search_concept "," s_search_concepts
s_search_concept	: :=	simple_concept | "any(" simple_concept ")"

A query is either a search concept or a Boolean combination of search concepts. Search concepts come in two sorts: Simple search concepts (called `s_search_concept` in the table) and coordinated search concepts (called `c_search_concept` in the table). Simple and coordinated search concepts are like simple and coordinated index concepts (see Table 1), except that search concepts may contain superconcepts (including super-coordinators) that have to be expanded to all their subconcepts in searching. The preterminals `simple_concept` and `coordinator` are defined as in Table 1.

small example, involving a database of document representations and a small concept hierarchy, to illustrate the query engine's behavior. We thus provide a functional specification of the query engine.

5.2. Basics

The query engine first and foremost has to handle queries of the form:

1. `aspirin`
2. `cures(aspirin, headache)`

(Queries will be numbered consecutively throughout the present section for the purpose of cross-reference.) On query No. 1, the query engine has to return all documents that have the uncoordinated concept `aspirin` in their representation plus documents that have `aspirin` as argument in a coordinated concept. After all, if a document is about, say, aspirin as a cure for headache, then certainly it is about aspirin. Query No. 2, on the other hand, only retrieves documents that have the coordinated concept as such in their representation. In fact, the query engine returns sets of keys of documents, to be called *result sets* here, rather than documents themselves. This is evident to IR researchers and we will often use the less precise terminology here.

A small example database is specified in Table 3. It refers to documents by means of their unique keys. For this database, the result sets returned by the query engine are as follows:

1. `aspirin` returns 112 through 115 inclusive, 117, and 118
2. `cures(aspirin, headache)` returns 114 and 115

5.3. Superconcept Expansion

The concept hierarchy allows queries that involve superconcept expansion. In traditional settings, a simple query like `medicine` is ambiguous. The user may seek only documents about medicines in general, or those plus documents about particular medicines. The distinction is made in Condorcet by means of a special syntax. Consider the two queries:

3. `medicine`
4. `any(medicine)`

Query No. 3 is like query No. 1 and retrieves only documents whose representation includes the concept `medicine` as such. Query No. 4 is different. It instructs the query engine to expand `medicine` to the set that consists of all of its subconcepts plus the concept `medicine` itself, to treat each concept thus found in the manner of query No. 1, and to return the union of all sets found as the result set.

Whether a concept is a superconcept or not is entirely determined by the concept hierarchy: If a concept occurs as the second argument in a `subconcept`-statement in the hierarchy, it is a superconcept of at least one other concept. The search concept of query No. 3 is a superconcept in this sense. To make the special status evident of search concepts like the one in query No. 4, we will call such concepts *expandable superconcepts*.

We have specified a small concept hierarchy in Table 4. Combined with the example database of Table 3, the result sets returned by the query engine are:

3. `medicine` returns only 108
4. `any(medicine)` returns all documents from the database except 109

TABLE 3. Example database of document representations.

```
about(108, medicine).
about(109, headache).
about(110, paracetamol).
about(112, aspirin).
about(113, aspirin).
about(114, cures(aspirin, headache)).
about(114, causes(paracetamol, nephritis)).
about(115, cures(aspirin, headache)).
about(116, cures(paracetamol, headache)).
about(117, causes(aspirin, headache)).
about(118, cures(aspirin, carditis)).
```

TABLE 4. Example concept hierarchy.

```
subconcept(aspirin, medicine).
subconcept(paracetamol, medicine).
subconcept(headache, disease).
subconcept(carditis, disease).
subconcept(nephritis, disease).
subconcept(cures, coordinator).
subconcept(causes, coordinator).
```

5.4. Combining Coordination and Superconcept Expansion

As explained in Section 1, superconcept expansion as specified above is already put to practice in existing systems such as EMBASE. In our design, however, coordination can be combined with superconcept expansion to allow searching within a large range of granularity. Every constituent of a coordinated search concept, including the coordinator itself, can be an expandable superconcept. See Table 4, where the coordinator `coordinator` is a superconcept that has the by now familiar coordinators `cures` and `causes` as subconcepts. In a coordinated search concept with expandable superconcepts, each occurrence of an expandable superconcept is combinatorially expanded to yield all subconcepts.

We illustrate the idea by means of five queries, where the result sets are included:

5. `cures(any(medicine), headache)` returns 114, 115, and 116
6. `cures(any(medicine),any(disease))` returns 114, 115, 116, and 118
7. `any(coordinator) (aspirin, headache)` returns 114, 115, and 117. For the present example, the subconcepts found by superconcept expansion are `cures-(aspirin, headache)` and `causes (aspirin, headache)`.
8. `any(coordinator) (any(medicine), headache)` returns 114 through 117 inclusive
9. `any(coordinator) (any(medicine), any(disease))`, finally, returns 114 through 118 inclusive

Search expressions are seen to depend on the order in which the arguments occur in document representations. Suppose, for an example, that there is also a coordinator called `strange`. This new coordinator takes a first argument that is a subconcept of `disease` and a second argument that is a subconcept of `medicine`. Then, if there is a document that has `strange(headache, aspirin)` in its representation, query No. 7 will not find it. The problem, if it is one, can be avoided by choosing coordinators in a consistent way or by employing a logic that abstracts from argument order, like a feature logic.

5.5. Boolean Search

To round off, we require that the query engine be able to handle expressions involving Boolean operators in the usual manner (compare Section 4.2). In this respect, our query engine is just like any other Boolean query engine except that, in our engine, the concepts can be any of those like in queries 1 through 9 above. Since the idea is sufficiently clear, we round off with a single query:

10. `aspirin NOT any(coordinator) (aspirin, any(disease))` returns 112 and 113

6. Implementation

6.1. Outline

The functional specification of the query engine leans heavily on a Prolog-like query handler, so it comes as no surprise that the implementation of the prototype in Prolog turned out to be remarkably easy. See Sterling & Shapiro (1994) for an excellent textbook on Prolog. The result, CQE, can be inspected and run at Condorcet's WWW site specified above.

The WWW interface contains a dialog box in which a user can specify a search expression of the form exemplified by queries 1 through 10 in the previous section. The search expression is evaluated by the underlying Prolog programs in the form of the Prolog query:

```
search(Query, Results).
```

where `Query` is a query as specified in Table 2. `Results` is a variable that is unified with the result set (in Prolog jargon: The list of keys that correspond to `Query`).

The CQE consists of three modules: A *parser*, a *search concept handler*, and a *superconcept expander*. Access to the engine is provided by the parser, which calls the search concept handler at appropriate places. The search concept handler, in turn, calls the superconcept expander if it encounters an expandable superconcept.

`Query` is first handed over to the CQE parser. Employing a Definite Clause Grammar, it analyses `Query` until it arrives at the level of what Table 2 calls search concepts, i.e., concepts that lack Boolean operators; examples are queries 1 through 9 of Section 5. At that point, the search concept handler is called to retrieve the list of keys that correspond to the search concept. The parser recognizes Boolean combinations by means of a simple grammar of queries that closely resembles Table 2. Nested Boolean combinations are recognized. Currently, the order of evaluation is determined by parentheses. In its nodes, the parser assembles lists of keys of documents. If a Boolean combination is recognized by a grammar rule, the definite clause that accompanies the grammar rule performs the appropriate set-theoretical operation (union, intersection, and asymmetric set difference; compare Section 4.1.). In this way, the parser assembles `Result`, the list of keys that correspond to `Query`, in the top node. In the simplest case, if `Query` consists of a single search concept, all the parser does is handing over the search concept to the search concept

handler and storing the list of keys returned by the search concept handler directly in its top node. Like any other parser, the CQE parser fails if the query is syntactically incorrect.

The search concept handler makes a distinction between search concepts that are syntactically identical to index concepts and search concepts that contain expandable superconcepts. Concepts of the former kind occur as such in document representations. Examples of queries that consist of this kind of concepts only are queries 1, 2, and 3 in Section 5. An inverted file directly maps such concepts onto lists of keys. The search concept handler consults the inverted file to obtain a list of keys, returns the list to the parser, and is done. Expandable superconcepts like in queries 4 through 9, by contrast, are handed over to the superconcept expander. The superconcept expander combinatorially constructs all search concepts that are subconcepts of the original search concept and returns the list to the search concept handler. The search concept handler now consults the inverted file for each search concept in the list, determines the union of the results, and returns this to the parser.

6.2. Discussion

The CQE is intended as a prototype for demonstration purposes only. We are confident that it can be implemented to match the runtime performance of state-of-the-art query engines that employ inverted files. This also holds for superconcept expansion, even though this process may take a lot of time. The time needed for superconcept expansion is a problem. Most users of IR systems refine their queries interactively, and small response times greatly facilitate smooth interaction.

Unfortunately, superconcept expansion is an inherently costly process no matter which query engine is at issue. It involves finding all subconcepts of a given superconcept, followed by determining the union of a possibly large number of sets. One way to ease the problem is to trade storage efficiency for runtime efficiency (after all, static storage is becoming cheaper by the week). The idea is to run the search concept handler and superconcept expander in compile time to determine the result sets of every possible search concept that contains expandable superconcepts. The outcome is used to simply include all such search concepts in the inverted file, too. This results in a very large inverted file, which negatively affects response times, but not too much. This method works for simple hierarchies like the one of Table 4, but not for the very intricate hierarchies spanned in ontologies with composite concepts (van der Vet & Mars, 1998). Another way to ease the problem is to implement the concept hierarchy in an imperative language like C or Pascal, using pointer structures. This undoubtedly gives optimal (but still possibly large) response times, but it runs counter to established rules of good practice in software engineering. The reason is that such data structures are very hard to maintain. Changes in the concept hierarchy can easily lead to errors or even inconsistencies. Within the knowledge-based systems community, a major research effort is devoted to the development of systems that support careful maintenance of concept hierarchies. A prime example is formed by the class of systems based on KL-ONE-like languages. As we have explained in Section 3, however, they have other drawbacks.

The other drawback of CQE is its user interface. It is unfriendly. It also lacks a conformance checker that immediately rejects a query like:

```
cures(headache, aspirin)
```

on the ground that it is ill-formed. (The CQE instead simply returns an empty list of keys.) One way to improve the interface is to offer the user menus of concepts which can be coordinated by clicking on coordinators in a menu. This would also obviate the need for a conformance checker, because the underlying program can be designed such that it constructs only well-formed queries. On the other hand, and apparently contrary to the received view (as expressed by, for instance, Sparck Jones, 1991), we think that in particular searchers with a scientific background will have little trouble with, and may well prefer, queries written in a logical language. Conformance checking is easy and can be added.

Another extension that has to be realized in the near future is an interface that communicates with the outside world according to the Z39.50 protocol.

7. Concluding Remarks

We have argued that the sparing use of precoordination constitutes a missed opportunity. Coordinated index concepts and queries allow for searching at many levels of granularity, so that the searcher can fine-tune the search expression to a far larger degree than hitherto perceived. The possibilities span a range from queries more fine-grained than simple Boolean expressions through the use of coordinators to queries that are more coarse-grained through the use of superconcepts which may be expanded or not at the searcher's discretion. Better tuning of the query will lead to higher precision under equal circumstances.

The accompanying implementation CQE demonstrates that the improvement can be realized now. Further steps are taken in the Condorcet project, which has set itself the aim of developing a system able to semiautomatically assign coordinated index concepts to documents.

Acknowledgments

The authors are indebted to the other Condorcet team members: Bas van Bakel, Reinier Boon, Jeroen Nijhuis, and Erik Oltmans for their contributions to discussions on coordinated index concepts and their comments on the present article. The prototype of CQE has been turned into a decent set of Prolog programs by Reinier Boon, who also designed and implemented the WWW interface. The Condorcet project is a 4-year project funded by the Dutch Technology

Foundation (STW).

References

Blair, D.C., & Maron, M.E. (1985). An evaluation of retrieval effectiveness for a full-text document-retrieval system. Communications of the ACM, 28, 289–299.

Brachman, R.J. (1977). What's in a concept: Structural foundations for semantic networks. International Journal of Man-Machine Studies, 9, 127–152.

Brachman, R.J., McGuiness, D.L., Patel-Schneider, P.F., Alperin Resnick, L., & Borgida, A. (1991). Living with CLASSIC: When and how to use a KL-ONE-like language. In J.F. Sowa (Ed.), Principles of semantic networks. Explorations in the representation of knowledge (pp. 401–456). San Mateo, CA: Morgan Kaufmann.

Brachman, R.J., & Schmolze, J.G. (1985). An overview of the KL-ONE knowledge representation system. Cognitive Science, 9, 171–216.

Foskett, A.C. (1982). The subject approach to information. London: Clive Bingley.

Gruber, T.R. (1993). A translation approach to portable ontology specifications. Knowledge Acquisition, 5, 199–220.

Gruber, T.R. (1995). Towards principles for the design of ontologies used for knowledge sharing. International Journal of Human-Computer Studies, 43, 907–928.

Harman, D.K. (Ed.) (1995). The third text retrieval conference (TREC-3). NIST Special Publication 500-225. Gaithersburg MD: U.S. Department of Commerce, National Institute of Standards and Technology.

Harman, D.K. (Ed.) (1996). The fourth text retrieval conference (TREC-4). NIST Special Publication 500-236. Gaithersburg MD: U.S. Department of Commerce, National Institute of Standards and Technology.

Hayes, P.J. (1985). Naive physics I: Ontology for liquids. In J.R. Hobbs & R.C. Moore (Eds.), Formal theories of the commonsense world (pp. 71–107). Norwood, NJ: Ablex.

Hoppe, T., Kindermann, C., Quantz, J.J., Schmiedel, A., & Fischer, M. (1993). BACK V5 Tutorial & Manual. KIT-Report 100. Berlin: Technische Universität Berlin.

Mars, N.J.I. (1995). What is an ontology? In A. Goodall (Ed.), The impact of ontologies on reuse, interoperability and distributed processing (pp. 9–19). Uxbridge, Middlesex, UK: Unicom.

Meghini, C., Sebastiani, F., Straccia, U., & Thanos, C. (1993). A model of information retrieval based on a terminological logic. In Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (pp. 298–307), Pittsburgh, PA.

NLM (1997). Unified Medical Language System knowledge sources, eighth experimental edition. Bethesda MD: U.S. National Library of Medicine.

Peltason, C. (1991). The BACK system—An overview. SIGART bulletin, 2, 114–119.

Russell, S., & Norvig, P. (1994). Artificial intelligence: A modern approach. Englewood Cliffs, NJ: Prentice Hall.

Sebastiani, F. (1994). A probabilistic terminological logic for modelling information retrieval. In W.B. Croft & C.J. van Rijsbergen (Eds.), Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval (pp. 122–130). London, UK: Springer.

Sparck Jones, K. (1991). The role of artificial intelligence in information retrieval. Journal of the American Society for Information Science, 42, 558–565.

Speel, P.-H. (1995). Selecting knowledge representation systems. Ph.D. thesis, University of Twente, Enschede, the Netherlands.

Speel, P.-H., Mars, N.J.I., & van der Vet, P.E. (1991). A knowledge-based approach to semi-automatic indexing. In A. McCray (Ed.), Proceedings of the Workshop on Language and Information Processing, October 27, 1991, Washington, DC (pp. 49–58).

Speel, P.-H., van Raalte, F., van der Vet, P.E., & Mars, N.J.I. (1995). Scalability of the performance of knowledge representation systems. In N.J.I. Mars (Ed.), Towards very large knowledge bases. Knowledge Building and Knowledge Sharing 1995 (pp. 173–183). Amsterdam: IOS Press.

Sterling, L., & Shapiro, E. (1994). The art of Prolog. Advanced programming techniques. Cambridge, MA: MIT Press.

Van Bakel, B., Boon, R.T., Mars, N.J.I., Nijhuis, J., Oltmans, E. & van der Vet, P.E. (1996). Condorcet annual report. September 1996. Memoranda Informatica 96-12. Enschede, the Netherlands: University of Twente.

Van der Vet, P.E., & Mars, N.J.I. (1993). Structured system of concepts for storing, retrieving, and manipulating chemical information. Journal of Chemical Information and Computer Sciences, 33, 564–568.

Van der Vet, P.E., & Mars, N.J.I. (1994). Concept systems as an aid for sharing and reuse of knowledge bases in materials science. In J.K. McDowell & K.J. Meltsner (Eds.), Knowledge-based applications in materials science and engineering (pp. 43–55). Warrendale, PA: The Minerals, Metals & Materials Society.

Van der Vet, P.E., & Mars, N.J.I. (1996). Kennistechnologie en information retrieval. In G.M. van Trier, H. Prins & D.W.K. Jansen (Eds.), Handboek Informatiewetenschap voor bibliotheek en archief, deel I (pp. 300-1–300-23). Houten, the Netherlands: Bohn Stafleu Van Loghum.

Van der Vet, P.E., & Mars, N.J.I. (1998). The bottom-up approach to building ontologies. IEEE Transactions on Knowledge and Data Engineering, 10, 513–526.

Van der Vet, P.E., Speel, P.-H., & Mars, N.J.I. (1995). Ontologies for very large knowledge bases in materials science: A case study. In N.J.I. Mars Ed.), Towards very large knowledge bases. Knowledge Building and Knowledge Sharing 1995 (pp. 73–83). Amsterdam: IOS Press.